Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Causal Inference using Graphical Models with the R Package pcalg

Markus Kalisch
ETH Zurich

Martin Mächler
ETH Zurich

Diego Colombo
ETH Zurich

Marloes H. Maathuis
ETH Zurich

Peter Bühlmann
ETH Zurich

Abstract

The **pcalg** package for R (R Development Core Team (2010)) can be used for the following two purposes: Causal structure learning and estimation of causal effects from observational data. In this document, we give a brief overview of the methodology, and demonstrate the package's functionality in both toy examples and applications.

Keywords: IDA, PC, RFCI, FCI, do-calculus, causality, graphical model, R.

1. Introduction

Understanding cause-effect relationships between variables is of primary interest in many fields of science. Usually, experimental intervention is used to find these relationships. In many settings, however, experiments are infeasible because of time, cost or ethical constraints.

We therefore consider the problem of inferring causal information from observational data. Under some assumptions, the PC algorithm (see Spirtes, Glymour, and Scheines (2000)), the FCI algorithm (see Spirtes *et al.* (2000) and Spirtes, Meek, and Richardson (1999a)) and the RFCI algorithm (see Colombo, Maathuis, Kalisch, and Richardson (2012)) can infer information about the causal structure from observational data. Thus, these algorithms tell us which variables could or could not be a cause of some variable of interest. They do not, however, give information about the size of the causal effects. We therefore developed the IDA method (see Maathuis, Kalisch, and Bühlmann (2009)), which can infer bounds on causal effects based on observational data under some assumptions. IDA was validated on a large-scale biological system (see Maathuis, Colombo, Kalisch, and Bühlmann (2010)).

For broader use of these methods, well documented and easy to use software is indispensable. We therefore wrote the R-package **pcalg**, which contains implementations of the PC-, FCI- and RFCI-algorithms, as well as of the IDA method. The objective of this paper is to introduce the R package **pcalg**, explain the range of functions on simulated data sets and summarize some applications.

To get started, we show how two of the main functions (one for causal structure learning and one for estimating causal effects from observational data) can be used in a typical application. Suppose we have a system described by some variables and many observations of this system. Furthermore, assume that it seems plausible that there are no hidden variables and no feedback loops in the underlying causal system. The causal structure of such a system can be conveniently represented by a directed acyclic graph (DAG), where each node represents a variable and each directed edge represents a direct cause. To fix ideas, we have simulated an example data set with $p = 8$ continuous variables with Gaussian noise and $n = 5000$ observations, which we will now analyse. First, we load the package **pcalg** and the data set.

```
R> library("pcalg")
R> data("gmG")
```

In the next step, we use the function `pc()` to produce an estimate of the underlying causal structure. Since this function is based on conditional independence tests, we need to define two things. First, we need a function that can compute conditional independence tests in a way that is suitable for the data at hand. For standard data types (Gaussian, discrete and binary) we provide predefined functions. See the example section in the help file of `pc()` for more details. Secondly, we need a summary of the data (sufficient statistic) on which the conditional independence function can work. Each conditional independence test can be performed at a certain significance level `alpha`. This can be treated as a tuning parameter. In the following code, we use the predefined function `gaussCItest()` as conditional independence test and create the corresponding sufficient statistic, consisting of the correlation matrix of the data and the sample size. Then we use the function `pc()` to estimate the causal structure and plot the result.

```
R> suffStat <- list(C = cor(gmG$x), n = nrow(gmG$x))
R> pc.fit <- pc(suffStat, indepTest = gaussCItest,
              p = ncol(gmG$x), alpha = 0.01)
R> stopifnot(require(Rgraphviz))# needed for all our graph plots
R> par(mfrow = c(1,2))
R> plot(gmG$g, main = "")
R> plot(pc.fit, main = "")
```

As can be seen in Fig. 1, there are directed and bidirected edges in the estimated causal structure. The directed edges show the presence and direction of direct causal effects. A bidirected edge means that the PC-algorithm was unable to decide whether the edge orientation should be $\leftarrow$ or $\rightarrow$. Thus, bidirected edges represent some uncertainty in the resulting model. They reflect the fact that in general one cannot estimate a unique DAG from observational data, not even with an infinite amount of data, since several DAGs can describe the same conditional independence information.

On the inferred causal structure, we can estimate the causal effect of an intervention. Denote the variable corresponding to node i in the graph by V_i. For example, suppose that, by external intervention, we first set the variable V_1 to some value $\tilde{x}$, and then to the value $\tilde{x}+1$.

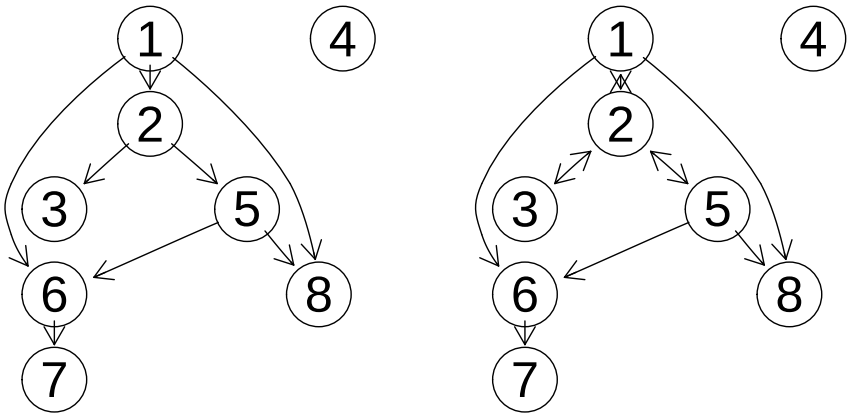

Figure 1: True underlying causal DAG (left) and estimated causal structure (right), representing a Markov equivalence class of DAGs that all encode the same conditional independence information. (Due to the large sample size, there were no sampling errors.)

The recorded average change in variable V_6 is the (total) causal effect of V_1 on V_6. More precisely, the causal effect $C(V_1, V_6, \tilde{x})$ of V_1 from $V_1 = \tilde{x}$ on V_6 is defined as

$$\begin{aligned} C(V_1, V_6, \tilde{x}) &= E(V_1|do(V_6 = \tilde{x} + 1)) - E(V_1|do(V_6 = \tilde{x})) \text{ or} \\ C(V_1, V_6, \tilde{x}) &= \frac{\partial}{\partial x} E(V_1|do(V_6 = x))|_{x=\tilde{x}}, \end{aligned}$$

where $do(V_6 = x)$ denotes Pearl's do-operator (see Pearl (2000)). If the causal relationships are linear, these two expressions are equivalent and do not depend on $\tilde{x}$.

Since the causal structure was not identified uniquely in our example, we cannot expect to get a unique number for the causal effect. Instead, we get a set of possible causal effects. This set can be computed by using the function `ida()`. To provide full quantitative information, we need to pass the covariance matrix in addition to the estimated causal structure.

```
R> ida(1, 6, cov(gmG$x), pc.fit@graph)

[1] 0.75364 0.54878
```

Since we simulated the data, we know that the true value of the causal effect is 0.71. Thus, one of the two estimates is indeed close to the true value. Since both values are larger than zero, we can conclude that variable V_1 has a positive causal effect on variable V_6. Thus, we can always estimate a lower bound for the absolute value of the causal effect. (Note that at this point we have no p-value to control the sampling error.)

If we would like to know the effect of a unit increase in variable V_1 on variables V_4, V_5 and V_6, we could simply call `ida()` three times. However, a faster way is to call the function `idaFast()`, which was tailored for such situations.

```
R> idaFast(1, c(4,5,6), cov(gmG$x), pc.fit@graph)
```

```
     [,1]     [,2]
4 0.01027 0.012014
5 0.23875 0.017935
6 0.75364 0.548776
```

Each row in the output shows the estimated set of possible causal effects on the target variable indicated by the row names. The true values for the causal effects are 0, 0.2, 0.71 for variables V_4, V_5 and V_6, respectively. The first row, corresponding to variable V_4, quite accurately indicates a causal effect that is very close to zero or no effect at all. The second row of the output, corresponding to variable V_5, is rather uninformative: Although one entry comes close to the true value, the other estimate is close to zero. Thus, we cannot be sure if there is a causal effect at all. The third row, corresponding to V_6 was already discussed above.

2. Methodological background

In Section 2.1 we propose methods for estimating the causal structure from observational data. In particular, we discuss the PC algorithm (see Spirtes *et al.* (2000))), the FCI algorithm (see Spirtes *et al.* (2000) and Spirtes *et al.* (1999a)) and the RFCI algorithm (see Colombo *et al.* (2012)). In Section 2.2 we describe the IDA method (see Maathuis *et al.* (2009)) to obtain bounds on causal effects from observational data. This method is based on first estimating the causal structure and then applying do-calculus (see Pearl (2000))

2.1. Estimating causal structures with graphical models

Graphical models can be thought of as maps of dependence structures of a given probability distribution or a sample thereof (see for example Lauritzen (1996)). In order to illustrate the analogy, let us consider a road map. In order to be able to use a road map, one needs two given factors. First, one needs the physical map with symbols such as dots and lines. Second, one needs a rule for interpreting the symbols. For instance, a railroad map and a map for electric circuits might look very much alike, but their interpretation differs a lot. In the same sense, a graphical model is a map. First, a graphical model consists of a graph with dots, lines and potentially edge marks like arrowheads or circles. Second, a graphical model always comes with a rule for interpreting this graph. In general, nodes in the graph represent (random) variables and edges represent some kind of dependence.

