More Causal Inference with Graphical Models in R Package pcalg

Markus Kalisch
ETH Zurich

Martin Mächler
ETH Zurich

Diego Colombo
ETH Zurich

Alain Hauser
University of Bern

Marloes H. Maathuis
ETH Zurich

Peter Bühlmann
ETH Zurich

Abstract

The **pcalg** package for R (R Development Core Team 2014) can be used for the following two purposes: Causal structure learning and estimation of causal effects from observational and/or interventional data. In this document, we give a brief overview of the methodology, and demonstrate the package's functionality in both toy examples and applications.

This vignette is an updated and extended (FCI, RFCI, etc) version of Kalisch *et al.* (2012) which was for **pcalg** 1.1-4.

Keywords: IDA, PC, RFCI, FCI, GES, GIES, 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 algorithms PC (see Spirtes *et al.* 2000), FCI (see Spirtes *et al.* 2000, 1999a), RFCI (see Colombo *et al.* 2012) and GES (see Chickering 2002) can infer information about the causal structure from observational data; there also exists a generalization of GES to interventional data, GIES (see Hauser and Bühlmann 2012). 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 (Maathuis *et al.* 2009), which can infer bounds on causal effects based on observational data under some assumptions and in particular that no hidden and selection variables are present. IDA is a two step approach that combines the PC algorithm and Pearl's backdoor criterion (Pearl 1993), which has been designed for DAGs without latent variables. IDA was validated on a large-scale biological system (see Maathuis *et al.* 2010). Since the assumption of no latent variables is a strong assumption when working with real data and therefore often violated in practice, we generalized Pearl's backdoor criterion (see Maathuis and Colombo 2013) to more general types of graphs, i.e. CPDAGs, MAGs, and PAGs, that describe Markov equivalence classes of DAGs with and without latent variables but without

selection variables.

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 algorithms PC, FCI, RFCI, GES and GIES, as well as of the IDA method and the generalized Pearl's backdoor criterion. 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 analyze. First, we load the package **pcalg** and the data set.

```
> library("pcalg")
> 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.

```
> suffStat <- list(C = cor(gmG8$x), n = nrow(gmG8$x))
> pc.gmG <- pc(suffStat, indepTest = gaussCItest,
               p = ncol(gmG8$x), alpha = 0.01)
```

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 $\longleftarrow$ or $\longrightarrow$. 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 $\tilde{x}+1$. The recorded average change in variable V_6 is the (total) causal effect of V_1 on V_6. More

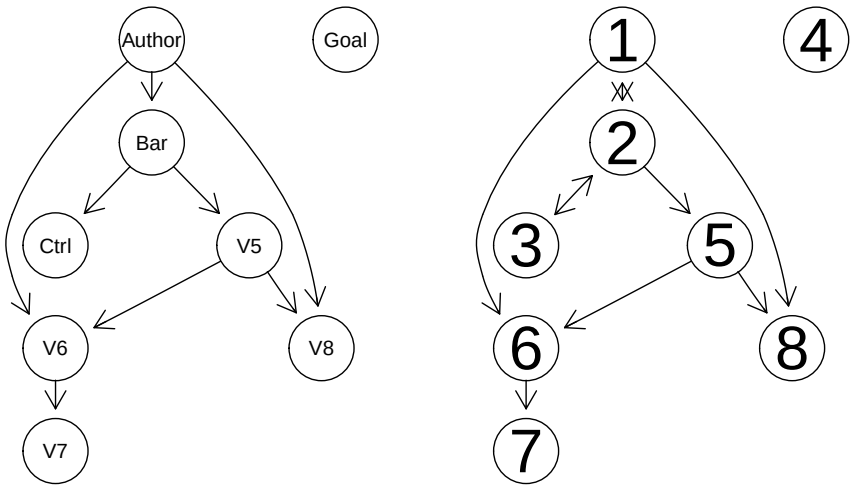

Figure 1: True underlying causal DAG (left) and estimated causal structure (right), representing a Markov equivalence class of DAGs that all encode the same conditional independence information. (Due to the large sample size, there were no sampling errors.)

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

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

where $\operatorname{do}(V_6 = x)$ 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.

```
> ida(1, 6, cov(gmG8$x), pc.gmG@graph)

[1] 0.5321 0.5304
```

Since we simulated the data, we know that the true value of the causal effect is 0.52. 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.

```
> idaFast(1, c(4,5,6), cov(gmG8$x), pc.gmG@graph)

