Description of the *graphComp* package: Visual comparison of graphs on the same node set

Khadija El Amrani [*][†]   Ulrich Mansmann[*]

June 3, 2011

Contents

[*]Division of Biometrics and Bioinformatics, IBE, University of Munich, 81377 Munich, Germany

[†]Package maintainer ,Email: `Khadija.Amrani@campus.lmu.de`

1 Abstract

The *graphComp* package for R [8] provides functions for visual comparison of graphs. It is designed to compare graphs defined on the same set of nodes.

This vignette demonstrates how the package *graphComp* can be used. For this purpose, a comparison of estimated graphs with the packages *GeneNet* [7], *pcalg*[6] and *glasso* [3] is performed. These packages are available at *the Comprehensive R Archive Network* (CRAN) at `http://cran.r-project.org`.

2 Introduction

The *graphComp* package contains functions for visual graph comparison. The package is designed to compare graphs defined on the same set of nodes. Given two *graphNEL* graphs, the *graphComp* package enables the visualization and comparison of the input graphs in a merged graph, and allows the analyst to visually compare the differences and similarities of the graphs that are distinguished by colors. In order to compare the proportion of degrees of nodes in both graphs, the nodes are represented as pie charts.

For a realistic estimation of graphs a real data set from ALL data is used. The data are available in the R package *ALL*. To reduce computation time a small data set to B-cell ALL is selected.

```
> library(ALL)
> data("ALL")
> ALL

ExpressionSet (storageMode: lockedEnvironment)
assayData: 12625 features, 128 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 01005 01010 ... LAL4 (128 total)
  varLabels: cod diagnosis ... date last seen (21 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
```

```
  pubMedIds: 14684422 16243790
Annotation: hgu95av2
```

The `ALL` data consist of micorarrays from 128 different individuals with acute lymphoblastic leukemia (ALL). There are 95 samples with B-cell ALL and 33 with T-cell ALL. These are different tissues and quite different diseases. A number of additional covariates is available, but not used in this example. The focus here will be on B-cell ALL tumors. The data have been normalized (using rma [5]) and are presented in the form of an 'ExpressionSet' object. The data set is available in the package *ALL* and more information can be found in [2].

To select the corresponding data set to B-cell ALL the information from the annotation slot 'BT' is used.

```
> BcellIds <- grep("^B", ALL$BT)
> ALL_B <- ALL[, BcellIds]
> dim(ALL_B)

Features  Samples
   12625       95
```

2.1 Nonspecific filtering

The function `nsFilter()` is available in the package *genefilter*. It identifies and removes features that appear to be less informative. Use cases for this function are: variable selection for subsequent sample clustering or classification tasks; independent filtering of features used in subsequent hypothesis testing, with the aim of increasing the detection rate.

```
> library(genefilter)
> library(hgu95av2.db)
> nALL_B <- nsFilter(ALL_B)
> dim(nALL_B$eset)

Features  Samples
    4399       95
```

2.2 Gene selection

To reduce further computation time, for clearness and for biological interpretation, only some genes belonging to the p53 signaling pathway will be analysed. This pathway play crucial roles in tumor development. It is available at the KEGG database and has the ID 'hsa04115'. The following code extracts the probe names for the p53 signaling pathway from the KEGG database and builds an expression matrix with the selected probes.

```
> library(hgu95av2.db)
> xx <- as.list(hgu95av2PATH2PROBE)
> ids <- xx$"04115"
> mat_B <- exprs(nALL_B$eset)
> ids <- ids[ids %in% rownames(mat_B)]
> mat_B <- mat_B[ids, ]
> dim(mat_B)
```

```
[1] 40 95
```

The following function is used to translate the probe names into gene names:

```
> get.genes <- function(IDs = c("")) {
+     result <- 1:length(IDs)
+     require(hgu95av2.db)
+     xx <- as.list(hgu95av2SYMBOL)
+     for (i in 1:length(IDs)) {
+         if (is.na(xx[[IDs[i]]]) == TRUE) {
+             result[i] <- IDs[i]
+         }
+         else {
+             result[i] <- xx[[IDs[i]]]
+         }
+     }
+     return(result)
+ }
> genes <- get.genes(rownames(mat_B))
> length(rownames(mat_B))
```

```
[1] 40
```

```
> length(genes)

[1] 40
```

In this example each gene is represented by one probe set, therefore the rownames of the matrix could be replaced by genes. If there is at least one gene that is represented by more than one probe set, you should estimate the probe set graph and translate it in gene graph.

```
> rownames(mat_B) <- get.genes(rownames(mat_B))
```

3 Estimating Graphs

In the following, the gene-graphs to the data set will be estimated with the above mentioned methods.

3.1 GeneNet

The input data must be arranged in a matrix where columns correspond to genes and rows correspond to individual measurements. For this purpose, the matrix should be transposed. Then the partial correlation matrix is estimated with the function `ggm.estimate.pcor()`. An edge is considered to be ”significant” if the probability of an edge to be ”present” is larger than a given value. A graph object must be generated containing all significant edges. The function `geneNetGraph()` given below gets a matrix and a value *pval* (between 0 and 1) as parameters. It returns the estimated graph to the given matrix with *GeneNet* [7]. The estimated graph is visualized in Figure˜1.

```
> geneNetGraph <- function(matrix, pval) {
+     require(GeneNet)
+     corr <- ggm.estimate.pcor(t(matrix))
+     results <- ggm.test.edges(corr, plot = F)
+     sig <- results$prob > pval
+     geneNetGr <- ggm.make.graph(results[sig, ], colnames(t(matrix)))
+     return(geneNetGr)
+ }
```

We use the value 0.8 for the estimation of the graph.

```
> geneNetGr <- geneNetGraph(mat_B, pval = 0.8)
> require(Rgraphviz)
> plot(geneNetGr, main = "B-cell ALL")
```

