Graphical Markov Models with Mixed Graphs in R

by Kayvan Sadeghi and Giovanni M. Marchetti

Abstract In this paper we provide a short tutorial illustrating the new functions in the package **ggm** that deal with ancestral, summary and ribbonless graphs. These are mixed graphs (containing three types of edges) that are important because they capture the modified independence structure after marginalisation over and conditioning on nodes of directed acyclic graphs. We provide functions to verify whether a mixed graph implies that A is independent of B given C for any disjoint sets of nodes, and to generate maximal graphs inducing the same independence structure of non-maximal graphs. Finally, we provide functions to decide on the Markov equivalence of two graphs with the same node set but different type of edges.

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 edges to some type of conditional dependencies.

Directed acyclic graphs. In the literature of graphical models the two most used classes of graphs are the directed acyclic graphs (DAGs) and the undirected graphs. DAGs have been proven useful, among other things, to specify the data generating processes when the variables satisfy an underlying partial ordering.

For instance, suppose that we have 4 observed variables Y, the ratio of systolic to diastolic blood pressure, X the diastolic blood pressure, both on log scale, Z, the body mass and W, the age, and that a possible generating process is the following linear recursive regression model

$$\begin{aligned} Y &= \gamma_{YZ}Z + \gamma_{YU}U + \epsilon_Y \\ X &= \gamma_{XW}W + \gamma_{XU}U + \epsilon_X \\ Z &= \gamma_{ZV}W + \epsilon_Z \\ W &= \epsilon_W;\ U = \epsilon_U, \end{aligned}$$

where all the variables are mean-centered and the ϵs are zero mean, mutually independent Gaussian random errors. In this model we assume that there exists a genetic factor U influencing the ratio and levels of blood pressure.

Then, this model can be represented by the DAG in Figure 1(a) with nodes associated with the variables and edges, indicating the dependencies, represented by the regression coefficients γs.

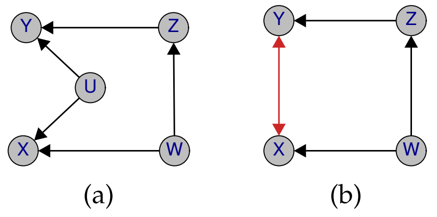


Figure 1: (a) A DAG. (b) A regression chain graph.

From the graph it is seen for instance that the ratio of the two blood pressures is directly influenced by body mass but not by age. Thus a consequence of the model is that the variables must satisfy a set of conditional independencies: for example the ratio of the blood pressure Y is independent of the age W given the body mass Z, written as $Y \perp\!\!\!\perp W | Z$.

A remarkable result is that the independencies can be deduced from the graph alone, without reference to the equations, by using a criterion called d-separation. In fact, in the graph of Figure 1(a), the nodes Y and W are d-separated given Z as can be checked using special graph algorithms included for example in packages **gRain** and **ggm**. For more details on DAG models and their implementation in R see the extensive discussion in Højsgaard et al. (2012).

Hidden variables and induced graphs. The model has 4 observed variables but includes an unobserved variable, that is the genetic factor U. When U is hidden the model for the observed variables becomes

$$\begin{aligned} Y &= \gamma_{YZ}Z + \eta_Y \\ X &= \gamma_{XW}W + \eta_X \\ Z &= \gamma_{ZV}W + \epsilon_Z \\ W &= \epsilon_W; \end{aligned}$$

with two correlated errors $\eta_Y = \gamma_{YU}U + \epsilon_Y$ and $\eta_X = \gamma_{XU}U + \epsilon_X$, such that $\text{cov}(\eta_Y, \eta_X) = \omega_{YX}$. As a consequence the model is still a recursive model and the parameters have a regression parameter interpretation, but contain some correlated residuals.

The induced model is said to be obtained after marginalisation over U. In this model some of the original independencies are lost, but we can observe the implied independencies $Y \perp\!\!\!\perp W | Z$ and $X \perp\!\!\!\perp Z | W$. Also it can be shown that in a DAG model defined for the 4 observed variables it is impossible to represent such independencies. Therefore, we say that DAG models are not stable under marginalisation.

