Package: dosearch (via r-universe)

September 14, 2024

Title Causal Effect Identification from Multiple Incomplete Data Sources

Version 1.0.11

Description Identification of causal effects from arbitrary observational and experimental probability distributions via do-calculus and standard probability manipulations using a search-based algorithm by Tikka, Hyttinen and Karvanen (2021) <doi:10.18637/jss.v099.i05>. Allows for the presence of mechanisms related to selection bias (Bareinboim and Tian, 2015) <doi:10.1609/aaai.v29i1.9679>, transportability (Bareinboim and Pearl, 2014) <http://ftp.cs.ucla.edu/pub/stat_ser/r443.pdf>, missing data (Mohan, Pearl, and Tian, 2013) <http://ftp.cs.ucla.edu/pub/stat_ser/r410.pdf>) and arbitrary combinations of these. Also supports identification in the presence of context-specific independence (CSI) relations through labeled directed acyclic graphs (LDAG). For details on CSIs see (Corander et al., 2019) <doi:10.1016/j.apal.2019.04.004>.

License GPL (>= 3)

Depends R (>= 4.0)

Suggests covr, dagitty, DiagrammeR, DOT, igraph, knitr, mockr, rmarkdown, testthat (>= 3.0.0)

Imports Rcpp

LinkingTo Rcpp

VignetteBuilder knitr

BugReports https://github.com/santikka/dosearch/issues

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

Encoding UTF-8

NeedsCompilation yes

Roxygen list(markdown = TRUE, roclets = c(``namespace", ``rd", ``srr::srr_stats_roclet"))

RoxygenNote 7.3.2
Config/testthat/edition 3
LazyData true
Repository https://santikka.r-universe.dev
RemoteUrl https://github.com/santikka/dosearch
RemoteRef HEAD
RemoteSha bfebae71592361a28cc8ade1f7947df755566711

Contents

`dosearch-package` *Causal Effect Identification from Multiple Incomplete Data Sources*

Description

Solves causal effect identifiability problems from arbitrary observational and experimental distributions using a heuristic search. Allows for the presence of advanced data-generating mechanisms. See Tikka et al. (2021) doi:10.18637/jss.v099.i05 for further details.

See also

- The package vignette.
- `dosearch()` for instructions and various examples.
- https://github.com/santikka/dosearch/issues/ to submit a bug report.

Author(s)

Santtu Tikka, Antti Hyttinen, Juha Karvanen

References

S. Tikka, A. Hyttinen and J. Karvanen. "Causal effect identification from multiple incomplete data sources: a general search-based approach." *Journal of Statistical Software*, 99(5):1–40, 2021.

See Also

Useful links:

- https://github.com/santikka/dosearch
- Report bugs at https://github.com/santikka/dosearch/issues

bivariate_missingness *Systematic Analysis of Bivariate Missing Data Problems*

Description

This data set contains the results of a systematic analysis of all missing data problems of two variables. Each problem is associated with a graph containing two vertices, X and Y, and their response indicators, R_X and R_Y.

Usage

```
data(bivariate_missingness)
```

Format

A data frame with 6144 rows and 8 variables:

graph the graph of the instance.

nedges number of edges in the graph (directed and bidirected).

arrowXtoY whether the graph contains an arrow from X to Y or not.

jointXY identifiability of the joint distribution of X and Y

marginX identifiability of the marginal distribution of X.

marginY identifiability of the marginal distribution of Y.

YcondX identifiability of the conditional distribution of Y given X.

YdoX identifiability of the causal effect of X on Y.

Source

Tikka et al. https://arxiv.org/abs/1902.01073

dosearch *Identify a Causal Effect from Arbitrary Experiments And Observations*

Description

Identify a causal `query` from available `data` in a semi-Markovian causal model described by a `graph` that is a directed acyclic graph (DAG) or a labeled directed acyclic graph (LDAG). In a semi-Markovian causal model, each unobserved variable has exactly two children and they are denoted by bidirected edges. For DAGs, special mechanisms related to transportability of causal effects, recoverability from selection bias and identifiability under missing data can also be included. See 'Details' for the syntax of each argument. Note that all `character` type arguments are case-sensitive.

`is_identifiable` returns the a logical value describing the identifiability of a causal query of an object of class `dosearch`.

`get_formula` returns the identifying formula describing a causal query of an object of class `dosearch`. If no formula is available, returns `NULL`.

`get_derivation` returns the derivation of a causal query of an object of class `dosearch`. If no derivation is available, returns `NULL`.

