Graphical Markov Models with Mixed Graphs in R

Keyvan Sadeghi
Department of Statistics, University of Oxford,
1 South Parks Road, OX1 3TG, Oxford, UK, `sadeghi@stats.ox.ac.uk`

Giovanni M. Marchetti,
Department of Statistics, "G. Parenti", University of Florence
viale Morgagni, 59, 50134 Firenze, Italy, `giovanni.marchetti@ds.unifi.it`

November 23, 2011

Abstract

In this paper we provide a short tutorial illustrating some new functions in the package **ggm** to deal with some problems related to graphical models that use mixed graphs. Some of these functions provide methods and implement new graph-theoretical algorithms that generate ribbonless, summary, and ancestral graphs, which are mixed graphs that induce the modified independence structure after marginalization over and conditioning on nodes of directed acyclic graphs. We also provide functions to test m-separation on these graphs, and to generate maximal graphs inducing the same independence structure of non-maximal graphs. Finally, we provide functions that deal with different problems related to Markov equivalences of several classes of graphical models.

1 Introduction and background

Graphical Markov models have become a part of the mainstream of statistical theory and application in recent years. These models use graphs to represent conditional independencies among sets of random variables. Nodes of the graph correspond to random variables and missing edges between nodes i and j typically indicate independencies in the form of $i \perp\!\!\!\perp j \mid C(i,j)$ with a specific definition for $C(i,j)$ depending on the model.

Mixed graphs and separation criteria. General criteria, determining whether two disjoint subsets of variables A and B in a graphical model are independent given a third subset C are typically path-based and called separation criteria, verifying whether there is a path of a specific type, determined by C, between A and B. For more details see, for instance, Lauritzen (1996).

The more general class of graphs that have been used in the literature of graphical models is the class of so-called *mixed graphs.* These graphs contain three types of edge, denoted by lines, $—$, arrows, $\rightarrowtail$, and arcs (bi-directed arrows), $\leftarrowtail\!\rightarrowtail$. The class of mixed graphs contains subclasses of graphs with fewer types of edge, such as undirected and directed acyclic graphs. The same separation criterion, known as *m-separation*, applies to all mixed graphs (Sadeghi and Lauritzen, 2011) and can be used to determine the conditional independence of the random variables when the joint probability is faithful to the graph; see Pearl (1988); Richardson and Spirtes (2002).

While for many subclasses of graphs a missing edge corresponds to some independence statement, for the more complex classes of mixed graphs this is not necessarily true. A graph where each of its missing edges is related to an independence statement is called a *maximal graph*. For a more detailed discussion on maximality of graphs and graph-theoretical conditions for maximal graphs, see Richardson and Spirtes (2002) and Sadeghi and Lauritzen (2011).

Marginalization, conditioning and graph stability. An essential motivation for defining classes of graphs with three types of edge is that they are *stable under marginalization and conditioning*. This means that in the related models when some variables are ignored (marginalized over) or when other variables are fixed by selecting and fixing their value (conditioned on), there exists a graph of the same class that captures the modified independence statements. The resulting graph, with a reduced node set obtained by removing nodes involved in the marginalization and conditioning, conforms to the modified independence statements coming from marginalization and conditioning for the corresponding random variables.

The graphs generated after marginalization and conditioning are of particular interest when there are unobserved or selection variables in directed acyclic graph (DAG) models. DAG models are useful since they suggest a recursive data generating process, but unfortunately DAGs are not stable under marginalization and conditioning. We consider three known classes of *stable mixed graphs* that contain directed acyclic graphs, which are the ribbonless graphs (RGs), (Sadeghi, 2011a) (defined as a modification of MC-graphs (Koster, 2002)), the summary graphs (SGs) (Wermuth, 2011), and the ancestral graphs (AGs) (Richardson and Spirtes, 2002).

Sadeghi (2011a) defines the algorithms generating stable mixed graphs of a specific type for a given DAG or for a stable mixed graphs of the same type.

Markov equivalence. Two graphs of different types, or even two graphs of the same type, may induce the same independencies under m-separation. Such graphs are said to be Markov equivalent. In the literature several problems related to Markov equivalences have been discussed. These include (a) verifying the Markov equivalence of given graphs, (b) presenting conditions under which a graph of a specific type can be Markov equivalent to a graph of another type, and (c) providing algorithms for generating Markov equivalent graphs of certain type from a given graph.

