Package ‘cfid’

June 10, 2021

Type Package

Title Identification of Counterfactual Queries in Causal Models

Version 0.1.2

Maintainer Santtu Tikka <santtuth@gmail.com>

Description Facilitates the identification of counterfactual queries in structural causal models via the ID* and IDC* algorithms by Shpitser, I. and Pearl, J. (2008) <https://jmlr.org/papers/v9/shpitser08a.html>. Provides a simple interface for defining causal graphs and counterfactual conjunctions. Construction of parallel worlds graphs and counterfactual graphs is done automatically based on the counterfactual query and the causal graph.

License GPL-3

Encoding UTF-8

URL https://github.com/santikka/cfid

RoxygenNote 7.1.1

Suggests covr, dagitty, igraph, mockery, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Santtu Tikka [aut, cre] (<https://orcid.org/0000-0003-4039-4342>)

Repository CRAN

Date/Publication 2021-06-10 07:40:02 UTC

R topics documented:

`cfid-package` *The 'cfid' package*

Description

Identification of Counterfactual Queries in Causal Models

Details

This package provides tools necessary for identifying counterfactual queries in causal models. Causal graphs, counterfactual variables, and counterfactual conjunctions are defined via a simple interface.

Counterfactuals::
In simple terms, counterfactual are statements involving multiple conceptual 'worlds' where the observed state of the worlds is different. As an example, consider two variables, Y = "headache", and X = "aspirin". A counterfactual statement could be "If I have a headache and did not take aspirin, would I not have a headache, had I taken aspirin instead". This statement involves two worlds: the real or actual world, where aspirin was not taken, and a hypothetical world, where it was taken. In more formal terms, this statement involves a counterfactual variable Y_x that attains two different values in two different worlds, forming a counterfactual conjunction: y_x AND $y'_{x'}$, where y and y' are two different values of Y, and x and x' are two different values of X.

Identifiability:
Pearl's ladder of causation consists of the associational, interventional and counterfactual levels, with counterfactual being the highest level. The goal of identification is to find a transformation from a higher level query into a lower level one. For the interventional case, this transformation is known as causal effect identifiability, where interventional distributions are expressed in terms of observational quantities. Tools for this type of identification are readily available, such as in causaleffect, dagitty, pcalg, and dosearch packages. Transformation from the highest counterfactual level, is more difficult, both conceptually and computationally, since to reach the observational level, we must first find a transformation of our counterfactual query into a interventional query, and then transfrom this yet again to observational. Also, there transformations may not always exist, for example in the presence of latent unobserved confounders, meaning that the queries are non-identifiable. This package deals with the first transformation, i.e., expressing the counterfactual queries in terms of interventional queries (and observational, when possible).

Algorithms:
Identification is carried out in terms of $P*$ and G, where $P*$ is the set of all observational and interventional distributions, and G is a directed acyclic graph (DAG) depicting the causal model in question (a causal graph for short). The goal is to transform a counterfactual probability into an expression which can be represented solely in terms of interventional distributions. Identification is carried out using the ID* and IDC* algorithms by Shpitser and Pearl (2008). These algorithms are sound and complete, meaning that their output is always correct, and in the case of a non-identifiable counterfactual, one can always construct a counterexample, witnessing non-identifiability.

Graphs::

The causal graph associated with the causal model is given via a simple text-based interface, similar to `dagitty::dagitty()`. Directed edges are given as `X -> Y`, and bidirected edges as X <-> Y, which is a shorthand notation for latent confounders. For more details on graph construction, see `dag()`.

Counterfactual variables and conjunctions:

Counterfactual variables are defined by their name, value and the conceptual world that they belong to. A world is defined by a unique set of actions (interventions) via the do-operator (Pearl, 2009). We can define the two counterfactual variables of the headache/aspirin example as follows:

```
cf(var = "Y", obs = 0, int = c(X = 0))
cf(var = "Y", obs = 1, int = c(X = 1))
```

Here, `var` defines the name of the variable, `obs` gives level the variable is assigned to (not the actual value), and `int` defines the vector of interventions that define the counterfactual world. For more details, see `CounterfactualVariable()`. Counterfactual conjunctions on the other hand, are simply counterfactual statements (variables) that are observed at the same time. For more details, see `CounterfactualConjunction()`.

