



Causal Inference using Graphical Models with the Package **pcalg**

Markus Kalisch Martin Mächler Diego Colombo Marloes H. Maathuis Peter Bühlmann
ETH Zurich ETH Zurich ETH Zurich ETH Zurich ETH Zurich

Abstract

The **pcalg** package for R is a tool for estimating intervention effects when the true underlying causal structure is unknown. To this end, **pcalg** contains functions for estimating the causal structure using graphical models (functions **skeleton**, **pc** and **fci**) and functions for estimating intervention effects given an estimate of the causal structure (functions **ida** and **idaFast**). In this document, we will give a brief overview of the methodological background of estimating intervention effects. Moreover, we will demonstrate how the package **pcalg** can be used for estimating intervention effects and graphical models using examples.

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

1. Introduction

THIS DOCUMENTATION IS STILL UNDER CONSTRUCTION!

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.

Recently, we proposed and mathematically justified a statistical method (IDA) to obtain bounds on total causal effects based solely on observational data under some assumptions (see ?). Furthermore, we presented an experimental validation of our method on a large-scale biological system (see ?).

For further validation and broader use of this method, well documented and easy to use software is indispensable. Therefore, we wrote the R package **pcalg**, which incorporates the above mentioned method (IDA).

The objective of this paper is to introduce the R package **pcalg** and explain the range of functions.

To get started, we show how two of the main functions 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 arrow represents a direct cause. We are interested in the change of variable Y if we changed the variable X by intervention, i.e., we seek the causal effect of X on Y . To fix ideas, we have simulated an example data set with $p = 8$ continuous variables with Gaussian noise and $n = 5000$ observations, which we will now analyse. First, we load the package **pcalg** and the data set.

```
> 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. The sufficient statistic consists 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(gmG$x), n = nrow(gmG$x))
> pc.fit <- pc(suffStat, indepTest = gaussCitest,
+             p = ncol(gmG$x), alpha = 0.01)
> require(Rgraphviz)
> par(mfrow = c(1,2))
> plot(gmG$g, main = "True DAG")
> plot(pc.fit, main = "Estimated DAG")
```

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. The direction of the bidirected edges, however, could not be decided by our method. Thus, they represent some uncertainty in the resulting model. A fundamental property of our method is, that some uncertainty of this kind sometimes remains, even if an infinite amount of data is available.

Based on the inferred causal structure, we can estimate the causal effect of an intervention. For example, suppose, we would increase variable $V1$ by external intervention by one unit. The recorded increase in variable $V6$ is the (total) causal effect of $V1$ on $V6$. More precisely, the causal effect $C(V1, V6, \tilde{x})$ of $V1$ from $V1 = \tilde{x}$ on $V6$ is defined as

$$\begin{aligned} C(V1, V6, \tilde{x}) &= E(V1|do(V6 = \tilde{x} + 1)) - E(V1|do(V6 = \tilde{x})) \text{ or} \\ C(V1, V6, \tilde{x}) &= \frac{\partial}{\partial x} E(V1|do(V6 = x))|_{x=\tilde{x}} \end{aligned}$$

If the causal relationships are linear, the causal effect is independent of \tilde{x} .

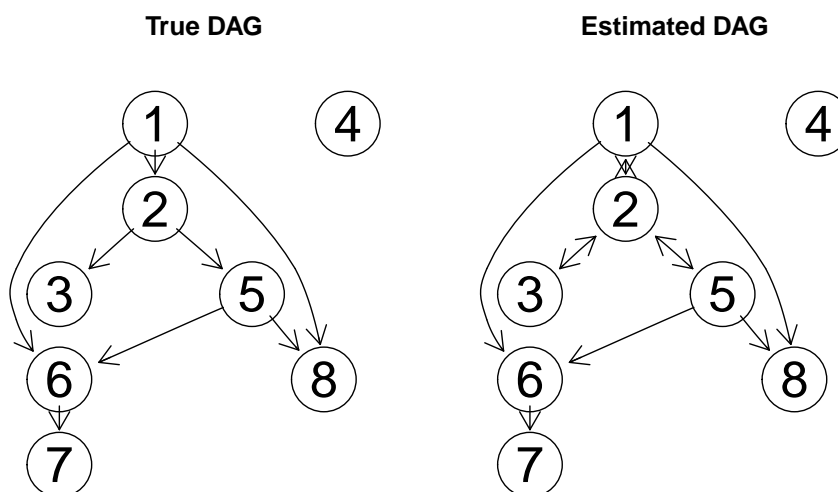


Figure 1: True and estimated causal structure represented by a Directed Acyclic Graph (DAG).

Since the causal structure was not identified perfectly in our example, we cannot expect to get a unique number for the causal effect. Instead, we will 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(gmG$x), pc.fit@graph)
```

```
[1] 0.75364 0.54878
```

Since we simulated the data, we know that the true value of the causal effect is 0.71. Thus, one of the two estimates is indeed close to the true value. Since both values are larger than zero, we can conclude, that variable $V1$ has a positive causal effect on variable $V6$. Thus, we can always estimate a lower bound to 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 $V1$ on variables $V4$, $V5$ and $V6$, 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(gmG$x), pc.fit@graph)
```

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

Each row in the output shows the estimated set of possible causal effects on the target variable indicated by the row names. The true values for the causal effects are 0, 0.2, 0.71 for variables $V4$, $V5$ and $V6$, respectively. The first row, corresponding to variable $V4$, 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 $V5$, is rather uninformative: Although one entry comes close to the true value, the other estimate is 0. Thus, we cannot be sure if there is a causal effect at all. The third row, corresponding to $V6$ was already discussed above.