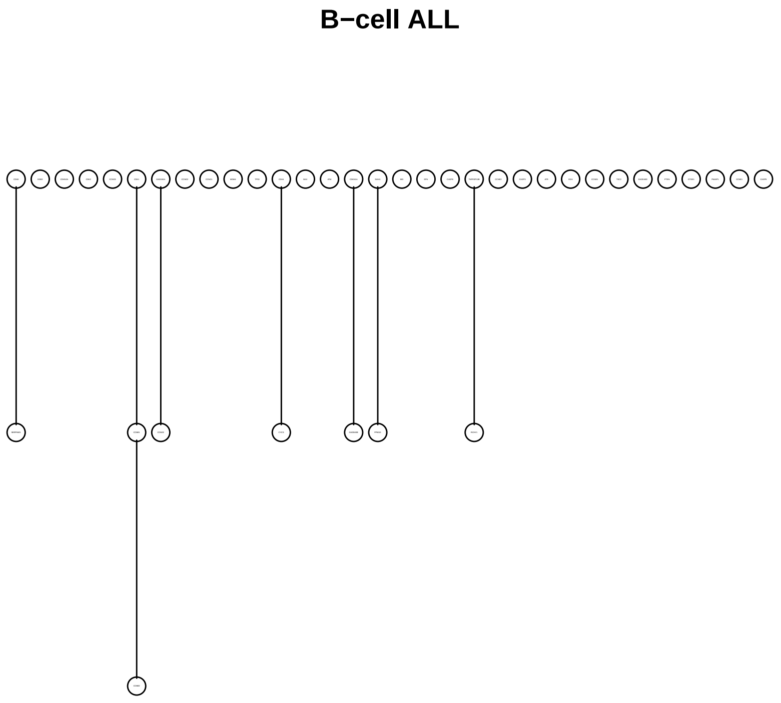


Figure 1: The estimated graph with *GeneNet*

3.2 PC-Algorithm

The PC-Algorithm is a method for estimating the skeleton of a very high-dimensional Directed Acyclic Graph (DAG) with corresponding Gaussian distribution [6]. The skeleton of a DAG G is the undirected graph obtained from G by substituting undirected edges for directed edges. `alpha` is a parameter of significance level for the individual partial correlation tests. The following function `pcGraph()` returns the estimated graph with the PC-Algorithm. This graph is visualized in Figure˜2.

```
> pcGraph <- function(mat, alpha) {
+     require(pcalg)
+     pcGr <- pcAlgo(t(mat), alpha = alpha)
+     pcGr <- pcGr@graph
+     nodes(pcGr) <- rownames(mat)
+     return(pcGr)
+ }
> pcGr <- pcGraph(mat_B, alpha = 0.05)
```

```
This function is deprecated and is only kept for backward compatibility.
 Please use skeleton, pc, or fci instead
```

```
> plot(pcGr, main = "B-cell ALL")
```

3.3 glasso

In the following, a function `lassoGraph()` is given. This function returns for a given matrix the estimated graph with `glasso()`. The function `cov()` returns the corresponding covariance matrix for the given matrix. The graph estimated with the glasso-Algorithm [3] is visualized in Figure˜3.

```
> lassoGraph <- function(matrix, rho){
+ library(glasso)
+ c <- cov(t(matrix))
+ gl <- glasso(c, rho=rho, zero=NULL, thr=1.0e-4,
+       maxit=1e4, approx=F, penalize.diagonal=TRUE)
+ wi <- gl$wi
+ for(i in 1:length(wi[1,])){
+ for(j in 1:length(wi[,1])){
+ if(wi[j,i]!=0) {
+ wi[j,i]=1
+ }
+
+ }
+ }
+ diag(wi)=0
+ g_las <- as(wi,"graphNEL")
+ nodes(g_las) <- rownames(matrix)
```

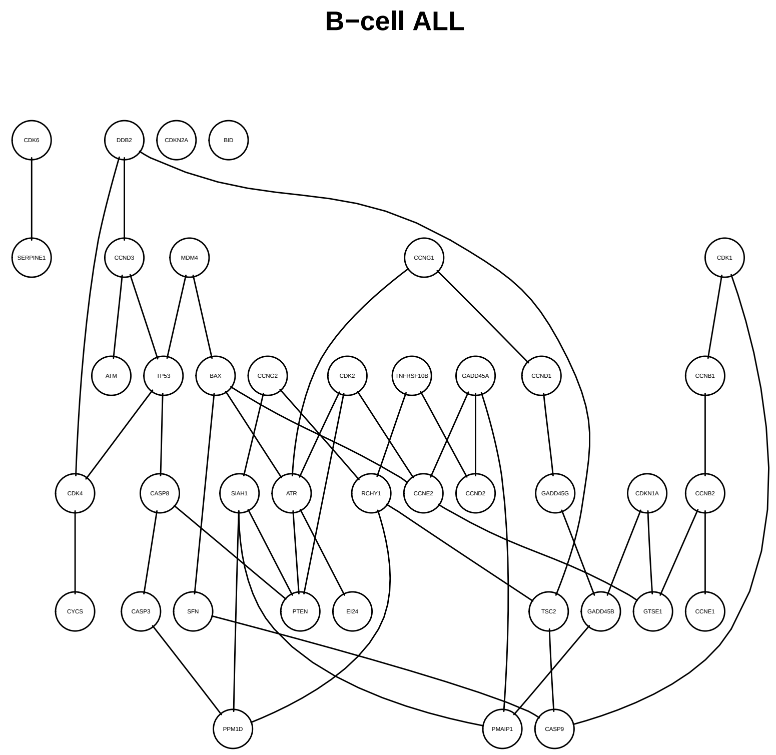


Figure 2: The estimated graph with PC-Algorithm

```
+ return(g_las)
+ }
```

The following code chunk plots the estimated graph.

```
> lassoGr <- lassoGraph(mat_B, 0.15)
> plot(lassoGr, main = "B-cell ALL")
```