`get_benchmark` returns the benchmarking information of an object of class `dosearch`. If no benchmark is available, returns `NULL`.

Usage

```
dosearch(
  data,
  query,
  graph,
  transportability = NULL,
  selection_bias = NULL,
  missing_data = NULL,
  control = list()
)

## S3 method for class 'dosearch'
summary(object, ...)

## S3 method for class 'dosearch'
plot(x, ...)

## S3 method for class 'dosearch'
print(x, max_chars = 300L, ...)

is_identifiable(x)

get_formula(x, run_again = FALSE)

get_derivation(x, run_again = FALSE, draw_all = FALSE)

get_benchmark(x, run_again = FALSE, include_rules = FALSE)
```

Arguments

`data`	A `character` string describing the available distributions in the package syntax. Alternatively, a list of character vectors.
`query`	A `character` string describing the target distribution in the package syntax. Alternatively, a character vector.
`graph`	A `character` string describing either a DAG or an LDAG in the package syntax. Alternatively, an **igraph** graph as used in the **causaleffect** package or a DAG constructed using the **dagitty** package.

`transportability`
: A `character` string describing the transportability nodes of the model in the package syntax (for DAGs only).

`selection_bias`
: A `character` string describing the selection bias nodes of the model in the package syntax (for DAGs only).

`missing_data`
: A `character` string describing the missing data mechanisms of the model in the package syntax (for DAGs only).

`control`
: A `list` of control parameters.

`object`
: An object of class `dosearch`.

`...`
: Additional arguments passed to `base::format()`.

`x`
: An object of class `dosearch`.

`max_chars`
: Maximum number of characters of the formula to display. The default is 300.

`run_again`
: If `TRUE`, runs the search again in an attempt to obtain the formula, for example if `control$formula` was `FALSE` in the call to `dosearch()`, but the query itself is identifiable.

`draw_all`
: A logical value. If `TRUE`, the derivation will contain every step taken by the search. If `FALSE`, only steps that resulted in identification are returned.

`include_rules`
: A `logical` value. If `TRUE`, also benchmarks the time taken by each inference rule separately.

Details

Argument `data` is used to list the available input distributions. When `graph` is a DAG the distributions should be of the form

$$P(A_i|do(B_i), C_i)$$

Individual variables within sets should be separated by a comma. For example, three input distributions:

$$P(Z|do(X)), P(W, Y|do(Z, X)), P(W, Y, X|Z)$$

should be given as follows:

```
> data <- "
+  P(Z|do(X))
+  P(W,Y|do(Z,X))
+  P(W,Y,X|Z)
+"
```

The use of multiple do-operators is not permitted. Furthermore, when both conditioning variables and a do-operator are present, every conditioning variable must either precede the do-operator or follow it. When `graph` is an LDAG, the do-operation is represented by an intervention node, i.e.,

$$P(Y|do(X), Z) = P(Y|X, Z, I_X = 1)$$

For example, in the case of the previous example in an LDAG, the three input distributions become:

```
> data <- "
+  P(Z|X,I_X = 1)
+  P(W,Y|Z,X,I_X=1,I_Z=1)
+  P(W,Y,X|Z)
+"
```

The intervention nodes I_X and I_Z must be explicitly defined in the `graph` along with the relevant labels for the edges.

Argument `query` is the target distribution of the search. It has the same syntax as `data`, but only a single distribution should be given. Multiple simultaneous target distributions are not supported.

Argument `graph` is a description of a directed acyclic graph where directed edges are denoted by `->` and bidirected arcs corresponding to unobserved confounders are denoted by `<->` (or by `--`). As an example, a DAG with two directed edges and one bidirected edge is constructed as follows:

```
> graph <- "
+  X -> Z
+  Z -> Y
+  X <-> Y
+"
```

Some alternative formats for DAGs are supported as well. Graphs created using the **igraph** package in the **causal.effect** package syntax can be used for **dosearch** as well. DAGs created using the **dagitty** package are also supported. Note that both time and space complexity of the underlying search algorithm are exponential in the number of vertices, but instances with up to ten nodes are routinely solved in under a second.

LDAGs are constructed similarly with the addition of labels and with the omission bidirected edges (latent variables must be explicitly defined). As an example, an LDAG with two labeled edges can be constructed as follows:

```
> graph <- "
+  X -> Z : A = 0
+  Z -> Y : A = 1
+  A -> Z
+  A -> Y
+"
```

Here the labels indicate that the edge from X to Z vanishes when A has the value 0 and the edge from Z to Y vanishes when A has the value 1. Multiple labels on the same edge should be separated by a semi-colon, and individual assignments within each label should be separated by a comma.