2. Methodological background

Our proposed method consists of two major steps. In the first step, the causal structure is estimated. This is done by estimating a graphical model. A graphical model is a map of the dependence structure of the data and can thus be an interesting object by itself. In the second step, we use the estimated causal structure and the do-calculus (see ?) to calculate bounds on causal effects.

2.1. Estimating 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 ?). 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. Firstly, one needs the physical map with symbols such as dots and lines. Secondly, 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. Firstly, a graphical model consists of a graph with dots, lines and potentially edge marks like arrowheads or circles. Secondly, 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 Directed Acyclic Graph (DAG) model. The physical map here is a graph consisting of nodes and arrows (only one arrowhead per line) connecting the nodes. As a further restriction, the arrows must be directed in a way, so that it is not possible to trace a circle when following the arrowheads. The interpretation rule for this graph is called d-separation. This rule is a bit more intricate and we refer the reader to ? for more details. The interpretation rule for this graph can be used in the following way when given a DAG model: Two nodes X and Y are d-separated by a set of nodes S , if and only if the corresponding random variables X and Y are conditionally independent given the set of random variables S . For the following, we will only deal with distributions where the following holds: For each distribution, it is possible to find a DAG, whose list of d-separation relations perfectly matches the list of conditional independencies of the distribution. Such distributions are called faithful. It has been shown that the set of distributions that are faithful is the overwhelming majority (?), so that the assumption does not seem to be very strict in practice.

Since the DAG model encodes conditional independences, 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 ?; PC stands for the initials of its inventor Peter Spirtes and Clark Glymour) which was proven to reconstruct the structure of the underlying DAG model given a conditional independence oracle up to some ambiguity (equivalence class) to be discussed 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 ?).

It is possible, that several DAGs 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 $i \rightarrow j \leftarrow k$ on the nodes i , j and k where i and k are

not connected. 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 for the same conditional independence statements if and only if the corresponding DAGs have the same skeleton and the same v-structures (see ?). Two such DAGs are called equivalent. In this way, the space of DAGs can be partitioned into equivalence classes, where each member of an equivalence class encodes 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 can not determine the DAG directly, but only the corresponding equivalence class of the DAG. Each DAG in this equivalence class would be equally valid to describe the conditional independence information. The 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. Arrows 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 Completed Partially Directed Acyclic Graph (CPDAG).

Algorithm 1 Outline of the PC-algorithm

Input: Vertex set V , conditional independence information, significance level α

Output: Estimated CPDAG \hat{G} , separation sets \hat{S}

Edge types: \rightarrow , $-$

(P1) Form the complete undirected graph on the vertex set V

(P2) Test conditional independence given subsets of adjacency sets at a given significance level α and delete edges if conditional independent

(P3) Orient v-structures

(P4) Orient remaining edges.

We will now describe the PC-algorithm, which is shown in Algorithm 1, in more detail. The PC-algorithm starts with a complete undirected graph, G_0 , as stated in **(P1)** of Algorithm 1. In stage **(P2)**, a series of conditional independence tests is done and edges are deleted in the following way. First, all pairs of nodes are tested for marginal independence. If two nodes i, j are judged to be marginally independent, 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 directly connected to node i by some edge). If there is any node k that makes i, j conditionally independent, the edge between i and j is removed and node k is saved as separation sets $\hat{S}[i, j]$ and $\hat{S}[j, i]$. If all adjacent pairs have been tested given one neighboring 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 neighborhoods 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 pair i, k and the pair j, k are each adjacent in the skeleton but i, j are not, is oriented based on the information saved in the conditioning sets $\hat{S}[i, j]$ and $\hat{S}[j, i]$. More precisely, $i - k - j$ is directed $i \rightarrow k \leftarrow j$ if k is neither in $\hat{S}[j, i]$ nor in $\hat{S}[i, j]$. Finally, in **(P4)** it may be possible to direct some of these 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 ?, p.85. The

resulting output is the equivalence class (CPDAG) for the given initial graph, in which every edge is either undirected (or bidirected, which has the same meaning in the context of the CPDAG) or directed.

With hidden or selection variables

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 don't know all hidden and selection variables. Therefore, it is practically unfeasible to find the true underlying DAG.

A more modest goal could be trying to find a structure, that represents all independence relationships among the observed variables (but not the latent and selection variables) of the underlying causal structure. It turns out, that this is possible. However, the resulting object will in general not be a DAG for the following reason. Suppose, we have a DAG including observational, latent and selection variables and we'd 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. I.e., DAGs are not closed under marginalization or conditioning and are therefore not suited for representing the conditional independence structure on the observed variables only.