       [,1]       [,2]
4 -0.010329 -0.0098746
```

```
5  0.043770 -0.0056205
6  0.532096  0.5303967
```

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, 0.05, 0.52 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. In particular, we discuss algorithms for structure learning

- in the absence of hidden variables from observational data such as PC (see Spirtes *et al.* 2000), GES (see Chickering 2002), and the dynamic programming approach of Silander and Myllymäki (2006),
- from observational data accounting for hidden variables such as FCI (see Spirtes *et al.* 2000, 1999a) and RFCI (see Colombo *et al.* 2012),
- in the absence of hidden variables from jointly observational and interventional data such as GIES (see Hauser and Bühlmann 2012).

In Section 2.2 we first describe the IDA method (see Maathuis *et al.* 2009) to obtain bounds on causal effects from observational data when no latent and selection variables are present. This method is based on first estimating the causal structure and then applying do-calculus (see Pearl 2000). We then propose the generalized Pearl's backdoor criterion (see Maathuis and Colombo 2013) that works with DAGs, CPDAGs, MAGs, and PAGs as input and it assumes that there are arbitrarly many latent variables but no selection variables. This method is based on two steps: first it checks if the total causal effect of one variable X onto another variables Y is identifiable via the generalized backdoor criterion in the given type of graph, and if this is the case it explicitly gives a set of variables that satisfies the generalized backdoor criterion with respect to X and Y in the given graph.

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 ($\longleftarrow$ or $\longrightarrow$). 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 whose list of conditional independencies perfectly matches the list of d-separation relations of some DAG; 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 $i \longrightarrow j \longleftarrow k$ 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 1990). 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 (Spirtes *et al.* 2000), or essential graph (Andersson *et al.* 1997).

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

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: $\longrightarrow$, $---$
(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.

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 $i\longrightarrow k\longleftarrow j$ 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.)

It is known that the PC algorithm is order-dependent in steps **(P2)**–**(P4)**, meaning that the output depends from the order in which the variables are given. Colombo and Maathuis (2013) proposed several modifications of the PC algorithm (see Sections 3.1 and 3.2) that partly or fully remove these order-dependence issues in each step.

The PC algorithm presented so far is based on conditional independence tests. Score-based methods form an alternative approach to causal inference. They try to find a CPDAG that maximizes a *score*, typically a model selection criterion, which is calculated from data.

One of the most popular scores is the Bayesian information criterion (BIC) because its maximization leads to *consistent* model selection in the classical large-sample limit (Haughton 1988; Geiger *et al.* 2001). However, computing its maximum is an NP-hard problem (Chick-

ering 1996). An exhaustive search is computationally infeasible due to the size of the search space, the space of DAGs or CPDAGs, respectively. Silander and Myllymäki (2006) have presented an exact dynamic programming algorithm with an exponential time complexity. Its execution is feasible for models with a few dozen variables.

The greedy equivalence search (GES) of Chickering (2002) makes the maximization of the BIC computationally feasible for much larger graphs. As the name of the algorithm implies, GES maximizes the BIC in a *greedy* way, but still guarantees consistency in the large-sample limit. It still has exponential-time complexity in the worst case, but only polynomial complexity in the average case where the size of the largest clique in a graph grows only logarithmically with the number of nodes (Grimmett and McDiarmid 1975).

GES greedily optimizes the BIC in two phases:

- In the *forward phase*, the algorithm starts with the empty graph. It then sequentially moves to larger CPDAGs by operations that correspond to adding single arrows in the space of DAGs. This phase is aborted if no augmentation of the BIC is possible any more.
- In the *backward phase*, the algorithm moves again into the direction of *smaller* graphs by operations that correspond to removing single arrows in the space of DAGs. The algorithm terminates as soon as no augmentation of the BIC is possible any more.

A key ingredient for the fast exploration of the search space in GES is an evaluation of the greedy steps in a local fashion which avoids enumerating all representative DAGs of an equivalence class and which exploits the decomposability of the BIC score (Chickering 2002).

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 (Theorem 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 ($\leftarrow\!\!\rightarrow$) (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: $\circ\!\!-\!\!\circ$, $\circ\!\!-\!\!-$, $\circ\!\!\rightarrow$, $-\!\!\rightarrow$, $\leftarrow\!\!\rightarrow$, $-\!\!-$. 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 $\circ$-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 *et al.* (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 3.4 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.

Since both these algorithms are build up from the PC algorithm, they are also order-dependent, meaning that the output depends from the order in which the variables are given. Starting from the solution proposed for the PC algorithm, Colombo and Maathuis (2013) proposed several modifications of the FCI and the RFCI algorithms (see Sections 3.1, 3.4, and 3.5) that partly or fully remove these order-dependence issues in each of their steps.

From a mixture of observational and interventional data

We often have to deal with interventional data in causal inference. In cell biology for example, data is often measured in different mutants, or collected from gene knockdown experiments, or simply measured under different experimental conditions. An intervention, denoted by Pearl's do-calculus (see Section 1), changes the joint probability distribution of the system; therefore, data samples collected from different intervention experiments are *not* identically distributed (although still independent).

The algorithms PC and GES both rely on the i.i.d. assumption and are not suited for causal inference from interventional data. The GIES algorithm, which stands for "greedy interventional equivalence search", is a generalization of GES to interventional data (see Hauser and Bühlmann 2012). It does not only make sure that interventional data points are handled correctly (instead of being wrongly treated as observational data points), but also accounts for the improved identifiablity of causal models under interventional data by returning an *interventional essential graph*. Just as in the observational case, an interventional essential graph is a partially directed graph representing an (interventional) Markov equivalence class of DAGs: a directed edge between two vertices stands for an arrow with common orientation among all representatives of the equivalence class, an undirected edge stands for an arrow that has different orientations among different representative DAGs; for more details, see Hauser and Bühlmann (2012).

GIES traverses the search space of interventional essential graphs in a similar way as GES traverses the search space of observational essential graphs. In addition, a new search phase

was introduced by Hauser and Bühlmann (2012) with movements which correspond to turning single arrows in the space of DAGs.

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 $P[V_y|\,\mathrm{do}(V_x = x)]$. 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 $P(V_x = 1|\,\mathrm{do}(V_y = 1)) = P(V_x = 1) = 0.1$. 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., $\frac{\partial}{\partial x}E[V_y|\,\mathrm{do}(V_x = x)]$, as a general measure for the causal effect of V_x on V_y. For multivariate Gaussian random variables, $E[V_y|\,\mathrm{do}(V_x = x)]$ 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 $E[V_y|\,\mathrm{do}(V_x = x + 1)] - E[V_y|\,\mathrm{do}(V_x = x)]$. For binary random variables (with domain $\{0, 1\}$) we define the causal effect of V_x on V_y as $E(V_y|\,\mathrm{do}(V_x = 1)) - E(V_y|\,\mathrm{do}(V_x = 0)) = P(V_y = 1|\,\mathrm{do}(V_x = 1)) - P(V_y = 1|\,\mathrm{do}(V_x = 0))$.

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 3.7 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 but no selection variables

If the causal DAG is known and no latent and selection variables are present, one can estimate causal effects from observational data using for example Pearl's backdoor criterion, as done in IDA.

However, in practice the assumption of no latent variables is often violated. Therefore, Maathuis and Colombo (2013) generalized Pearl's backdoor criterion to more general types of graphs that describe Markov equivalence classes of DAGs when allowing arbitrarily many latent but no selection variables. This generalization works with DAGs, CPDAGs, MAGs, and PAGs as input and it is based on a two step approach. In a first step, the causal effect of one variable X onto another variable Y under investigation is checked to be identifiable via the generalized backdoor criterion, meaning that there exists a set of variables W for which the generalized backdoor criterion is satisfied with respect to X and Y in the given graph. If the effect is indeed identifiable, in a second step the set W is explicitly given.

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.

GES algorithm: No hidden or selection variables; consistency in a standard asymptotic regime with a fixed number of variables (see Chickering 2002).

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.

GIES algorithm: No hidden or selection variables; mix of observational and interventional data. Interventional data alone is sufficient if there is no variable which is intervened in *all* data points.

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.

Generalized Backdoor Criterion: allows for arbitrarily many hidden but no selection variables. See Maathuis and Colombo (2013) for more details.

3. Package pcalg

This package has two goals. First, it is intended to provide fast, flexible and reliable implementations of the PC, FCI, RFCI, GES and GIES 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, and it also provides a genralization of Pearl's backdoor criterion to DAGs, CPDAGs, MAGs, nad PAGs, when hidden but no selection variables are allowed.

In the following, we describe the main functions of our package for achieving these goals. The functions `skeleton()`, `pc()`, `fci()`, `rfci()`, `ges()`, `gies()` and `simy()` are intended for estimating graphical models. The functions `ida()` and `idaFast()` are intended for estimating causal effects from observational data, and the function `backdoor()` is intended for checking if a causal effect is identifiable or not using the generalized backdoor criterion and if it is identifiable for estimating a set that actually satisfies the generalized backdoor criterion.

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

3.1. skeleton

The function `skeleton()` estimates the skeleton of a DAG without latent and selection variables using the PC algorithm (steps (P1) and (P2) in Algorithm 1), and it estimates an initial

```
> ## using  data("gmG", package="pcalg")
> suffStat <- list(C = cor(gmG8$x), n = nrow(gmG8$x))
> skel.gmG <- skeleton(suffStat, indepTest = gaussCItest,
                       p = ncol(gmG8$x), alpha = 0.01)
> par(mfrow = c(1,2))
> plot(gmG8$g, main = ""); plot(skel.gmG, main = "")
```

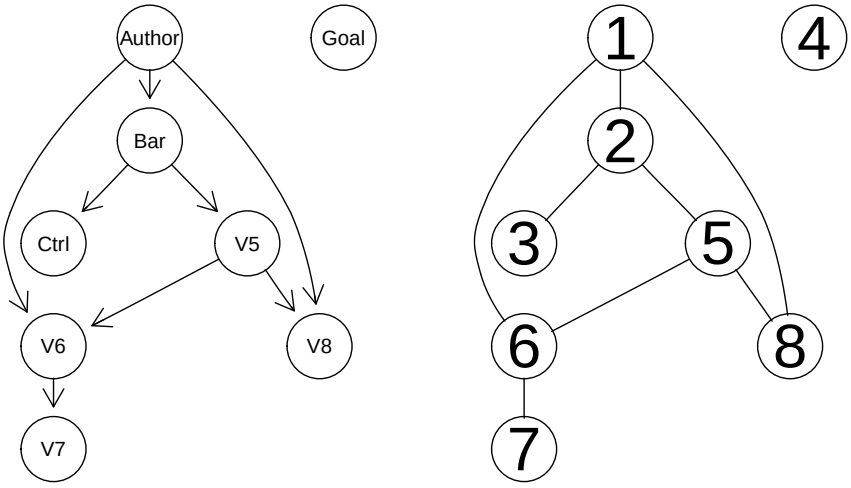

Figure 2: True underlying DAG (left) and estimated skeleton (right) fitted on the simulated Gaussian data set `gmG`.

skeleton of a DAG with arbitrarily many latent and selection variables using the FCI and the RFCI algorithms. The function can be called with the following arguments

