User manual: R software for learning Marginal AMP Chain Graphs

October 26, 2015

Contents

1 Introduction

This R code was used for performing the experiments presented in the paper. It is focused on testing the features of the algorithm proposed for learning MAMP chain graphs (MAMPCG). These experiments are directed towards two main directions:

- to check the performance of the algorithm when learning MAMP chain graphs from databases (section 5.1). The experiments consider three random models with **15** nodes and using different probabilities for introducing directed, undirected and bidirected edges. The model named **artificial1** was generated with probabilities 0.2, 0.2 and 0.6 respectively and it contains 17 edges (5 directed, 4 undirected and 8 bidirected). The probabilities used for **artificial2** were 0.2, 0.6 and 0.2; the model contains 32 edges: 3 directed, 24 undirected and 5 bidirected. For **artificial3** the probabilities were 0.6, 0.2 and 0.2; it contains 17 edges: 7 directed, 4 undirected and 6 bidirected.

- to compare MAMPCG versus PC algorithms and using databases generated from Bayesian networks (section 5.2). The experiments consider the following set of networks: **asia** (8 nodes, 8 directed edges), **sachs** (11 nodes, 17 directed edges), **child** (20 nodes, 25 directed edges), **insurance** (27 nodes, 52 directed edges), **mildew** (35 nodes and 46 directed edges), **alarm** (37 nodes and 46 directed edges) and **barley** (48 nodes and 84 directed edges).

The code offers functions for:

- random generation of databases from Bayesian networks.

· random generation of MAMPCG models and databases sampling from them.

· execution of the learning algorithm, checking learned models against the true ones (those where data came from) and storing and showing the results.

· computing significance tests.

The software is configured for using the folders described below:

· **ddbb**: it stores the databases used for experiments. The paper describes a set of experiments using 30 databases for each model and 500, 1000, 5000, 10000 and 50000 as sample sizes. The folder contains as many sub-folders as models (Bayesian networks and MAMPCG). These sub folders contains a new sub folder for each sample size. For example, **./ddbb/asia/** will contain folders named 500, 1000, 5000, 10000 and 50000. These folders are the final containers of databases. The names used for identifying the databases have the following structure:

$$modelName - sampleSize - id.db$$

where *sampleSize* is one of the sample sizes previously mentioned and *id* a number between 1 and 30. As an example, the folder **./ddbb/asia/50000** will contain 3*aa*0 files with names from **alarm-50000-1.db** to **alarm-50000-30.db**.

· **networks**: it stores the definition of the models (Bayesian networks and MAMPCG models). There are a file for each model. The extension of the files described its content (**net** for Bayesian networks **mampcg** for MAMPCG models).

· **results**. This folder will store files storing the results of comparing the product of the learning algorithm with respect to the true model. Each file contains four values: precision and recall for links and precision and recall for v-structures. These results can be used for generating tables, executing significance tests, etc.

a

2 Database generation from Bayesian networks

The generation of databases from Bayesian network is implemented in **generateDatabasesBN.R**, using some auxiliary functions defined in **auxiliarFunctions.R**. One of the most important functions in this last file consists of reading the Bayesian network to sample from: **readNetFile**. It reads the description of the Bayesian network from a file with **net** extension. All the networks employed for the experiments were taken from **bnlearn** Bayesian network repository `http://www.bnlearn.com/bnrepository/` and stored in a folder named **networks**.

The file **generateDatabasesBN** contains a set of functions:

· **generateDatabases.R**. Arguments: Bayesian network, number of files (databases) to generate, number of samples, path where databases must be stored and Bayesian network file name (in order to give a proper name to databases). It is the main function devoted to produce the random samples.

- **generateDatabasesForNet**. Arguments: Bayesian network file name, number of files (databases) to generate, number of desired samples, path where Bayesian networks (net files) are stored and path for databases storage. This function uses the previous one after preparing the required elements: creates the folder where databases will be stored (if needed) and reads the Bayesian network file.
- **generateDatabasesForNets**: function for generating databases for all the Bayesian networks stored in a given folder (it uses the previous one).
- **generateDatabasesForNetName**: function for creating the databases for a Bayesian network with a given name. It bases its work on producing a call to the function **generateDataBasesForNet**.