A mixed graph with arrows and arcs, shown in Figure 1(b) can however be used to represent the induced independence model after marginalisation over U. In this representation, beside the arrows, represented by the γs, we have the arc $Y \prec\!\!\!\longrightarrow\!\!\!\succ X$ associated with the (partial) correlation $\omega_{YX},$.

The graph of Figure 1(b) belongs to a class of models called *regression chain graph models*. This class generalises the recursive generating process of DAGs by permitting joint responses, coupled in the graph by arcs, and thus appears an essential extension for applications; see Cox and Wermuth (1996). Regression chain graphs can be used as a conceptual framework for understanding multivariate dependencies for example in longitudinal studies. The variables can then be arranged in a sequence of blocks, such that all variables in one block are of equal standing and any dependences between them being is represented by an arc; all variables in one block are responses to variables in all blocks to their right, so that any dependences between them are directed, represented by an arrow pointing from right to left. The graph shows then how the data analysis can be broken down into a series of regressions and informs about which variables should or should not be controlled for in each regression.

More general induced graphs. The class of regression chain graphs is however itself not stable under marginalisation. For instance, suppose that the generating process for the blood pressure data is defined by the more general regression chain graph of Figure 2(a) where L is a further variable representing a common hidden cause of systolic good pressure and body mass.

Then, after marginalisation over L, the model can still be described by a linear system of equations with correlated residuals and can be represented by the mixed graph shown in Figure 2(b). But the resulting graph is not a DAG nor a regression chain graph because it contains the pair of variables (Y, Z) coupled both by a directed edge and by a path composed by bi-directed arcs. Thus Y cannot be interpreted as a pure response to Z and in addition Y and Z are not two joint responses.

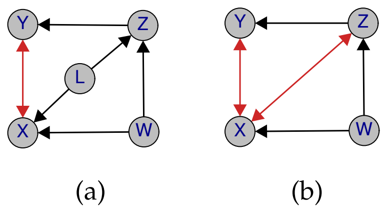


Figure 2: (a) A regression chain graph model; (b) the mixed graph obtained after marginalisation over L, which is not a regression chain graph.

Stable mixed graphs. The previous illustrations show that when there are unobserved variables, DAG or regression chain graph models are no longer appropriate. The discussion could be extended to situations where there are some selection variables, that are hidden variables that are conditioned on.

This motivates the introduction of a more general class of *mixed graphs*, which contains three types of edges, denoted by lines, $—$, arrows, $\longrightarrow\!\!\!\succ$, and arcs (bi-directed arrows), $\prec\!\!\!\longrightarrow\!\!\!\succ$. In the case of regression models, explained above, lines generally link pairs of joint context (explanatory) variables and arcs generally link pairs of joint response variables.

There are at least three known classes of mixed graphs without self loops that remain in the same class, i.e. that are *stable under marginalisation and conditioning*. The largest one is that of *ribbonless graphs* (RGs), (Sadeghi, 2012a), defined as a modification of MC-graphs (Koster, 2002). Then, there is the subclass of *summary graphs* (SGs) (Wermuth, 2011), and finally the smallest class of the *ancestral graphs* (AGs) (Richardson and Spirtes, 2002).

Four tasks of the current paper. In this paper, we focus on the implementation of four important tasks performed on the class of mixed graphs in R:

1. Generating different types of stable mixed graphs after marginalisation and conditioning.

2. Verifying whether an independency of form $Y \perp\!\!\!\perp W | Z$ holds by using a separation criterion called m-separation.

3. Generating a graph that induces the same independence structure as an input mixed graph such that the generated graph is *maximal*, i.e. each missing edge of the generated graph implies at least an independence statement.

4. Verifying whether two graphs are *Markov equivalent*, i.e. they induce the same independencies, and whether, given graph of a specific type, there is a graph of a different type that is Markov equivalent to it.

Package ggm. The previous tasks are illustrated by using a set of new functions introduced into the R package **ggm**. In the next section we give the details on how general mixed graphs are defined. Next in each of the following sections we respectively deal with each of the tasks described above. For each task we give a brief introduction at the beginning of its corresponding section.