Conditions for Markov equivalence for maximal ancestral graphs (MAGs) have been given in Ali and Richardson (2004), whereas simpler conditions for Markov equivalence for regression chain graphs (RCGs), which are subclass of MAGs, have been presented in Wermuth and Sadeghi (2011). A summary of conditions and algorithms for the above mentioned problems have been provided in Sadeghi (2011b) for MAGs, RCGs, and other subclasses of graphs including undirected graphs (used in concentration models), bidirected (used in covariance models), and DAGs.

Outline. In this paper we provide a tutorial explaining how the algorithms related to stable mixed graphs, m-separation, Markov equivalences, and maximality can be implemented in R.

We first introduce the different ways of defining mixed graphs in **ggm**. We then introduce functions that generate different types of stable mixed graphs from DAGs or from graphs of the same class. We discuss how to generate a maximal graph that induces the same independence structure from a non-maximal graph. We then illustrate how to verify the validity of given independence statements over the defined or generated graphs by using the m-separation criterion. Finally, we illustrate functions to deal with different problems related to Markov equivalence.

The functions and codes outlined in this paper have been merged as a new set of functions into the R package **ggm**. For some previous contributions of **ggm**, see Marchetti (2006). The package **ggm** has been improved and it is now more integrated with other contributed packages related to graphs, such as **graph**, **igraph** (Csardi and Nepusz, 2006), and **gRbase** (Dethlefsen and Højsgaard, 2005), which are now required for representing and plotting graphs. In particular, in addition to adjacency matrices, all the functions in the package now accept `graphNEL` and `igraph` objects as input, as well as a new character string representation, which will be discussed later in this paper. Notice however that all the algorithms internally use the adjacency matrix representation.

A list of available packages in graphical models with some descriptions can be found at CRAN Task View gRaphical Models in R at `http://cran.r-project.org/web/views/gR.html`.

2 Defining directed acyclic graphs

We start by illustrating some ways of defining directed acyclic graphs in R. We shall use four graph representations, that is (a) by `graphNEL` objects, (b) by `igraph` objects, (c) by adjacency matrices, (d) by character vectors. These same representations will be extended to general mixed graphs in the next section.

Defining DAGs as graphNEL objects. The **graph** package provides the `graphNEL` class for graphs that are represented in terms of nodes and an edge list. These graphs can be either directed or undirected, with no multiple edges or self-loops. The **gRbase** package provides the function `dag` for easily defining DAGs as `graphNEL` objects. One way in which the graph can be specified is by a list of formulas. For example, one can define a DAG by the following formula, where, for example, `h*p*r` implies that h has parents p and r. For reference, we plot the DAG after the explanations related to the formula, by using the **ggm** function `plotGraph`.

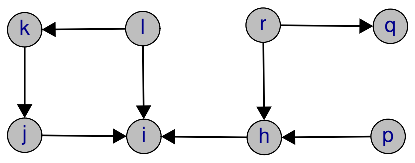
```
> exdag <- dag(~r, ~q*r, ~p, ~h*p*r, ~l,
               ~k*l, ~j*k, ~i*h*l*j)
> exdag

A graphNEL graph with directed edges
Number of Nodes = 8
Number of Edges = 8
```

Notice that one cannot use integers as node labels in `graphNEL` objects. In addition, instead of "*" a ":" can be used in the specification. To retrieve the nodes and edges of the DAG the functions `nodes` and `edges` are used. Thus for instance,

```
> nodes(exdag)

[1] "r" "q" "p" "h" "l" "k" "j" "i"
```

To plot a DAG as a `graphNEL` object one can use the function `plot` in the **Rgraphviz** package or various plotting options within the **igraph** package (see below). Our recommended way is to use a defined function `plotGraph` as follows:

```
> plotGraph(exdag)
```



Defining DAGs as igraph objects. In package **igraph** there are many ways of defining DAGs as `igraph` objects; for the full list see Csardi and Nepusz (2006). Probably the most handy way of generating small graphs is to use `graph.formula`. In this function we use "-" as edges and "+" as arrowheads. For example, for the same DAG as before, we use

```
> exdag <- graph.formula(q+-r-+h+-p,
           h-+i+-l-+k,   i+-j+-k)
> exdag
```

```
Vertices: 8
Edges: 8
Directed: TRUE
Edges:

[0] 'r' -> 'q'
[1] 'r' -> 'h'
[2] 'p' -> 'h'
[3] 'h' -> 'i'
[4] 'l' -> 'i'
[5] 'l' -> 'k'
[6] 'j' -> 'i'
[7] 'k' -> 'j'
```

It is always possible to convert `igraph` graphs to `graphNEL` objects or back using the `igraph.to.graphNEL` and `igraph.from.graphNel` functions.