Argument `transportability` enumerates the nodes that should be understood as transportability nodes responsible for discrepancies between domains. Individual variables should be separated by a comma. See e.g., (Bareinboim and Pearl, 2014) for details on transportability.

Argument `selection_bias` enumerates the nodes that should be understood as selection bias nodes responsible for bias in the input data sets. Individual variables should be separated by a comma. See e.g., (Bareinboim and Tian, 2015) for details on selection bias recoverability.

Argument `missing_data` enumerates the missingness mechanisms of the model. The syntax for a single mechanism is `M_X : X` where M_X is the mechanism for X. Individual mechanisms should be

separated by a comma. Note that both M_X and X must be present in the graph if the corresponding mechanism is given as input. Proxy variables should not be included in the graph, since they are automatically generated based on `missing_data`. By default, a warning is issued if a proxy variable is present in an input distribution but its corresponding mechanism is not present in any input. See e.g., (Mohan, Pearl and Tian, 2013) for details on missing data as a causal inference problem. Note that `dosearch` is not complete for missing data problems, meaning that if `dosearch` is not able to identify the `query`, it might still be identifiable via some other means.

The `control` argument is a list that can supply any of the following components:

- `benchmark`: a `logical` value. If `TRUE`, the search time is recorded and returned (in milliseconds). Defaults to `FALSE`.
- `benchmark_rules`: a `logical` value. If `TRUE`, the time taken by each individual inference rule is also recorded in the benchmark (in milliseconds). Defaults to `FALSE`.
- `draw_derivation`: a `logical` value. If `TRUE`, a string representing the derivation steps as a DOT graph is returned. If the `DiagrammeR` package is installed, the DOT graph can be plotted by calling `plot` on the return object. The DOT graph can also be exported as an .svg file by using the `DOT` package. Defaults to `FALSE`.
- `draw_all`: a `logical` value. If `TRUE` and if `draw_derivation = TRUE`, the derivation will contain every step taken by the search. If `FALSE`, only the steps that resulted in an identifiable target are returned. Defaults to `FALSE`. `empty`: a `logical` value. If `TRUE`, an empty `dosearch` object is returned without running the search.
- `formula`: a `logical` value. If `TRUE`, a string representing the identifiable query is returned when the target query is identifiable. If `FALSE`, only a logical value is returned that takes the value `TRUE` for an identifiable target and `FALSE` otherwise. Defaults to `TRUE`.
- `heuristic`: a `logical` value. If `TRUE`, new distributions are expanded during the search according to a search heuristic (see Tikka et al., 2021, for details). Otherwise, distributions are expanded in the order in which they were identified. Defaults to `FALSE`.
- `md_sym`: a single `character` describing the symbol to use for active missing data mechanisms. Defaults to `"1"`.
- `time_limit`: a `numeric` value giving a time limit for the search (in hours). Defaults to a negative value that disables the time limit.
- `verbose`: a `logical` value. If `TRUE`, diagnostic information is printed to the console during the search. Defaults to `FALSE`.
- `warn`: a `logical` value. If `TRUE`, a warning is issued for possibly unintentionally misspecified but syntactically correct input distributions. A warning is also raised if both lower-case and upper-case node or variable names are used simultaneously in the inputs

Value

`dosearch` returns an object of class `dosearch` which is a list with the following components by default. See the `control` options on how to obtain a graphical representation of the derivation or how to benchmark the search.

- `identifiable`: a `logical` value that is `TRUE` if the target quantity is identifiable and `FALSE` otherwise.
- `formula`: a `character` string describing the formula for an identifiable query or an empty character vector for a non-identifiable effect.

`summary` returns a `summary.dosearch` object.

`plot` returns a `htmlwidget` object or `NULL` (invisibly)

`print` returns `x` invisibly.

`is_identifiable` returns a logical value. If `TRUE`, the target distribution was identifiable from the available inputs.

`get_formula` returns a `character` string representing the query in terms of the input data or `NULL` if the query is not identifiable.

`get_derivation` returns a graphical representation of the derivation steps that resulted in identification. The return object is a `character` string in `DOT` syntax.

`get_benchmark` returns a `list` with one or two elements or `NULL`. The first element of the list is always a numeric value of the total time taken by the search in milliseconds. The second is a numeric vector of the time taken by each inference rule (in the internal C++ implementation) of the search in milliseconds if `include_rules` is `TRUE`.