Without hidden and selection variables

An example of a graphical model is the DAG model. The physical map here is a graph consisting of nodes and directed edges ($\leftarrow$ or $\rightarrow$). As a further restriction, the edges must be directed in a way, so that it is not possible to trace a cycle when following the arrowheads (i.e., no directed cycles). The interpretation rule is called d-separation. This rule is a bit intricate and we refer the reader to Lauritzen (1996) for more details. This interpretation rule can be used in the following way: If two nodes x and y are d-separated by a set of nodes S, then the corresponding random variables V_x and V_y are conditionally independent given the set of random variables V_S. For the following, we only deal with distributions where the following holds: For each distribution, it is possible to find a DAG, whose list of d-separation relations perfectly matches the list of conditional independencies of the distribution. Such distributions are called faithful. It has been shown that the set of distributions that are

faithful is the overwhelming majority (Meek (1995)), so that the assumption does not seem to be very strict in practice.

Since the DAG model encodes conditional independencies, it seems plausible that information on the latter helps to infer aspects of the former. This intuition is made precise in the PC algorithm (see Spirtes *et al.* (2000); PC stands for the initials of its inventors Peter Spirtes and Clark Glymour) which was proven to reconstruct the structure of the underlying DAG model given a conditional independence oracle up to its Markov equivalence class which is discussed in more detail below. In practice, the conditional independence oracle is replaced by a statistical test for conditional independence. For situations without hidden variables and under some further conditions it has been shown that the PC algorithm using statistical tests instead of an independence oracle is computationally feasible and consistent even for very high-dimensional sparse DAGs (see Kalisch and Bühlmann (2007)).

As mentioned before, several DAGs can encode the same list of conditional independencies. One can show that such DAGs must share certain properties. To be more precise, we have to define a v-structure as the subgraph $i \to j \leftarrow k$ on the nodes i, j and k where i and k are not adjacent (i.e., , there is no edge between i and k). Furthermore, let the skeleton of a DAG be the graph that is obtained by removing all arrowheads from the DAG. It was shown that two DAGs encode the same conditional independence statements if and only if the corresponding DAGs have the same skeleton and the same v-structures (see Verma and Pearl (1991)). Such DAGs are called Markov-equivalent. In this way, the space of DAGs can be partitioned into equivalence classes, where all members of an equivalence class encode the same conditional independence information. Conversely, if given a conditional independence oracle, one can only determine a DAG up to its equivalence class. Therefore, the PC algorithm cannot determine the DAG uniquely, but only the corresponding equivalence class of the DAG.

An equivalence class can be visualized by a graph that has the same skeleton as every DAG in the equivalence class and directed edges only where all DAGs in the equivalence class have the same directed edge. Edges that point into one direction for some DAGs in the equivalence class and in the other direction for other DAGs in the equivalence class are visualized by bidirected edges (sometimes, undirected edges are used instead). This graph is called a completed partially directed acyclic graph (CPDAG).

Algorithm 1 Outline of the PC-algorithm

Input: Vertex set V, conditional independence information, significance level α
Output: Estimated CPDAG $\hat{G}$, separation sets $\hat{S}$
Edge types: $\to$, $-$
(P1) Form the complete undirected graph on the vertex set V
(P2) Test conditional independence given subsets of adjacency sets at a given significance level α and delete edges if conditional independent
(P3) Orient v-structures
(P4) Orient remaining edges.

We now describe the PC-algorithm, which is shown in Algorithm 1, in more detail. The PC-algorithm starts with a complete undirected graph, G_0, as stated in **(P1)** of Algorithm 1. In stage **(P2)**, a series of conditional independence tests is done and edges are deleted in the following way. First, all pairs of nodes are tested for marginal independence. If two nodes i and j are judged to be marginally independent at level α, the edge between them is deleted

and the empty set is saved as separation sets $\hat{S}[i,j]$ and $\hat{S}[j,i]$. After all pairs have been tested for marginal independence and some edges might have been removed, a graph results which we denote by G_1. In the second step, all pairs of nodes (i,j) still adjacent in G_1 are tested for conditional independence given any single node in $\text{adj}(G_1,i)\setminus\{j\}$ or $\text{adj}(G_1,j)\setminus\{i\}$ ($\text{adj}(G,i)$ denotes the set of nodes in graph G that are adjacent to node i) . If there is any node k such that V_i and V_j are conditionally independent given V_k, the edge between i and j is removed and node k is saved as separation sets (sepset) $\hat{S}[i,j]$ and $\hat{S}[j,i]$. If all adjacent pairs have been tested given one adjacent node, a new graph results which we denote by G_2. The algorithm continues in this way by increasing the size of the conditioning set step by step. The algorithm stops if all adjacency sets in the current graph are smaller than the size of the conditioning set. The result is the skeleton in which every edge is still undirected. Within **(P3)**, each triple of vertices (i,k,j) such that the pairs (i,k) and (j,k) are each adjacent in the skeleton but (i,j) are not (such a triple is called an "unshielded triple"), is oriented based on the information saved in the conditioning sets $\hat{S}[i,j]$ and $\hat{S}[j,i]$. More precisely, an unshielded triple $i-k-j$ is oriented as $i\to k\leftarrow j$ if k is not in $\hat{S}[j,i]=\hat{S}[i,j]$. Finally, in **(P4)** it may be possible to orient some of the remaining edges, since one can deduce that one of the two possible directions of the edge is invalid because it introduces a new v-structure or a directed cycle. Such edges are found by repeatedly applying rules described in Spirtes *et al.* (2000), p.85. The resulting output is the equivalence class (CPDAG) that describes the conditional independence information in the data, in which every edge is either undirected or directed. (To simplify visual presentation, undirected edges are depicted as bidirected edges in the output as soon as at least one directed edge is present. If no directed edge is present, all edges are undirected.)

A causal structure without feedback loops and without hidden or selection variable can be visualized using a DAG where the edges indicate direct cause-effect relationships. Under some assumptions, Pearl (2000) showed (Th. 1.4.1) that there is a link between causal structures and graphical models. Roughly speaking, if the underlying causal structure is a DAG, we observe data generated from this DAG and then estimate a DAG model (i.e., a graphical model) on this data, the estimated CPDAG represents the equivalence class of the DAG model describing the causal structure. This holds if we have enough samples and assuming that the true underlying causal structure is indeed a DAG without latent or selection variables. Note that even given an infinite amount of data, we usually cannot identify the true DAG itself, but only its equivalence class. Every DAG in this equivalence class can be the true causal structure.

With hidden or selection variables

When discovering causal relations from nonexperimental data, two difficulties arise. One is the problem of hidden (or latent) variables: Factors influencing two or more measured variables may not themselves be measured. The other is the problem of selection bias: Values of unmeasured variables or features may influence whether a unit is included in the data sample.

In the case of hidden or selection variables, one could still visualize the underlying causal structure with a DAG that includes all observed, hidden and selection variables. However, when inferring the DAG from observational data, we do not know all hidden and selection variables.

We therefore seek to find a structure that represents all conditional independence relationships among the observed variables given the selection variables of the underlying causal structure. It turns out that this is possible. However, the resulting object is in general not a DAG for the following reason. Suppose, we have a DAG including observed, latent and selection variables and we would like to visualize the conditional independencies among the observed variables only. We could marginalize out all latent variables and condition on all selection variables. It turns out that the resulting list of conditional independencies can in general not be represented by a DAG, since DAGs are not closed under marginalization or conditioning (see Richardson and Spirtes (2002)).

A class of graphical independence models that is closed under marginalization and conditioning and that contains all DAG models is the class of ancestral graphs. A detailed discussion of this class of graphs can be found in Richardson and Spirtes (2002). In this text, we only give a brief introduction.

Ancestral graphs have nodes, which represent random variables and edges which represent some kind of dependence. The edges can be either directed ($\leftarrow$ or $\rightarrow$), undirected (-) or bidirected ($\leftrightarrow$) (note that in the context of ancestral graphs, undirected and bidirected edges do *not* mean the same). There are two rules that restrict the direction of edges in an ancestral graph:

1: If i and j are joined by an edge with an arrowhead at i, then there is no directed path from i to j. (A path is a sequence of adjacent vertices, and a directed path is a path along directed edges that follows the direction of the arrowheads.)

2: There are no arrowheads present at a vertex which is an endpoint of an undirected edge.

Maximal ancestral graphs (MAG), which we will use from now on, also obey a third rule:

3: Every missing edge corresponds to a conditional independence.

The conditional independence statements of MAGs can be read off using the concept of m-separation, which is a generalization the concept of d-separation. Furthermore, part of the causal information in the underlying DAG is represented in the MAG. If in the MAG there is an edge between node i and node j with an arrowhead at node i, then there is no directed path from node i to node j nor to any of the selection variables in the underlying DAG (i.e., , i is not a cause of j or of the selection variables). If, on the other hand, there is a tail at node i, then there is a directed path from node i to node j or to one of the selection variables in the underlying DAG (i.e., , i is a cause of j or of a selection variable).

Recall that finding a unique DAG from an independence oracle is in general impossible. Therefore, one only reports on the equivalence class of DAGs in which the true DAG must lie. The equivalence class is visualized using a CPDAG. The same is true for MAGs: Finding a unique MAG from an independence oracle is in general impossible. One only reports on the equivalence class in which the true MAG lies.