The file **testGenerateDatabasesBN.R** includes the sentences for producing 30 variants of each sample size (500, 1000, 5000, 10000 and 50000) for **alarm** network. The code in this file is included below:

```
1  # sets relevant paths to nets folder and databases
2  path <- "./networks/"
3  pathddbb <- "./ddbb/"
4
5  # generates the databases for all these sample sizes
6  ns <- c(500,1000,5000,10000,50000)
7
8  # sets the number of databases to generate for each dataset
9  numberFiles <- 30
10
11 # sets the name of the net used for generating samples
12 netname <- "alarm"
13
14 # considers each sample size
15 for(i in 1:length(ns)){
16   # selects the corresponding sample size
17   nsamples <- ns[i]
18
19   # generate the databases
20   generateDatabasesForNetName(path, netname, pathddbb, nsamples,
         numberFiles)
21 }
```

The first sentences define the environment: path to networks definition, path for storing databases, set of sample sizes, number of variants per sample size and Bayesian network to generate from.

After that a loop (line 15 to 21) iterates on the sample sizes and calls the function **generateDatabasesForNetName** passing as arguments the environment information. At the end of the loop the databases will be stored into **ddbb** folder, using the structure described above.

3 Random generation of MAMP models and databases

This functionality is implemented in **generateMAMPCG.R**. The file is organized in several functions related to some specific tasks (all of them focused on generating random models and databases).

· Section 1: utility functions for managing edges and nodes, getting neighbors, spouses, etc.

  – **getNewFileName(baseFileName, pathNet)**: gets a unique name for a new model. This name will contain a prefix given by **baseFileName** argument (**artificial** for the experiments) and an **id** (determined in order to avoid repeated names) that will be concatenated to the prefix.
  – **createEdge(errorNode, node, edges, leftDeco, rightDeco)**: creates a new edge between two nodes with decorations for both sides.
  – **deleteEdge(from, to, edges)**: deletes the edge between the nodes $from$ and to.
  – **deleteEdgeAtRandom(path, edges)**: given a path between nodes and the set of edges in the model, it deletes a randomly selected edge in the path. This function is required in order to repair a model in order to guarantee it fulfills the three required conditions for MAMP chain graphs.
  – **checkUndirectedEdge(nodeA, nodeB, edges)**: checks if the set of edges contains an undirected edge between two nodes.
  – **getUndirectedEdges(node, edges)**: gets the set of undirected edges in the model containing a given node.
  – **getNodesInEdges(node, edge1, edge2)**: gets the nodes involved in a given pair of edges, being node a common one for both of them.
  – **getOppositeNode(node, edge)**: given an edge this function returns the node involved in the opposite side to node.
  – **getSpousesForNode(node, edges)**: gets the set of spouses of a given node.
  – **getSpouses(edges)**: gets the spouses of each node.
  – **getNeighbours(node, edges)**: gets the neighbors of a certain node.
  – **getNeighboursWithDirections(node, edges**: gets the neighbors of node taking into account edges directions. If $A \to B$ is in edges, then B is neighbor of A (but A is not neighbor of B).
  – **getNeighboursWithUndirectedEdges(node,edges)**: gets the neighbors of node but taking into account undirected edges only.
  – **getPath(from, to=from, visited, edges, neighboursFunction)**: gets the path (if there is such a path) between $from$ and to nodes. The function used for deciding the next node to visit is passed as the last argument. It is a recursive function.

· Section 2: MAMPCG functions

  – **checkCondition1(edges)**: checks the compliance of the first condition of MAMPCG (the graph does not contain undirected cycles).
  – **checkCondition2(edges)**: checks second condition (G has no cycle $V_1 \dots V_n = V_1$ such that $V_1 \leftrightarrow V_2$ is in G and $V_i - V_{i+1}$ is in G for all $1 < i < n$).
  – **checkCondition3(edges)**: checks third condition (if $V_1 - V_2 - V_3$ is in G and $spG(V_2) \neq \emptyset$, then $V_1 - V_3$ is in G too).
  – **checksMAMPCG(edges)**: checks if a model defined by a set of edges fulfills the conditions mentioned before. The methods checking conditions can modify the set of edges in order to get a valid model (adding and removing edges).

- **generateRandomGraph(numberNodes, probDirected, probUndirected, probBidirected)**: base function for the generation. It uses a function of **bnlearn** package for producing a random graph. This graph will be used as base for the final model: its arcs are converted to directed, undirected or bidirected at random.
- **generateRandomMAMPCG(numberNodes, probs)**: generates a random graph with the desired number of nodes and using the probabilities passed as argument for classifying the edges as directed, undirected or bidirected. The model is checked in order to guarantee it is a valid MAMPCG.