Also, the package **igraph** contains several options for plotting graphs such as the function `tkplot`. This is a quite flexible function providing an interactive graph drawing facility based on Tcl-Tk. Our `plotGraph` function is just a convenient wrapper of this function.

Defining DAGs with adjacency matrices. Any graph can be defined by an *adjacency matrix*. The adjacency matrix of a loopless directed graph G with node set V and edge set E is the matrix $A = (a_{ij})$ of size $|V| \times |V|$ with zeros on the diagonal and

$$a_{ij} = \begin{cases} 1, & \text{if } i \rightarrowtail j; \\ 0 & \text{otherwise.} \end{cases}$$

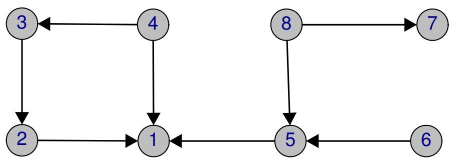
An adjacency matrix can be defined directly without defining a `graphNEL` or `igraph` object. For instance the adjacency matrix of the previous DAG can be created via:

```
> A <- matrix(0,8,8)
> A[8,7] <- A[8,5] <- A[6,5] <- A[5,1] <-
  A[4,1] <- A[4,3] <- A[2,1] <- A[3,2] <- 1
> A
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    0    0    0    0    0    0    0
[2,]    1    0    0    0    0    0    0    0
[3,]    0    1    0    0    0    0    0    0
[4,]    1    0    1    0    0    0    0    0
[5,]    1    0    0    0    0    0    0    0
[6,]    0    0    0    0    1    0    0    0
[7,]    0    0    0    0    0    0    0    0
[8,]    0    0    0    0    1    0    1    0
```

All functions in package **ggm** accept adjacency matrices as defining graphs. If the node names are not provided, the functions would specify by default as labels the integers $\langle 1, \ldots, n\rangle$, where n is the number nodes. Thus the defined adjacency matrix `A` defines the following graph:

```
> plotGraph(A)
```



The node labels are obtained by including appropriate `dimnames` to the matrix:

```
> V <- c("i","j","k","l","h","p","q","r")
> dimnames(A) = list(V, V)
```

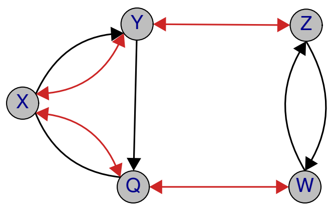
An adjacency matrix for the DAG can be also obtained using the function `DAG` in **ggm**. This function accepts as input a model formula an returns the adjacency matrix with the appropriate labels. For the same DAG, which we call `exdag`, we can write

```
> exdag <- DAG(i ~ j+l+h, k~l, h~r+p, q ~ r)
```

The function also verifies whether the resulting graph is acyclic.

3 Defining mixed graphs

A mixed graph is a graph with at most three types of edge: directed, undirected and bi-directed, with possibly multiple edges of different types connecting two nodes. For instance the following graph is a mixed graph:



In **ggm** we provide some tools for mixed graphs that are not present in other packages. Here we briefly illustrate some methods to define mixed graphs and use the same function, `plotGraph`, to plot them.

The first method is based on a generalization of the adjacency matrix described in the previous section. The second uses a descriptive vector and is easy to use for small graphs. The third uses a special function `makeMG` that allows to combine the directed, undirected and bi-directed components of a mixed graph.

Adjacency matrices for mixed graphs. In the adjacency matrix of a mixed graph we code the three different edges with a binary indicator: 1 for directed, 10 for undirected and 100 for bi-directed edges. When there are multiple edges the codes are added.