Some of the functions generalise previous contributions of **ggm** discussed in Marchetti (2006). The **ggm** package has been improved and it is now more integrated with other contributed packages related to graph theory, such as **graph**, **igraph** (Csardi and Nepusz, 2006), and **gRbase** (Dethlefsen and Højsgaard, 2005), which are now required for representing and plotting graphs. Specifically, 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. A more detailed list of available packages for graphical models can be found at CRAN Task View *gRaphical Models in R* at http://cran.r-project.org/web/views/gR.html.

Defining mixed graphs in R

For a comprehensive discussion on the ways of defining a directed acyclic a graph, see Højsgaard et al. (2012). A mixed graph is a more general graph type with at most three types of edge: directed, undirected and bi-directed, with possibly multiple edges of different types connecting two nodes. In **ggm** we provide some special tools for mixed graphs that are not present in other packages. Here we briefly illustrate some methods to define mixed graphs and we plot them with a new function, `plotGraph`, which uses a Tk GUI for basic interactive graph manipulation.

The first method is based on a generalisation of the adjacency matrix. 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

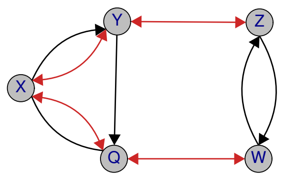
$$b_{ij} = \begin{cases} 1, & \text{if and only if } i \longrightarrow j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases}$$

$$s_{ij} = s_{ji} = \begin{cases} 10, & \text{if and only if } i \frac{\quad}{} j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases}$$

$$w_{ij} = w_{ji} = \begin{cases} 100, & \text{if and only if } i \longleftrightarrow j \text{ in } H; \\ 0, & \text{otherwise.} \end{cases}$$

Notice that because of the symmetric nature of lines and arcs S and W are symmetric, whereas B is not necessarily symmetric.

For instance consider the following general mixed graph.:



Notice that this graph is not of much interest per se, because it is not a stable graph, but it is introduced just to illustrate the structure of the adjacency matrix.