An equivalence class of a MAG can be uniquely represented by a partial ancestral graph (PAG) (see e.g., Zhang (2008)). A PAG contains the following types of edges: o–o, o–, o–>, –>, <–>, –. Roughly, the bidirected edges come from hidden variables, and the undirected edges come from selection variables. The edges have the following interpretation: (i) There is an edge between x and y if and only if V_x and V_y are conditionally dependent given V_S for

all sets V_S consisting of all selection variables and a subset of the observed variables; (ii) a tail on an edge means that this tail is present in all MAGs in the equivalence class; (iii) an arrowhead on an edge means that this arrowhead is present in all MAGs in the equivalence class; (iv) a o-edgemark means that there is a at least one MAG in the equivalence class where the edgemark is a tail, and at least one where the edgemark is an arrowhead.

An algorithm for finding the PAG given an independence oracle is the FCI algorithm ("fast causal inference"; see Spirtes *et al.* (2000) and Spirtes, Meek, and Richardson (1999b)). The orientation rules of this algorithm were slightly extended and proven to be complete in Zhang (2008). FCI is very similar to PC but makes additional conditional independence tests and uses more orientation rules (see Section 3.3 for more details). We refer the reader to Zhang (2008) or Colombo *et al.* (2012) for a detailed discussion of the FCI algorithm. It turns out that the FCI algorithm is computationally infeasible for large graphs. The RFCI algorithm ("really fast causal inference"; see Colombo *et al.* (2012)), is much faster than FCI. The output of RFCI is in general slightly less informative than the output of FCI, in particular with respect to conditional independence information. However, it was shown in Colombo *et al.* (2012) that any causal information in the output of RFCI is correct and that both FCI and RFCI are consistent in (different) sparse high-dimensional settings. Finally, in simulations the estimation performances of the algorithms are very similar.

2.2. Estimating bounds on causal effects

One way of quantifying the causal effect of variable V_x on V_y is to measure the state of V_y if V_x is forced to take value $V_x = x$ and compare this to the value of V_y if V_x is forced to take the value $V_x = x + 1$ or $V_x = x + \delta$. If V_x and V_y are random variables, forcing $V_x = x$ could have the effect of changing the distribution of V_y. Following the conventions in Pearl (2000), the resulting distribution after manipulation is denoted by $P[V_y|do(V_x = x)]$. Note that this is different from the conditional distribution $P[V_y|V_x = x]$. To illustrate this, imagine the following simplistic situation. Suppose we observe a particular spot on the street during some hour. The random variable V_x denotes whether it rained during that hour ($V_x = 1$ if it rained, $V_x = 0$ otherwise). The random variable V_y denotes whether the street was wet at the end of that hour ($V_y = 1$ if it was wet, $V_y = 0$ otherwise). If we assume $P(V_x = 1) = 0.1$ (rather dry region), $P(V_y = 1|V_x = 1) = 0.99$ (the street is almost always still wet at the end of the hour when it rained during that hour) and $P(V_y = 1|V_x = 0) = 0.02$ (other reasons for making the street wet are rare), we can compute the conditional probability $P(V_x = 1|V_y = 1) = 0.85$. So, if we observe the street to be wet, the probability that there was rain in the last hour is about 0.85. However, if we take a garden hose and force the street to be wet at a randomly chosen hour, we get $P(V_x = 1|do(V_y = 1)) = P(V_x = 1) = 0.1$. Thus, the distribution of the random variable describing rain is quite different when making an observation versus making an intervention.

Oftentimes, only the change of the target distribution under intervention is reported. We use the change in mean, i.e., $\frac{\partial}{\partial x} E[V_y|do(V_x = x)]$, as a general measure for the causal effect of V_x on V_y. For multivariate Gaussian random variables, $E[V_y|do(V_x = x)]$ depends linearly on x. Therefore, the derivative is constant which means that the causal effect does not depend on x, and can also be interpreted as $E[V_y|do(V_x = x + 1)] - E[V_y|do(V_x = x)]$. For binary random variables (with domain $\{0, 1\}$) we define the causal effect of V_x on V_y as $E(V_y|do(V_x = 1)) - E(V_y|do(V_x = 0)) = P(V_y = 1|do(V_x = 1)) - P(V_y = 1|do(V_x = 0))$.

The goal in the remainder of this section is to estimate the effect of an intervention if only observational data is available.

Without hidden and selection variables

If the causal structure is a known DAG and there are no hidden and selection variables, Pearl (2000) (Th 3.4.1) suggested a set of inference rules known as "do-calculus" whose application transforms an expression involving a "do" into an expression involving only conditional distributions. Thus, information on the interventional distribution can be obtained by using information obtained by observations and knowledge of the underlying causal structure.

Unfortunately, the causal structure is rarely known in practice. However, as discussed in Section 2.1, we can estimate the Markov equivalence class of the true causal DAG. Taking this into account, we conceptually apply the do-calculus on each DAG within the equivalence class and thus obtain a possible causal effect for each DAG in the equivalence class (in practice, we developed a local method that is faster but yields a similar result; see Section 3.5 for more details). Therefore, even if we have an infinite amount of observations we can in general report on a multiset of possible causal values (it is a multiset rather than a set because it can contain duplicate values). One of these values is the true causal effect. Despite the inherent ambiguity, this result can still be very useful when the multiset has certain properties (e.g., all values are much larger than zero). These ideas are incorporated in the IDA method (**I**ntervention calculus when the **D**AG is **A**bsent).

In addition to this fundamental limitation in estimating a causal effect, errors due to finite sample size blur the result as with every statistical method. Thus, we can typically only get an estimate of the set of possible causal values. It was shown that this estimate is consistent in sparse high-dimensional settings under some assumptions by Maathuis *et al.* (2009).

It has recently been shown empirically that despite the described fundamental limitations in identifying the causal effect uniquely and despite potential violations of the underlying assumptions, the method performs well in identifying the most important causal effects in a high-dimensional yeast gene expression data set (see Maathuis *et al.* (2010)).

With hidden and selection variables

At the moment, we can not yet estimate causal effects when hidden variables or selection variables are present.

2.3. Summary of assumptions

For all proposed methods, we assume that the data is faithful to the unknown underlying causal DAG. For the individual methods, further assumptions are made.

PC-algorithm: No hidden or selection variables; consistent in high-dimensional settings (the number of variables grows with the sample size) if the underlying DAG is sparse, the data is multivariate Normal and satisfies some regularity conditions on the partial correlations, and α is taken to zero appropriately. See Kalisch and Bühlmann (2007) for full details. Consistency in a standard asymptotic regime with a fixed number of variables follows as a special case.

FCI-algorithm: Allows for hidden and selection variables; consistent in high-dimensional

settings if the so-called Possible-D-SEP sets (see Spirtes *et al.* (2000)) are sparse, the data is multivariate Normal and satisfies some regularity conditions on the partial correlations, and α is taken to zero appropriately. See Colombo *et al.* (2012) for full details. Consistency in a standard asymptotic regime with a fixed number of variables follows as a special case.

RFCI-algorithm: Allows for hidden and selection variables; consistent in high-dimensional settings if the underlying MAG is sparse (this is a much weaker assumption than the one needed for FCI), the data is multivariate Normal and satisfies some regularity conditions on the partial correlations, and α is taken to zero appropriately. See Colombo *et al.* (2012) for full details. Consistency in a standard asymptotic regime with a fixed number of variables follows as a special case.

IDA: No hidden or selection variables; all conditional expectations are linear; consistent in high-dimensional settings if the underlying DAG is sparse, the data is multivariate Normal and satisfies some regularity conditions on the partial correlations and conditional variances, and α is taken to zero appropriately. See Maathuis *et al.* (2009) for full details.

3. Package pcalg

This package has two goals. First, it is intended to provide fast, flexible and reliable implementations of the PC, FCI and RFCI algorithms for estimating causal structures and graphical models. Second, it provides an implementation of the IDA method, which estimates bounds on causal effects from observational data when no causal structure is known and hidden or selection variables are absent.

In the following, we describe the main functions of our package for achieving these goals. The functions `skeleton()`, `pc()`, `fci()` and `rfci()` are intended for estimating graphical models. The functions `ida()` and `idaFast()` are intended for estimating causal effects from observational data.

Alternatives to this package for estimating graphical models in R include: Scutari (2010), Bottcher and Dethlefsen (2011), Hojsgaard (2012), Hojsgaard, Dethlefsen, and Bowsher (2012) and Hojsgaard and Lauritzen (2011).

