Package 'caugi'

December 4, 2025

Title Causal Graph Interface

Version 0.3.1

Description Create, query, and modify causal graphs. 'caugi' (Causal Graph Interface) is a causality-first, high performance graph package that provides a simple interface to build, structure, and examine causal relationships.

License MIT + file LICENSE

URL https://frederikfabriciusbjerre.github.io/caugi/

BugReports https://github.com/frederikfabriciusbjerre/caugi/issues

Depends R (>= 4.2)

Imports data.table, fastmap, S7, stats, methods

Suggests bnlearn, dagitty, devtools, ggm, graph, gRbase, igraph, knitr, MASS, Matrix, rextendr, rmarkdown, testthat

VignetteBuilder knitr

Config/rextendr/version 0.4.2

Encoding UTF-8

RoxygenNote 7.3.3

SystemRequirements Cargo (Rust's package manager), rustc >= 1.80.0, xz

Config/Needs/website rmarkdown

NeedsCompilation yes

Author Frederik Fabricius-Bjerre [aut, cre, cph], Johan Larsson [aut] (ORCID: <https://orcid.org/0000-0002-4029-5945>), Michael Sachs [aut] (ORCID: <https://orcid.org/0000-0002-1279-8676>)

Maintainer Frederik Fabricius-Bjerre <frederik@fabriciusbjerre.dk>

Repository CRAN

Date/Publication 2025-12-04 12:00:02 UTC

Contents

`adjustment_set` *Compute an adjustment set*

Description

Computes an adjustment set for `X -> Y` in a DAG.

Usage

```
adjustment_set(
  cg,
  X = NULL,
  Y = NULL,
  X_index = NULL,
  Y_index = NULL,
  type = c("optimal", "parents", "backdoor")
)
```

Arguments

`cg`	A `caugi` object.
`X`, `Y`	Node names.
`X_index`, `Y_index`	Optional numeric 1-based indices.
`type`	One of `"parents"`, `"backdoor"`, `"optimal"`. The `optimal` option computes the O-set.

Details

Types supported:

- `"parents"`: $\bigcup \mathrm{Pa}(X)$ minus $X \cup Y$
- `"backdoor"`: Pearl backdoor formula
- `"optimal"`: O-set (only for single `x` and single `y`)

Value

A character vector of node names representing the adjustment set.

See Also

Other adjustment: `all_backdoor_sets()`, `d_separated()`, `is_valid_backdoor()`

Examples

```
cg <- caugi(
  C %-->% X,
  X %-->% F,
  X %-->% D,
  A %-->% X,
  A %-->% K,
  K %-->% Y,
  D %-->% Y,
  D %-->% G,
  Y %-->% H,
  class = "DAG"
)

adjustment_set(cg, "X", "Y", type = "parents") # C, A
adjustment_set(cg, "X", "Y", type = "backdoor") # C, A
adjustment_set(cg, "X", "Y", type = "optimal") # K
```

`aid` *Adjustment Identification Distance*

Description

Compute the Adjustment Identification Distance (AID) between two graphs using the `gadjid` Rust package.

Usage

```
aid(truth, guess, type = c("oset", "ancestor", "parent"), normalized = TRUE)
```

Arguments

`truth`	A `caugi` object.
`guess`	A `caugi` object.
`type`	A character string specifying the type of AID to compute. Options are `"oset"` (default), `"ancestor"`, and `"parent"`.
`normalized`	Logical; if `TRUE`, returns the normalized AID. If `FALSE`, returns the count.

Value

A numeric representing the AID between the two graphs, if `normalized = TRUE`, or an integer count if `normalized = FALSE`.

See Also

Other metrics: `hd()`, `shd()`

Examples

```
set.seed(1)
truth <- generate_graph(n = 100, m = 200, class = "DAG")
guess <- generate_graph(n = 100, m = 200, class = "DAG")
aid(truth, guess) # 0.0187
```

`all_backdoor_sets` *Get all backdoor sets up to a certain size.*

Description

This function returns the backdoor sets up to size `max_size`, which per default is set to 10.

Usage

```
all_backdoor_sets(
  cg,
  X = NULL,
  Y = NULL,
  X_index = NULL,
  Y_index = NULL,
  minimal = TRUE,
  max_size = 3L
)
```

Arguments

`cg`	A `caugi`.
`X`, `Y`	Single node name.
`X_index`, `Y_index`	Optional 1-based indices (exclusive with name args).
`minimal`	Logical; if `TRUE` (default), only minimal sets are returned.
`max_size`	Integer; maximum size of sets to consider (default 3).