· Section 3: functions for databases generation

- **transformToBayesianNetwork(edges)**: this method converts a model into a Bayesian network. This is required in order to obtain the samples of the databases. The method produces two lists: one with the edges of the Bayesian network obtained from the model and another with the set of immoralities. These must be considered afterwards for sampling.
- **createBnlearnNet(edges, check)**: this method uses the set of edges created with the previous method. The Bayesian network is required for generating the databases.
- **generateDistributionForRootNodesfunction(net)**: generates the gaussian distribution for root nodes. The method receives as argument the Bayesian network created with the previous function.
- **generateDistributionForNonRootNodes**: generates gaussian distributions for nodes with parents.
- **setParameters(net, params)**: sets the parameters generated for root and non root nodes to network.
- **removeEvidentialVariables(sample, evidenceVariables)**: given a sample and a list of evidential variables produces a new set of data but removing evidential variables.
- **generateSample(model, sampleSize, threshold, cl)**: produces the desired sample using a model passed as first argument. The model contains a name (**artificiali**, being i an unique identifier obtained analyzing the folder where models are stored), a set of edges, a Bayesian network (derived from the model defined by the edges), the complete Bayesian network used for sampling and a set of immoralities produced by the conversion to Bayesian network. The number of samples is given by sampleSize. Threshold is a parameter used in the expression used for the evidence: all the nodes contained in the list of immoralities must be included as evidence. Let us assume there are n nodes in the list of immoralities. The evidence expression will have the following form:

$$node_1 >= -threshold \ \&node_1 <= threshold \ \& \\ \ldots node_n >= -threshold \ \&node_n <= threshold$$

- **storeDatabase(sample, id, numberSamples, path, filename)**: stores the samples in a file: the last 4 arguments are used for deriving the name of the file to use.

- **storeMode(model, folder**: stores a model in a file in order to its posterior use. The model is a list with 5 entries: name, set of edges, basic Bayesian network, Bayesian network used for sampling (with error nodes, distributions, etc) and list of nodes producing immoralities. All the models will be stored in files with **mampcg** as extension.
- **retrieveModel(modelname, folder)**: gets a model from a file.
- **prepareMAMPCGForSampling(edges, basename, folder**: given a set of edges produced by the previous method, it builds a complete model ready for databases generation.
- **generateDatabases(model, variants, sampleSizes, threshold, pathDb, cluster)**: the model passed as first argument is used for producing the desired number of databases for each sample size. The databases will be stored in the folder passed as fifth argument.

The file **testGenerateMAMPCG.R** contains an example of generation of a new model and the corresponding databases. It also shows how can be retrieved a previous model in order to produce more databases.

```
# definition of global parameters: paths, base file name (artificial),
# sample sizes, vector of edges probs for the variants to consider
# (1. 20% for directed edges, 20% undirected, 60% bidirected;
#  2. 20% directed, 60% undirected, 20% bidirected;
#  3. 60% directed, 20% undirected, 20% bidirected)
# number of nets to generate, number of nodes (15), number of databases
# to generate during the sample procedure and seed (NOTE: remove this
# sentence for a real random behaviour; in this cases is fixed just to
# focus the software on the tree models used for the experiments
    described
# in the paper)
pathDb <- "./ddbb/"
pathNet <- "./networks/"
baseFileName <- "artificial"
sampleSizes <- c(500, 1000, 5000, 10000, 50000)
s
# these are the probs for directed, undirected and bidirected used for
# artificial1, artificial2 and artificial3 respectively
edgesProbs <- list(c(0.2, 0.2, 0.6), c(0.2, 0.6, 0.2), c(0.6, 0.2, 0.2)
    )
numberNets <- 3
numberNodes <- 15
variants <- 30

# This parameter defines the threshold used for the evidence. We have
# used 0.2 for artificial1 and artificial3 and 0.9 for artificial2
threshold <- 0.2

# use parallel execution if possible. The argument defines the number
    of
# cores to use
cl=makeCluster(2)

# normal procedure
# 1. generates a new graphical model
graphModel <- generateRandomMAMPCG(numberNodes, edgesProbs[[1]])