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

Ancestral graphs have nodes, which represent random variables and edges which represent some kind of dependence. The edges can be either directed, undirected or bidirected (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:

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

In practice, a slight variation of ancestral graphs is used, that incorporates a third rule:

- 3:** Every missing edge corresponds to a conditional independence.

Ancestral graphs that obey all of the three above rules are called *Maximal Ancestral Graphs (MAG)*.

The conditional independence statements of MAGs can be read off using the concept of m-separation, which is almost identical to the concept of d-separation in DAGs.

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)*. A PAG contains the following types of edges: o-o, o-, o->, ->, <->, -. Roughly, the

bidirected edges come from hidden variables, and the undirected edges come from selection variables. The edges have the following interpretation: (i) there is an edge between x and y if and only if x and y are conditionally dependent given S for all sets S consisting of all selection variables and a subset of the observed variables; (ii) a tail on an edge means that this tail is present in all MAGs in the equivalence class; (iii) an arrowhead on an edge means that this arrowhead is present in all MAGs in the equivalence class; (iv) a o-edgemark means that there is 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 (see ?). This algorithm was slightly extended and proven to be sound and complete in ?. We refer the reader to ? for a detailed discussion of the FCI algorithm. Suffice it to say, that FCI is very similar to PC but makes additional conditional independence tests and uses more orientation rules (see section 3.3 for more details).

2.2. Estimating bounds on causal effects

One way of quantifying the causal effect of variable X on Y is to measure the state of Y if X is forced to take value $X = x$. If X and Y are random variables, forcing $X = x$ could have the effect of changing the distribution of Y . Following the conventions in ?, the resulting distribution after manipulation is denoted by $P[Y|do(X = x)]$. Note that this is different from the conditional distribution $P[Y|X = x]$. To illustrate this, imagine the following simplistic situation. Suppose we observe every hour a particular spot on the street. The random variable X denotes whether it rained during that hour ($X = 1$ if it rained, $X = 0$ otherwise). The random variable Y denotes whether the street was wet or damp at the end of that hour we looked ($Y = 1$ if it was wet, $Y = 0$ otherwise). If we assume $P(X = 1) = 0.1$ (rather dry region), $P(Y = 1|X = 1) = 0.99$ (the street is almost always still wet when we look after and it rained during the last hour) and $P(Y = 1|X = 0) = 0.02$ (other reasons for making the street wet are rare), we can compute the conditional probability $P(X = 1|Y = 1) = 0.85$. 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 $P(X = 1|do(Y = 1)) = P(X = 1) = 0.1$. Thus, the distribution of the random variable describing rain is quite different when making an observation and making an intervention. Oftentimes, only the change of the target distribution under intervention is reported. We will use the change in mean, i.e. $\frac{\partial}{\partial x} E[Y|do(X = x)]$, as measure for the causal effect for Gaussian random variables. For Gaussian random variable Y , $E[Y|do(X = x)]$ depends linearly on x . Therefore, the derivative is constant which means that the causal effect does not depend on x . For binary random variables (with domain $\{0, 1\}$) we use $E(X|do(Y = 1)) - E(X|do(Y = 0)) = P(X = 1|do(Y = 1)) - P(X = 1|do(Y = 0))$.

The goal in the remaining 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, ? (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. ? restructured the rules of “do-calculus” into algorithmic form and showed that they are complete in a sense

that, if the algorithm doesn't find a transformation, there exists none.

Unfortunately, the causal structure is rarely known in practice. Under some assumptions, Pearl (1995) showed (Th. 1.4.1) that there is a link between causal structures and graphical models. Roughly speaking, if the underlying causal structure is a DAG, we observe data generated from this DAG and then estimate a DAG model (i.e. a graphical model) on this data, the estimated CPDAG represents the equivalence class of the DAG model describing the causal structure. This holds if we have enough samples and assuming that the true underlying causal structure is indeed a DAG without latent or selection variables. Note that even given an infinite amount of data, we usually cannot identify the true DAG itself, but only its equivalence class. Every DAG in this equivalence is equally likely to represent the true causal structure. Taking this into account, we conceptually apply the do-calculus on each DAG within the equivalence class and thus obtain not a unique value for a causal effect but one value for each DAG in the causal structure (in practice, we developed a local method that is faster but yields a similar result; see 3.4 for more details). Therefore, even if we have an infinite amount of observations we can only 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 oftentimes still be very useful, when the set has certain properties (e.g. all values are much larger than zero).

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, in a real world situation, we only get an estimate of the set of possible causal values. It was shown under some assumptions, that this estimate is consistent (see Pearl (1995)).

It has recently been shown empirically that despite the described fundamental limitations in identifying the causal effect uniquely, the method performs well in identifying the most important causal effects in a high-dimensional setting (see Pearl (2016)).

3. Package *pcalg*

This package has three goals. First, it is intended to provide fast, flexible and reliable implementations of the PC and the FCI algorithm. Secondly, it is intended to provide a tool for estimating the causal effect among continuous variables given a causal structure using a form of the do-calculus known as back-door criterion. Finally, combining these two applications of the package, it is possible to estimate causal effects when no causal structure is known. In the following, we describe the main functions of our package for achieving these goals. The functions `skeleton`, `pc` and `fci` are intended for estimating graphical models. The functions `ida` and `idaFast` are intended for estimating causal effects when the causal structure is known or was estimated.