3.1. skeleton

The function `skeleton()` implements (P1) and (P2) from Algorithm 1. The function can be called with the following arguments

```
skeleton(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
    fixedEdges = NULL, NAdelete = TRUE, m.max = Inf)
```

As was discussed in Section 2.1, the main task in finding the skeleton is to compute and test several conditional independencies. To keep the function flexible, `skeleton()` takes as argument a function `indepTest()` that performs these conditional independence tests and returns a p-value. All information that is needed in the conditional independence test can be passed in the argument `suffStat`. The only exceptions are the number of variables `p`

```
R> require("pcalg")
R> data("gmG")
R> suffStat <- list(C = cor(gmG$x), n = nrow(gmG$x))
R> pc.fit <- skeleton(suffStat, indepTest = gaussCItest,
                      p = ncol(gmG$x), alpha = 0.01)
R> par(mfrow = c(1,2))
R> plot(gmG$g, main = "")
R> plot(pc.fit, main = "")
```

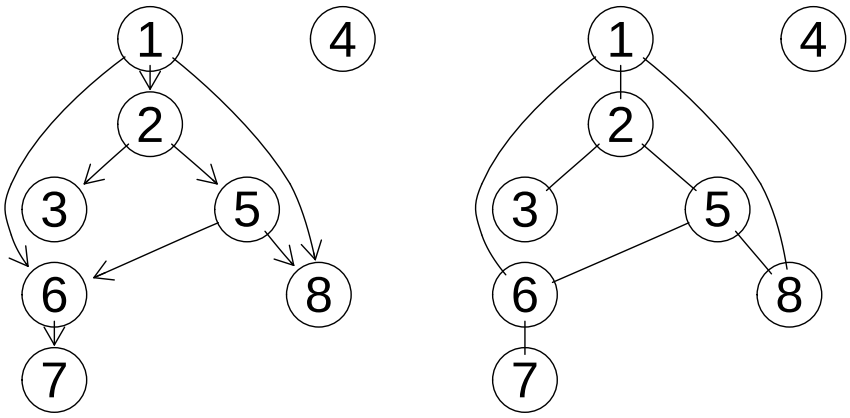

Figure 2: True underlying DAG (left) and estimated skeleton (right) fitted on the simulated Gaussian data set `gmG`.

and the significance level `alpha` for the conditional independence tests, which are passed separately. For convenience, we have preprogrammed versions of `indepTest()` for Gaussian data (`gaussCItest()`), discrete data (`disCItest()`), and binary data (`binCItest()`). Each of these independence test functions needs different arguments as input, described in the respective help files. For example, when using `gaussCItest()`, the input has to be a list containing the correlation matrix and the sample size of the data. In the following code, we estimate the skeleton on the data set `gmG` (which consists of $p = 8$ variables and $n = 5000$ samples) and plot the results. The estimated skeleton and the true underlying DAG are shown in Fig. 2.

To give another example, we show how to fit a skeleton to the example data set `gmD` (which consists of $p = 5$ discrete variables with 3, 2, 3, 4 and 2 levels and $n = 10000$ samples). The predefined test function `disCItest()` is based on the G^2 statistic and takes as input a list containing the data matrix, a vector specifying the number of levels for each variable and an option which indicates if the degrees of freedom must be lowered by one for each zero count. Finally, we plot the result. The estimated skeleton and the true underlying DAG are shown in Fig. 3.

In some situations, one may have prior information about the underlying DAG, for example that certain edges are absent or present. Such information can be incorporated into the algorithm via the arguments `fixedGaps` (absent edges) and `fixedEdges` (present edges). The

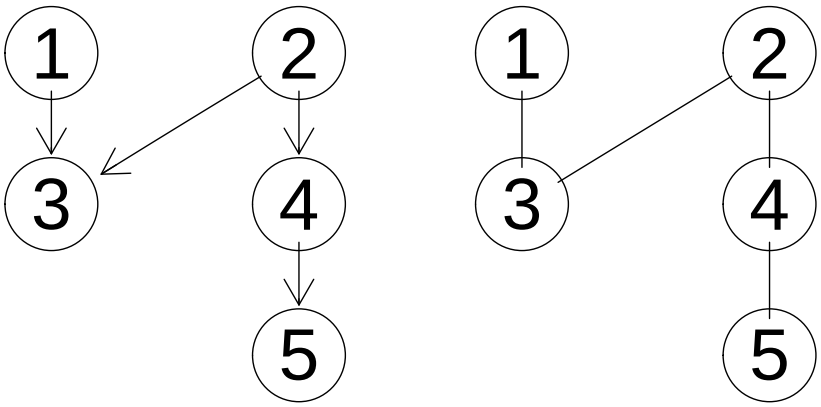

Figure 3: True underlying DAG (left) and estimated skeleton (right) fitted on the simulated discrete data set `gmD`.

information in `fixedGaps` and `fixedEdges` is used as follows. The gaps given in `fixedGaps` are introduced in the very beginning of the algorithm by removing the corresponding edges from the complete undirected graph. Thus, these edges are guaranteed to be absent in the resulting graph. Pairs (i, j) in `fixedEdges` are skipped in all steps of the algorithm, so that these edges are guaranteed to be present in the resulting graph.

If `indepTest()` returns `NA` and the option `NAdelete` is `TRUE`, the corresponding edge is deleted. If this option is `FALSE`, the edge is not deleted.

The argument `m.max` is the maximum size of the conditioning sets that are considered in the conditional independence tests in (P2) of Algorithm 1.

Throughout, the function works with the column positions of the variables in the adjacency matrix, and not with the names of the variables.

3.2. pc

The function `pc()` implements all steps (P1) to (P4) of the PC algorithm shown in display algorithm 1. First, the skeleton is computed using the function `skeleton()` (steps (P1) and (P2)). Then, as many edges as possible are oriented (steps (P3) and (P4)). The function can be called as

```
pc(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
    fixedEdges = NULL, NAdelete = TRUE, m.max = Inf, u2pd = "rand",
    conservative = FALSE)
```

where the arguments `suffStat`, `indepTest`, `p`, `alpha`, `fixedGaps`, `fixedEdges`, `NAdelete` and `m.max` are identical to those of `skeleton()`.

Sampling errors (or hidden variables) can lead to conflicting information about edge directions. For example, one may find that $a - b - c$ and $b - c - d$ should both be oriented as v-structures. This gives conflicting information about the edge $b - c$, since it should be oriented as $b \leftarrow c$ in v-structure $a \rightarrow b \leftarrow c$, while it should be oriented as $b \rightarrow c$ in v-structure $b \rightarrow c \leftarrow d$. In

such cases, we simply overwrite the directions of the conflicting edge. In the example above this means that we obtain $a \to b \to c \leftarrow d$ if $a - b - c$ was visited first, and $a \to b \leftarrow c \leftarrow d$ if $b - c - d$ was visited first.

Sampling errors or hidden variables can also lead to invalid CPDAGs, meaning that there does not exist a DAG that has the same skeleton and v-structures as the graph found by the algorithm. An example of this is an undirected cycle consisting of the edges $a - b - c - d$ and $d - a$. In this case it is impossible to direct the edges without creating a cycle or a new v-structure. The optional argument `u2pd` specifies what should be done in such a situation. If it is set to `"relaxed"`, the algorithm simply outputs the invalid CPDAG. If `u2pd` is set to `"rand"`, all direction information is discarded and a random DAG is generated on the skeleton. The corresponding CPDAG is then returned. If `u2pd` is set to `"retry"`, up to 100 combinations of possible directions of the ambiguous edges are tried, and the first combination that results in a valid CPDAG is chosen. If no valid combination is found, an arbitrary CPDAG is generated on the skeleton as with `u2pd = "rand"`.

By setting the argument `conservative = TRUE`, the conservative PC algorithm (see Ramsey, Zhang, and Spirtes (2006)) is chosen. The conservative PC is a slight variation of the PC algorithm and is intended to be more robust against sampling errors in its edge orientations. After the skeleton is computed, all unshielded triplets $a-b-c$ are checked in the following way. We test whether V_a and V_c are conditionally independent given any subset of the adjacency set of a or any subset of the adjacency set of c. If b is in no such conditioning set (and not in the original sepset) or in all such conditioning sets (and in the original sepset), the triple is marked as *faithful*. If, however, b is in some conditioning sets but not in others, or if there was no subset S of the adjacency set of a nor of c such that V_a and V_c are conditionally independent given V_S, the triple is marked as *"unfaithful"*. Only faithful triples are oriented as v-structures. Furthermore, orientation rules that need to know whether $a - b - c$ is a v-structure or not are only applied to faithful triples. (For more details, see the help file of the internal function `pc.cons.intern()` which is called with the argument `version.unf = c(2,2)`.)

As with the skeleton, the PC algorithm works with the column positions of the variables in the adjacency matrix, and not with the names of the variables. When plotting the object, undirected and bidirected edges are equivalent.

As an example, we estimate a CPDAG of the Gaussian data used in the example for the skeleton in Section 3.1. Again, we choose the predefined `gaussCItest()` as conditional independence test and create the corresponding test statistic. Finally, we plot the result. The estimated CPDAG and the true underlying DAG are shown in Fig. 4.

3.3. fci

The FCI algorithm is a generalization of the PC algorithm, in the sense that it allows arbitrarily many latent and selection variables. Under the assumption that the data are faithful to a DAG that includes all latent and selection variables, the FCI algorithm estimates the equivalence class of MAGs that describe the conditional independence relationships between the observed variables given the selection variables.