```

```
35 # 2. prepare the model for sampling. This also requires to get a valid
36 # name for the model (it will be artificial, bit the id will be
       assigned
37 # examining the folder considered for storing the models (pathNet))
38 completeModel <- prepareMAMPCGForSampling(graphModel, baseFileName,
       pathNet)
39
40 # 3. If needed, store the model for posterior use. The procedure
41 # will check all the files named artificial and will add a create
42 # a new one. This method only saves complete models defined by
43 # edges, a bnet and a set of inmoralities
44 storeModel(completeModel,pathNet)
45
46 # 4. The model can be used to generate the samples
47 generateDatabases(completeModel, variants, sampleSizes, threshold,
       pathDb, cl)
48
49 # 5. We can retrieve a previously generated model for producing new
       datasets
50 completeModel <- retrieveModel("artificial1",pathNet)
51
52 # 6. Now it is possible to generateDatasets as before....
53 # generateDatabases(completeModel, variants, sampleSizes, threshold,
       pathDb, cl)
```

The code begins defining the environment for the generation: path for databases storage, path to models definition, base name used for the models (**artificial**), sample sizes and three different configurations used for producing the models in the paper (**artificial1**, **artificial2** and **artificial3**), number of nodes, number of databases to generate and threshold for evidence expression (lines 1 to 25).

If possible several cores will be used for speeding up the work (sentence giving value to **cl**; line 29). The first step would be the creation of a new graphical model with **generateRandomMAMPCG**. This model is completed in order to be used for database generation (**prepareMAMPCGForSampling**). It is convenient to store the model allowing a posterior databases generation, check, etc. This can be done using the function **storeModel**. Anyway, once a model is prepared for generating databases the function **generateDatabases** can be used. Sometimes it is interesting to use a previous model for producing new databases (see steps 5 and 6 of code; lines 49 and 52).

4 Procedure for comparing MAMP versus PC algorithms

The execution of this comparison requires several conditions:

- there are a set of Bayesian networks (stored as **net** format files) into a folder named **networks** (the experiments are based of learning from *asia*, *sachs*, *child*, *insurance*, *mildew*, *alarm* and *barley*).
- the set of databases used for the experiments included in the paper are stored in a folder named **ddbb**. Its sub-folders are termed as the corresponding Bayesian network. As we have tested several samples sizes, each sub folder contains itself several sub folders (500, 1000, 5000, 10000 and 50000). These sub folders contain the databases, being 30 variants for each sample size.

The main file for this experiment is **executeExperiment.R**. It contains two functions: **learn** and **execute**.

1. **learn**, with arguments defining the path to net files, path to databases, net name, id of the database to use, number of samples in the database, moral (boolean flag stating the way to deal with v-structures), pc (boolean flag denoting the algorithm to use; false for MAMPCG algorithm and true for PC algorithm), mode (string storing the kind of true model to compare with: **net** for Bayesian networks and **mampcg** for MAMPCG models. This function makes the following operations:

   · reads the true model.

   · reads the database.

   · builds an object of **MAMPCGSearch** class. This class contains the data and methods needed for the execution of the learning algorithms used in the paper: MAMPCG and PC algorithms. The behaviour of the object is defined through **pc** and **mode** flags.

2. **execute**, with arguments for the path to network file, database, net name, database id, number of samples, mode (kind of true model) and debug flags. The flag **mode** sets up the experiments to perform. MAMPCG algorithm is required for both experiments and PC algorithm only for the comparison. Therefore the first experiments presented in the paper require applying MAMPCG algorithm on databases derived from MAMPCG models; the second ones will use MAMPCG and PC algorithms on databases coming from Bayesian networks. The operations performed in this method are:

   · executes the algorithms.

   · performs the comparisons between learned and true models.

   · store the results on files.