3.1. `skeleton`

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

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

As was shown 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 independence tests and returns a p-value. All information that is needed in the independence test can be passed in the argument `suffStat`. The


```

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

```

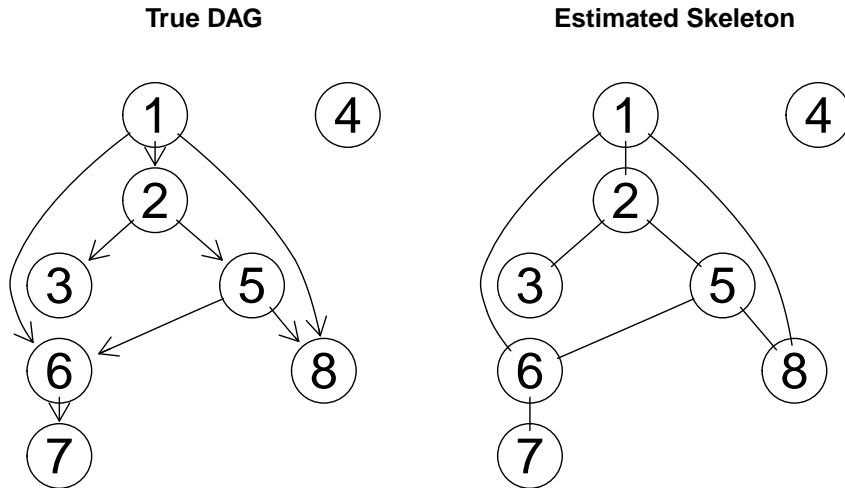


Figure 2: True DAG and estimated skeleton fitted on the Gaussian data set provided by the package using the function `skeleton`.

only exceptions are the number of variables p and the significance level α 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 (see `binCItest`). Each of these independence test functions need different arguments as input, which are described in the helpfiles of the functions. 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 `gaussianData` (which consists of $p = 8$ variables and $n = 5000$ samples) and plot the results. The estimated skeleton and the true underlying DAG are shown in Fig. 2.

To give another example, we show how to fit a skeleton to the example data set `gmD` (which consists of $p = 5$ discrete variables with 3, 2, 3, 4 and 2 levels and $n = 10000$ samples). The predefined test function `disCItest()` is based on the G^2 statistics 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 whether to lower the degrees of freedom by one for each zero count. Finally, we plot the result. The estimated skeleton and the true underlying DAG are shown in Fig. 3.

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 gaps are guaranteed to be 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 in the resulting graph.

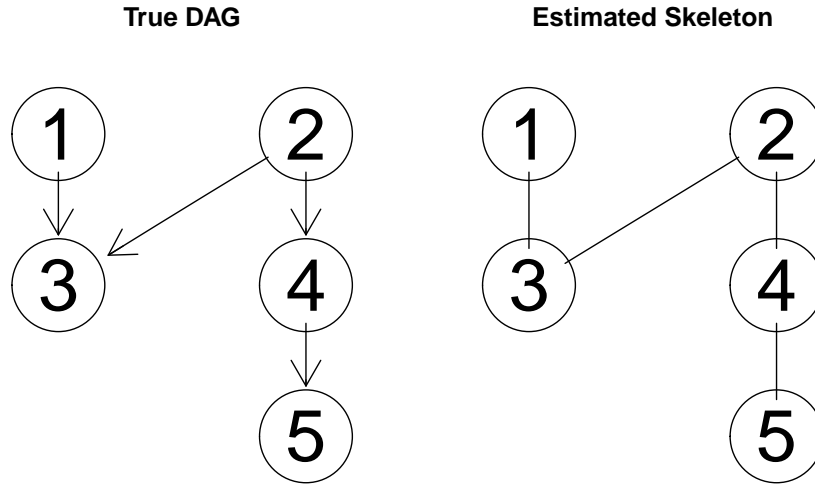


Figure 3: True DAG and estimated skeleton fitted on the discrete data set provided by the package using function `skeleton()`.

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 maximal size of the conditioning sets that are considered in the conditional independence tests in (P2) of Algorithm 1.

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

3.2. `pc`

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

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

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

Sampling errors (or hidden variables) can lead to conflicting information about edge directions. For example, one may find that $a - b - c$ and $b - c - d$ should both be 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 $b \rightarrow c \leftarrow d$. 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.

Sampling errors or hidden variables can also lead to invalid CPDAGs, meaning that there does not exist a DAG that has the same skeleton and v-structures as the graph found by the algorithm. An example of this is an undirected cycle consisting of the edges $a - b - c - d$ and $d - a$. In this case it is impossible to direct the edges without creating a cycle or a new v-structure. The option `u2pd` specifies what should be done in such a situation. If the

option is set to **"relaxed"**, the algorithm simply outputs the invalid CPDAG. If the option is set to **"rand"**, all direction information is discarded and a random DAG is generated on the skeleton. If the option 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 DAG is generated on the skeleton as in the option **"rand"**.