This graph 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 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 ⟨`type, label1, label2`⟩, where the `type` is the edge type and `label1` and `label2` are the labels of the two nodes. The edge type accepts `"a"` for a directed arrow , `"b"` for an arc 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 `makeMG`. 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),
              ug = UG(~ X*Q),
              bg = UG(~ Y*X + X*Q + Q*W + Y*Z))
```

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

Generating stable mixed graphs

There are four general classes of stable mixed graphs.

The more general class is that of ribbonless graphs: these are mixed graphs without a specific set of subgraphs called ribbons. Figure 3 below shows two examples of ribbons. The exact definition of ribbons is given in Sadeghi (2012a).

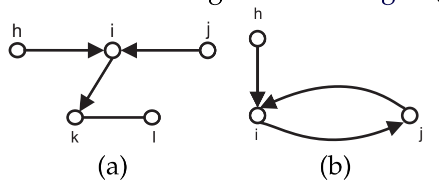


Figure 3: Two commonly seen ribbons $\langle h,i,j \rangle$.

The lack of ribbons ensures that, for any RG, there is a DAG whose independence structure, i.e. the set of all conditional independence statements that it induces, after marginalisation over and conditioning on two disjoint subsets of its node set can be represented by the given RG. This is essential as it shows that the independence structures corresponding to RGs are probabilistic, that is there exists a probability distribution P that is faithful with respect to any RG, i.e. for random vectors X_A, X_B, and X_C with probability distribution P, $X_A \perp\!\!\!\perp X_B \,|\, X_C$ if and only if $\langle A, B \,|\, C \rangle$ is in the induced independence structure by the graph. This probability distribution is the marginal and conditional of a probability distribution that is faithful to the generating DAG.

The other classes of stable graphs are further simplification of the class of ribbonless graphs. Summary graphs have in addition the property that there are neither arrowheads pointing to lines (i.e. $\prec\!\!-\!\!\succ \circ$ $-\!\!-$ or $-\!\!\succ \circ$ $-\!\!-$) nor directed cycles with all its arrows pointing towards one direction.

Ancestral graphs have the same constraints of summary graphs plus the additional prohibition of *bows*, i.e. arcs with one endpoint that is an ancestor of the other endpoint; see Richardson and Spirtes (2002).

However, notice that for some ribbonless and summary graphs the corresponding parametrisation is sometimes not available even in the case of standard joint Gaussian distribution.

If we suppose that stable mixed graphs are only used to represent the independence structure after marginalisation and conditioning, we can consider all types as equally appropriate. However, each of the three types has been used in different context and for different purposes. RGs have been introduced in order to straightforwardly deal with the problem of finding a class of graphs that is closed under marginalisation and conditioning by a simple process of deriving them from DAGs. SGs are used when the generating DAG is known and to trace the effects in the sets of regressions as described above. AGs are simple graphs, meaning that they do not contain multiple edges and the lack of bows ensures that they satisfy many desirable statistical properties.

In addition, when one traces the effects in regression models with latent and selection variables (as described in the introduction) ribbonless graphs are more alerting to possible distortions (due to indirect effects) than summary graphs, and summary graphs are more alerting than ancestral graphs; see also Wermuth and Cox (2008). For the exact definition and a thorough discussion of all such graphs, see Sadeghi (2012a).

Sadeghi (2012a) also defines the algorithms for generating stable mixed graphs of a specific type for a given DAG or for a stable mixed graphs of the same type after marginalisation and conditioning such that they induce the marginal and conditional DAG-independence structure. We implement these algorithms in this paper.

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

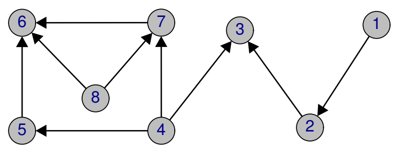
Functions to generate the three main types of stable mixed graphs. Three main functions `RG`, `SG`, and `AG` are available to generate and plot ribbonless, summary, and ancestral graphs from DAGs, using the algorithms in Sadeghi (2012a). These algorithms look for the paths with three nodes and two edges in the graph whose inner nodes are being marginalised 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 marginalisation and conditioning; see Sadeghi (2012a) for more details.

The functions `RG`, `SG`, and `AG` all have three arguments: `a`, the given input graph, `M`, the marginalisation 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)
```



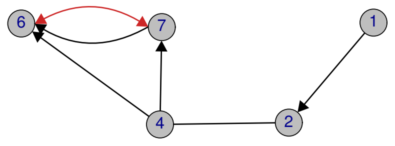
Then we define two disjoint sets M and C to marginalise over and condition on

```
> 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)
```

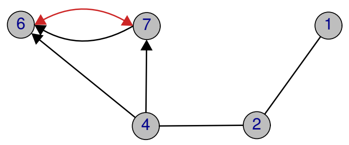
```
  1  2  4   6   7
1 0  1  0   0   0
2 0  0 10   0   0
4 0 10  0   1   1
6 0  0  0   0 100
7 0  0  0 101   0
```



The summary graph is also plotted:

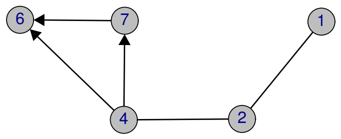
```
> plotGraph(SG(ex,M,C))
```

```
   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)
```



Verifying *m*-separation

To globally verify whether an independence statement of form $A \perp\!\!\!\perp B \mid C$ is implied by a mixed graph we use a separation criterion called *m-separation*. This has been defined in Sadeghi (2012a) for the general class of loopless mixed graphs and is the same as the *m*-separation criterion defined in Richardson and Spirtes (2002) for ancestral graphs. It is also a generalisation of the *d*-separation criterion for DAGs (Pearl, 1988). This is a graphical criterion that looks if the graph contains special paths connecting two sets A and B and involving a third set C of the nodes. These special paths are said to be active or *m*-connecting. For example, a directed path from a node in A to a node in B that does not contain any node of C is *m*-connecting A and B. However, if such a path intercepts a node in C then A and B are said to be *m*-separated given C. However, this behaviour can change if the path connecting A and B contains a collision node or a *collider* for short, that is a node c where the edges meet head-to-head, like e.g., $\longrightarrow\!\!\!\succ c \prec\!\!\!\longleftarrow$ or $\longrightarrow\!\!\!\succ c \prec\!\!\!\longrightarrow\!\!\!\succ$.