The file **testExecuteComparisonExperiments.R** contains the sentences required for the experiments performed on Bayesian networks databases:

```
library(foreach)
library(doParallel)

# This sentences configure the environment for performing the
# experiment. Includes the code required for these experiments
source("R/auxiliarFunctions.R")
source("R/utilResults.R")
source("R/MAMPCGSearch.R")
source("R/executeExperiment.R")

#sets the number of digits for the final report
options(digits=2)

# sets the paths to the databases and to the networks. This has to
# be changed when the software is installed on another machine
pathdb <- "./ddbb/"
pathnet <- "./networks/"

# sets the name of the network to work with
netName <- "asia"

# clean ths sink in order to initiate a new trace if required. If this
# is the case, the last of these three lines must be operative removing
```

```
# the comment mark
sink()
sink()
traceFileName <- paste0(netName,"-trace")
sink(traceFileName)

# uses parallelism if possible. The next sentence sets the number of
# cores to use (depending on the execution machine)
registerDoParallel(cores=1)
getDoParWorkers()

# set the different sample sizes to consider
samples <- c(500, 1000, 5000, 10000, 50000)

# sets the number of variants for each sample size
repetitions <- 30

cat("Learning process start\n")

# initializes globalResults with an empty list
globalResults <- list()

# sets this var to show the origin of the true model (net or
# edges). Now it is set  to "net" in order to learn a BN. Change
# this value to edges if the objective if to learn (and compare)
# with respect to a mamp model
trueModel <- "net"

# sets debug flag
debug <- FALSE

# considers each sample size
for(i in 1:length(samples)){
  cat("Learning for sample size: ",samples[i],"\n")
  pathdbsample <- ""
  # composes the path where the ddbb is located
  pathdbsample <- paste(pathdb,netName,sep="")
  pathdbsample <- paste(pathdbsample,samples[i],sep="/")
  pathdbsample <- paste(pathdbsample,"/",sep="")
  cat("ddbb path: ",pathdbsample,"\n")

  # creates the result matrix for this sample size: it will
  # contain 8 rows (results of mampc and pc) and as many
  # columns as the number of variants for this sample size
  partialResults <- matrix(NA,8,repetitions)

  # considers every repetition with a parallel approach
  partialResults <- foreach(j=1:repetitions, .combine='cbind') %dopar%
     {
    # learn from the ddbb, with ths corresponding sample size and
    # the variant given by j.
    execute(pathnet, pathdbsample, netName, j, samples[i], trueModel,
       debug)
  }

  # stores the results into globalResults
  colnames(partialResults) <- (c(1:repetitions))
```

```
79    globalResults[[i]] <- partialResults
80  }
81
82  # generate latex table from data (globalResult) (if required). Only
83  # one of these sentences must be employed. It is included here in
84  # order to check the algorithm
85  generateLatexTableFromData(netName, samples, globalResults)
86
87  # generate latex table from files (if required)
88  #generateLatexTableFromFiles(netName, samples, repetitions)
89
90  # gets sure no sink is open
91  sink()
92  sink()
```

The configuration of the execution environment (lines 1 50 53) consists of:

- sets the number of digits to consider for producing the final output.
- sets the folders containing the databases and the true Bayesian networks.
- sets the name of the Bayesian network to use (now the script is prepared for **asia** network; the assignment of **netName** variable must be changed to work with another ones).
- prepares the storage of trace messages if required; this set of **sink** sentences can be commented to avoid the generation of a trace file.
- sets the parameters for using parallelism taking advantage of the processor cores.
- defines the number of sample sizes to consider for the experiment (giving value to **samples** variable).
- defines the number of repetitions for each sample size (the value of **repetitions** is 30 but it can be changed as well).
- initializes the structure employed for storing the results in memory (**globalResults**; anyway execution results will be stored in files as well).
- sets a value for **trueModel** variable. This allow to use this same set of functions for learning on **MAMPCG** models (and not Bayesian networks). The learning of Bayesian network requires setting the value **"net"** to this parameter.
- defines if debug information will be generated during the execution. This parameter must be set to **FALSE** except for a detailed trace of the execution.

After that the main loop (lines 56 to 80) iterates on the possible values of sample sizes. This loop contains an internal loop (lines 71 to 75) prepared for a parallel execution on the databases for a given combination of net and sample size. Finally it is possible to gather all the results into a latex table (line 88). This can be done using the results stored on memory or disk (this double mechanism was implemented just for producing the results when required). These lines are commented right now. The script finishes cleaning all the output redirection.

5 Procedure for learning MAMP models

These experiments try to check the behavior of the algorithm when learning from databases coming from MAMPCG models. The execution requires:

- there are a set of MAMPCG models (stored as rds files; data with **R** binary format and **mampcg** extension) into **networks** folder. The experiments used three artificial models. As it was explained before there are functions for producing new random models as well.
- MAMPCG models can be used for generating random databases with logic sampling for continuous variables. This was done for the set of networks included in the experiments: *artificial1*, *artificial2* and *artificial3*. The set of corresponding databases are stored in **ddbb** folder with the same organization explained for databases coming from Bayesian networks.