Thus the *adjacency matrix of a mixed graph* H with node set N and edge set F is an $|N| \times |N|$ matrix obtained as $A = B + S + W$ by adding three matrices $B = (b_{ij})$, $S = (s_{ij})$ and $W = (w_{ij})$ defined by

$$\begin{aligned}
b_{ij} &= \begin{cases} 1, & \text{if and only if } i \succ\!\!\!\longrightarrow j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases} \\
s_{ij} = s_{ji} &= \begin{cases} 10, & \text{if and only if } i \,\text{———}\, j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases} \\
w_{ij} = w_{ji} &= \begin{cases} 100, & \text{if and only if } i \prec\!\!\!\longrightarrow\!\!\!\succ j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases}
\end{aligned}$$

Notice that because of the symmetric nature of lines and arcs S and W are symmetric, whereas B is not necessarily symmetric. Therefore, the mixed graph shown before can be defined by the commands

```
> mg <- matrix(c( 0, 101,   0,   0, 110,
                100,   0, 100,   0,   1,
                  0, 110,   0,   1,   0,
                  0,   0,   1,   0, 100,
                110,   0,   0, 100,   0),
        5,5, byrow = TRUE)
> N <- c("X","Y","Z","W","Q")
> dimnames(mg) <- list(N, N)
> mg
```

```
    X   Y   Z   W   Q
X   0 101   0   0 110
Y 100   0 100   0   1
Z   0 110   0   1   0
W   0   0   1   0 100
Q 110   0   0 100   0
```

and can be plotted with `plotGraph(mg)`.

Defining mixed graphs by using vectors. A more convenient way of defining small mixed graphs is based on a simple vector coding as follows. The graph is defined by a character vector of length $3f$, where $f = |F|$ is the number of edges, and the vector contains a sequence of triples $\langle$`type, label1, label2`$\rangle$, where the `type` is the edge type and `label1` and `label2` are the labels of the two nodes. The edge type accepts `"a"` for an directed arrow , `"b"` for an arcs and `"l"` for a line. Notice that isolated nodes may not be created by this method. For example, the vector representation of the previous mixed graph is

```
> mgv <- c("b","X","Y","a","X","Y","l","X","Q",
           "b","Q","X","a","Y","Q","b","Y","Z",
           "a","Z","W","a","W","Z","b","W","Q")
```

Once again as in the DAG case we can use `plotGraph(mgv)` to plot the defined graph.

Mixed graph using the function `makeMG`. Finally the adjacency matrix of a mixed graph may be built up with the function `makeAG`. This function requires three arguments `dg`, `ug` and `bg`, corresponding respectively to the three adjacency matrices B, S and W composing the mixed graph. These may also be obtained by the constructor functions `DG` and `UG` of **ggm** for directed and undirected graphs respectively. Thus for the previous mixed graph we can issue the command

```
> mg <- makeMG(dg = DG(Y~X, Z~W, W~Z, Q~X),
               ug = UG(~ X*Q),
               bg = UG(~ Y*X + X*Q + Q*W +Y*Z))
```

obtaining the same adjacency matrix (up to a permutation).

4 Generating stable mixed graphs

By "generating graphs" we mean applying the defined algorithms, e.g. those for generating stable mixed to graphs, in order to generate new graphs.

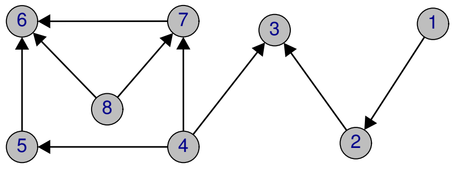
Three main functions to generate stable mixed graphs. Three main functions `RG`, `SG`, and `AG` are available to generate and plot ribbonless, summary, and ancestral graphs from DAGs, using the algorithms in Sadeghi (2011a). These algorithms look for the paths with three nodes and two edges in the graph whose inner nodes are being marginalized over or condition on, and generate appropriate edges between the endpoints. These have two important properties: (a) they are well-defined in the sense that the process can be performed in any order and produces always the same final graph; (b) the generated graphs induce the modified independence structure after marginalization and conditioning; see Sadeghi (2011a) for more details.

The functions `RG`, `SG`, and `AG` all have three arguments: `a` the given input graph, `M`, the marginalization set and `C`, the conditioning set. The graph may be of class `"graphNEL"` or of class `"igraph"` or may be represented by a character vector, or by an adjacency matrix, as explained in the previous sections. The sets `M` and `C` (default `c()`) must be disjoint vectors of node labels, and they may possibly be empty sets. The output is always the adjacency matrix of the generated graph. There are two additional logical arguments `showmat` and `plot` to specify whether the adjacency matrix must be explicitly printed (default `TRUE`) and the graph must be plotted (default `FALSE`).