The first part of the FCI algorithm is analogous to the PC algorithm. It starts with a complete undirected graph and estimates an initial skeleton using the function `skeleton()`. All edges of this skeleton are of the form o–o. Now, the v-structures are oriented as in the

```
R> suffStat <- list(C = cor(gmG$x), n = nrow(gmG$x))
R> pc.fit <- pc(suffStat, indepTest=gaussCItest, p = ncol(gmG$x), alpha = 0.01)
R> par(mfrow = c(1,2))
R> plot(gmG$g, main = "")
R> plot(pc.fit, main = "")
```

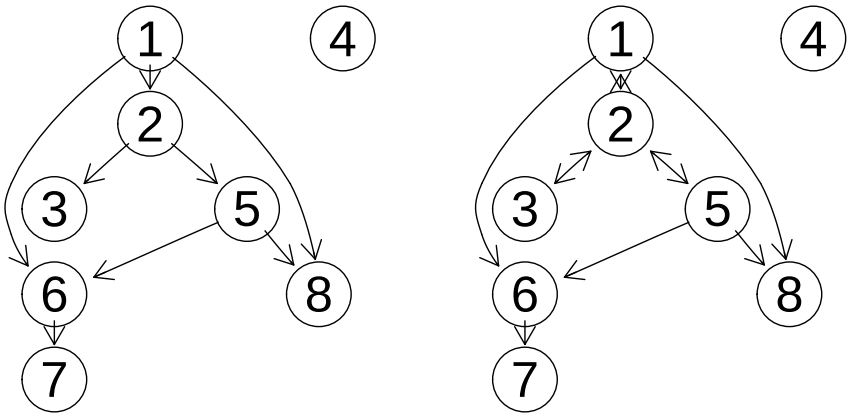

Figure 4: True underlying DAG (left) and estimated CPDAG (right) fitted on the simulated Gaussian data set `gmG`.

PC algorithm. However, due to the presence of hidden variables, it is no longer sufficient to consider only subsets of the adjacency sets of nodes x and y to decide whether the edge $x - y$ should be removed. Therefore, the initial skeleton may contain some superfluous edges. These edges are removed in the next step of the algorithm. To decide whether edge x o–o y should be removed, one computes Possible-D-SEP(x, y) and Possible-D-SEP(y, x) and performs conditional independence tests of V_x and V_y given all subsets of Possible-D-SEP(x, y) and of Possible-D-SEP(y, x) (see helpfile of function `pdsep()`). Subsequently, all edges are transformed into o–o again and the v-structures are newly determined (using information in sepset). Finally, as many undetermined edge marks (o) as possible are determined using (a subset of) the 10 orientation rules given by Zhang (2008).

The function can be called with the following arguments:

```
fci(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
    fixedEdges = NULL, NAdelete = TRUE, m.max = Inf, rules = rep(TRUE,
        10), doPdsep = TRUE, conservative = c(FALSE, FALSE),
    biCC = FALSE, cons.rules = FALSE, labels = NA)
```

where the arguments `suffStat`, `indepTest`, `p`, `alpha`, `fixedGaps`, `fixedEdges`, `NAdelete` and `m.max` are identical to those in `skeleton()`. The option `rules` contains a logical vector of length 10 indicating which rules should be used when directing edges, where the order of the rules is taken from Zhang (2008).

The option `doPdsep` indicates whether Possible-D-SEP should be computed for all nodes, and all subsets of Possible-D-SEP are considered as conditioning sets in the conditional independence tests. If FALSE, Possible-D-SEP is not computed, so that the algorithm simplifies to

the Modified PC algorithm of Spirtes *et al.* (2000).

The argument `conservative` can be used to invoke the conservative FCI algorithm which consists of two parts. In the first part (done if `conservative[1]` is `TRUE`), we call the internal function `pc.cons.internal()` with argument `version.unf = c(1,2)` after computing the skeleton. This is a slight variation of the conservative PC algorithm (which used `version.unf = c(2,2)`): If V_a is independent of V_c given some V_S in the skeleton (i.e., , the edge $a - c$ dropped out), but V_a and V_c remain dependent given all subsets of the adjacency set of either a or c, we call $a-b-c$ *"unambiguous"*. This is because in the FCI algorithm, the true separating set might be outside the adjacency sets of a or c. The purpose of this conservative version is to speed up the algorithm in the sample version (less v-structures lead to smaller Possible-D-SEP sets). In the second part (done if `conservative[2]` is `TRUE`), we call `pc.cons.internal(..., version.unf = c(1,2))` again after Possible-D-SEP was found and the graph potentially lost some edges. Therefore, new triples might have occurred. If this second part is done, the resulting information on sepset and faithful triples overwrites the previous and will be used for the subsequent orientation rules. The purpose of this conservative version is used in the hope to obtain better edge orientations. See Colombo *et al.* (2012) for more details.

By setting the argument `biCC = TRUE`, Possible-D-SEP(a, c) is defined as the intersection of the original Possible-D-SEP(a, c) and the set of nodes that lie on a path between a and c. This method uses biconnected components to find all nodes on a path between nodes a and c. The smaller Possible-D-SEP sets lead to faster computing times, while Colombo *et al.* (2012) showed that the algorithm is identical to the original FCI algorithm given oracle information on the conditional independence relationships.

The argument `cons.rules` manages the way in which the information about unfaithful triples affects the orientation rules for directing edges. If `cons.rules = TRUE`, an orientation rule that needs information on definite non-colliders is only applied, if the corresponding subgraph relevant for the rule does not involve an unfaithful triple.

Using the argument `labels`, one can pass names for the vertices of the estimated graph. By default, this argument is set to `NA`, in which case the node names `as.character(1:p)` are used.

As an example, we estimate the PAG of a graph with five nodes using the function `fci()` and the predefined function `gaussCItest()` as conditional independence test. In Fig. 5 the true DAG and the PAG estimated with `fci()` are shown. Random variable V_1 is latent. We can read off that both V_4 and V_5 cannot be a cause of V_2 and V_3, which can be confirmed in the true DAG.

```
R> data("gmL")
R> suffStat1 <- list(C = cor(gmL$x), n = nrow(gmL$x))
R> pag.est <- fci(suffStat1, indepTest = gaussCItest,
                  p = ncol(gmL$x), alpha = 0.01, labels = as.character(2:5))
R> par(mfrow = c(1,2))
R> plot(gmL$g, main = "")
R> plot(pag.est)
```

3.4. rfci

The function `rfci()` is rather similar to `fci()`. However, it does not compute any Possible-

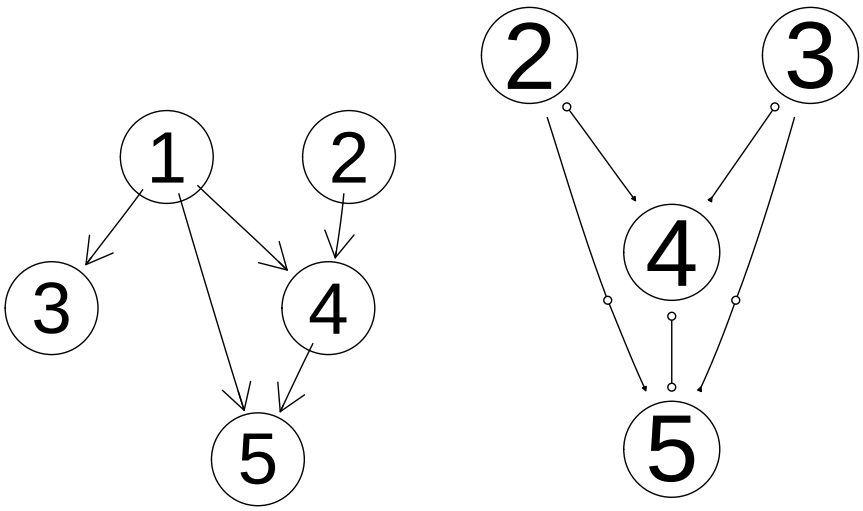

Figure 5: True underlying DAG (left) and estimated PAG (right), when applying the FCI and RFCI algorithms to the data set `gmL`. The output of FCI and RFCI is identical. Variable V_1 of the true underlying DAG is latent.

D-SEP sets and thus does not make tests conditioning on them. This makes `rfci()` much faster than `fci()`. The orientation rule for v-structures and the orientation rule for so-called discriminating paths (rule 4) were modified in order to produce a PAG which, in the oracle version, is guaranteed to have correct ancestral relationships.

The function can be called in the following way:

```
rfci(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
    fixedEdges = NULL, NAdelete = TRUE, m.max = Inf)
```

where the arguments `suffStat`, `indepTest`, `p`, `alpha`, `fixedGaps`, `fixedEdges`, `NAdelete` and `m.max` are identical to those in `skeleton()`.

As an example, we re-run the example from Section 3.3 and show the PAG estimated with `rfci()` in Figure 5. The PAG estimated with `fci()` and the PAG estimated with `rfci()` are the same.

```
R> data("gmL")
R> suffStat1 <- list(C = cor(gmL$x), n = nrow(gmL$x))
R> pag.est <- rfci(suffStat1, indepTest = gaussCItest,
                   p = ncol(gmL$x), alpha = 0.01, labels = as.character(2:5))
```

A note on implementation: As `pc()`, `fci()` and `rfci()` are similar in the result they produce, their resulting values are of (S4) class `pcAlgo` and `fciAlgo` (for both `fci()` and `rfci()`), respectively. Both classes extend the class (of their "communalities") `gAlgo`.