The file **testExecuteMAMPCGExperiments.R** contains the sentences required for learning from these databases. Is has a similar structure to the code used for learning from databases produced from Bayesian networks. The difference is the value assigned to **trueModel** variable. This is enough to change the behavior of the software in order to:

- reads the true model for a **R** file with **mampcg** extension (instead of using a **net** file with the definition of a Bayesian network).
- executes MAMPCG learning algorithm only (there will not be comparison respect to PC algorithm).
- read the databases treating the values as generated from continuous variables.
- changes the kind of independence test to perform.
- changes the way of checking the v-structures of learned versus true model (using the list of edges for both models).

```
library(foreach)
library(doParallel)

# This sentences configure the environment for performing the
# experiment
source("R/auxiliarFunctions.R")
source("R/utilResults.R")
source("R/MAMPCGSearch.R")
source("R/executeExperiment.R")

#sets the number of digits for the final report
options(digits=2)

# sets the paths to the databases and to the networks. This has to
# be changed when the software is installed on another machine
pathdb <- "./ddbb/"
pathnet <- "./networks/"

# sets the name of the network to work with
netName <- "artificial1"
```

```
21
22 # clean ths sink in order to initiate a new trace if required. If this
23 # is the case, the last of these three lines must be operative removing
24 # the comment mark
25 sink()
26 sink()
27 traceFileName <- paste0(netName,"-trace")
28 sink(traceFileName)
29
30 # uses parallelism if possible. The next sentence sets the number of
31 # cores to use (depending on the execution machine)
32 registerDoParallel(cores=8)
33 getDoParWorkers()
34
35 # set the different sample sizes to consider
36 samples <- c(500, 1000, 5000, 10000, 50000)
37
38 # sets the number of variants for each sample size
39 repetitions <- 30
40
41 # shows the start of the learning process
42 cat("Learning process start\n")
43
44 # initializes globalResults with an empty list
45 globalResults <- list()
46
47 # sets this var to show the origin of the true model (net or
48 # edges). Now it is set  to "net" in order to learn a BN. Change
49 # this value to edges if the objective if to learn (and compare)
50 # with respect to a mamp model
51 trueModel <- "mampcg"
52
53 # sets debug flag
54 debug <- FALSE
55
56 # considers each sample size
57 for(i in 1:length(samples)){
58   cat("Learning for sample size: ",samples[i],"\n")
59   pathdbsample <- ""
60   # composes the path where the ddbb is located
61   pathdbsample <- paste(pathdb,netName,sep="")
62   pathdbsample <- paste(pathdbsample,samples[i],sep="/")
63   pathdbsample <- paste(pathdbsample,"/",sep="")
64   cat("ddbb path: ",pathdbsample,"\n")
65
66   # creates the result matrix for this sample size: it will
67   # contain 8 rows (results of mampc and pc) and as many
68   # columns as the number of variants for this sample size
69   partialResults <- matrix(NA,8,repetitions)
70
71   # considers every repetition with a parallel approach
72   partialResults <- foreach(j=1:repetitions, .combine='cbind') %dopar%
       {
73     # learn from the ddbb, with ths corresponding sample size and
74     # the variant given by j.
75     execute(pathnet, pathdbsample, netName, j, samples[i], trueModel,
         debug)
```

```
  }

  # stores the results into globalResults
  colnames(partialResults) <- (c(1:repetitions))
  globalResults[[i]] <- partialResults
}

# generate latex table from data (globalResult) (if required). Only
# one of these sentences must be employed. It is included here in
# order to check the algorithm
generateLatexTableFromData(netName, samples, globalResults)

# generate latex table from files (if required)
#generateLatexTableFromFiles(netName, samples, repetitions)

# gets sure no sink is open
sink()
sink()
```

6 Significance test computation

The file named **significanceTest.R** contains a single function for comparing the results of executing MAMPCG and PC algorithms determining if there is a significant difference between them. This function is based on the existence of result files previously generated. Its code contains a main loop for iterating over the sample sizes. Given a concrete sample size the method gathers the results for all the variants analyzed (all the databases used for this sample size) and compares the series of values: for precision, recall, precision for v-structures and recall for v-structures. The file **testSignificanceTest.R** shows an example of use for **asia** network:

```
# these sentences show how to perform the analysis for a given
# network. The experiments for this network must be already
# available
netName <- "asia"
samples <- c(500, 1000, 5000, 10000, 50000)
repetitions <- 30

# prepares the output to a file
sink()
sink()
sink()

# compose the name of the file to generate with the results
traza <- paste0(netName,"-tests")

# redirect output to the file
sink(traza)

# perform the test
results <- significanceTest(netName, samples, repetitions)
sink()
sink()
```