References

S. Tikka, A. Hyttinen, J. Karvanen. "Causal Effect Identification from Multiple Incomplete Data Sources: A General Search-based Approach." *Journal of Statistical Software*, 99(5):1–40, 2021.

E. Bareinboim, J. Pearl. "Transportability from Multiple Environments with Limited Experiments: Completeness Results." In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems*, 280–288, 2014.

E. Bareinboim, J. Tian. "Recovering Causal Effects from Selection Bias " In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 3475–3481, 2015.

K. Mohan, J. Pearl, J. Tian. "Graphical Models for Inference with Missing Data." In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 1277–1285, 2013.

Examples

```
# A simple back-door formula
data1 <- "P(x,y,z)"
query1 <- "P(y|do(x))"
graph1 <- "
  x -> y
  z -> x
  z -> y
"
dosearch(data1, query1, graph1)

# A simple front-door formula
data2 <- "P(x,y,z)"
query2 <- "P(y|do(x))"
graph2 <- "
  x -> z
  z -> y
  x <-> y
"
dosearch(data2, query2, graph2)
```

```
# A scenario with combined transportability and selection bias
# in this case using the search heuristic provides a simpler formula
data <- "
  p(x,z,y|s)
  p(y,z|t,do(x))
"
query <- "p(y|do(x))"
graph <- "
  x -> z
  z -> y
  x -> s
  t -> z
  x <-> y
"
dosearch(
  data,
  query,
  graph,
  transportability = "t",
  selection_bias = "s",
  control = list(heuristic = TRUE, improve = FALSE)
)

# A simple case-control design
data <- "
  p(x*,y*,r_x,r_y)
  p(y)
"
graph <- "
  x -> y
  y -> r_y
  r_y -> r_x
"
md <- "r_x : x, r_y : y"
dosearch(data, query, graph, missing_data = md)

# Graph input using 'igraph' in the 'causaleffect' syntax
if (requireNamespace("igraph", quietly = TRUE)) {
  g_igraph <- igraph::graph.formula(
    x -+ z, z -+ y, x -+ y, y -+ x,
    simplify = FALSE
  )
  g_igraph <- igraph::set.edge.attribute(g_igraph, "description", 3:4, "U")
  dosearch(data2, query2, g_igraph)
}

# Graph input with 'dagitty'
if (requireNamespace("dagitty", quietly = TRUE)) {
  g_dagitty <- dagitty::dagitty("dag{x -> z -> y; x <-> y}")
  dosearch(data2, query2, g_dagitty)
}

# Alternative distribution input style using lists and vectors:
```

```
# Each element of the list describes a single distribution
# Each element is a character vector that describes the role
# of each variable in the distribution as follows:
# For a variable V and a distribution P(A|do(B),C) we have
#   V = 0, if V is in A
#   V = 1, if V is in B
#   V = 2, if V is in C
data_alt <- list(
  c(x = 0, y = 0, z = 0) # = P(x,y,z)
)
query_alt <- c(x = 1, y = 0) # = P(y|do(x))
dosearch(data_alt, query_alt, graph2)

## Not run:
# Additional examples
# Multiple input distributions (both observational and interventional)
data3 <- "
  p(z_2,x_2|do(x_1))
  p(z_1|x_2,do(x_1,y))
  p(x_1|w_1,do(x_2))
  p(y|z_1,z_2,x_1,do(x_2))
  p(w|y,x_1,do(x_2))
"
query3 <- "p(y,x_1|w,do(x_2))"
graph3 <- "
  x_1 -> z_2
  x_1 -> z_1
  x_2 -> z_1
  x_2 -> z_2
  z_1 -> y
  z_2 -> y
  x_1 -> w
  x_2 -> w
  z_1 -> w
  z_2 -> w
"
dosearch(data3, query3, graph3)

# Selection bias
data4 <- "
  p(x,y,z_1,z_2|s)
  p(z_1,z_2)
"
query4 <- "p(y|do(x))"
graph4 <- "
  x   -> z_1
  z_1 -> z_2
  x   -> y
  y   -- z_2
  z_2 -> s
"
dosearch(data4, query4, graph4, selection_bias = "s")
```

```
# Transportability
data5 <- "