Using the option `conservative = TRUE`, the conservative PC algorithm done. The conservative PC is a slight variation of the PC algorithm and is intended to be more robust against sampling errors. After the skeleton is computed, all potential v-structures $a-b-c$ are checked in the following way. We test whether a and 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), no further action is taken and the usual PC is continued. If, however, b is in only some conditioning sets, the triple $a-b-c$ is marked 'unfaithful'. Moreover, if in the conservative step there is no subset among the neighbors that makes a and c independent, the triple is marked 'unfaithful'. An unfaithful 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. (For more detail, see the help file of the internal function `pc.cons.intern` which is called with the argument `version.unf = c(2,2)`.)

As with the skeleton, the 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 function `gaussCitest` as conditional independence test and create the corresponding test statistic. Finally, we plot the result. The estimated CPDAG and the true underlying DAG are shown in Fig. 4.

3.3. fci

This function is a generalization of the PC algorithm (see section 3.2), 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 (Fast Causal Inference algorithm) estimates the equivalence class of MAGs that describe the conditional independence relationships between the observed variables.

The first part of the FCI algorithm is analogous to the PC algorithm. It starts with a complete undirected graph and estimates an initial skeleton using the function `skeleton`. All edges of this skeleton are of the form $o-o$. Now, the v-structures are directed as in the PC algorithm. 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 $x-y$ should be removed. Therefore, the initial skeleton may contain some superfluous edges. These edges are removed in the next step of the algorithm. To decide whether edge $x-o-o-y$ should be removed, one computes `Possible-D-SEP(x)` and `Possible-D-SEP(y)` and performs conditional independence tests of x and y given all possible subsets of `Possible-D-SEP(x)` and of `Possible-D-SEP(y)` (see helpfile of function `pdsep`). Subsequently, all edges are transformed into $o-o$ again and the v-structures are determined (using information in `sepset`). Finally, as many as possible undetermined edge marks (o) are determined using (a subset of) the 10 orientation rules given by ?.

The function can be called with the following arguments:

```

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

```

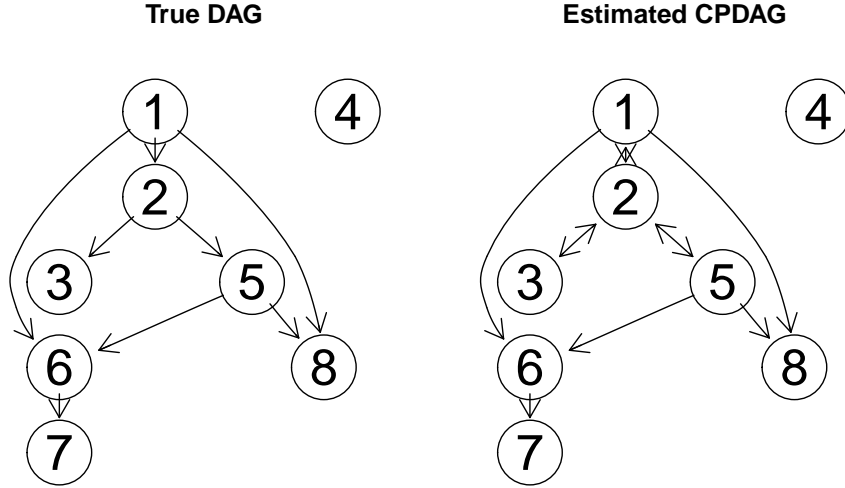


Figure 4: True DAG and estimated CPDAG fitted on the Gaussian data set provided by the package using the function `pc`.

```

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

```

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

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

Using the option `conservative` one can invoke the conservative FCI which consists of two parts. In the first part (done if `conservative[1]` is `TRUE`), we call the internal function `pc.cons.internal` with argument `version.unf = c(1,2)` after computing the skeleton. This is a slight variation of the conservative PC (which used `version.unf = c(2,2)`): If a is independent of c given some S in the skeleton (i.e., the edge $a - c$ dropped out), but a and c remain dependent given all subsets of neighbors of either a or c , we will call all triples $a - b - c$ 'faithful'. This is because in the FCI, the true separating set might be outside the neighborhood of either a or c . In the second part (done if `conservative[2]` is `TRUE`), we call `pc.cons.internal(..., version.unf = c(1,2))` again after Possible-D-Sep was found and the graph potentially lost some edges. Therefore, new triples might have occurred. If this second part is done, the resulting information on sepset and faithful triples overwrites the previous and will be used for the subsequent orientation rules.

```

> data(gmL)
> suffStat1 <- list(C = cor(gmL$x), n = nrow(gmL$x))
> pag.est <- fci(suffStat1, indepTest= gaussCIttest,
+               p = ncol(gmL$x), alpha = 0.01)
> par(mfrow = c(1,2))
> plot(gmL$g, main = "True DAG")
> plot(pag.est, main = "Estimated PAG")