3.5. ida

To illustrate the function `ida()`, consider the following example. We simulated 10000 samples from seven multivariate Gaussian random variables with a causal structure given on the left of Fig. 6. We assume that the causal structure is unknown and want to infer the causal effect

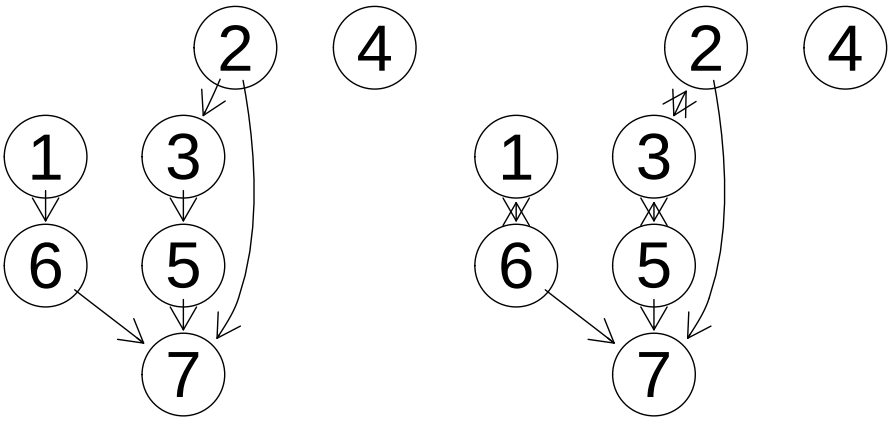

Figure 6: True DAG (left) and estimated CPDAG (right).

of V_2 on V_5. First, we estimate the equivalence class of DAGs that describe the conditional independence relationships in the data, using the function `pc()` (see Section 3.2).

```
R> data("gmI")
R> suffStat <- list(C = cor(gmI$x), n = nrow(gmI$x))
R> pc.fit <- pc(suffStat, indepTest=gaussCItest,
              p = ncol(gmI$x), alpha = 0.01)
```

Comparing the true DAG with the CPDAG in Fig. 6, we see that the CPDAG and the true DAG have the same skeleton. Moreover, the directed edges in the CPDAG are also directed in that way in the true DAG. Three edges in the CPDAG are bidirected. Recall that undirected and bidirected edges bear the same meaning in a CPDAG, so they can be used interchangeably.

Since there are three undirected edges in the CPDAG, there might be up to $2^3 = 8$ DAGs in the corresponding equivalence class. However, the undirected edges $2 - 3 - 5$ can be oriented as a new v-structure. As mentioned in Section 2.1, DAGs in the equivalence class must have exactly the same v-structures as the corresponding CPDAG. Thus, $2 - 3 - 5$ can only be oriented as $2 \to 3 \to 5$, $2 \leftarrow 3 \leftarrow 5$ or $2 \leftarrow 3 \to 5$, and not as $2 \to 3 \leftarrow 5$. There is only one remaining undirected edge $(1 - 6)$, which can be oriented in two ways. Thus, there are six valid DAGs (i.e., they have no new v-structures and no directed cycles) and these form the equivalence class represented by the CPDAG. In Fig. 7, all DAGs in the equivalence class are shown. DAG 6 is the true DAG.

For each DAG G in the equivalence class, we apply Pearl's do-calculus to estimate the total causal effect of V_x on V_y. Since we assume Gaussianity, this can be done via a simple linear regression: If y is not a parent of x, we take the regression coefficient of V_x in the regression `lm(Vy ~ Vx + pa(Vx))`, where `pa(Vx)` denotes the parents of x in the DAG G (z is called a parent of x if G if G contains the edge $z \to x$); if y is a parent of x in G, we set the estimated causal effect to zero.

If the equivalence class contains k DAGs, this yields k estimated total causal effects. Since

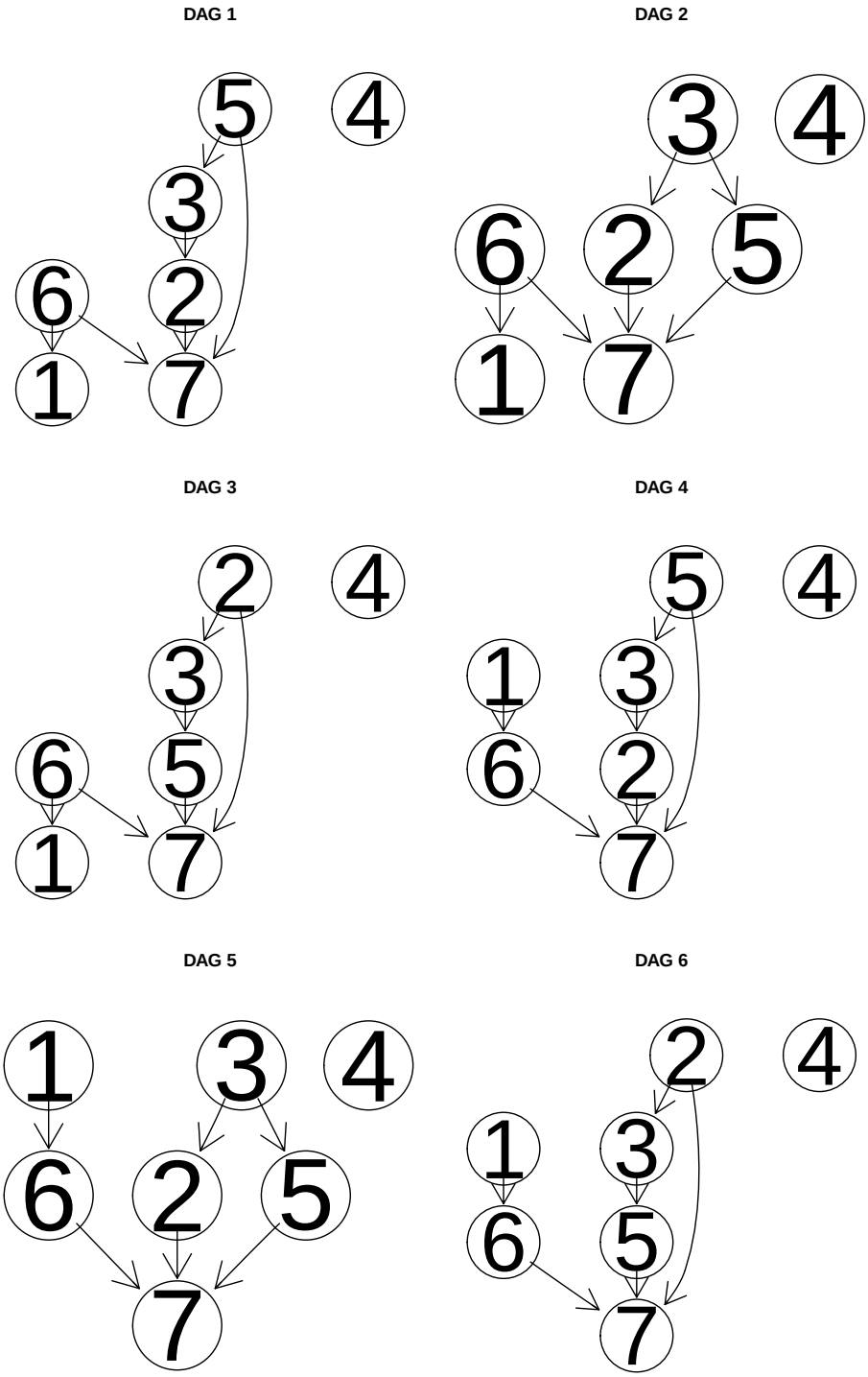

Figure 7: All DAGs belonging to the same equivalence class as the true DAG shown in Fig. 6.

we do not know which DAG is the true causal DAG, we do not know which estimated total causal effect of V_x on V_y is the correct one. Therefore, we return the entire multiset of k estimated effects.

In our example, there are six DAGs in the equivalence class. Therefore, the function `ida()` (with `method = "global"`) produces six possible values of causal effects, one for each DAG.

```
R> ida(2, 5, cov(gmI$x), pc.fit@graph, method = "global", verbose = FALSE)

[1] -0.0049012 -0.0049012  0.5421360 -0.0049012 -0.0049012  0.5421360
```

Among these six values, there are only two unique values: -0.0049 and 0.5421. This is because we compute `lm(V5 ~ V2 + pa(V2))` for each DAG and report the regression coefficient of V_2. Note that there are only two possible parent sets of node 2 in all six DAGs: In DAGs 3 and 6, there are no parents of node 2. In DAGs 1, 2, 4 and 5, however, the parent of node 2 is node 3. Thus, exactly the same regressions are computed for DAGs 3 and 6, and the same regressions are computed for DAGs 1, 2, 4 and 5. Therefore, we end up with two unique values, one of which occurs twice, while the other occurs four times.

Since the data was simulated from a model, we know that the true value of the causal effect of V_2 on V_5 is 0.5529. Thus, one of the two unique values is indeed very close to the true causal effect (the slight discrepancy is due to sampling error).

The function `ida()` can be called as

```
ida(x.pos, y.pos, mcov, graphEst, method = "local", y.notparent = FALSE,
    verbose = FALSE, all.dags = NA)
```

where `x.pos` denotes the column position of the cause variable, `y.pos` denotes the column position of the effect variable, `mcov` is the covariance matrix of the original data, and `graphEst` is a graph object describing the causal structure (this could be given by experts or estimated by `pc()`).

If `method="global"`, the method is carried out as described above, where all DAGs in the equivalence class of the estimated CPDAG are computed. This method is suitable for small graphs (say, up to 10 nodes). The DAGs can (but need not) be precomputed using the function `allDags()` and passed via argument `all.dags`.

If `method="local"`, we do not determine all DAGs in the equivalence class of the CPDAG. Instead, we only consider the local neighborhood of x in the CPDAG. In particular, we consider all possible directions of undirected edges that have x as endpoint, such that no new v-structure is created. For each such configuration, we estimate the total causal effect of V_x on V_y as above, using linear regression.