```
skeleton(suffStat, indepTest, alpha, labels, p, method = c("stable", "original",
    "stable.fast"), m.max = Inf, fixedGaps = NULL, fixedEdges = NULL, NAdelete = TRUE,
    numCores = 1, verbose = FALSE)
```

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` 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

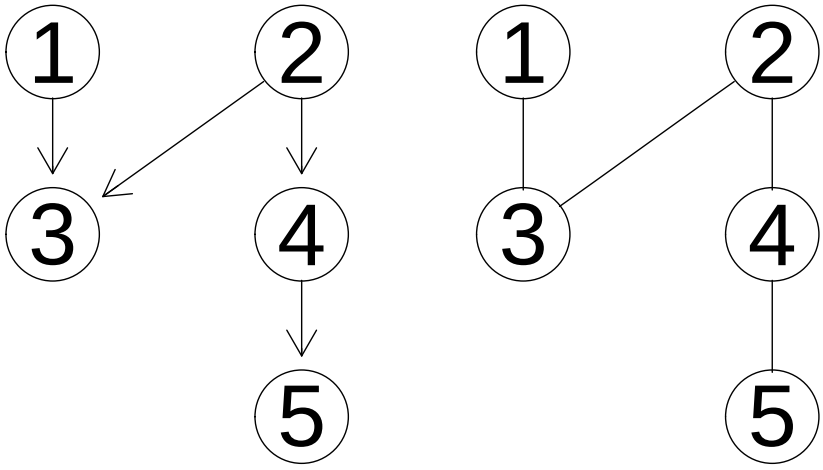

Figure 3: True underlying DAG (left) and estimated skeleton (right) fitted on the simulated discrete data set `gmD`.

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 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.

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

The PC algorithm is known to be order-dependent, in the sense that the output depends on the order in which the variables are given. Therefore, Colombo and Maathuis (2013) proposed a simple modification, called PC-stable, that yields order-independent adjacencies in the skeleton. In this function we implement their modified algorithm (the old order-dependent implementation can be found in version 1.1-5).

Since the FCI and RFCI algorithms are build up from the PC algorithm, they are also order-dependent in the skeleton. To resolve their order-dependence issues in the skeleton is more

involved, see Colombo and Maathuis (2013). However, this function estimates an initial order-independent skeleton in these algorithms (for additional details on how to make the final skeleton of FCI fully order-independent see 3.4 and Colombo and Maathuis (2013)).

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, alpha, labels, p, fixedGaps = NULL, fixedEdges = NULL,
    NAdelete = TRUE, m.max = Inf, u2pd = c("relaxed", "rand", "retry"),
    skel.method = c("stable", "original", "stable.fast"), conservative = FALSE,
    maj.rule = FALSE, solve.confl = FALSE, numCores = 1, verbose = FALSE)
```

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

The conservative PC algorithm (`conservative = TRUE`) is a slight variation of the PC algorithm (see Ramsey *et al.* 2006). 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 independent conditioning on any subset of the neighbors of a or any subset of the neighbors 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 *unambiguous*, no further action is taken and the usual PC is continued. If, however, b is in only some conditioning sets, 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 $a - b - c$ is marked as *ambiguous*. An ambiguous triple is not oriented as a v-structure. Furthermore, no later orientation rule that needs to know whether $a - b - c$ is a v-structure or not is applied. Instead of using the conservative version, which is quite strict towards the v-structures, Colombo and Maathuis (2013) introduced a less strict version for the v-structures called majority rule. This adaptation can be called using `maj.rule = TRUE`. In this case, the triple $a - b - c$ is marked as *ambiguous* if b is in exactly 50 percent of such conditioning sets, if it is in less than 50 percent it is set as a v-structure, and if in more than 50 percent as a non v-structure, for more details see Colombo and Maathuis (2013). The usage of both the conservative and the majority rule versions resolve the order-dependence issues of the determination of the v-structures, see Colombo and Maathuis (2013) for more details.

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 directed as v-structures. This gives conflicting information about the edge $b - c$, since it should be directed as $b \leftarrow c$ in v-structure $a \rightarrow b \leftarrow c$, while it should be directed as $b \rightarrow c$ in v-structure $b \rightarrow c \leftarrow d$. With the option `solve.confl = FALSE`, in such cases, we simply overwrite the directions of the conflicting edge. In the example above this means that we obtain $a \rightarrow b \rightarrow c \leftarrow d$ if $a - b - c$ was visited first, and $a \rightarrow b \leftarrow c \leftarrow d$ if $b - c - d$ was visited first, meaning that the final orientation on the edge depends on the ordering in which the edges are oriented. With the option `solve.confl = TRUE` (which is only supported with option `u2pd = "relaxed"`), we first generate a list of all (unambiguous) v-structures (in the example above $a - b - c$ and $b - c - d$), and then we simply orient them allow both

directions on the edge $b - c$, namely we allow the bi-directed edge $b \leftrightarrow c$ resolving the order-dependence issues on the edge orientations. We denote bi-directed edges in the adjacency matrix M of the graph as $M[b, c] = 2$ and $M[c, b] = 2$. In a similar way using lists for the candidate edges for each orientation rule and allowing bi-directed edges, the order-dependence issues in the orientation rules can be solved. Note that bi-directed edges merely represents a conflicting orientation and they should not to be interpreted causally. The usage of these lists for the candidate edges and allowing bi-directed edges resolve the order-dependence issues on the orientation of the v-structures and on the edges using the three orientation rules, see Colombo and Maathuis (2013) for more details.

Note that calling (`conservative = TRUE`) or `maj.rule = TRUE`, together with `solve.confl = TRUE` produces a fully order-independent output, see Colombo and Maathuis (2013).

Sampling errors, non faithfulness, 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"`.

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. ges

The PC algorithm presented in the previous section is based on conditional independence tests. To apply it, we first had to calculate a sufficient statistic and to specify a conditional independence test function. For the score-based GES algorithm, we have to define a score object before applying the inference algorithm.

A score object is an instance of a class derived from the base class `Score`. This base class is implemented as a virtual reference class. At the moment, the **pcalg** package only contains classes derived from `Score` for Gaussian data: `GaussL0penObsScore` for purely observational data, and `GaussL0penIntScore` for a mixture of observational and interventional data; for the GES algorithm, we only need the first one here, but we will encounter the second one in Section 3.6 again. The implementation of score classes for discrete data is planned for future versions of the **pcalg** package. However, the flexible implementation using class inheritance allows the user to implement own score classes for different scores.

The predefined score-class `GaussL0penObsScore` implements a an ℓ_0-penalized maximum-likelihood estimator for observational data from a Gaussian causal model. A instance is

```
> suffStat <- list(C = cor(gmG8$x), n = nrow(gmG8$x))
> pc.fit <- pc(suffStat, indepTest=gaussCItest, p = ncol(gmG8$x), alpha = 0.01)
> par(mfrow= c(1,2));  plot(gmG8$g, main = ""); plot(pc.fit, main = "")
```

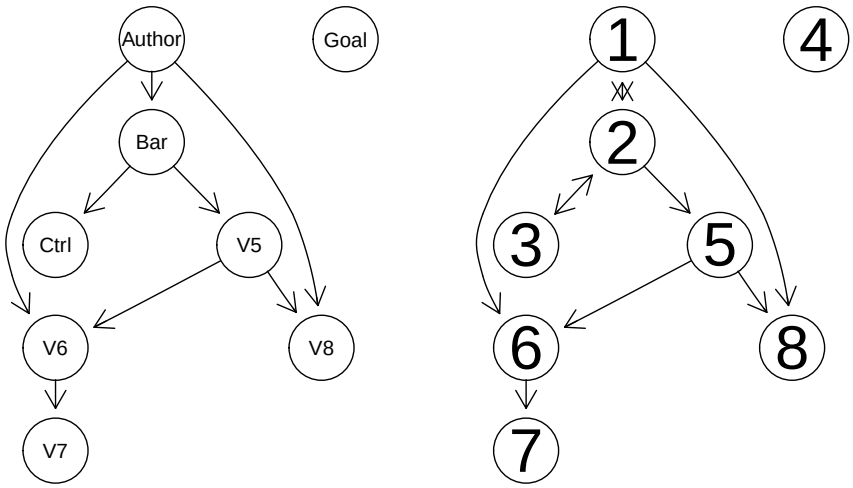

Figure 4: True underlying DAG (left) and estimated CPDAG (right) fitted on the simulated Gaussian data set `gmG`.

generated as follows:

```
> score <- new("GaussL0penObsScore", data = matrix(1, 1, 1),
      lambda = 0.5*log(nrow(data)), intercept = FALSE, use.cpp = TRUE, ...)
```

The data matrix is provided by the argument `data`. The penalization constant is specified by `lambda`. The default value of `lambda` corresponds to the BIC score; the AIC score is realized by setting `lambda` to 1. The argument `intercept` indicates whether the model should allow for intercepts in the linear structural equations, or for a non-zero mean of the multivariate normal distribution, which is equivalent. The last argument `use.cpp` indicates whether the internal C++ library should be used for calculation, which is in most cases the best choice due to velocity issues.

Once a score object is defined, the GES algorithm is called as follows:

```
ges(score, labels = score$getNodes(), fixedGaps = NULL, adaptive = c("none",
    "vstructures", "triples"), phase = c("forward", "backward", "turning"),
    iterate = length(phase) > 1, turning = NULL, maxDegree = integer(0),
    verbose = FALSE, ...)
```

The argument `score` is a score object defined before. The argument `turning` indicates whether the novel turning phase (see Section 2.1) not present in the original implementation of Chickering (2002) should be used, and `maxdegree` can be used to bound the vertex degree of the estimated graph. More details can be found in the help file of `ges()`.