```

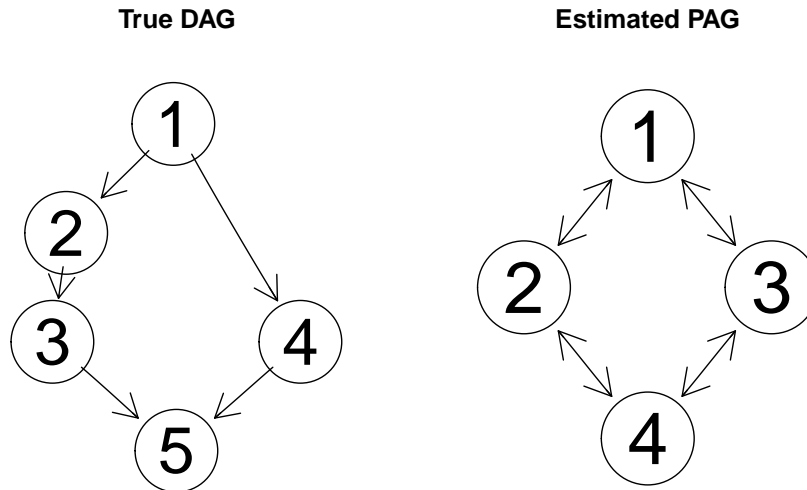


Figure 5: True data generating DAG and estimated PAG. Variable 1 of the data generating DAG is latent. One can see that the PAG was estimated correctly.

Using the option `biCC = TRUE` only nodes on paths between a and c are considered to be in `sepset(a,c)`. This method uses biconnected components to find all paths between nodes a and c .

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

As an example, we estimate the PAG of a graph with five nodes using the function `fci` and the predefined function `gaussCIttest` as conditional independence test. Finally, we plot the result. Node V_1 is latent. In Fig. 5 the true DAG and the estimated PAG are shown.

3.4. ida

It is assumed that we have observational data that are multivariate Gaussian and faithful to the true (but unknown) underlying causal DAG (without hidden variables). Under these assumptions, this function estimates the multiset of possible total causal effects of x on y , where the total causal effect is defined via Pearl's do-calculus as $\frac{\partial}{\partial z} E[Y|do(X = z)]$ (this value does not depend on z , since Gaussianity implies that conditional expectations are linear). We estimate a multiset of possible total causal effects instead of the unique total causal effect, since it is typically impossible to identify the latter when the true underlying causal DAG is unknown (even with an infinite amount of data).

To illustrate this, consider the following example. We have data from seven Gaussian variables