At first sight, it is not clear that such a local configuration corresponds to a DAG in the equivalence class of the CPDAG, since it may be impossible to direct the remaining undirected edges without creating a directed cycle or a v-structure. However, Maathuis *et al.* (2009) showed that there is at least one DAG in the equivalence class for each such local configuration. As a result, it follows that the multisets of total causal effects of the `global` and the `local` method have the same unique values. They may, however, have different multiplicities.

Recall, that in the example using the global method, we obtained two unique values with multiplicities two and four yielding six numbers in total. Applying the local method, we obtain the same unique values, but the multiplicities are 1 for both values.

```
R> ida(2,5, cov(gmI$x), pc.fit@graph, method = "local")
```

```
[1]  0.5421360 -0.0049012
```

One can take summary measures of the multiset. For example, the minimum absolute value provides a lower bound on the size of the true causal effect: If the minimum absolute value of all values in the multiset is larger than one, then we know that the size of the true causal effect (up to sampling error) must be larger than one. The fact that the unique values of the multisets of the `global` and `local` method are identical implies that summary measures of the multiset that only depend on the unique values (such as the minimum absolute value) can be found using the local method.

In some applications, it is clear that some variable is definitivly a cause of other variables. Consider for example a bacterium producing a certain substance, taking the amount of produced substance as response variable. Knocking out genes in the bacterium might change the ability to produce the substance. By measuring the expression levels of genes, we want to know which genes have a causal effect on the product. In this setting, it is clear that the amount of substance is the effect and the activity of the genes is the cause. Thus in the causal structure, the response variable cannot be a parent node of any variable describing the expression level of genes. This background knowledge can be easily incorporated: By setting the option `y.notparent = TRUE`, all edges in the CPDAG that have the response variable as endpoint (no matter whether directed or undirected) are overwritten so that they are oriented towards the response variable.

3.6. idaFast

In some applications it is desirable to estimate the causal effect of one variable on a set of response variables. In these situations, the function `idaFast()` should be used. Imagine for example, that we have data on several variables, that we have no background knowledge about the causal effects among the variables and that we want to estimate the causal effect of each variable onto each other variable. To this end, we could consider for each variable the problem: What is the causal effect of this variable on all other variables. Of course, one could solve the problem by using `ida()` on each pair of variables. However, there is a more efficient way which uses the fact that a linear regression of a fixed set of explanatory variables on several different response variables can be computed very efficiently.

The function `idaFast()` can be called with the following arguments

```
idaFast(x.pos, y.pos.set, mcov, graphEst)
```

The arguments `x.pos`, `mcov`, `graphEst` have the same meaning as the corresponding arguments in `ida()`. The argument `y.pos.set` is a vector containing the column positions of all response variables of interest.

This call performs `ida(x.pos, y.pos, mcov, graphEst, method="local", y.notparent=FALSE, verbose=FALSE)` for all values of `y.pos` in `y.pos.set` at the same time and in an efficient way. Note that the option `y.notparent = TRUE` is not implemented.

Consider the example from Section 3.5, where we computed the causal effect of V_2 on V_5. Now, we want to compute the effect of V_2 on V_5, V_6 and V_7 using `idaFast()` and compare the results with the output of `ida()`. As expected, both methods lead to the same results.

```
R> data("gmI")
R> suffStat <- list(C = cor(gmI$x), n = nrow(gmI$x))
R> pc.fit <- pc(suffStat, indepTest=gaussCItest, p=ncol(gmI$x), alpha = 0.01)
```

```
R> (eff.est1 <- ida(2,5, cov(gmI$x), pc.fit@graph, method="local"))
[1]  0.5421360 -0.0049012
R> (eff.est2 <- ida(2,6, cov(gmI$x), pc.fit@graph, method="local"))
[1] -0.0058532 -0.0062310
R> (eff.est3 <- ida(2,7, cov(gmI$x), pc.fit@graph, method="local"))
[1] 1.00861 0.89708
R> (eff.estF <- idaFast(2, c(5,6,7), cov(gmI$x), pc.fit@graph))
        [,1]       [,2]
5  0.5421360 -0.0049012
6 -0.0058532 -0.0062310
7  1.0086138  0.8970766
```

3.7. Using a user specific conditional independence test

In some cases it might be desirable to use a user specific conditional independence test instead of the provided ones. The **pcalg** package allows the use of any conditional independence test defined by the user. In this section, we illustrate this feature by using a conditional independence test for Gaussian data that is not predefined in the package.

The functions `skeleton()`, `pc()` and `fci()` all need the argument `indepTest`, a function of the form `indepTest(x, y, S, suffStat)` to test conditional independence relationships. This function must return the p-value of the conditional independence test of V_x and V_y given V_S and some information on the data in the form of a sufficient statistic (this might be simply the data frame with the original data), where x, y, S indicate column positions of the original data matrix. We will show an example that illustrates how to construct such a function.

A simple way to compute the partial correlation of V_x and V_y given V_S for some data is to solve the two associated linear regression problems $\texttt{lm}(Vx \sim VS)$ and $\texttt{lm}(Vy \sim VS)$, get the residuals, and calculate the correlation between the residuals. Finally, a correlation test between the residuals yields a p-value that can be returned. The argument `suffStat` is an arbitrary object containing several pieces of information that are all used within the function to produce the p-value. In the predefined function `gaussCItest()` for example, a list containing the correlation matrix and the number of observations is passed. This has the advantage that any favorite (e.g., robust) method of computing the correlation matrix can be used before partial correlations are computed. Oftentimes, however, it suffices to just pass the complete data set in `suffStat`. We choose this simple method in our example. An implementation of the function `myCItest()` could look like this.

```
R> myCItest <- function(x,y,S, suffStat) {
   if (length(S) == 0) {
     x. <- suffStat[,x]
     y. <- suffStat[,y]
   } else {
     rxy <- resid(lm.fit(y= suffStat[,c(x,y)], x= cbind(1, suffStat[,S])))
     x. <- rxy[,1];  y. <- rxy[,2]
   }
```

```
   cor.test(x., y.)$p.value
 }
```

We can now use this function together with `pc()`.

```
R> pc.myfit <- pc(suffStat = gmG$x, indepTest = myCItest,
                  p = 8, alpha = 0.01)
R> par(mfrow = c(1,2))
R> plot(pc.fit, main = "")
R> plot(pc.myfit, main = "")
```

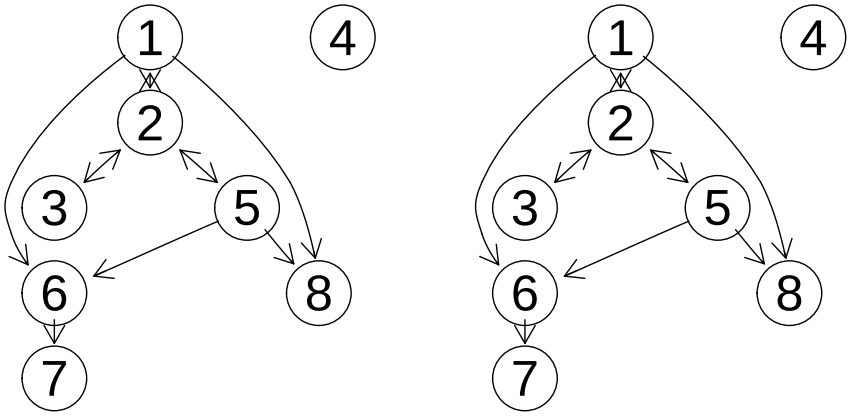

Figure 8: The estimated CPDAGs using the predefined conditional independence test `gaussCItest()` (left) and the user specified conditional independence test `myCItest()` (right) are identical for the `gmG` data.

As expected, the resulting CPDAG (see Fig. 8) is the same as in Section 3.2 where we used the function `gaussCItest()` as conditional independence test. Note however that using `gaussCItest()` is considerably faster than using `myCItest()` (on our computer 0.059 seconds using `gaussCItest()` versus 1.05 seconds using `myCItest()`).

4. Applications

The **pcalg** package has been used for applications in epidemiology (see Kalisch, Fellinghauer, Grill, Maathuis, Mansmann, Bühlmann, and Stucki (2010)), biology (see Maathuis *et al.* (2010) and Nagarajan, Datta, Scutari, Beggs, Nolen, and Peterson (2010)) and the social sciences (see Danenberg and Marsella (2010)). We will discuss two applications in more detail below.

4.1. Graphical Models and Causal Effects in Human Functioning

In Kalisch *et al.* (2010), the development of WHO's International Classification of Functioning, Disability and Health (ICF) on the one hand and recent developments in graphical modeling on the other hand were combined to deepen the understanding of human functioning. The

objective of the paper was to explore how graphical models can be used in the study of ICF data. It was found that graphical models could be used successfully for visualization of the dependence structure of the data set, dimension reduction, and the comparison of subpopulations. Moreover, estimations of bounds on causal effects using the IDA method yielded plausible results. All analyses were done with the **pcalg** package.