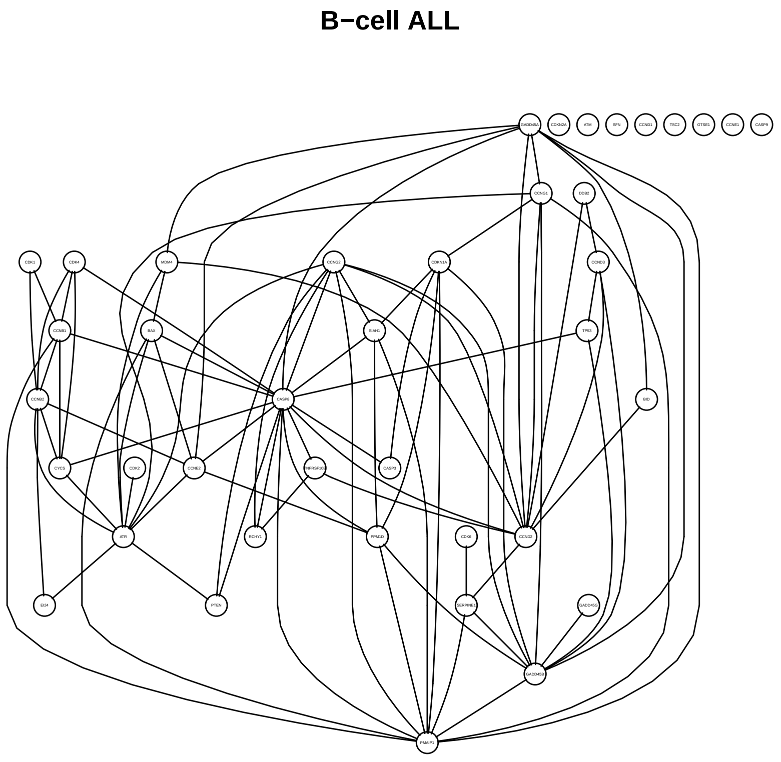


Figure 3: The estimated graph with *glasso*

4 Function Description

4.1 Function: compare2Graphs()

The `compare2Graphs()` function is a routine to compare two graphs on the same node set. The function needs at least the mandatory parameter `graphList`, this is a list of two *graphNEL* graphs. For more details about the optional parameters, see the help files. The function returns an *Ragraph* object that can be further used as input parameter to the following subfunctions: `get.edges()`, `get.hubs()` and `getMixed.hubs()` to get edges or hubs. Additionally the function plots the graphs in a merged graph where edges of both graphs and common edges are distinguished by colors. It prints the number of edges to the used colors and the total number of edges in the comparative graph. In order to compare the proportion of degrees of nodes in both graphs, the nodes are represented as pie charts using the function `pieGlyph()` from the package *Rgraphviz* [4]. This representation of nodes is used in all implemented functions.

```
> compare2Graphs(list(graph1, graph2))
```

4.1.1 Function: get.edges()

The function `get.edges()` needs an *Ragraph* object (returned from the function `compare2Graphs()`) and one color from the color list, which is used by the function `compare2Graphs()` to color the edges of the comparative graph, as input parameters (both parameters are necessary). This function returns edges of the given *Ragraph* object that are colored with the given color.

4.1.2 Function: get.hubs()

The function `get.hubs()` needs an *Ragraph* object (returned from the function `compare2Graphs()`), one color from the color list, which is used by the function `compare2Graphs()` to color the edges of the comparative graph and a threshold value p (value between 1 and 100, default: 20) as input parameters. This function returns nodes which highly interact, the hubs. A node is considered a hub if it is incident to at least p percent of the total number of edges in the comparative graph. In contrast to the function `getMixed.hubs()`, this function returns hubs incident to edges colored with the given color only.

4.1.3 Function: getMixed.hubs()

The function `getMixed.hubs()` is similar to the function `get.hubs()`, however it returns hubs incident to edges colored with at least two colors. Therefore, the color parameter is not necessary.

4.2 Function: compGraphs.vis()

The function `compGraphs.vis()` needs only the mandatory parameter *graphList*. The other optional parameters could be passed via the control panel that appears after calling the function. The control panel is constructed using functions from the *rpanel* package [1]. Additionally to the visualization of the comparative graph, this function enables the visualization of the graph with edges that exist in only one of the compared graphs and the graph with common edges only. In this way, a side-by-side visual comparison of graphs is also provided.

4.3 Function: compGraphs.interactive()

The function `compGraphs.interactive()` needs at least the mandatory parameter *graphList* (list of two *graphNEL* graphs). For more details about the optional parameters, see the help files. Additionally to the visualization of the comparative graph, this function offers the user the possibility to click on any node to visualize the subgraph with the clicked node and its direct neighbors in a new window.

5 Graphical comparison

Now the generated graphs in section ˜3 will be compared with the implemented methods in the *graphComp* package.

5.1 Function: compare2Graphs

5.1.1 Comparison of the *GeneNet* and PC graphs

The following code chunk compares the *GeneNet* and PC graphs. The returned *Ragraph* graph from the function `compare2Graphs()` is `Ragr`.

```
> library(graphComp)

Package `rpanel', version 1.0-6
type help(rpanel) for summary information

> Ragr <- compare2Graphs(list(geneNetGr, pcGr), cexx=1.5,
+ graphTitle="The comparative graph of the GeneNet and PC graphs",
+ legendGr1="Edges of GeneNet graph",
+ legendGr2="Edges of the PC graph")

   Color Number.edges
1   blue            0
2 green4           41
3    red            8
4    sum           49
```

The resulting graph is visualized in Figure~4. The *GeneNet* and PC graphs have 8 edges in common that are colored red. The edges of PC graph are colored green. There are no edges that exist only in the *GeneNet* graph and do not exist in the PC graph, therefore the color for these edges is missing in the comparative graph. The nodes are represented as pie charts to illustrate the proportion of colored edges that are incident to each node.

5.1.2 Comparison of the PC and *glasso* graphs

The following code chunk compares the PC and *glasso* graphs.

```
> compare2Graphs(list(pcGr, lassoGr), cexx=1.5,
+ graphTitle="The comparative graph of the PC and glasso graphs",
+ legendGr1="Edges of PC",legendGr2="Edges of glasso graph",
+ legendPosition="bottomleft")

   Color Number.edges
1   blue           21
2 green4           55
3    red           28
4    sum          104
[1] "A graph with 40 nodes."
```

The resulting graph is visualized in Figure~5. The graphs of PC and *glasso* have 29 edges in common and other edges that exist only in one of the graphs. Therefore, the edges of the comparative graph are colored with three colors: red for common edges, blue for edges of the PC graph and green for edges of the *glasso* graph.