Some examples. We start from a DAG defined in two ways, as an adjacency matrix and as a character vector:

```
> ex <- matrix(c(0,1,0,0,0,0,0,0,
                 0,0,1,0,0,0,0,0,
                 0,0,0,0,0,0,0,0,
                 0,0,1,0,1,0,1,0,
                 0,0,0,0,0,1,0,0,
                 0,0,0,0,0,0,0,0,
                 0,0,0,0,0,1,0,0,
                 0,0,0,0,0,1,1,0),8,8,byrow=TRUE)
> exvec <- c("a",1,2,"a",2,3,"a",4,3,
             "a",4,5,"a",4,7,"a",5,6,
             "a",7,6,"a",8,6,"a",8,7)
```

```
> plotGraph(ex)
```



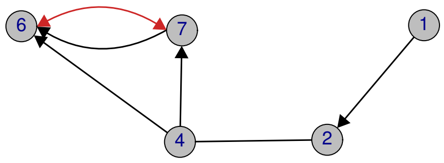
Then we define two disjoint sets M and C to marginalize over and condition on

```
> M <- c(5,8)
> C <- 3
```

and we generate the ribbonless, summary and ancestral graphs from the DAG with the associated plot.

```
> RG(ex, M, C, plot = TRUE)
```

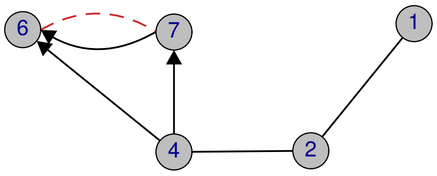
```
  1  2  4   6   7
1 0  1  0   0   0
2 0  0 10   0   0
4 0 10  0   1   1
6 0  0  0   0 100
7 0  0  0 101   0
```



The summary graph is plotted with dashed undirected edges instead of bi-directed edges:

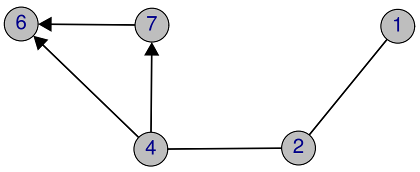
```
> plotGraph(SG(ex,M,C), dashed = TRUE)
```

```
   1  2  4   6   7
1  0 10  0   0   0
2 10  0 10   0   0
4  0 10  0   1   1
6  0  0  0   0 100
7  0  0  0 101   0
```



The induced ancestral graph is obtained from the DAG defined as a vector.

```
> AG(exvec,M,C,showmat=FALSE, plot=TRUE)
```

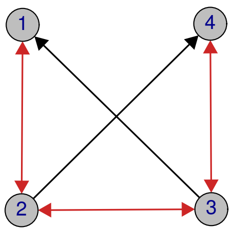


5 Maximality, m-separation, and Markov equivalences

Function for generating maximal graphs. Given a non-maximal graph, we can obtain the adjacency matrix of a maximal graph that induces the same independence statements with the function `Max`. This function uses the algorithm by Sadeghi (2011b), which is an extension of the implicit algorithm presented in Richardson and Spirtes (2002). This algorithm searches for all non-adjacent pairs of nodes that are connected by a so-called *primitive inducing path* and generate an appropriate edge between the pairs. The related functions `MAG`, `MSG`, and `MRG`, are just handy wrappers to obtain maximal AGs, SGs and AGs, respectively. For example,

```
> H <- matrix(c(0  ,100,  1,  0,
                100,0  ,100,  0,
                0  ,100,  0,100,
                0,  1  ,100,  0),4,4)
```

```
> plotGraph(H)
```

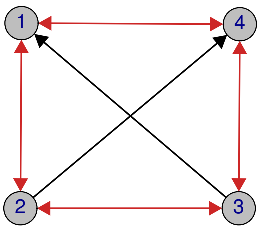


is a mixed non-maximal graph, with the missing edge between nodes 1 and 4 that is not associated with any independence statement (because $\langle 1, 2, 3, 4\rangle$ is a primitive inducing path). Its associated maximal graph is obtained by

```
> Max(H)
```

```
    1   2   3   4
1   0 100   0 100
2 100   0 100   1
3   1 100   0 100
4 100   0 100   0
```

```
> plotGraph(Max(H))
```