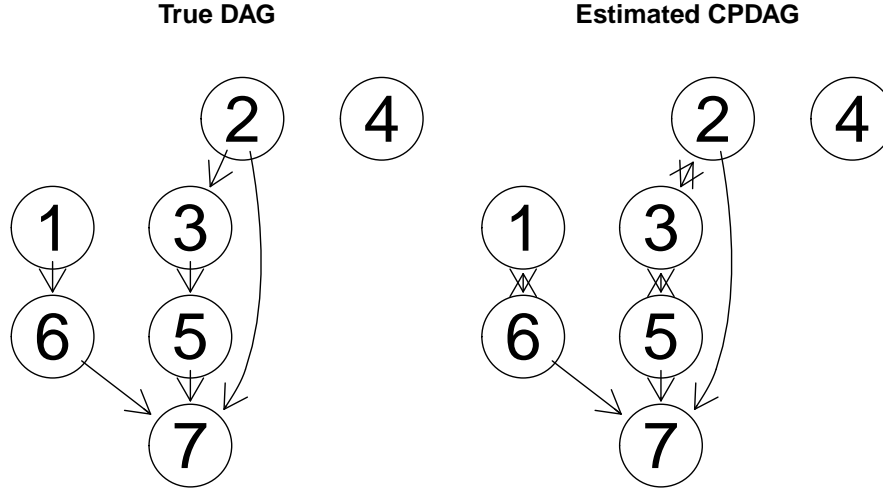


Figure 6: True DAG and estimated CPDAG for illustrating the function `ida`

with a causal structure given on the left of Fig. 6. Denote node i in the graph by V_i . 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)
> suffStat <- list(C = cor(gmI$x), n = nrow(gmI$x))
> pc.fit <- pc(suffStat, indepTest=gaussCIttest,
+             p = ncol(gmI$x), alpha = 0.01)
```

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

Since there are three undirected edges in the CPDAG, there might be up to $2^3 = 8$ DAGs in the corresponding equivalence class. However, the undirected edges $V_2 - V_3 - V_5$ can be formed to a new v-structures. As mentioned in section 2.1, DAGs in the equivalence class must have exactly the same v-structures as the corresponding CPDAG. Thus, $V_2 - V_3 - V_5$ can only be directed as $V_2 \rightarrow V_3 \rightarrow V_5$, $V_2 \leftarrow V_3 \leftarrow V_5$ or $V_2 \leftarrow V_3 \rightarrow V_5$ but not as $V_2 \rightarrow V_3 \leftarrow V_5$, since this would introduce a new colliding v-structure. There is only one remaining undirected edge ($V_1 - V_6$), which can be directed in two ways. Thus, there are six valid DAGs (i.e. they have no new v-structure and no cycle) and these form the equivalence class represented by the CPDAG. In Fig. 7, all DAGs in the equivalence class are shown. DAG 4 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 x on y . This can be done via a simple linear regression: if y is not a parent of x , we take the regression coefficient of x in the regression $\text{lm}(y \sim x + \text{pa}(x))$, where $\text{pa}(x)$ denotes the parents of x in the DAG G ; 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 will yield k estimated total causal effects.

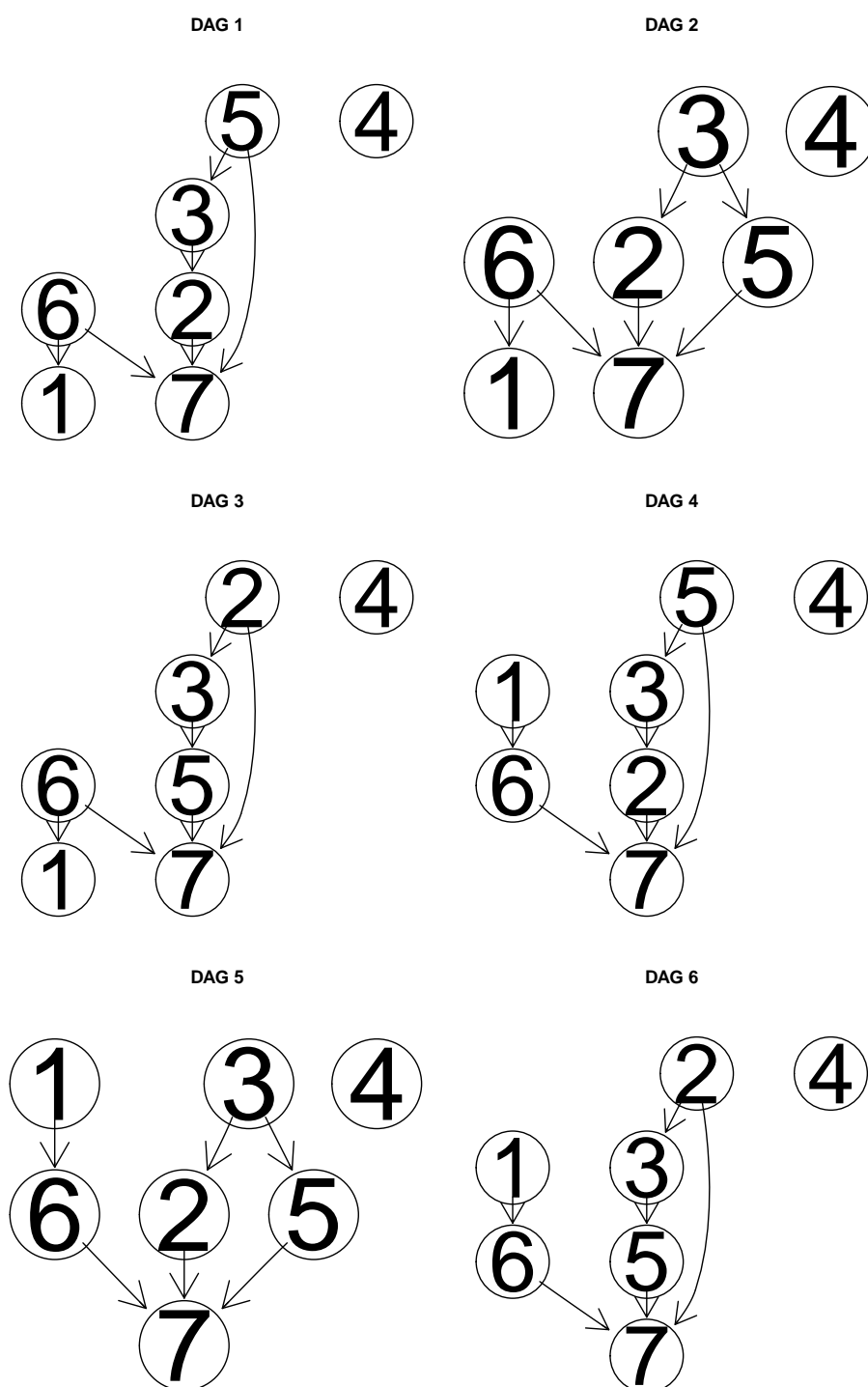


Figure 7: All DAGs belonging to the same equivalence class in which the true DAG shown in Fig. 6 is.

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

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

```
> ida(2, 5, cov(gmI$x), pc.fit@graph, method = "global", verbose = FALSE)
[1] -0.0049012 -0.0049012  0.5421360 -0.0049012 -0.0049012  0.5421360
```

Among these six values, there are only two unique values: -0.0049 and 0.5421 . This is because we compute $\text{lm}(V5 \sim V2 + \text{pa}(V2))$ (V_i denotes node i) for each DAG and report the regression coefficient of $V2$. Note that there are only two possible parent sets of $V2$ in all six DAGs: In DAGs 3 and 6, there are no parents of $V2$. In DAGs 1, 2, 4 and 5, however, the parent of $V2$ is $V3$. 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 two times, 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 $V2$ on $V5$ is 0.5529 . Thus, one of the two unique values is indeed very close to the true causal effect (the slight discrepancy is due to sampling error).

The function `ida` can be called as

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

where `x.pos` denotes the column position of the cause variable, `y.pos` denotes the column position of the effect variable, `mcov` is the covariance matrix of the original data, and `graphEst` is a graph object describing the causal structure (this could be given by experts or estimated using `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 x on 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, ? 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 (they need not always be one).