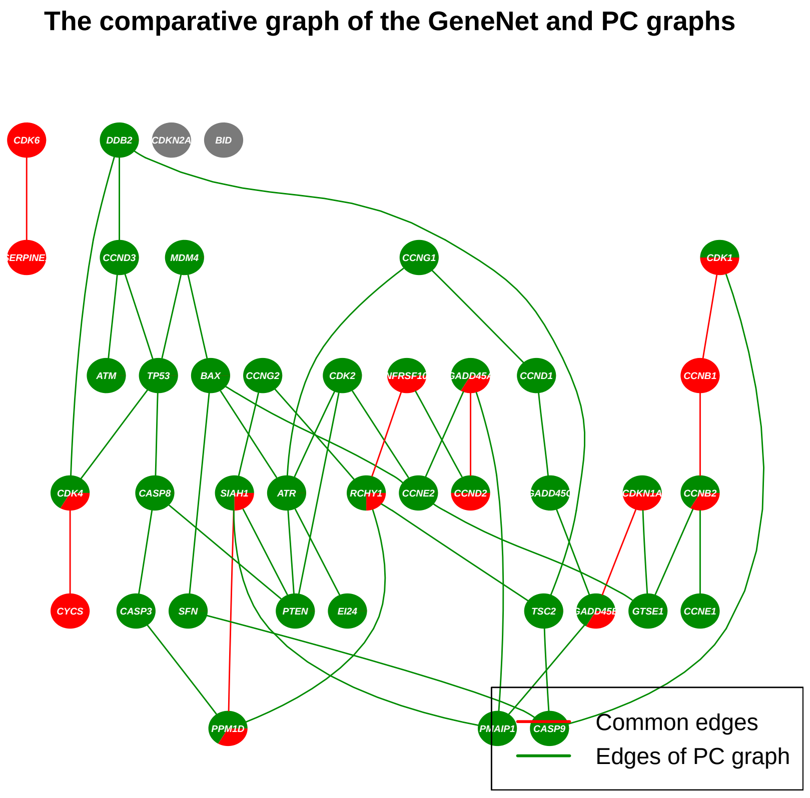


Figure 4: The comparative graph of *GeneNet* and PC graphs

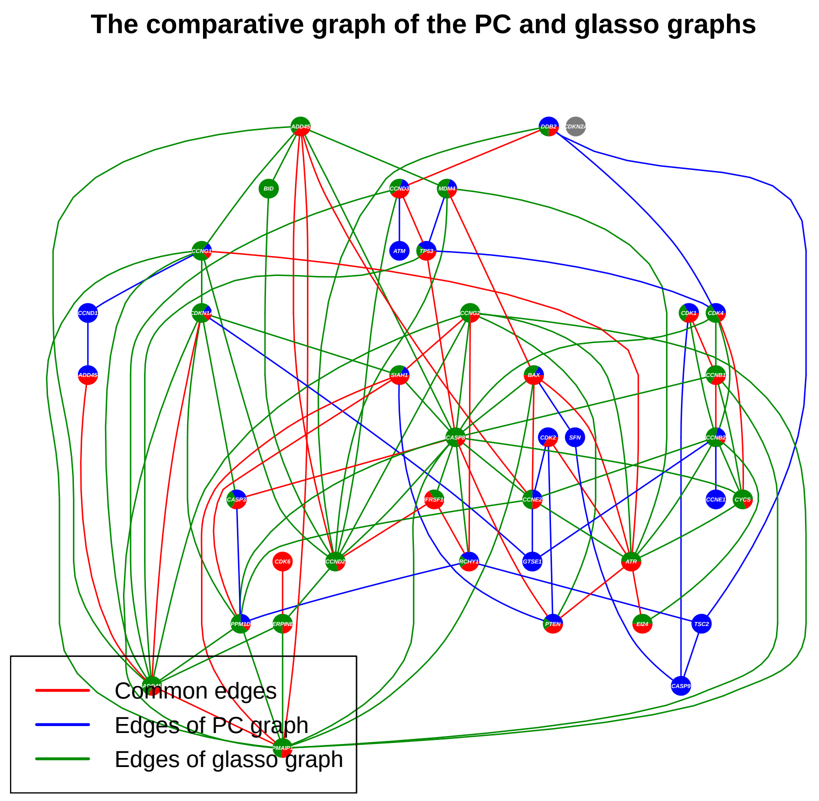


Figure 5: The comparative graph of PC and *glasso* graphs

5.1.3 Comparison of the *GeneNet* and *glasso* graphs

The following code chunk compares the *GeneNet* and *glasso* graphs. The *GeneNet* and *glasso* graphs have 8 edges in common that are colored red. The edges of *glasso* graph are colored green. There are no edges that exist only in the *GeneNet* graph and do not exist in the *glasso* graph, therefore the color for these edges is missing in the comparative graph.

```
> compare2Graphs(list(geneNetGr, lassoGr), cexx=1.5,
+ graphTitle="The comparative graph of the GeneNet and glasso graphs",
+ legendGr1="Edges of GeneNet graph",
+ legendGr2="Edges of glasso graph")
```

```
   Color Number.edges
1   blue            0
2 green4           75
3    red            8
4    sum           83
[1] "A graph with 40 nodes."
```

The resulting graph is visualized in Figure˜6.

In the following, the returned graph from `compare2Graphs()` by the comparison of *GeneNet* and PC graphs is given as parameter to the functions `get.edges()`, `get.hubs()` and `getMixed.hubs()`.