In Fig. 5, we re-analyze the data set used in the example of Fig. 4 with the GES algorithm. The estimated graph is exactly the same in this case.

```
> score <- new("GaussL0penObsScore", gmG8$x)
> ges.fit <- ges(score)
> par(mfrow=1:2); plot(gmG8$g, main = ""); plot(ges.fit$essgraph, main = "")
```

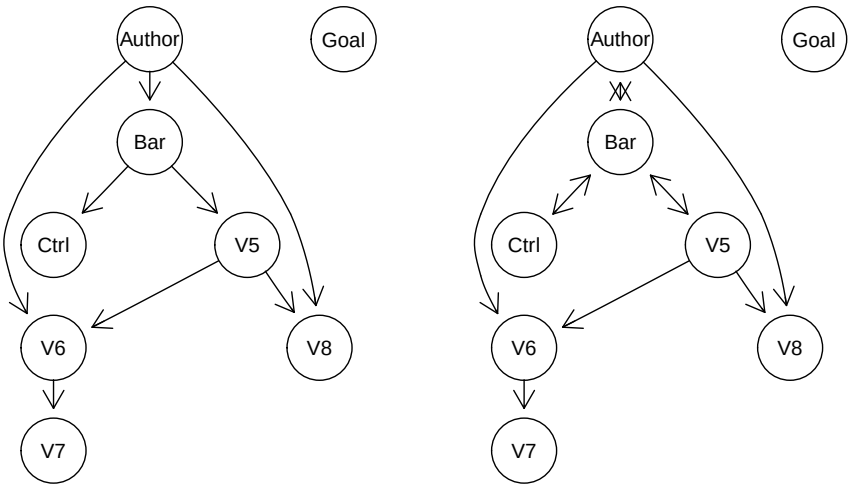

Figure 5: True underlying DAG (left) and essential graph (right) estimated with the GES algorithm fitted on the simulated Gaussian data set `gmG`.

3.4. 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()`, which produces an initial order-independent skeleton. All edges of this skeleton are of the form $\circ\!\!-\!\!\circ$. 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 which requires some orientations. Therefore, the v-structures are determined using the conservative method (see discussion on `conservative` below). All potential v-structures $a - b - c$ are checked in the following way. We test whether V_a and V_c are independent conditioning on any subset of the neighbors of a or any subset of the neighbors of c. If b is in no such conditioning set or in all such conditioning sets, no further action is taken. If, however, b is in only some conditioning sets, the triple $a - b - c$ is marked as *ambiguous*. If V_a is independent of V_c given some S in the skeleton (i.e., the edge $a - c$ dropped out), but V_a and V_c remain dependent given all subsets of neighbors of either a or c, we will call all such triples $a - b - c$ *unambiguous*. This is because in the FCI, the true separating set might be outside the neighborhood of either a or c. An ambiguous triple is not oriented as a v-structure. After the v-structures have been oriented, Possible-D-SEP sets for each node in the graph are computed at once. To decide whether edge $x \circ\!\!-\!\!\circ y$ should

be removed, one performs conditional independence tests of V_x and V_y given all subsets of Possible-D-SEP(x) and of Possible-D-SEP(y) (see help file of function `pdsep()`). The edge is removed if a conditional independence is found. This will produce a fully order-independent final skeleton as explained in Colombo and Maathuis (2013). Subsequently, all edges are transformed into ○–○ 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, alpha, labels, p, skel.method = c("stable",
    "original", "stable.fast"), type = c("normal", "anytime", "adaptive"),
    fixedGaps = NULL, fixedEdges = NULL, NAdelete = TRUE, m.max = Inf,
    pdsep.max = Inf, rules = rep(TRUE, 10), doPdsep = TRUE, biCC = FALSE,
    conservative = FALSE, maj.rule = FALSE, numCores = 1, verbose = FALSE)
```

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

The argument `pdsep.max` indicates the maximum size of Possible-D-SEP for which subsets are considered as conditioning sets in the conditional independence tests. If the nodes `x` and `y` are adjacent in the graph and the size of Possible-D-SEP(`x`)\{`x`,`y`} is bigger than `pdsep.max` the edge is simply left in the graph. Note that if `pdsep.max` is less than Inf, the final PAG may be a supergraph than the one computed with `pdsep.max = Inf`, because less tests may have been performed in the former.

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 not defined otherwise in `pdsep.max`. If FALSE, Possible-D-SEP is not computed, so that the algorithm simplifies to the Modified PC algorithm of Spirtes *et al.* (2000).

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.

Conservative versions of FCI, Anytime FCI, and Adaptive Anytime FCI (see below) are computed if the argument of `conservative` is `TRUE`. After the final skeleton is computed, all potential v-structures a — b — c are checked in the following way. We test whether V_a and V_c are independent conditioning on any subset of the neighbors of a or any subset of the neighbors of c. When a subset makes V_a and V_c conditionally independent, we call it a separating set. If b is in no such separating set or in all such separating sets, no further action is taken and the normal version of the FCI, Anytime FCI, or Adaptive Anytime FCI algorithm is continued. If, however, b is in only some separating sets, the triple a — b — c is marked *ambiguous*. If V_a is independent of V_c given some S in the skeleton (i.e., the edge a — c dropped out), but V_a and V_c remain dependent given all subsets of neighbors of either a or c, we will call all triples a — b — c *unambiguous*. This is because in the FCI algorithm, the

true separating set might be outside the neighborhood of either a or c. An ambiguous triple is not oriented as a v-structure. Furthermore, no further orientation rule that needs to know whether a——b——c is a v-structure or not is applied. Instead of using the conservative version, which is quite strict towards the v-structures, Colombo and Maathuis (2013) introduced a less strict version for the v-structures called majority rule. This adaptation can be called using `maj.rule = TRUE`. In this case, the triple a —— b —— c is marked as *ambiguous* if and only if b is in exactly 50 percent of such separating sets or no separating set was found. If b is in less than 50 percent of the separating sets it is set as a v-structure, and if in more than 50 percent it is set as a non v-structure, for more details see Colombo and Maathuis (2013). Colombo and Maathuis (2013) showed that with both these modifications, the final skeleton and the decisions about the v-structures of the FCI algorithm are fully order-independent.

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.

The argument `type` specifies the version of the FCI that has to be used. Per default it is `normal` and so the normal FCI algorithm is called. If set as `anytime`, the Anytime FCI Spirtes (2001) is called and in this case `m.max` must be specified by the user. The Anytime FCI algorithm can be viewed as a modification of the FCI algorithm that only performs conditional independence tests up to and including order m.max when finding the initial skeleton, using the function `skeleton`, and the final skeleton, using the function `pdsep`. Thus, Anytime FCI performs fewer conditional independence tests than FCI. If set as `adaptive`, the Adaptive Anytime FCI Colombo *et al.* (2012) is called and in this case m.max is not used. The first part of the algorithm is identical to the normal FCI described above. But in the second part when the final skeleton is estimated using the function `pdsep`, the Adaptive Anytime FCI algorithm only performs conditional independence tests up to and including order m.max, where m.max is the maximum size of the conditioning sets that were considered to determine the initial skeleton using the function `skeleton`.

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. 6 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.

3.5. rfci

The function `rfci()` is rather similar to `fci()`. However, it does not compute any Possible-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, alpha, labels, p, skel.method = c("stable",
    "original", "stable.fast"), fixedGaps = NULL, fixedEdges = NULL, NAdelete = TRUE,
    m.max = Inf, rules = rep(TRUE, 10), conservative = FALSE, maj.rule = FALSE,
    numCores = 1, verbose = FALSE)
```

where the arguments `suffStat`, `indepTest`, `p`, `alpha`, `fixedGaps`, `fixedEdges`, `NAdelete`