In general, a path is said to be *m-connecting* given C if all its collider nodes are in C or in the set of ancestors of C, and all its non-collider nodes are outside C. For two disjoint subsets A and B of the node set, we say that C *m*-separates A and B if there is no *m*-connecting path between A and B given C.

Function for verifying *m*-separation. The *m*-separation criterion has been implemented in **ggm** and is available by using the function `msep`. Note that there is still a function `dSep` in **ggm** for *d*-separation, although it is superseded by `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`.

Examples. For example, consider the DAG of Figure 1(a):

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

Then, we see that Y and W are *m*-separated given Z:

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

```
[1] TRUE
```

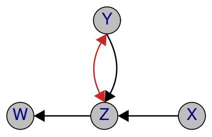
and the same statement holds for the induced ancestral graph after marginalisation over `U`:

```
> b <- AG(a, M = "U")
> msep(b,"Y","W","Z")

[1] TRUE
```

This was expected because the induced ancestral graph respects all the independence statements induced by m-separation in the DAG, and not involving the variable `U`.

As a more complex example, consider the following summary graph,

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

> plotGraph(a)
```



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

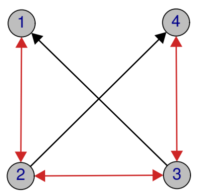
```
> msep(a,"X","Y","Z")

[1] FALSE

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

[1] TRUE
```

Verifying maximality

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 (2012). Sadeghi and Lauritzen (2012) also gave an algorithm for generating maximal ribbonless graphs that induces the same independence structure to an input non-maximal ribbonless graph. This algorithm has been implemented here in **ggm** as illustrated below.

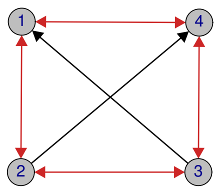
Function for generating maximal graphs. Given a non-maximal graph, we can obtain the adjacency matrix of a maximal graph that induces the same independence statements with the function `Max`. This function uses the algorithm by Sadeghi (2012b), which is an extension of the implicit algorithm presented in Richardson and Spirtes (2002). 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 non-maximal ancestral graph, with the missing edge between nodes 1 and 4 that is not associated with any independence statement. 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
```

Verifying Markov equivalence

Two graphical models are said to be Markov equivalent when their associated graphs, although nonidentical, imply the same independence structure, that is the same set of independence statements. Thus two Markov equivalent models cannot be discriminated on the basis of statistical tests of independence, even for arbitrary large samples. For instance, it is easy to verify that the two directed acyclic graphs models $X \prec\!\!-\!\!- U \!\!-\!\!-\!\!\succ Y$ and $X \prec\!\!-\!\!- U \prec\!\!-\!\!- Y$ both imply the same independence statements, and are, therefore, Markov equivalent.

Sometimes, we can check whether graphs of different types are Markov equivalent. For instance the

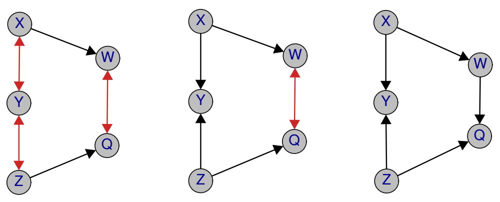
DAG $X \rightarrow U \leftarrow Y$ is Markov equivalent to the bi-directed graph $X \leftrightarrow U \leftrightarrow Z$.

Markov equivalent models may be useful in applications because (a) the may suggest alternative interpretations of a given well-fitting model or (b) on the basis of the equivalence one can choose a simpler fitting algorithm. For instance, the previous bi-directed graph model may be fitted, using the Markov equivalent DAG, in terms of a sequence of univariate regressions.