5.1.4 Function: get.edges()

The function `get.edges()` is called with the returned graph from `compare2Graphs()` by the comparison of *GeneNet* and PC graphs and the color blue as parameters to get edges that exist only in the *GeneNet* graph. The returned value is NULL because there are no edges that are present only in the *GeneNet* graph, see Figure˜4

```
> get.edges(Ragr, EdgColor = "blue")
```

```
NULL
```

The following function call returns edges of the PC graph.

```
> get.edges(Ragr, EdgColor = "green4")
```

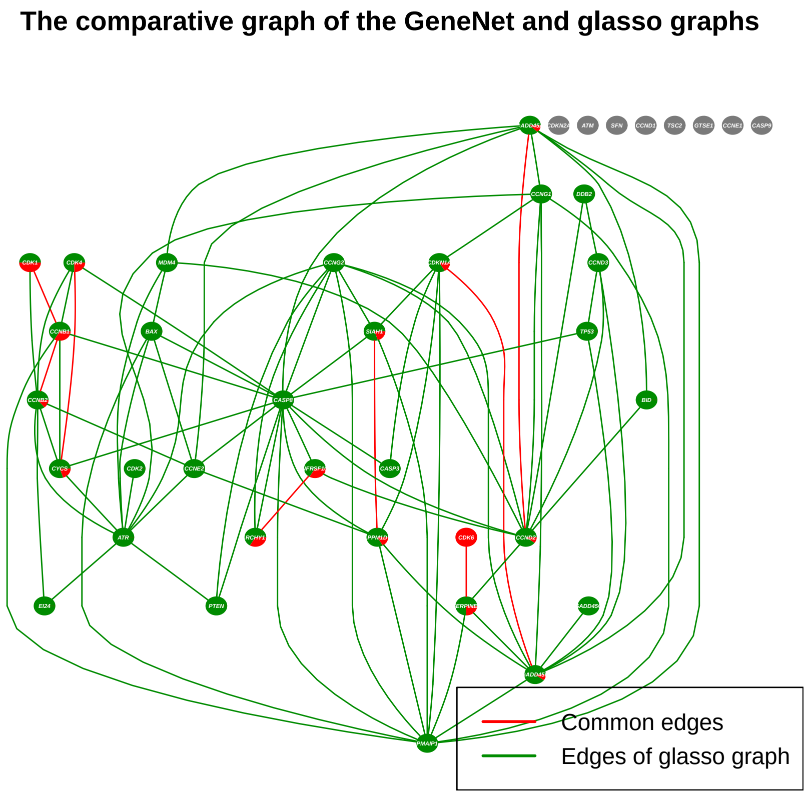


Figure 6: The comparative graph of *GeneNet* and *glasso*

```
 [1] "CCND3~DDB2"       "CDK4~DDB2"        "TSC2~DDB2"
 [4] "CCNE2~CDK2"       "ATR~CDK2"         "PTEN~CDK2"
 [7] "TP53~CCND3"       "ATM~CCND3"        "CASP9~CDK1"
[10] "CCNE2~GADD45A"    "PMAIP1~GADD45A"   "SIAH1~CCNG2"
[13] "RCHY1~CCNG2"      "ATR~CCNG1"        "CCND1~CCNG1"
[16] "TP53~MDM4"        "BAX~MDM4"         "CDK4~TP53"
[19] "CASP8~TP53"       "SFN~BAX"          "CCNE2~BAX"
[22] "ATR~BAX"          "GTSE1~CDKN1A"     "PTEN~SIAH1"
[25] "PMAIP1~SIAH1"     "GTSE1~CCNB2"      "CCNE1~CCNB2"
[28] "CASP9~SFN"        "CASP3~CASP8"      "PTEN~CASP8"
[31] "CCND2~TNFRSF10B" "GTSE1~CCNE2"      "PPM1D~RCHY1"
[34] "TSC2~RCHY1"       "PPM1D~CASP3"      "EI24~ATR"
[37] "PTEN~ATR"         "GADD45G~CCND1"    "CASP9~TSC2"
[40] "GADD45B~GADD45G" "PMAIP1~GADD45B"
```

Finally we get common edges of *GeneNet* and PC graphs as follows:

```
> get.edges(Ragr, EdgColor = "red")
```

```
[1] "SERPINE1~CDK6"   "CCNB1~CDK1"      "CCND2~GADD45A"
[4] "CYCS~CDK4"       "CCNB2~CCNB1"     "GADD45B~CDKN1A"
[7] "PPM1D~SIAH1"     "RCHY1~TNFRSF10B"
```

5.1.5 Function: get.hubs()

The function `get.hubs()` is used here to get hubs that are present in the *GeneNet* graph only. In the following, the value p is used to define a node as a hub, p=30 means that a node will be considered as a hub if it is incident to at least 30 percent edges of the total number of edges in the comparative graph. It is expected that there are no hubs in the *GeneNet* graph, because there are no edges of this graph in the comparative graph (blue edges), see Figure~4 .

```
> get.hubs(Ragr, color = "blue", p = 30)
```

```
[1] "There are no hubs to the given color"
```

To get hubs (incident to at least 30 percent edges) that exist in the PC graph only, the function `get.hubs()` is called with the corresponding color.

```
> get.hubs(Ragr, color = "green4", p = 30)
```

```
[1] "There are no hubs to the given threshold"
```

There are no hubs with the threshold p=30, this value is decreased from 30 to 5.

```
> get.hubs(Ragr, color = "green4", p = 5)
```

```
   ATR  CCNE2   PTEN   TP53    BAX  CCND3   DDB2   TSC2   CDK2  CASP9
     5      4      4      4      4      3      3      3      3      3
PMAIP1  CASP8  GTSE1
     3      3      3
```

The numbers under the nodes are the degrees of these nodes in the comparative graph. To get hubs that are present in the *GeneNet* and PC graphs, the function `get.hubs()` is called with the color red as parameter.

```
> get.hubs(Ragr, color = "red", p = 30)
```

```
[1] "There are no hubs to the given threshold"
```

```
> get.hubs(Ragr, color = "red", p = 5)
```

```
[1] "There are no hubs to the given threshold"
```

5.1.6 Function: getMixed.hubs()

In contrast to the function `get.hubs()`, the function `getMixed.hubs()` returns hubs incident to edges colored with at least two colors.

```
> getMixed.hubs(Ragr, p = 30)
```

```
[1] "There are no mixed hubs to the given threshold"
```

```
> getMixed.hubs(Ragr, p = 5)
```

```
[[1]]
        red green4
GADD45A   1      2
CDK4      1      2
CCNB2     1      2
GADD45B   1      2
PPM1D     1      2
SIAH1     1      3
RCHY1     1      3
```

By decreasing the threshold to define hubs from 30 to 5, the function returns some hubs. The printed matrix illustrates the number of colored edges that are incident to each hub.

5.2 Function: compGraphs.vis()

To illustrate the functionality of the function `compGraphs.vis()`, the graphs estimated with PC-Algorithm and *glasso* will be compared.