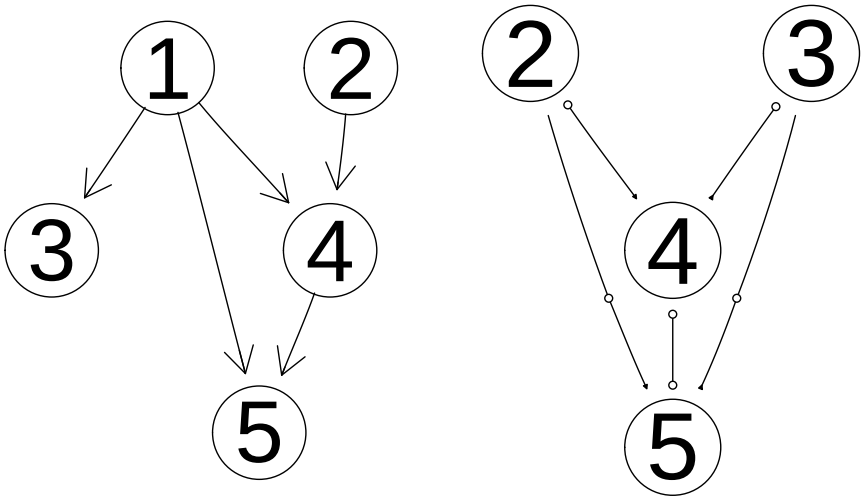

Figure 6: True underlying DAG (left) and estimated PAG (right), when applying the FCI and RFCI algorithms to the data set `gmL`. The output of FCI and RFCI is identical. Variable V_1 of the true underlying DAG is latent.

and `m.max` are identical to those in `skeleton()`.

The argument `rules` is similar to the one in `fci` but modified to produce a PAG with correct ancestral relationships, in the oracle version.

The first part of the RFCI algorithm is analogous to the PC and FCI algorithm. It starts with a complete undirected graph and estimates an initial skeleton using the function `skeleton`, which produces an initial order-independent skeleton, see `skeleton` for more details. All edges of this skeleton are of the form $\circ\!\!-\!\!\circ$. Due to the presence of hidden variables, it is no longer sufficient to consider only subsets of the neighborhoods of nodes `x` and `y` to decide whether the edge `x o-o y` should be removed. The FCI algorithm performs independence tests conditioning on subsets of Possible-D-SEP to remove those edges. Since this procedure is computationally infeasible, the RFCI algorithm uses a different approach to remove some of those superfluous edges before orienting the v-structures and the discriminating paths in orientation rule 4.

Before orienting the v-structures, we perform the following additional conditional independence tests. For each unshielded triple $a - b - c$ in the initial skeleton, we check if both V_a and V_b and V_b and V_c are conditionally dependent given the separating of a and c (sepset(a, c)). These conditional dependencies may not have been checked while estimating the initial skeleton, since sepset(a, c) does not need to be a subset of the neighbors of a nor of the neighbors of c. If both conditional dependencies hold and b is not in the sepset(a, c), the triple is oriented as a v-structure $a \longrightarrow b \longleftarrow c$. On the other hand, if an additional conditional independence relationship may be detected, say V_a is independent from V_b given the sepset(a, c), the edge between a and c is removed from the graph and the set responsible for that is saved in sepset(a, b). The removal of an edge can destroy or create new unshielded triples in the graph. To solve this problem we work with lists (for details see Colombo *et al.* 2012).

Before orienting discriminating paths, we perform the following additional conditional in-

dependence tests. For each triple $a \leftarrow\!\bullet\, b \circ\!\!-\!\!\bullet\, c$ with $a \longrightarrow c$, the algorithm searches for a discriminating path $p = \langle d, ..., a, b, c\rangle$ for b of minimal length, and checks that the vertices in every consecutive pair (V_f, V_g) on p are conditionally dependent given all subsets of $\text{sepset}(d, c) \setminus V_f, V_g$. If we do not find any conditional independence relationship, the path is oriented as in rule (R4). If one or more conditional independence relationships are found, the corresponding edges are removed, their minimal separating sets are stored.

Conservative RFCI can be computed if the argument of `conservative` is `TRUE`. After the final skeleton is computed and the additional local tests on all unshielded triples, as described above, have been done, all potential v-structures $a - b - c$ are checked in the following way. We test whether V_a and V_c are independent conditioning on any subset of the neighbors of a or any subset of the neighbors of c. When a subset makes V_a and V_c conditionally independent, we call it a separating set. If b is in no such separating set or in all such separating sets, no further action is taken and the normal version of the RFCI algorithm is continued. If, however, b is in only some separating sets, the triple $a - b - c$ is marked *ambiguous*. If V_a is independent of V_c given some S in the skeleton (i.e., the edge $a - c$ dropped out), but V_a and V_c remain dependent given all subsets of neighbors of either a or c, we will call all triples $a - b - c$ *unambiguous*. This is because in the RFCI algorithm, the true separating set might be outside the neighborhood of either a or c. An ambiguous triple is not oriented as a v-structure. Furthermore, no further orientation rule that needs to know whether $a - b - c$ is a v-structure or not is applied. Instead of using the conservative version, which is quite strict towards the v-structures, Colombo and Maathuis (2013) introduced a less strict version for the v-structures called majority rule. This adaptation can be called using `maj.rule = TRUE`. In this case, the triple $a - b - c$ is marked as *ambiguous* if and only if b is in exactly 50 percent of such separating sets or no separating set was found. If b is in less than 50 percent of the separating sets it is set as a v-structure, and if in more than 50 percent it is set as a non v-structure, for more details see Colombo and Maathuis (2013).

The implementation uses the stabilized skeleton `skeleton()`, which produces an initial order-independent skeleton. The final skeleton and edge orientations can still be order-dependent, see Colombo and Maathuis (2013).

As an example, we re-run the example from Section 3.4 and show the PAG estimated with `rfci()` in Figure 6. The PAG estimated with `fci()` and the PAG estimated with `rfci()` are the same.

```
> data("gmL")
> suffStat1 <- list(C = cor(gmL$x), n = nrow(gmL$x))
> 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`.

3.6. gies and simy

As we noted in Section 2.1, the GIES algorithm is a generalization of the GES algorithm to a mix of interventional and observational data. Hence the usage of `gies()` is similar to that of `ges()` (see Section 3.3). Actually, the function `ges()` is only an internal wrapper for `gies()`.

A data set with jointly interventional and observational data points is *not* i.i.d. In order to use it for causal inference, we must specify the intervention target each data point belongs to. This is done by specifying the arguments `target` and `target.index` when generating an instance of `GaussL0penIntScore` (see Section 3.3):

```
> score <- new("GaussL0penIntScore", data = matrix(1, 1, 1),
      targets = list(integer(0)), target.index = rep(as.integer(1), nrow(data)),
      lambda = 0.5*log(nrow(data)), intercept = FALSE, use.cpp = TRUE, ...)
```

The argument `targets` is a list of all (mutually different) targets that have been intervened in the experiments generating the data set. The allocation of sample indices to intervention targets is specified by the argument `target.index`. This is an integer vector whose first entry specifies the index of the intervention target in the list `targets` of the first data point, whose second entry specifies the target index of the second data point, and so on. An example is given in Figure 7.

Once a score object for interventional data is defined, the GIES algorithm is called as follows:

```
gies(score, labels = score$getNodes(), targets = score$getTargets(),
    fixedGaps = NULL, adaptive = c("none", "vstructures", "triples"), phase = c("forward",
        "backward", "turning"), iterate = length(phase) > 1, turning = NULL,
    maxDegree = integer(0), verbose = FALSE, ...)
```

Most arguments coincide with those of `ges()` (see Section 3.3). The only additional argument is `targets`: it takes the same list of (unique) intervention targets as the constructor of the class `GaussL0penIntScore` (see above). This list of targets specifies the space of corresponding interventional Markov equivalence classes or essential graphs (see Section 2.1).

The data set `gmInt` consists of 5000 data points sampled from the DAG in Figure 2, among them 3000 observational ones, 1000 originating from an intervention at vertex 3 and 1000 originating from an intervention at vertex 5. It can be loaded by calling

```
> data(gmInt)
```

The underlying causal model (or its interventional essential graph, respectively) is estimated in Figure 7.

As an alternative to GIES, we can also use the dynamic programming approach of Silander and Myllymäki (2006) to estimate the interventional essential graph from this interventional data set. This algorithm is implemented in the function `simy()` which has the same arguments as `gies()`. As noted in Section 2.1, this approach yields the *exact* optimum of the BIC score at the price of a computational complexity which is exponential in the number of variables. On the small example based on 8 variables, using this algorithm is feasible; however, it is not feasible for more than approximately 20 variables, depending on the processor and memory of the machine. In this example, we get exactly the same result as with `gies()` (see Figure 7).

3.7. 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. 8. We assume that the causal structure is unknown and want to infer the causal effect 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).