For complete examples of identifiable counterfactual queries, see `identifiable()`, which is the main function of the package.

References

Pearl, J. (1995) Causal diagrams for empirical research. *Biometrika*, **82(4)**:669–688.

Pearl, J. (2009) *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition.

Shpitser, I. and Pearl, J. (2008). Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, **9(64)**:1941–1979.

`CounterfactualConjunction`

Counterfactual Conjunction

Description

Defines a conjunction of counterfactual statements (variables).

Usage

```
CounterfactualConjunction(...)

conj(...)
```

Arguments

`...`	Objects of class `CounterfactualVariable`.

Details

A counterfactual conjunction is a conjunction (or a set in some contexts) of counterfactual statements that are assumed to hold simultaneously.

For example, "The value of Y was observed to be y, and the value of Y was observed to be y' under the intervention $do(X = x)$" consists of two variables: variable Y without intervention, and Y under the intervention $do(X = x)$. Conjunctions can also be constructed via the alias `conj` or iteratively from `CounterfactualVariable` objects (see examples).

Value

An object of class `CounterfactualConjunction`.

See Also

`CounterfactualVariable()`

Examples

```
# The conjunction described in 'details'
v1 <- cf("Y", 0)
v2 <- cf("Y", 1, c("X" = 0))
c1 <- conj(v1, v2)

# Alternative construction
c1 <- v1 + v2

# Adding further variables
v3 <- cf("X", 1)
c2 <- c1 + v3

# A specific variable (a unique combination of `var` and `int`)
# can only appear once in a given conjunction,
# otherwise the conjunction would be trivially inconsistent
v4 <- cf("Y", 0, c("X" = 0))
v5 <- cf("Y", 1, c("X" = 0))
c3 <- try(conj(v4, v5))
```

`CounterfactualVariable`

Counterfactual Variable

Description

Defines a counterfactual variable y_x.

Usage

```
CounterfactualVariable(var, obs = integer(0), int = integer(0))

cf(var, obs = integer(0), int = integer(0))
```

Arguments

`var`	A character vector of length one naming the variable (i.e., Y).
`obs`	An integer vector of length one or zero. If given, denotes the observed value of `var` (i.e., $Y = 0$)
`int`	A named integer vector where the names correspond to the variables intervened on (i.e., X) and values to the value assignments (their levels).

Details

Assume that Y is a single variable and X is a vector of variables. Here, The notation y_x means that the variable Y (`var`) attains the value y (`obs`) under the intervention $do(X = x)$ (`int`).

Note that different values of `obs` for a two variables with the same `var` and the same `int` do not denote their actual values, but the levels (i.e., `obs = 0` is different from `obs = 1`, but the variables do not actually attain values 0 and 1). For more information about the do-operator, see Pearl (2009). The shortcut alias `cf` can also be used to construct counterfactual variables variables.

Value

An object of class `CounterfactualVariable`.

References

Pearl, J. (2009) *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition.

See Also

`CounterfactualConjunction()`

Examples

```
# Y without an assigned value or any interventions
cf("Y")

# Y with a value assignment y, but no interventions
cf("Y", 0)

# Y with a different value y', but no interventions
cf("Y", 1)

# Y with the same value as the previous under the intervention do(X = x)
cf("Y", 1, c("X" = 0))
```

```
# Y with yet another value y'', under the intervention
# do(X = x', Z = z), i.e., the intervention on X has a different value
# than the previous (x != x') and Z is also assigned the value z
cf("Y", 2, c("X" = 1, "Z" = 0))
```

---

dag *Directed Acyclic Graph*

Description

Define a directed acyclic graph (DAG).

Usage

```
dag(x)
```

Arguments

x	A character string containing a sequence of definitions of edges in the form `X -> Y`, `X <-Y` or X <-> Y. See details for more advanced constructs.

Details

The syntax for `x` follows closely that of `dagitty::dagitty()` for compatibility. The resulting adjacency matrix of the definition is checked for cycles.

Directed edges are defined as `X -> Y` meaning that there is an edge from `X` to `Y` in the graph. Edges can be combined in sequence to create paths for concise descriptions, for example `X -> Y <-Z -> W`.

Unobserved latent confounders are defined using bidirected edges as X <-> Y which means that there is an additional variable `U[X,Y]` in the graph, and the edges `X <-U[X,Y] -> Y`, respectively.

Groups of vertices can be defined by enclosing the vertices within curly braces. For example X -> {Y Z} defines that the dag has an edge from `X` to both `Y` and `Z`. Different statements in `x` are automatically distinguished from one another without any additional delimiters, but semicolons, commas and line breaks can be used if desired.

Note that in the context of this package, vertex labels will always be converted into upper case, meaning that typing `Z` or `z` within `x` will always represent the same variable. This is done to enforce the notation of counterfactual variables, where capital letter denote variables, and small letters denote their value assignments.

Value

An object of class `dag`, which is a square adjacency matrix with the following attributes:

- `labels` A character vector (or a list) of vertex labels.
- `latent` A logical vector indicating latent variables.
- `order` An integer vector giving a topological order for the vertices. .

Examples

```
dag("x -> {y z} <- w <-> g")

# Groups can appear on both sides of an edge
dag("{x z} -> {y w}")

# Semicolons can be used to distinguish individual statements
dag("x -> z -> y; x <-> y")

# Commas can be used to distinguish variables for example
dag("{x, y, z} -> w")

# Line breaks are also allowed
dag("z -> w
     x -> y")
```

`export_graph` *Export Graph*

Description

Convert a valid graph object to a supported external format.

Usage

```
export_graph(
  g,
  type = c("dagitty", "causaleffect", "dosearch"),
  use_bidirected = TRUE,
  ...
)
```

Arguments

`g`	An object of class `DAG`.
`type`	A character string matching one of the following: "dagitty", "causaleffect" or "dosearch".
`use_bidirected`	A logical value indicating if bidirected edges should be used. If `TRUE`, the result will have explicit X <-> Y type edges. If `FALSE`, an explicit latent variable `X <-U[X,Y] -> Y` will be used instead (only applicable `type` is `"dosearch"`).
`...`	Additional arguments passed to `format` for formatting vertex labels.

Value

If `type` is `"dagitty"`, returns a `dagitty` object. If `type` is `"causaleffect"`, return an `igraph` graph, with its edge attributes set according to the causaleffect package syntax. If `type` is `"dosearch"`, returns a character vector of length one that describes `g` in the dosearch package syntax.

`identifiable` *Identify a Counterfactual Query*

Description

Determine the identifiability of a (conditional) counterfactual conjunction.

Usage

```
identifiable(g, gamma, delta = NULL)
```

Arguments

`g`	A `DAG` object describing the causal graph (or an object that can be coerced, see `import_graph()`.
`gamma`	An object of class `CounterfactualConjunction` representing the counterfactual causal query.
`delta`	An object of class `CounterfactualConjunction` representing the conditioning conjunction (optional).

Details

To identify a non-conditional conjunction $p(\gamma)$, the argument `delta` should be `NULL`.

To identify a conditional conjunction $p(\gamma|\delta)$, both `gamma` and `delta` should be specified.

First, a parallel worlds graph is constructed based on the query. In a parallel worlds graph, for each do-action that appears in $gamma$ (and $delta$) a copy of the original graph is created with the new observational variables attaining their post-interventional values but sharing the latent variables. From this graph, a counterfactual graph is derived, where each variable is unique, which might not be the case in a parallel worlds graph.

Finally, the ID* (or IDC*) algorithm is applied to determine identifiability of the query. Similar to the ID and IDC algorithms for causal effects, these algorithms exploit the so called c-component factorization to split the query into smaller subproblems, which are then solved recursively.

Value

A list containing one or more of the following:

- `id` A logical value that is `TRUE` if the query is identifiable and `FALSE` otherwise. Note that in cases where `gamma` is itself inconsistent, the query will be identifiable, but with probability 0.
- `prob` An object of class `Probability` giving the formula of the query in LaTeX syntax via format or print, if identifiable. This expression is given in terms of $P*$, the set of all interventional distributions over g. For tautological statements, the resulting probability is 1, and for inconsistent statements, the resulting probability is 0.
- `undefined` A logical value that is `TRUE` if a conditional conjunction $p(\gamma|\delta)$ is undefined, for example when $p(\delta) = 0$, and `FALSE` otherwise.

References

Shpister, I. and Pearl, J. (2006). Identification of joint interventional distributions in semi-Markovian causal models. In *21st National Conference on Artificial Intelligence*

Shpitser, I. and Pearl, J. (2008). Complete identification methods for the causal hierarchy. *Journal of Machine Learning Research*, **9(64)**:1941–1979.

See Also

`dag()`, `CounterfactualVariable()`, `CounterfactualConjunction()`

Examples

```
# Examples that appears in Shpitser and Pearl (2008)
g1 <- dag("X -> W -> Y <- Z <- D X <-> Y")
g2 <- dag("X -> W -> Y <- Z <- D X <-> Y X -> Y")
v1 <- cf("Y", 0, c(X = 0))
v2 <- cf("X", 1)
v3 <- cf("Z", 0, c(D = 0))
v4 <- cf("D", 0)
c1 <- conj(v1)
c2 <- conj(v2, v3, v4)
c3 <- conj(v1, v2, v3, v4)

# Identifiable conditional conjunction
identifiable(g1, c1, c2)

# Identifiable conjunction
identifiable(g1, c3)

# Non-identifiable conjunction
identifiable(g2, c3)
```

`import_graph` *Import Graph*

Description

Import and construct a valid DAG from an external format.

Usage

```
import_graph(x)
```

Arguments

`x`	A graph object in a valid external format, see details.

Details

Argument x accepts `dagitty::dagitty()` graphs, igraph-graphs in the `causaleffect::causaleffect()` package syntax and character strings in the `dosearch::dosearch()` package syntax.

Value

A `DAG` object if successful.

`Probability` *Probability*

Description

Defines an interventional or observational probability $p(y|do(x))$.

Usage

```
Probability(
  val = NULL,
  var = NULL,
  do = NULL,
  sumset = NULL,
  summand = NULL,
  terms = NULL,
  numerator = NULL,
  denominator = NULL
)
```

Arguments

`val`	An integer value of either 0 or 1 for almost sure events.
`var`	A list of objects of class `CounterfactualVariable` (without interventions and with value assignments). `var` defines the observations y in $p(y|...)$.
`do`	A list of objects of class `CounterfactualVariable` (without interventions and with value assignments). If an interventional probability is defined, these depict the $do(.)$ variables.
`sumset`	A list of objects of class `CounterfactualVariable` (without interventions and with value assignments). If the probability depicts marginalization, `sumset` defines the set of variable to be marginalized over.
`summand`	An object of class `Probability`. If `sumset` is not `NULL`, this defines the probability being marginalized.
`terms`	A list of `Probability` objects if the object in question is meant to represent a product of terms.
`numerator`	An object of class `Probability`. If the probability depicts a conditional probability that cannot be expressed simply in terms of the set of inputs $P*$, this is the numerator of the quotient representation.

`denominator`	An object of class `Probability`. The denominator of the quotient representation.

Details

When formatted via `print` or `format`, the arguments are prioritized in the following order if conflicting definitions are given: `val`, (`var`, `do`), (`sumset`, `summand`), `terms`, (`numerator`, `denominator`)

Value

An object of class `Probability`.

Index