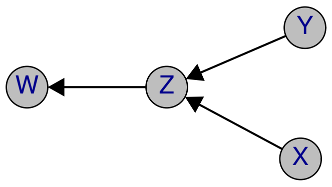
As the graph `H` is an ancestral graph (as may be verified by the function `isAG`), we obtain the same result with

```
> MAG(H)
```

```
    1   2   3   4
1   0 100   0 100
2 100   0 100   1
3   1 100   0 100
4 100   0 100   0
```

Function to verify m-separation. The fundamental way to verify if a graphical model implies an independence $A \perp\!\!\!\perp B|C$ is to test whether the disjoint subsets A and B of the node set are m-separated given a third set C, and this may be done with the function `msep`. The function has 4 arguments, where the first is the graph `a`, in one of the forms discussed before, and the other three are the disjoint sets `A`, `B` and `C`. For example, for the following graph,

```
> a <- DAG(W ~ Z, Z ~ Y + X)

> plotGraph(a)
```



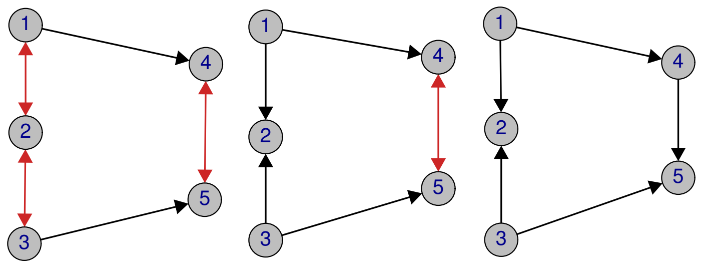
the two following statements verify whether `X` is m-separated from `Y` given `X`, and whether `X` is m-separated from `Y` (given the empty set):

```
> msep(a,"X","Y","Z")

[1] "NOT separated"

> msep(a,"X","Y")

[1] "Separated"
```

In this example m-separation is equivalent to its special case, the d-separation criterion for DAGs. (Note that there is another function `dSep` in **ggm** for d-separation.)

Functions for testing Markov equivalences. The function `MarkEqRcg` tests whether two regression chain graphs are Markov equivalent. This function simply finds the skeleton and all unshielded collider V-configurations in both graphs and tests whether they are identical, see Wermuth and Sadeghi (2011). The arguments of this function are the two graphs `a` and `b` in one of the allowed forms. For example,

```
> H1 <- matrix(c(  0,100,  0,  0,  0,
                 100,  0,100,  0,  0,
                   0,100,  0,  0,  0,
                   1,  0,  0,  0,100,
                   0,  0,  1,100,  0),5,5)
> H2 <- matrix(c(0,0,0,  0,  0,
                 1,0,1,  0,  0,
                 0,0,0,  0,  0,
                 1,0,0,  0,100,
                 0,0,1,100,  0),5,5)
> H3 <- matrix(c(0,0,0,0,0,
                 1,0,1,0,0,
                 0,0,0,0,0,
                 1,0,0,0,0,
                 0,0,1,1,0),5,5)

> plotGraph(H1); plotGraph(H2); plotGraph(H3)
```

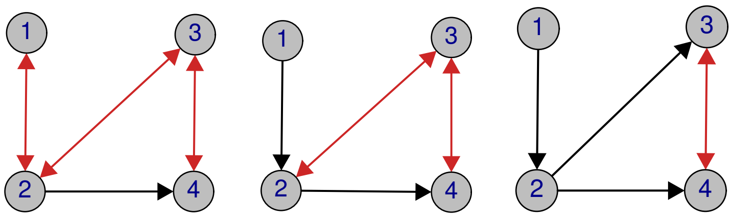


We can now verify Markov equivalence as follows

```
> MarkEqRcg(H1,H2)

[1] "Markov Equivalent"

> MarkEqRcg(H1,H3)

[1] "NOT Markov Equivalent"

> MarkEqRcg(H2,H3)

[1] "NOT Markov Equivalent"
```

To test Markov equivalence for maximal ancestral graphs the algorithm is computationally much more demanding (see Ali and Richardson (2004)) and, for this purpose, the function `MarkEqMag` has been provided. Of course, one can use this function for Markov equivalence of regression chain graphs (which are a subclass of MAGs). For example,