```
> data("gmI")
```

```
> score <- new("GaussL0penIntScore", gmInt$x, targets = gmInt$targets,
                target.index = gmInt$target.index)
> gies.fit <- gies(score)
> simy.fit <- simy(score)
> par(mfrow = c(1, 3)) ; plot(gmInt$g, main = "")
> plot(gies.fit$essgraph, main = "")
> plot(simy.fit$essgraph, main = "")
```

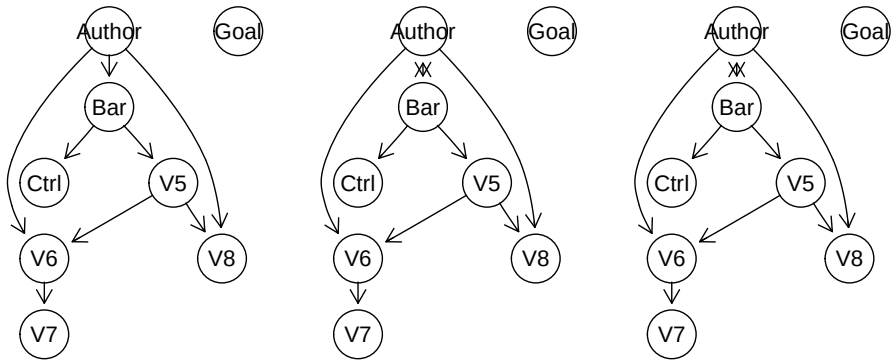

Figure 7: The underlying DAG (left) and the essential graph estimated with the GIES algorithm (middle) and the dynamic programming approach of Silander and Myllymäki (2006) (right) applied on the simulated interventional Gaussian data set `gmInt`. This data set contains data from interventions at vertices 3 and 5; accordingly, the orientation of all arrows incident to these two vertices becomes identifiable (see also Figure 5 for comparison with the observational case.

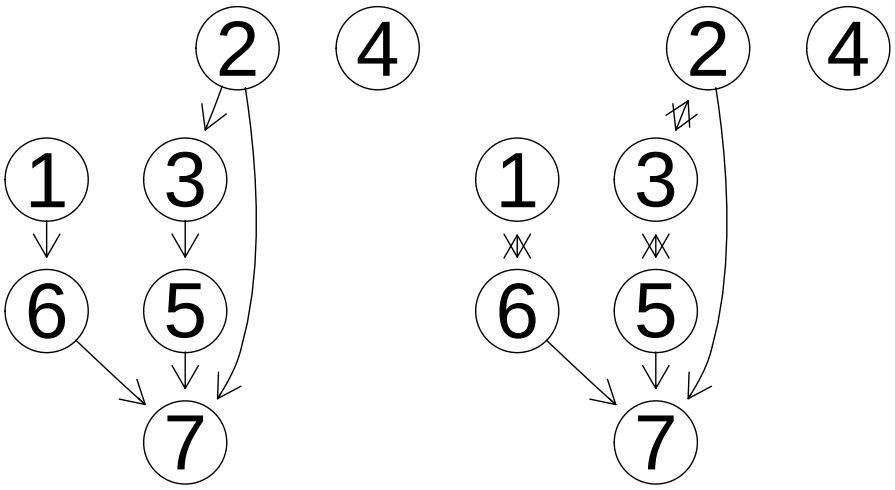

Figure 8: True DAG (left) and estimated CPDAG (right).

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

Comparing the true DAG with the CPDAG in Fig. 8, 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 $2 \longrightarrow 3 \longrightarrow 5$, $2 \longleftarrow 3 \longleftarrow 5$ or $2 \longleftarrow 3 \longrightarrow 5$, and not as $2 \longrightarrow 3 \longleftarrow 5$. 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. 9, 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 $z \longrightarrow x$); 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 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.

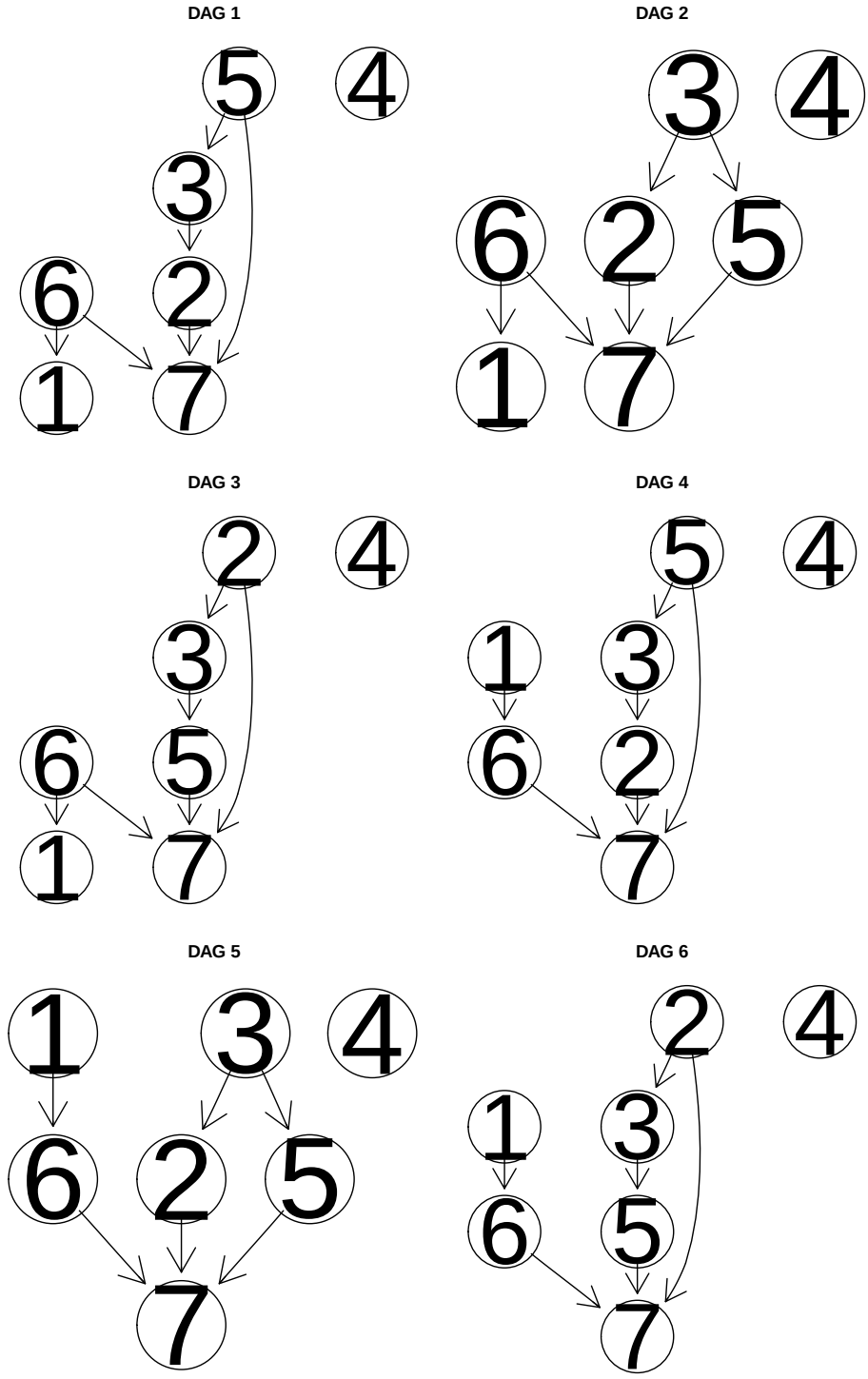

Figure 9: All DAGs belonging to the same equivalence class as the true DAG shown in Fig. 6.

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.

```
> ida(2, 5, cov(gmI$x), pc.gmI@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 = c("local", "global"),
    y.notparent = FALSE, verbose = FALSE, all.dags = NA, type = c("cpdag",
        "pdag"))
```

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.

```
> ida(2,5, cov(gmI$x), pc.gmI@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 definitively 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.