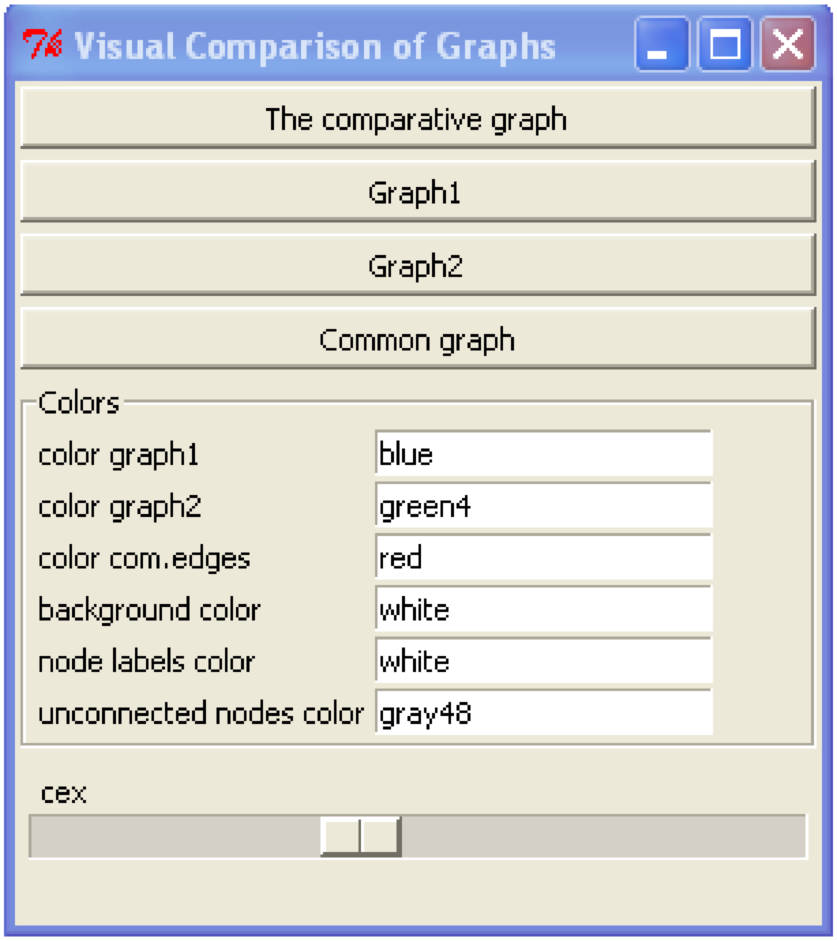
```
> compGraphs.vis(list(pcGr, lassoGr))
```

After calling the function, the control panel in Figure ~7 will appear. Pressing the button *The comparative graph* will plot the same graph as that visualized in Figure~5. Pressing the buttons *Graph1, Graph2 and Common graph* will plot the graphs visualized in Figures ~8, ~9 and ~10 respectively.

5.3 Function: compGraphs.interactive()

The following code compares the estimated graphs with PC-Algorithm and *glasso*. The resulted comparative graph is the same as that visualized in Figure~5. Clicking on the node **CASP8** for example results in plotting the subgraph with the direct neighbours of this node in a new window. The resulted subgraph is visualized in Figure~11.

```
> compGraphs.interactive(list(pcGr, lassoGr),
graphTitle="The comparative graph of the PC and glasso graphs",
legendGr1="Edges of the PC graph",
legendGr2="Edges of the glasso graph")
```



color graph1 blue
color graph2 green4
color com.edges red
background color white
node labels color white
unconnected nodes color gray48

Figure 7: The created control panel with the function `compGraphs.vis()`

6 Summary

This article proposes the new package *graphComp* for visual comparison of graphs defined on the same set of nodes. To illustrate how this package could be used, estimated graphs for a real data set to B-cell ALL data are compared. The graphs are estimated with the packages *GeneNet*, *pcalg* and *glasso*. The estimated graphs have more different edges than common edges. Table˜1 summarizes the edges found with each of the three algorithms and the common edges. An important question is: which of these gene interactions represent trustworthy biological relationships?

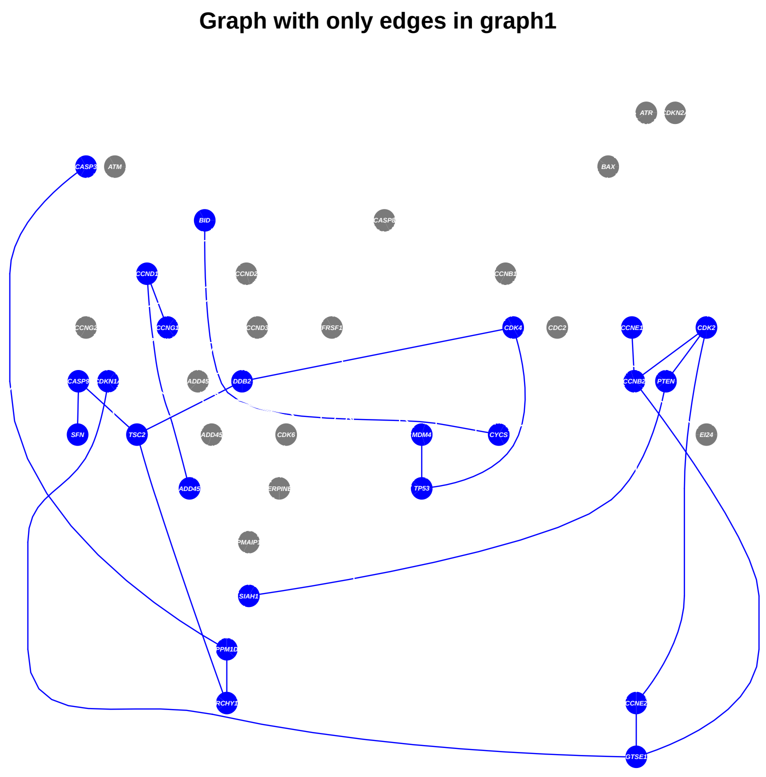


Figure 8: The graph with edges of PC graph only

`Edges found with the three algorithms`	`"CDC2~CCNB1" "CCNB2~CCNB1" "CYCS~CDK4" "SERPINE1~CDK6" "GADD45B~CDKN1A" "GADD45G~GADD45B" "PPM1D~SIAH1" "RCHY1~TNFRSF10B"`