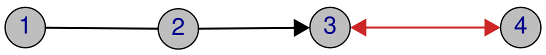
```
> A1<-matrix(c(  0,100,  0,  0,
               100,  0,100,  0,
                 0,100,  0,100,
                 0,  1,100,  0),4,4)
> A2<-matrix(c(0,  0,  0,  0,
               1,  0,100,  0,
               0,100,  0,100,
               0,  1,100,  0),4,4)
> A3<-matrix(c(0,0,  0,  0,
               1,0,  0,  0,
               0,1,  0,100,
               0,1,100,  0),4,4)

> plotGraph(A1); plotGraph(A2); plotGraph(A3)
```



```
> MarkEqMag(H1,H2)

[1] "Markov Equivalent"

> MarkEqMag(H1,H3)

[1] "NOT Markov Equivalent"

> MarkEqMag(H2,H3)

[1] "NOT Markov Equivalent"
```

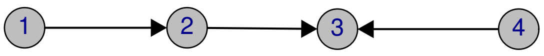
Functions for generating Markov equivalent graphs of a specific type. Sometimes is important to verify if a given graph is capable of being Markov equivalent to a graph of a specific class of graphs (such as DAGs, undirected graphs, or bidirected graphs), and if so, to obtain as a result such a graph. The functions `RepMarDAG`, `RepMarUG`, and `RepMarBG` do this for DAGs, undirected graphs, or bidirected graphs, respectively. For associated conditions and algorithms, see Sadeghi (2011b). For example, given the following graph

```
> H <- matrix(c( 0,10,  0,  0,
                10, 0,  0,  0,
                 0, 1,  0,100,
                 0, 0,100,  0),4,4)

> plotGraph(H)
```



we can see that it is Markov equivalent to a DAG, by

```
> RepMarDAG(H)

  1 2 3 4
1 0 1 0 0
2 0 0 1 0
3 0 0 0 0
4 0 0 1 0

> plotGraph(RepMarDAG(H))
```



On the other hand it is not Markov equivalent to an undirected graph or to a bidirected graph.

```
> RepMarUG(H)

[1] "NOT Markov equivalent to a UG"

> RepMarBG(H)

[1] "NOT Markov equivalent to a BG"
```

This has consequence on the possibility of using standard fitting algorithms for DAG models to estimate the parameters of a model for the first graph.

6 Acknowledgments

The authors are grateful to Steffen Lauritzen for helpful suggestions on codes and comments on an earlier version of the paper.

References

R. A. Ali and T. Richardson. Searching across Markov equivalence classes of maximal ancestral graphs. In *Proceedings of the Joint Statistical Meeting of the American Statistical Association*, Toronto, Canada, 2004.

G. Csardi and T. Nepusz. The `igraph` software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `http://igraph.sf.net`.

C. Dethlefsen and S. Højsgaard. A common platform for graphical models in r: The grbase package. *Journal of Statistical Software*, 14(17):1–12, 12 2005. ISSN 1548-7660. URL `http://www.jstatsoft.org/v14/i17`.

J. T. A. Koster. Marginalizing and conditioning in graphical models. *Bernoulli*, 8(6): 817–840, 2002.

S. L. Lauritzen. *Graphical Models.* Clarendon Press, Oxford, United Kingdom, 1996.

G. M. Marchetti. Independencies induced from a graphical Markov model after marginalization and conditioning: the R package ggm. *Journal of Statistical Software*, 15(6), 2006.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: networks of plausible inference.* Morgan Kaufmann Publishers, San Mateo, CA, USA, 1988.

T. S. Richardson and P. Spirtes. Ancestral graph Markov models. *Annals of Statistics*, 30 (4):962–1030, 2002.

K. Sadeghi. Stable mixed graphs. *submitted*, 2011a. URL `http://arxiv.org/abs/1110.4168`.

K. Sadeghi. Markov equivalences for subclasses of loopless mixed graphs. *submitted*, 2011b. URL `http://arxiv.org/abs/1110.4539`.

K. Sadeghi and S. L. Lauritzen. Markov properties for loopless mixed graphs. *submitted*, 2011. URL `http://arxiv.org/abs/1109.5909`.

N. Wermuth. Probability distributions with summary graph structure. *Bernoulli*, 17(3): 845–879, 2011.

N. Wermuth and K. Sadeghi. Sequences of regressions and their independences. *TEST*, To appear as an invited discussion paper, 2011. URL `http://arxiv.org/abs/1103.2523`.