3.8. 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 3.7, 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.

```
> (eff.est1 <- ida(2,5, cov(gmI$x), pc.gmI@graph, method="local"))

[1]  0.5421360 -0.0049012

> (eff.est2 <- ida(2,6, cov(gmI$x), pc.gmI@graph, method="local"))

[1] -0.0058532 -0.0062310
```

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

3.9. backdoor

This function is a generalization of Pearl's backdoor criterion, see Pearl (1993), defined for directed acyclic graphs (DAGs), for single interventions and single outcome variable to more general types of graphs (CPDAGs, MAGs, and PAGs) that describe Markov equivalence classes of DAGs with and without latent variables but without selection variables, for more details see Maathuis and Colombo (2013).

The motivation to find a set W that satisfies the generalized backdoor criterion with respect to X and Y in the given graph relies on the result of the generalized backdoor adjustment that says: "If a set of variables W satisfies the generalized backdoor criterion relative to X and Y in the given graph, then the causal effect of X on Y is identifiable and is given by:" $P(Y|\text{do}(X = x)) = \sum_W P(Y|X, W).P(W)$. This result allows to write post-intervention densities (the one written using Pearl's do-calculus) using only observational densities estimated from the data.

This function can be called in the following way:

`backdoor(amat, x, y, type = "pag", max.chordal = 10, verbose = FALSE)`

where `amat` is the adjacency matrix of the given graph, `x` denotes the column position of the cause variable, `y` denotes the column position of the effect variable, and `mcov` is the covariance matrix of the original data.

The argument `type` specifies the type of graph of the given adjacency matrix in `amat`. If the input graph is a DAG (`type="dag"`), this function reduces to Pearl's backdoor criterion for single interventions and single outcome variable, and the parents of X in the DAG satisfies the backdoor criterion unless Y is a parent of X. Therefore, if Y is a parent of X, there is no set W that satisfies the generalized backdoor criterion relative to X and Y in the DAG and NA is output. Otherwise, the causal effect is identifiable and a set W that satisfies the generalized backdoor criterion relative to X and Y in the DAG is given. If the input graph is a CPDAG C (`type="cpdag"`), a MAG M, or a PAG P (with both M and P not allowing selection variables), this function first checks if the total causal effect of X on Y is identifiable via the generalized backdoor criterion (see Maathuis and Colombo (2013), Theorem 4.1). If the effect is not identifiable, the output is NA. Otherwise, an explicit set W that satisfies the generalized backdoor criterion relative to X and Y in the given graph is found.

Note that if the set W is equal to the empty set, the output is NULL.

At this moment this function is not able to work with PAGs estimated using the `rfci` Algorithm.

It is important to note that there can be pair of nodes `x` and `y` for which there is no set W that satisfies the generalized backdoor criterion, but the total causal effect might be identifiable via some other technique.

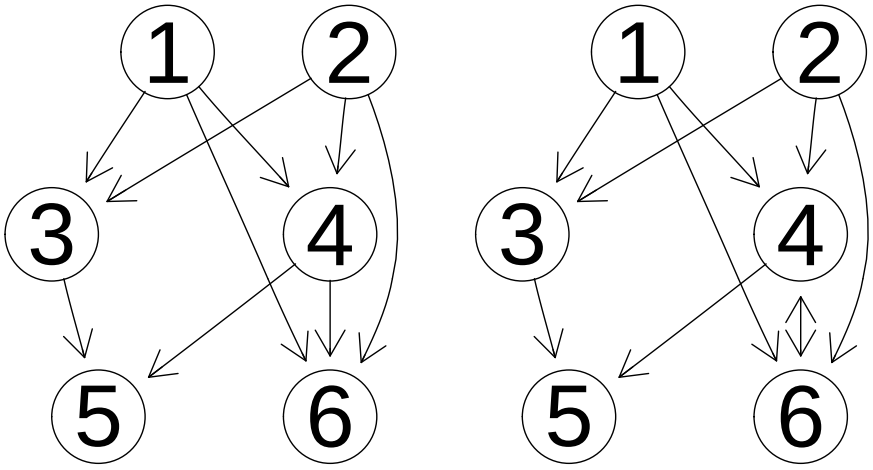

Figure 10: True DAG (left) and estimated CPDAG (right).

To illustrate this function, we use as example the CPDAG displayed in Figure 4, page 15 of Maathuis and Colombo (2013). The R-code below is used to generate a DAG `g` that belongs to the required equivalence class which is uniquely represented by the estimated CPDAG `myCPDAG`.

```
> p <- 6
> amat <- t(matrix(c(0,0,1,1,0,1, 0,0,1,1,0,1, 0,0,0,0,1,0,
                     0,0,0,0,1,1, 0,0,0,0,0,0, 0,0,0,0,0,0), 6,6))
> V <- as.character(1:6)
> colnames(amat) <- rownames(amat) <- V
> edL <- vector("list",length=6)
> names(edL) <- V
> edL[[1]] <- list(edges=c(3,4,6),weights=c(1,1,1))
> edL[[2]] <- list(edges=c(3,4,6),weights=c(1,1,1))
> edL[[3]] <- list(edges=5,weights=c(1))
> edL[[4]] <- list(edges=c(5,6),weights=c(1,1))
> g <- new("graphNEL", nodes=V, edgeL=edL, edgemode="directed")
> cov.mat <- trueCov(g)
> myCPDAG <- dag2cpdag(g)
> true.amat <- as(myCPDAG, "matrix")
> ## true.amat[true.amat != 0] <- 1
```

The DAG `g` and the CPDAG `myCPDAG` are shown in Figure 10.

Now, we want to check if the effect of V_6 on V_3 in the given CPDAG is identifiable using `backdoor()` and if this is the case know which set W satisfies the generalized backdoor criterion. As explained in Example 4 in Maathuis and Colombo (2013), the causal effect of V_6 on V_3 in the CPDAG `myCPDAG` is identifiable via the generalized backdoor criterion and there is a set W that satisfies the generalized criterion:

```
> backdoor(true.amat, 6, 3, type="cpdag")
```

```
[1] 1 2
```

3.10. 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 $\texttt{lm}(V_x \sim V_S)$ and $\texttt{lm}(V_y \sim V_S)$, 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.

```
> 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()`.