Edges found with glasso	"CASP8~ATM" "CCNG2~ATR" "CDK2~ATR" "MDM4~ATR" "CYCS~CCNB2" "CCNB2~ATR" "CCNE2~ATR" "CYCS~ATR" "PMAIP1~SERPINE1" "PMAIP1~BAX" "CCND2~BID" "GADD45A~BID" "GADD45A~CCNG1" "CCNB1~CASP8" "CCND2~CASP8" "CCNG2~CASP8" "CDKN1A~CASP3" "GADD45A~CASP8" "PMAIP1~CASP8" "SIAH1~CASP8" "PPM1D~CASP8" "TNFRSF10B~CASP8" "CCNE2~CASP8" "RCHY1~CASP8" "CYCS~CASP8" "CDK4~CCNB1" "PMAIP1~CCNB1" "CYCS~CCNB1" "CCND3~CCND2" "CCNG1~CCND2" "CCNG2~CCND2" "DDB2~CCND2" "PMAIP1~CCNG2" "SERPINE1~CCND2" "GADD45B~CCND3" "CDKN1A~CCNG1" "CDK4~CASP8" "GADD45B~CCNG1" "PMAIP1~CCNG1" "GADD45B~CCNG2" "MDM4~CCND2" "PTEN~CCNG2" "CCNB2~CDC2" "CCNB2~CDK4" "GADD45B~GADD45A" "SIAH1~CDKN1A" "PPM1D~CDKN1A" "MDM4~GADD45A" "PMAIP1~CDKN1A" "SERPINE1~GADD45B" "TP53~GADD45B" "PPM1D~GADD45B" "CASP8~BAX" "PPM1D~PMAIP1" "CCNE2~PPM1D" "CCNE2~CCNB2" "EI24~CCNB2"
Edges found with PC-Algorithm	"CCNG1~CCND1" "GADD45G~CCND1" "CYCS~BID" "PTEN~CDK2" "SFN~CASP9" "TSC2~CASP9" "CCNB2~CCNE1" "PPM1D~CASP3" "CCNB2~CDK2" "CCNE2~CDK2" "DDB2~CDK4" "SIAH1~PTEN" "GTSE1~CDKN1A" "TSC2~DDB2" "TP53~MDM4" "TP53~CDK4" "RCHY1~TSC2" "RCHY1~PPM1D" "GTSE1~CCNB2" "GTSE1~CCNE2"
Edges found with glasso and PC-Algorithm	"CCNG2~ATM" "PMAIP1~ATM" "BAX~ATR" "CCNG1~ATR" "PTEN~ATR" "EI24~ATR" "MDM4~BAX" "CCNE2~BAX" "CASP8~CASP3" "PTEN~CASP8" "TP53~CASP8" "GADD45A~CCND2" "TNFRSF10B~CCND2" "DDB2~CCND3" "TP53~CCND3" "SIAH1~CCNG2" "RCHY1~CCNG2" "PMAIP1~GADD45A" "CCNE2~GADD45A" "PMAIP1~GADD45B" "SIAH1~PMAIP1"

Table 1: The edges found with the three algorithms

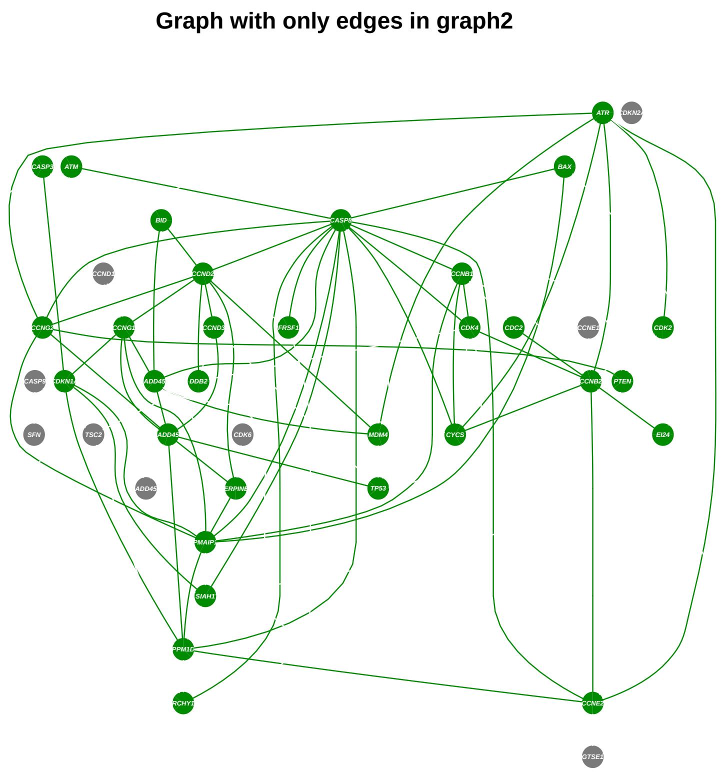


Figure 9: The graph with edges of *glasso* graph only

7 SessionInfo

This document was produced using

```
R version 2.12.1 (2010-12-16)
Platform: i486-pc-linux-gnu (32-bit)

locale:
 [1] LC_CTYPE=de_DE.utf8       LC_NUMERIC=C
 [3] LC_TIME=de_DE.utf8        LC_COLLATE=C
 [5] LC_MONETARY=C             LC_MESSAGES=de_DE.utf8
 [7] LC_PAPER=de_DE.utf8       LC_NAME=C
 [9] LC_ADDRESS=C              LC_TELEPHONE=C
[11] LC_MEASUREMENT=de_DE.utf8 LC_IDENTIFICATION=C
```