  p(x,y,z_1,z_2)
  p(x,y,z_1|t_1,t_2,do(z_2))
  p(x,y,z_2|t_3,do(z_1))
"
query5 <- "p(y|do(x))"
graph5 <- "
  z_1 -> x
  x   -> z_2
  z_2 -> y
  z_1 <-> x
  z_1 <-> z_2
  z_1 <-> y
  t_1 -> z_1
  t_2 -> z_2
  t_3 -> y
"
dosearch(data5, query5, graph5, transportability = "t_1, t_2, t_3")

# Missing data
# Proxy variables are denoted by an asterisk (*)
data6 <- "
  p(x*,y*,z*,m_x,m_y,m_z)
"
query6 <- "p(x,y,z)"
graph6 <- "
  z -> x
  x -> y
  x -> m_z
  y -> m_z
  y -> m_x
  z <-> y
"
dosearch(data6, query6, graph6, missing_data = "m_x : x, m_y : y, m_z : z")

# An LDAG
data7 <- "P(X,Y,Z)"
query7 <- "P(Y|X,I_X=1)"
graph7 <- "
  X -> Y : Z = 1
  Z -> Y
  Z -> X : I_X = 1
  I_X -> X
  H -> X : I_X = 1
  H -> Z
  Q -> Z
  Q -> Y : Z = 0
"
dosearch(data7, query7, graph7)

# A more complicated LDAG
# with multiple assignments for the edge X -> Z
```

```
data8 <- "P(X,Y,Z,A,W)"
query8 <- "P(Y|X,I_X=1)"
graph8 <- "
  I_X -> X
  I_Z -> Z
  A -> W
  Z -> Y
  A -> Z
  X -> Z : I_Z = 1; A = 1
  X -> Y : A = 0
  W -> X : I_X = 1
  W -> Y : A = 0
  A -> Y
  U -> X : I_X = 1
  U -> Y : A = 1
"
dosearch(data8, query8, graph8)

# Export the DOT diagram of the derivation as an SVG file
# to the working directory via the DOT package.
# By default, only the identifying part is plotted.
# PostScript format is also supported.
if (requireNamespace("DOT", quietly = TRUE)) {
  d <- get_derivation(
    data1,
    query1,
    graph1,
    control = list(draw_derivation = TRUE)
  )
  DOT::dot(d$derivation, "derivation.svg")
}

## End(Not run)

data <- "p(x,y,z)"
query <- "p(y|do(x))"
graph <- "
  x -> y
  Z -> x
  z -> y
"
x <- dosearch(data, query, graph)
y <- summary(x)

## Not run:
out <- dosearch(
  "p(x,y,z, w)",
  "p(y|do(x))",
  "x -> y \n z -> x \n w -> z \n x <-> w \n w <-> y",
  control = list(draw_derivation = TRUE)
)
if (requireNamespace("DiagrammeR", quietly = TRUE)) {
```

```
  plot(out)
}

## End(Not run)

data <- "p(x,y,z)"
query <- "p(y|do(x))"
graph <- "
  x -> z
  Z -> y
  x <-> y
"
x <- dosearch(data, query, graph)
print(x)

data <- "P(x,y,z)"
query <- "P(y|do(x))"
graph <- "
  x -> y
  z -> x
  z -> y
"
x <- dosearch(data, query, graph)
is_identifiable(x)
# TRUE

data <- "P(x,y,z)"
query <- "P(y|do(x))"
graph <- "
  x -> y
  z -> x
  z -> y
"
x <- dosearch(data, query, graph, control = list(formula = FALSE))
get_formula(x, run_again = TRUE)

data <- "P(x,y,z)"
query <- "P(y|do(x))"
graph <- "
  x -> y
  z -> x
  z -> y
"
x <- dosearch(data, query, graph, control = list(draw_derivation = FALSE))
get_derivation(x, run_again = TRUE)
data <- "P(x,y,z)"
query <- "P(y|do(x))"
graph <- "
  x -> y
  z -> x
  z -> y
"
x <- dosearch(data, query, graph, control = list(benchmark = FALSE))
```

```
get_benchmark(x, run_again = TRUE)
```

---

print.summary.dosearch

Print the Summary of a dosearch *Object*

Description

Print the Summary of a dosearch Object

Usage

```
## S3 method for class 'summary.dosearch'
print(x, max_chars = 300L, ...)
```

Arguments

x	An object of class summary.dosearch.
max_chars	Maximum number of characters of the formula to display. The default is 300.
...	Not used.

Value

x (invisibly)

Examples

```
data <- "p(x,y,z)"
query <- "p(y|do(x))"
graph <- "
  x -> y
  Z -> x
  z -> y
"
x <- dosearch(data, query, graph)
y <- summary(x)
print(y)
```

Index