```
> pc.myfit <- pc(suffStat = gmG8$x, indepTest = myCItest,
                 p = 8, alpha = 0.01)
> par(mfrow = c(1,2)); plot(pc.gmG, main = ""); plot(pc.myfit, main = "")
```

As expected, the resulting CPDAG (see Fig. 11) 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()`).

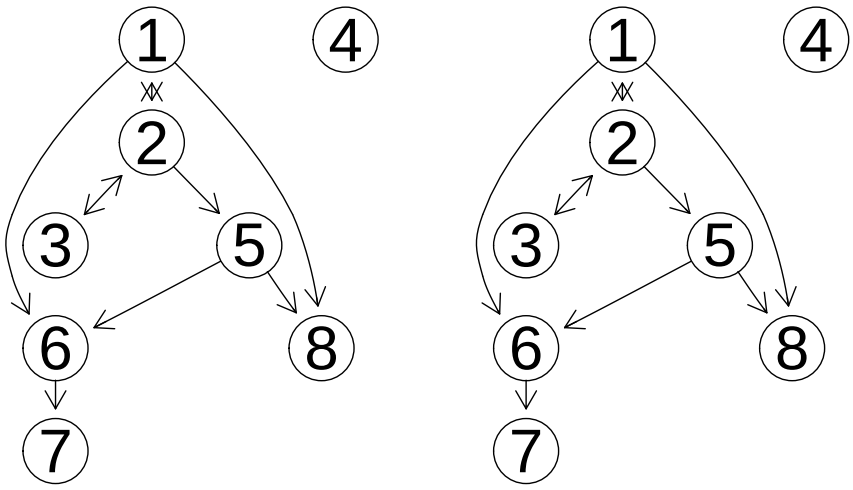

Figure 11: The estimated CPDAGs using the predefined conditional independence test `gaussCItest()` (left) and the user specified conditional independence test `myCItest()` (right) are identical for the `gmG` data.

4. Applications

The **pcalg** package has been used for applications in epidemiology (see Kalisch *et al.* 2010), biology (see Maathuis *et al.* 2010; Nagarajan *et al.* 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 *et al.* 2000), the goal was to identify the largest intervention effects between all pairs of genes. For the analysis, the **pcalg** package with version 1.1-5 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 12 one can see that `ida()`, using the function `skeleton()` with version 1.1-5, is clearly superior to the alternative methods (and random guessing) in terms of identifying effects in the target set. In Figure 13 one can see that `ida()` using the stable function `skeleton()` produces even better results than with the old version.

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 12 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.

Data from cell biology is usually interventional data: different measurements could originate from different mutants or different cell stems, or from experiments with gene knockout or knockdown experiments. The GIES algorithm is designed for causal inference from data of this kind.

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.

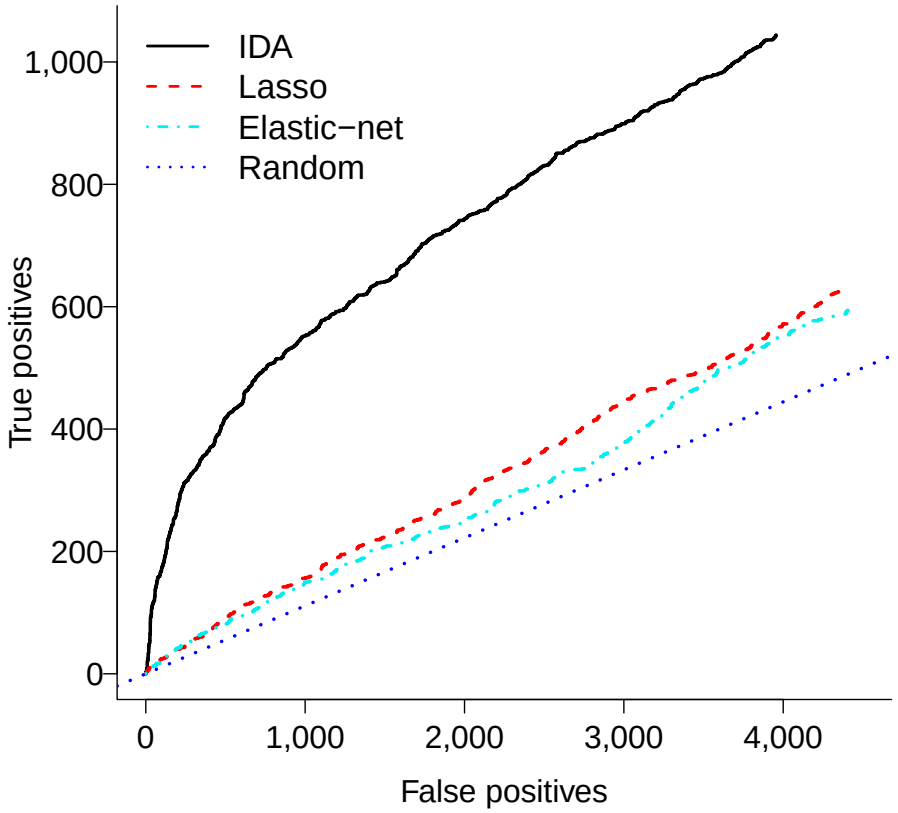

Figure 12: The largest 10% of the causal effects found in experiments among yeast genes are identified much better from observational data with IDA than with Lasso, Elastic Net or random guessing. The figure is essentially taken from Maathuis *et al.* (2010).

References

Andersson SA, Madigan D, Perlman MD (1997). "A characterization of Markov equivalence classes for acyclic digraphs." *Annals of Statistics*, **25**(2), 505–541.

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.

Chickering DM (1996). "Learning Bayesian networks is NP-complete." *Learning from data: Artificial intelligence and statistics V*, **112**, 121–130.

Chickering DM (2002). "Optimal structure identification with greedy search." *Journal of Machine Learning Research*, **3**(3), 507–554.

Colombo D, Maathuis MH (2013). "Order-independent constraint-based causal structre learning." *ArXiv preprint 1211.3295*. URL http://arxiv.org/abs/1211.3295.

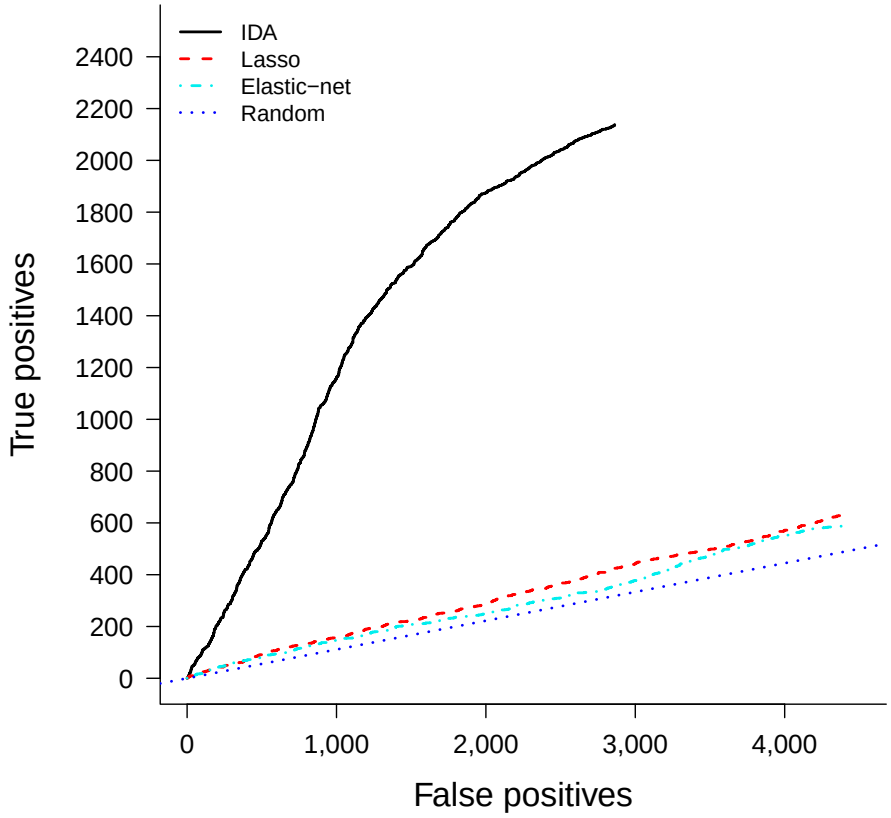

Figure 13: The largest 10% of the causal effects found in experiments among yeast genes are identified much better from observational data with IDA than with Lasso, Elastic Net or random guessing. The figure is essentially taken from Colombo and Maathuis (2013).

Colombo D, Maathuis MH, Kalisch M, Richardson TS (2012). "Learning High-Dimensional drected acyclic graphs with latent and selection variables." *Annals of Statistics*, **40(1)**, 294–321.

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

Geiger D, Heckerman D, King H, Meek C (2001). "Stratified exponential families: graphical models and model selection." *Annals of Statistics*, **29**(2), 505–529.

Grimmett GR, McDiarmid CJH (1975). "On colouring random graphs." *Mathematical Proceedings of the Cambridge Philosophical Society*, **77**, 313–324.

Haughton DDM (1988). "On the choice of a model to fit data from an exponential family." *Annals of Statistics*, **16**(1), 342–355.

Hauser A, Bühlmann P (2012). "Characterization and greedy learning of interventional

Markov equivalence classes of directed acyclic graphs." *Journal of Machine Learning Research*, **13**, 2409–2464.

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**.

Kalisch M, Mächler M, Colombo D, Maathuis MH, Bühlmann P (2012). "Causal Inference Using Graphical Models with the R Package pcalg." *Journal of Statistical Software*, **47**(11), 1–26. ISSN 1548-7660. URL http://www.jstatsoft.org/v47/i11.

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

Maathuis MH, Colombo D (2013). "A generalized backdoor criterion." *ArXiv e-prints*. 1307.5636, URL http://arxiv.org/abs/1307.5636.

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 11th Conference on Uncertainty in Artificial Intelligence (UAI 1995)*, 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 (1993). "Comment: Graphical models, causality and intervention." *Statistical Science*, **8**, 266–269.

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

R Development Core Team (2014). *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 22nd Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, 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/.

Silander T, Myllymäki P (2006). "A simple approach for finding the globally optimal Bayesian network structure." In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, pp. 445–452.

Spirtes P (2001). "An anytime algorithm for causal inference." In *Proc. of the Eighth International Workshop on Artificial Intelligence and Statistics*, pp. 213–221. Morgan Kaufmann, San Francisco.

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 (1990). "Equivalence and Synthesis of Causal Models." In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence (UAI 1990)*, pp. 255–270.

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