4.2. Causal effects among genes

In Maathuis *et al.* (2010), the authors aim at quantifying the effects of single gene interventions on the expression of other genes in yeast, allowing for better insights into causal relations between genes. With $n = 63$ samples of observational data measuring the expression of $p = 5361$ genes (see Hughes, Marton, Jones, Roberts, Stoughton, Armour, Bennett, Coffey, Dai, He, Kidd, King, Meyer, Slade, Lum, Stepaniants, Shoemaker, Gachotte, Chakraburtty, Simon, Bard, and Friend (2000)), the goal was to identify the largest intervention effects between all pairs of genes. For the analysis, the **pcalg** package was used.

Hughes *et al.* (2000) also provide gene expression measurements from 234 interventional experiments, namely from 234 single-gene deletion mutant strains. Using this data, we know the true causal effect of the knock-out genes on the remaining genes in good approximation. We can then quantify how well we can find the true intervention effects in the following way: We encode the largest 10% of the intervention effects computed from the interventional data as the target set of effects that we want to identify. We then check in an ROC curve, how well the ranking of the causal effects estimated by applying `ida()` to the observational data is able to identify effects in the target set. For comparison, the authors also used the (conceptually wrong) Lasso and Elastic Net to obtain rankings. In Figure 9 one can see that `ida()` is clearly superior to the alternative methods (and random guessing) in terms of identifying effects in the target set.

We note that the yeast data set is very high-dimensional ($n = 63$, $p = 5361$). Thus, unlike the toy examples used to illustrate the package in this manuscript, where n was much bigger than p and the causal structure was recovered exactly up to its equivalence class, the estimated causal structure for the yeast data is likely to contain many sampling errors. However, Figure 9 shows that it is still possible to extract useful information about causal effects.

5. Discussion

Causal structure learning and the estimation of causal effects from observational data has large potential. However, we emphasize that we do not propose causal inference methods based on observational data as a replacement for experiments. Rather, IDA should be used as a guide for prioritizing experiments, especially in situations where no clear preferences based on the context can be given.

Since many assumptions of the proposed methods are uncheckable, it is important to further validate the methods in a range of applications. We hope that the **pcalg** package contributes to this important issue by providing well-documented and easy to use software.

6. Session information

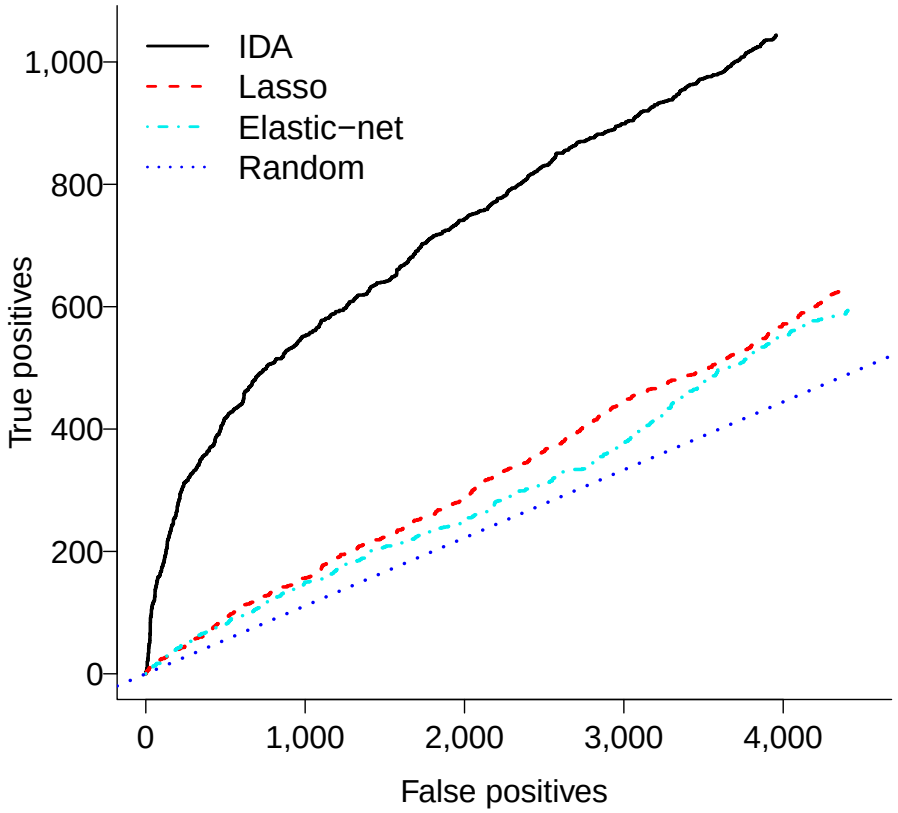

Figure 9: The largest 10% of the causal effects found in experiments among yeast genes are identified much better from observational data with IDA than with Lasso, Elastic Net or random guessing. The figure is essentially taken from Maathuis *et al.* (2010).

```
R> toLatex(sessionInfo())
```

- R version 2.14.1 (2011-12-22), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=de_CH.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=de_CH.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=de_CH.UTF-8`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=de_CH.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: abind 1.4-0, corpcor 1.6.2, fortunes 1.4-2, graph 1.30.0, pcalg 1.1-4, Rgraphviz 1.31.2, sfsmisc 1.0-19
- Loaded via a namespace (and not attached): ggm 1.99-2, RBGL 1.28.0, robustbase 0.7-8, tools 2.14.1

References

Bottcher SG, Dethlefsen C (2011). ***deal**: Learning Bayesian Networks with Mixed Variables*. R package version 1.2-34, URL http://CRAN.R-project.org/package=deal.

Colombo D, Maathuis MH, Kalisch M, Richardson TS (2012). "Learning High-Dimensional DAGs with Latent and Selection Variables." *Annals of Statistics, to appear*.

Danenberg P, Marsella S (2010). "Data-Driven Coherence Models." In *Proceedings of the 19th Conference on Behavior Representation in Modeling and Simulation*.

Hojsgaard S (2012). ***gRain**: Graphical Independence Networks*. R package version 0.8-12, URL http://CRAN.R-project.org/package=gRain.

Hojsgaard S, Dethlefsen C, Bowsher C (2012). ***gRbase**: A package for graphical modelling in R*. R package version 1.4.4, URL http://CRAN.R-project.org/package=gRbase.

Hojsgaard S, Lauritzen SL (2011). ***gRc**: Inference in Graphical Gaussian Models with Edge and Vertex Symmetries*. R package version 0.3.0, URL http://CRAN.R-project.org/package=gRc.

Hughes T, Marton M, Jones A, Roberts C, Stoughton R, Armour C, Bennett H, Coffey E, Dai H, He Y, Kidd M, King A, Meyer M, Slade D, Lum P, Stepaniants S, Shoemaker D, Gachotte D, Chakraburtty K, Simon J, Bard M, Friend S (2000). "Functional Discovery via a Compendium of Expression Profiles." *Cell*, **102**, 109–126.

Kalisch M, Bühlmann P (2007). "Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm." *Journal of Machine Learning Research*, **8**, 613–636.

Kalisch M, Fellinghauer B, Grill E, Maathuis MH, Mansmann U, Bühlmann P, Stucki G (2010). "Understanding Human Functioning Using Graphical Models." *BMC Medical Research Methodology*, **10:14**.

Lauritzen S (1996). *Graphical Models*. Oxford University Press.

Maathuis MH, Colombo D, Kalisch M, Bühlmann P (2010). "Predicting Causal Effects in Large-Scale Systems from Observational Data." *Nature Methods*, **7**, 261–278.

Maathuis MH, Kalisch M, Bühlmann P (2009). "Estimating High-Dimensional Intervention Effects from Observational Data." *Annals of Statistics*, **37**, 3133–3164.

Meek C (1995). "Strong Completeness and Faithfulness in Bayesian Networks." In *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 411–418. Morgan Kaufmann.

Nagarajan R, Datta S, Scutari M, Beggs ML, Nolen GT, Peterson CA (2010). "Functional Relationships between Genes Associated with Differentiation Potential of Aged Myogenic Progenitors." *Frontiers in Physiology*, **1**.

Pearl J (2000). *Causality*. Cambridge University Press.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Ramsey J, Zhang J, Spirtes P (2006). "Adjacency-Faithfulness and Conservative Causal Inference." In *Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pp. 401–408. AUAI Press, Arlington, Virginia.

Richardson TS, Spirtes P (2002). "Ancestral Graph Markov Models." *Annals of Statistics*, **30**, 962–1030.

Scutari M (2010). "Learning Bayesian Networks with the bnlearn R Package." *Journal of Statistical Software*, **35**(3), 1–22. URL http://www.jstatsoft.org/v35/i03/.

Spirtes P, Glymour C, Scheines R (2000). *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning, second edition. MIT Press, Cambridge.

Spirtes P, Meek C, Richardson T (1999a). *Computation, Causation and Discovery*, chapter An Algorithm for Causal Inference in the Presence of Latent Variables and Selection Bias, pp. 211–252. MIT Press.

Spirtes P, Meek C, Richardson TS (1999b). *An Algorithm for Causal Inference in the Presence of Latent Variables and Selection Bias*, pp. 211–252. MIT Press.

Verma T, Pearl J (1991). "Equivalence and Synthesis of Causal Models." In *UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 255–270. Elsevier Science Inc., New York, NY, USA.

Zhang J (2008). "On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias." *Artificial Intelligence*, **172**, 1873–1896.

Affiliation:

Markus Kalisch
Seminar für Statistik
ETH Zürich
8092 Zürich, Switzerland
E-mail: kalisch@stat.math.ethz.ch

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd