# Package 'cfid'

October 26, 2022

**Type** Package

**Title** Identification of Counterfactual Queries in Causal Models

**Version** 0.1.4

**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. (2007, 2008) <arXiv:1206.5294>, <https://jmlr.org/papers/v9/shpitser08a.html>. Provides a simple interface for defining causal diagrams and counterfactual conjunctions. Construction of parallel worlds graphs and counterfactual graphs is carried out automatically based on the counterfactual query and the causal diagram.

**License** GPL (>= 3)

**Encoding** UTF-8

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

**RoxygenNote** 7.2.1

**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** 2022-10-26 15:35:07 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 simple interfaces.

## 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: $y_x \wedge y'_{x'}$, 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 of 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 the `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 transform 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), as well as the second one, transforming interventional distributions to observational quantities.

## Algorithms

Identification is carried out in terms of $G$ and $P_*$ and where $G$ is a directed acyclic graph (DAG) depicting the causal model in question (a causal graph for short), and $P_*$ is the set of all interventional distributions in causal models inducing $G$. Identification is carried out by the ID* and IDC* algorithms by Shpitser and Pearl (2008) which aim to convenrt the input counterfactual probability into an expression which can be represented solely in terms of interventional distributions. 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` package syntax. 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, sub = c(X = 0))
cf(var = "Y", obs = 1, sub = c(X = 1))
```

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

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. (2007). What counterfactuals can be tested. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, 352–359.

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

Makhlouf, K., Zhioua, S. and Palamidessi, C. (2021). Survey on causal-based machine learning fairness notions. *arXiv:2010.09553*

`causal_effect` *Causal Effect Identification*

## Description

Identify a causal effect of the form $P(y|do(x), z)$ from $P(v)$ in $G$.

## Usage

```
causal_effect(g, y, x = character(0), z = character(0), v = integer(0))
```

## Arguments

| | |
|---|---|
| `g` | A `dag` object depicting the causal diagram $G$. |
| `y` | A `character` vector of response variables $Y$. |
| `x` | A `character` vector of intervention variables $X$. |
| `z` | An optional `character` vector of conditioning variables $Z$. |
| `v` | An optional named `integer` vector giving the value assignments for observed variables in the model, i.e. $V = v$ or for a subset of $V$. |

## Value

An object of class `query` which is a `list` with the following components:

- `id`
  A `logical` value that is `TRUE` if the query is identifiable and `FALSE` otherwise.
- `formula`
  A `functional` object expressing the causal effect in terms of the joint probability distribution $P(v)$ for identifiable queries or `NULL` if the query is not identifiable.
- `data`
  The available data, for `causal_effect` this is always `"observations"`
- `causaleffect`
  The original query $P(y|do(x), z)$ as a `probability` object.
- `undefined`
  A `logical` value, this is always `FALSE` for `causaleffect`

counterfactual_conjunction *Counterfactual Conjunction*

## Description

conj defines a conjunction of counterfactual statements (variables).

cf defines a counterfactual variable $y_x$.

## Usage

```
counterfactual_conjunction(...)

## S3 method for class 'counterfactual_conjunction'
format(x, var_sep = " /\\ ", ...)

## S3 method for class 'counterfactual_conjunction'
print(x, ...)

## S3 method for class 'counterfactual_conjunction'
e1 + e2

## S3 method for class 'counterfactual_conjunction'
x[i]

## S3 method for class 'counterfactual_variable'
e1 + e2

conj(...)

counterfactual_variable(var, obs = integer(0L), sub = integer(0L))

## S3 method for class 'counterfactual_variable'
format(x, use_primes = TRUE, ...)

## S3 method for class 'counterfactual_variable'
print(x, ...)

cf(var, obs = integer(0L), sub = integer(0L))
```

## Arguments

| | |
|---|---|
| ... | Additional arguments passed to format.counterfactual_variable(). |
| x | A counterfactual_variable or a counterfactual_conjunction object. |
| var_sep | A character string to separate counterfactual variables. |
| e1 | A counterfactual_variable or a counterfactual_conjunction object. |

| | |
|---|---|
| e2 | A counterfactual_variable or a counterfactual_conjunction object. |
| i | An integer index vector. |
| 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., $Y = y$) |
| sub | A named integer vector where the names correspond to the variables intervened on (via $do(X = x)$) and values to the value assignments (their levels, e.g., $x$). |
| use_primes | A logical value indicating whether primes should be used to differentiate between value assignments |

## Value

conj returns an object of class counterfactual_conjunction.

cf returns an object of class counterfactual_variable.

## Counterfactual Conjunctions

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

For example, the statement "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 $do(X = x)$ (which is $Y_x$). This conjunction can be succinctly written as $y \wedge y'_x$.

Conjunctions can also be constructed via the alias conj or iteratively from counterfactual_variable objects (see examples).

## Counterfactual Variables

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 $do(X = x)$ (sub).

Note that different values of obs for a two variables with the same var and the same sub 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). In other words, if var is different for two counterfactual variables, but they have the same value obs, this does not mean that these variables have the same value. They will only actually have the same value if they share both var and obs.

For more information about the $do$-operator, see Pearl (2009). The shortcut alias cf can also be used to construct counterfactual variables.

## Examples

```
# The conjunction described under '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 value of a variable (a unique combination of `var` and `sub`)
# 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))

# 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) describing the causal model.

## Usage

```
dag(x, u = character(0L))

## S3 method for class 'dag'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | A dag object |
| u | A `character` vector of variable names which should be considered unobserved (besides those defined by bidirected edges). These variables are subsequently removed via latent projection. Variable names not found in the graph are ignored. |

... Not used

## Details

The syntax for x follows the dagitty package closely for compatibility. However, not all features of dagitty graphs are supported. 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.

Different statements in x can be distinguished from one another using either semicolons, line breaks, or spaces.

Subgraphs can be defined by enclosing the definition within curly braces. For example `X -> {Y Z}` defines an edge from X to both Y and Z. Individual statements within a subgraph can be separated by a comma or semicolon, but this is optional. Edges can also be defined within subgraphs, and subgraphs can be nested. For example, `X -> {Z -> Y}` is the same definition as `X -> Z; X -> Y; Z -> Y`. Similarly `X <-> {Z -> {A B}} -> Y` is the same as `X <-> {Z A B} -> Y; Z -> {A B}`.

Note that in the context of this package, vertex labels will always be converted into upper case, meaning that typing Z or z will always represent the same variable. This is done to enforce the notation of counterfactual variables, where capital letters 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.
- `text`
  A `character` string giving representing the DAG. .

## Examples

```
dag("X -> {Y Z} <- W <-> G")

# Subgraphs 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 within groups if there
# are no edges within the group
```

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

# Edges within subgraphs is supported
dag("{X -> Z} -> {Y <-> W}")

# Nested subgraphs are supported
dag("{X -> {Z -> {Y <-> W}}}")

# Line breaks are also supported for statement separation
dag("
  Z -> W
  X -> Y
")
```

---

`export_graph` *Export Graph*

---

## Description

Convert a valid graph object into 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"`. For `"dagitty"` and `"causaleffect"`, the packages `dagitty` and `igraph` must be available, respectively. |
| `use_bidirected` | A logical value indicating if bidirected edges should be used in the resulting object. If `TRUE`, the result will have explicit `X <-> Y` edges. If `FALSE`, an explicit latent variable `X <- U[X,Y] -> Y` will be used instead (only applicable if `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"`, returns 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.

`functional` *Identifying Functional of a Counterfactual Query*

## Description

Identifying functionals are more complicated probabilistic expressions that cannot be expressed as simple observational or interventional

## Usage

```
functional(sumset = NULL, terms = NULL, numerator = NULL, denominator = NULL)

## S3 method for class 'functional'
format(x, use_primes = TRUE, use_do = FALSE, ...)

## S3 method for class 'functional'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `sumset` | A `list` of objects of class `counterfactual_variable` (without interventions and with value assignments). If the probability depicts marginalization, `sumset` defines the set of variables to be marginalized over. |
| `terms` | A `list` of `functional` objects if the object in question is meant to represent a product of terms. |
| `numerator` | A `functional` or a `probability` object. If the functional represents a conditional probability that cannot be expressed simply in terms of the set of inputs, this is the numerator of the quotient representation. |
| `denominator` | A `functional` or a `probability` object. The denominator of the quotient representation. |
| `x` | A `functional` object. |
| `use_primes` | A `logical` value. If `TRUE` (the default), any value assignment of a counterfactual variable with `obs` will be formatted with as many primes in the superscript as the value of `obs`, e.g., `obs = 0` outputs `"y"`, `obs = 1` outputs `"y'"`, `obs = 2` outputs `"y''"` and so forth. The alternative when `FALSE` is to simply denote the `obs` value via superscript directly as `"y^{(obs)}"`, where obs is evaluated. |
| `use_do` | A `logical` value. If `TRUE`, the explicit do-operation is used to denote interventional probabilities (e.g., $P(y\|do(x))$). If `FALSE` (the default), the subscript notation is used instead (e.g., $P_x(y)$). |
| `...` | Additional arguments passed to `format`. |

## Details

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

### Value

An object of class `functional`, which is a `list` containing all of the arguments of the constructor.

A `character` representation of the `functional` object in LaTeX syntax.

---

`identifiable` *Identify a Counterfactual Query*

---

### Description

Determine the identifiability of a (conditional) counterfactual conjunction.

### Usage

```
identifiable(
  g,
  gamma,
  delta = NULL,
  data = c("interventions", "observations", "both")
)
```

### Arguments

| | |
|---|---|
| `g` | A `dag` object describing the causal graph (to obtain a `dag` from another format, see `import_graph()`. |
| `gamma` | An R object that can be coerced into a `counterfactual_conjunction` object that represents the counterfactual causal query. |
| `delta` | An R object that can be coerced into a `counterfactual_conjunction` object that represents the conditioning conjunction (optional). |
| `data` | A `character` string that accepts one of the following: `"interventions"` (the default), `"observations"`or `"both"`. This argument defines the target level of identification. If `"interventions"` is used, the identification is attempted down to the intervention level. If `"observations"` is used, identification is attempted down to the observational level. If `"both"` is used, identification is carried out for each term to the lowest level where the term is still identifiable. |

### 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. This graph is known as a parallel worlds graph. From the parallel worlds graph, a counterfactual graph is derived such that 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. If argument `data` is `"observations"` or `"both"`, identification of interventional probabilities in the resulting functional is further attempted in terms of the joint probability distribution by using the ID and IDC algorithms (see causal_effect).

## Value

An object of class `query` which is a `list` containing one or more of the following:

- `id`
  A `logical` value that is `TRUE` if the query is identifiable and `FALSE` otherwise from the available `data` in `g`. Note that in cases where `gamma` itself is inconsistent, the query will be identifiable, but with probability 0.
- `formula`
  An object of class `functional` giving the identifying functional of the query in LaTeX syntax via `format` or `print`, if identifiable. This expression is given in terms of the available `data`. For tautological statements, the resulting probability is 1, and for inconsistent statements, the resulting probability is 0. For formatting options, see `format.functional()` and `format.probability()`.
- `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.
- `gamma`
  The original counterfactual conjunction..
- `delta`
  The original conditioning counterfactual conjunction.
- `data`
  The original data.

## See Also

`dag()`, `counterfactual_variable()`, `probability()`, `functional()`

## 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. Accepts `dagitty` graphs, `igraph` graphs in the `causaleffect` package syntax, and character strings in the `dosearch` package syntax.

## Usage

```
import_graph(x)
```

## Arguments

| | |
|---|---|
| `x` | A graph object in a valid external format. |

## Value

A dag object.

---

`probability` *Symbolic Probability Distributions*

---

## Description

Defines an interventional or observational (conditional) probability $P(y|do(x), z)$. For formatting options, see `format.probability()`.

## Usage

```
probability(val = NULL, var = NULL, do = NULL, cond = NULL)

## S3 method for class 'probability'
format(x, use_primes = TRUE, use_do = FALSE, ...)

## S3 method for class 'probability'
print(x, ...)
```

## Arguments

| | |
|---|---|
| val | An integer value of either 0 or 1 for almost sure events. |
| var | A list of objects of class counterfactual_variable (without interventions and with value assignments). var defines the observations $y$ in $P(y\|do(x), z)$. |
| do | A list of counterfactual_variable variable objects (without interventions and with value assignments). Defines the intervention set $x$ in $P(y\|do(x), z)$. |
| cond | A list of counterfactual_variable variable objects (without interventions and with value assignments).Defines the conditioning set $z$ in $P(y\|do(x), z)$. |
| x | A probability object. |
| use_primes | A logical value. If TRUE (the default), any value assignment of a counterfactual variable with obs will be formatted with as many primes in the superscript as the value of obs, e.g., obs = 0 outputs "y", obs = 1 outputs "y'", obs = 2 outputs "y''" and so forth. The alternative when FALSE is to simply denote the obs value via superscript directly as "y^{(obs)}", where obs is evaluated. |
| use_do | A logical value. If TRUE, the explicit do-operation is used to denote interventional probabilities (e.g., $P(y\|do(x))$). If FALSE (the default), the subscript notation is used instead (e.g., $P_x(y)$). |
| ... | Additional arguments passed to format. |

## Value

An object of class probability, which is a list containing all of the arguments of the constructor.

A character representation of the probability object in LaTeX syntax.

## See Also

counterfactual_variable(), functional()

## Examples

```
# Example from Makhlouf, Zhioua and Palamidessi (2021)
g2 <- dag("C -> A -> Y; C -> Y")
v1 <- cf("Y", 0, c(A = 1))
v2 <- cf("A", 0)
c1 <- conj(v1)
c2 <- conj(v2)
f <- identifiable(g2, c1, c2)$formula

# Default, using primes and subscript notation
format(f)

# Without primes, no do-operator
format(f, use_primes = FALSE)

# Primes, with do-operator
format(f, use_do = TRUE)

# Without primes, with do-operator
```

```
format(f, use_primes = FALSE, use_do = TRUE)
```

---

query *Query Objects*

---

## Description

Objects of class query describe the output of identifiable and causal_effect. They are list objects with a custom print method and contain data related to the identifiability results. See identifiable and causal_effect for details.

## Usage

```
## S3 method for class 'query'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | A query object |
| ... | Arguments passed to format.functional and format.counterfactual_conjunction |

# Index