```
> ida(2,5, cov(gmI$x), pc.fit@graph, method = "local")
[1]  0.5421360 -0.0049012
```


One can take summary measures of the multiset. For example, the minimum absolute value provides a lower bound on the size of the true causal effect: if the minimum absolute value of all values in the multiset is larger than one, then we know that the size of the absolute 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 not an effect but it might be a cause. 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 can not 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 connected to the response variable (no matter whether directed or undirected) are overwritten so that they are directed towards the response variable at the end.

3.5. `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

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

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 3.4, where we computed the causal effect of `V2` on `V5`. Now, we want to compute the effect of `V2` on `V5`, `V6` and `V7` using `idaFast` and compare the results with the output of `ida`. As expected, both methods lead to the same results.

```
> data(gmI)
> suffStat <- list(C = cor(gmI$x), n = nrow(gmI$x))
> pc.fit <- pc(suffStat, indepTest=gaussCIttest, p=ncol(gmI$x), alpha = 0.01)
> (eff.est1 <- ida(2,5, cov(gmI$x), pc.fit@graph, method="local"))
```

```

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

```

3.6. 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** need a function of the form **myCItest**(*x*, *y*, *S*, *suffStat*) to test conditional independence relationships. This function must return the p-value of the conditional independence test of *x* and *y* given *S* given 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 *x* and *y* given *S* for some data is to solve the two associated linear regression problems $x \sim S$ and $y \sim S$, 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 will suffice to just pass the complete data set in **suffStat**. We will 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) {
    res <- cor.test(suffStat[,x],suffStat[,y])$p.value
  } else {
    resx <- resid(lm(suffStat[,x] ~ suffStat[,S]))
    resy <- resid(lm(suffStat[,y] ~ suffStat[,S]))
    res <- cor.test(resx, resy)$p.value
  }
  res
}

```

We can now use this function together with the **pc()** one. As expected, the resulting CPDAG (see Fig. 8) is the same as in section 3.2 where we used the function **gaussCItest** as conditional

```

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

```

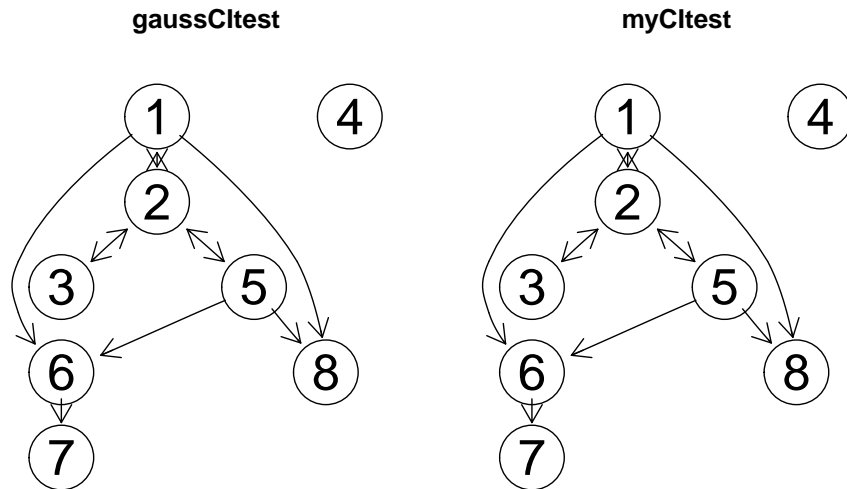


Figure 8: The estimated CPDAGs using the predefined and the user specified conditional independence test are identical.

independence test. Note, however, that using `gaussCitest` is considerably faster than using `myCitest` (on our computer 0.067 seconds using `gaussCitest` versus 4.448 seconds using `myCitest`).

4. Session Information

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

- R version 2.13.0 Under development (unstable) (2010-09-22 r52963),
x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=de_CH.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8,
LC_COLLATE=de_CH.UTF-8, LC_MONETARY=C, LC_MESSAGES=de_CH.UTF-8,
LC_PAPER=de_CH.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C,
LC_MEASUREMENT=de_CH.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: abind 1.1-0, colorspace 1.0-1, corpcor 1.5.7, graph 1.27.21,
MASS 7.3-7, pcalg 1.1-0, RBGL 1.25.1, Rgraphviz 1.27.0, sfsmisc 1.0-12, vcd 1.2-9
- Loaded via a namespace (and not attached): ggm 1.0.4, robustbase 0.5-0-1, tools 2.13.0

??? MENTION: S4-class `gAlgo`, `pcAlgo`, `fciAlgo` and inheritance.

Affiliation:

Markus Kalisch

Seminar für Statistik

ETH Zürich

8092 Zürich, Switzerland

E-mail: kalisch@stat.math.ethz.ch