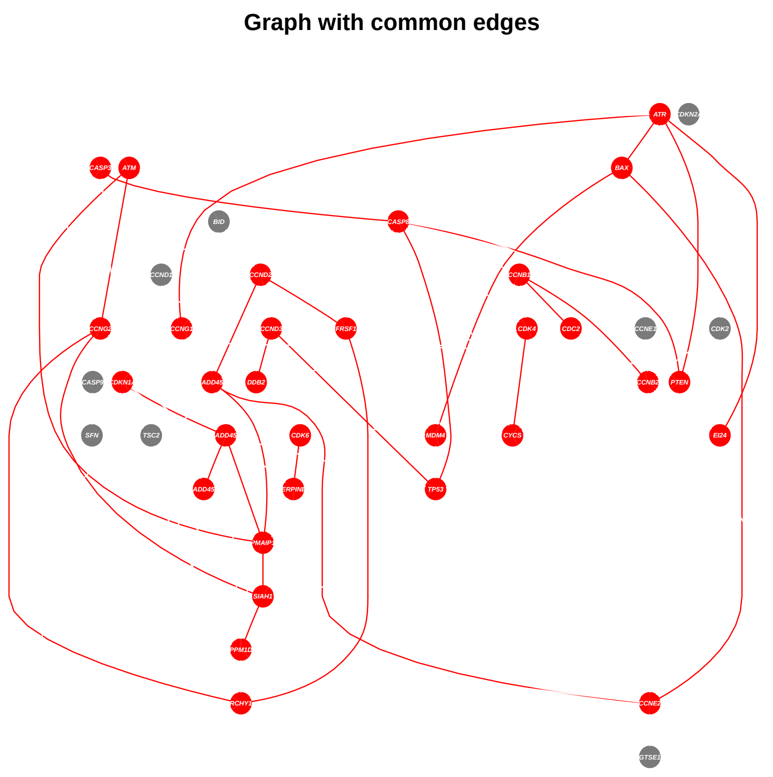


Figure 10: The graph with common edges

```
attached base packages:
[1] tcltk     grid      stats     graphics  grDevices utils
[7] datasets  methods   base

other attached packages:
 [1] graphComp_1.0        rpanel_1.0-6         glasso_1.4
 [4] pcalg_1.1-2          sfsmisc_1.0-14       abind_1.3-0
 [7] Rgraphviz_1.18.1     graph_1.28.0         GeneNet_1.2.4
[10] fdrtool_1.2.6        longitudinal_1.1.5   corpcor_1.5.7
[13] hgu95av2.db_2.4.5    org.Hs.eg.db_2.4.6   RSQLite_0.9-4
[16] DBI_0.2-5            AnnotationDbi_1.12.0 genefilter_1.32.0
[19] ALL_1.4.7            Biobase_2.10.0
```

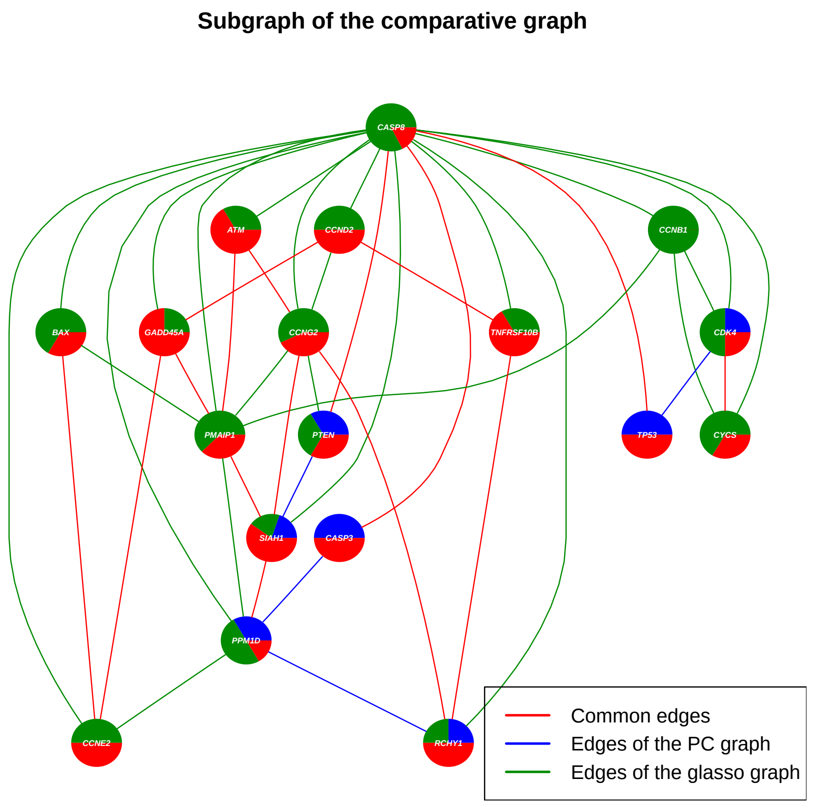


Figure 11: The plotted subgraph with the function `compGraphs.interactive()` by clicking on the node **CASP8**

```
loaded via a namespace (and not attached):
[1] RBGL_1.26.0         annotate_1.28.0    ggm_1.0.4
[4] robustbase_0.5-0-1 splines_2.12.1     survival_2.36-2
[7] tools_2.12.1        xtable_1.5-6
```

References

[1] A.~W. Bowman and E.~Crawford. *R package* **`rpanel`***: simple control panels (version 1.0-5).* University of Glasgow, UK, 2008.

[2] Sabina Chiaretti, Xiaochun Li, Robert Gentleman, Antonella Vitale,

Marco Vignetti, Franco Mandelli, Jerome Ritz, and Robin Foa. Gene expression profile of adult t-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103(7):2771–2778, Apr 2004.

[3] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. *glasso: Graphical lasso- estimation of Gaussian graphical models.* R package version 1.4.

[4] Jeff Gentry, Robert Gentleman, and Wolfgang Huber. *How To Plot A Graph Using Rgraphviz*, 2009.

[5] Rafael˜A Irizarry, Bridget Hobbs, Francois Collin, Yasmin˜D Beazer-Barclay, Kristen˜J Antonellis, Uwe Scherf, and Terence˜P Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, Apr 2003.

[6] Markus Kalisch and Martin Maechler. *pcalg: Estimating the skeleton and equivalence class of a DAG*, 2009. R package version 0.1-8.

[7] Juliane Schaefer, Rainer Opgen-Rhein, , and Korbinian Strimmer. *GeneNet: Modeling and Inferring Gene Networks*, 2008. R package version 1.2.3.

[8] R˜Development˜Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.