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 a certain type from a given graph.

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 (2012). The arguments of this function are the two graphs a and b in one of the allowed forms. For example,

```
> H1 = makeMG(dg = DAG(W ~ X, Q ~ Z),
       bg = UG(~ X*Y + Y*Z + W*Q))
> H2 = makeMG(dg = DAG(W ~ X, Q~ Z,
                       Y ~ X + Z),
              bg = UG(~ W*Q))
> H3 = DAG(W~X, Q~Z + W,Y~X + Z)

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

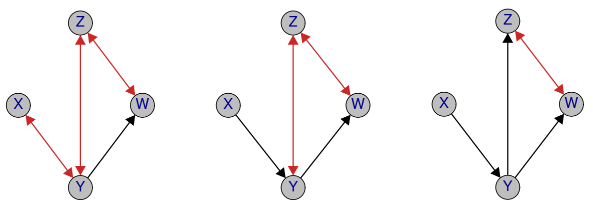


We can now verify Markov equivalence as follows

```
> MarkEqRcg(H1,H2)

[1] TRUE

> MarkEqRcg(H1,H3)

[1] FALSE

> MarkEqRcg(H2,H3)

[1] FALSE
```

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 maximal ancestral graphs). For example,

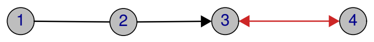
```
> A1<- makeMG(dg = DG(W~Y),
       bg =  UG(~ X*Y + Y*Z + Z*W))
> A2<- makeMG(dg = DG(W~Y, Y~X),
       bg = UG(~ Y*Z + Z*W))
> A3<- makeMG(dg = DG(W~Y, Y~X, Z~Y),
       bg = UG(~ Z*W))

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



```
> MarkEqMag(H1,H2)

[1] TRUE

> MarkEqMag(H1,H3)

[1] FALSE

> MarkEqMag(H2,H3)

[1] FALSE
```

Functions for generating Markov equivalent graphs of a specific type. To obtain an alternative interpretation of an independence structure by using different graphical models, it 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 (2012b). For example, given the following graph

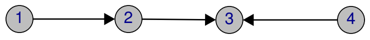
```
> H <- matrix(c( 0,10,  0,  0,
                10, 0,  0,  0,
                 0, 1,  0,100,
                 0, 0,100,  0),4,4)

> plotGraph(H)
```



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

```
> RepMarDAG(H)

$verify
[1] TRUE

$amat
  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)

$verify
[1] FALSE

$amat
[1] NA

> RepMarBG(H)

$verify
[1] FALSE

$amat
[1] NA
```

Acknowledgments

The authors are grateful to Steffen Lauritzen for helpful suggestions on codes and comments on an earlier version of the paper and Nanny Wermuth, the editor, and referees for their insightful comments.

Bibliography

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.

D. R. Cox and N. Wermuth. *Multivariate Dependencies : models, analysis and interpretation*. Chapman & Hall, London, United Kingdom, 1996.

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`.

S. Højsgaard, D. Edwards, and S. Lauritzen. *Graphical Models with R*. Springer-Verlag, Berlin-Heidelberg-New York, 2012.

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

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. *Bernoulli*, To appear, 2012a. URL `http://arxiv.org/abs/1110.4168`.

K. Sadeghi. Markov equivalences for subclasses of mixed graphs. *submitted*, 2012b. URL `http://arxiv.org/abs/1110.4539`.

K. Sadeghi and S. L. Lauritzen. Markov properties for mixed graphs. *submitted*, 2012. URL `http://arxiv.org/abs/1109.5909`.

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

N. Wermuth and D. R. Cox. Distortion of effects caused by indirect confounding. *Biometrika*, 95:17–33, 2008.

N. Wermuth and K. Sadeghi. Sequences of regressions and their independences. *TEST*, 21(2):215–252 and 274–279, 2012.

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

Giovanni M. Marchetti
Dipartimento di Statistica "G. Parenti"
University of Florence
viale Morgagni, 59, 50134, Florence
Italy
`giovanni.marchetti@ds.unifi.it`