Value

A list of character vectors, each an adjustment set (possibly empty).

See Also

Other adjustment: `adjustment_set()`, `d_separated()`, `is_valid_backdoor()`

Examples

```
cg <- caugi(
  C %-->% X,
  X %-->% F,
  X %-->% D,
  A %-->% X,
  A %-->% K,
  K %-->% Y,
  D %-->% Y,
  D %-->% G,
  Y %-->% H,
  class = "DAG"
)

all_backdoor_sets(cg, X = "X", Y = "Y", max_size = 3L, minimal = FALSE)
#> [[1]]
#> [1] "A"
#>
#> [[2]]
#> [1] "K"
#>
#> [[3]]
#> [1] "C" "A"
#>
#> [[4]]
#> [1] "C" "K"
#>
#> [[5]]
#> [1] "A" "K"
#>
#> [[6]]
#> [1] "C" "A" "K"

all_backdoor_sets(cg, X = "X", Y = "Y", max_size = 3L, minimal = TRUE)
#> [[1]]
#> [1] "A"
#>
#> [[2]]
#> [1] "K"
```

---

ancestors *Get ancestors of nodes in a* caugi

Description

Get ancestors of nodes in a caugi

Usage

```
ancestors(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
ancestors(cg, "A") # NULL
ancestors(cg, index = 2) # "A"
ancestors(cg, "B") # "A"
ancestors(cg, c("B", "C"))
#> $B
#> [1] "A"
#>
#> $C
#> [1] "A" "B"
```

---

as_adjacency *Convert a caugi to an adjacency matrix*

Description

Does not take other edge types than the one found in a PDAG.

Usage

```
as_adjacency(x)
```

Arguments

x	A caugi object.

Value

An integer 0/1 adjacency matrix with row/col names.

See Also

Other conversions: as_bnlearn(), as_caugi(), as_dagitty(), as_igraph()

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)
adj <- as_adjacency(cg)
```

---

as_bnlearn *Convert a caugi to a bnlearn network*

Description

Convert a caugi to a bnlearn network

Usage

```
as_bnlearn(x)
```

Arguments

x	A caugi object.

Value

A bnlearn DAG.

See Also

Other conversions: as_adjacency(), as_caugi(), as_dagitty(), as_igraph()

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)
g_bn <- as_bnlearn(cg)
```

as_caugi *Convert to a* caugi

Description

Convert an object to a caugi. The object can be a graphNEL, matrix, tidygraph, daggity, bn, or igraph.

Usage

```
as_caugi(
  x,
  class = c("DAG", "PDAG", "PAG", "UNKNOWN"),
  simple = TRUE,
  build = TRUE,
  collapse = FALSE,
  collapse_to = "---",
  ...
)
```

Arguments

x	An object to convert to a caugi.
class	"DAG", "PDAG", "PAG", or "UNKNOWN". "PAG" is only supported for integer coded matrices.
simple	logical. If TRUE (default) the graph will be simple (no multiple edges or self-loops).
build	logical. If TRUE (default) build the graph now, otherwise build lazily on first query or when using build().
collapse	logical. If TRUE collapse mutual directed edges to undirected edges. Default is FALSE.
collapse_to	Character string to use as the edge glyph when collapsing. Should be a registered symmetrical edge glyph. Default is "---".
...	Additional arguments passed to specific methods.

Details

For matrices, as_caugi assumes that the rows are the from nodes and the columns are the to nodes. Thus, for a graph, G: A –> B, we would have that G["A", "B"] == 1 and G["B", "A"] == 0. For PAGs, the integer codes are as follows (as used in pcalg):

- 0: no edge
- 1: circle (e.g., A o-o B or A o-- B)
- 2: arrowhead (e.g., A --> B or A o-> B)
- 3: tail (e.g., A o-- B or A --- B)

Value

A caugi object.

See Also

Other conversions: as_adjacency(), as_bnlearn(), as_dagitty(), as_igraph()

Examples

```
# igraph
ig <- igraph::graph_from_literal(A - +B, B - +C)
cg_ig <- as_caugi(ig, class = "DAG")

# graphNEL
gn <- graph::graphNEL(nodes = c("A", "B", "C"), edgemode = "directed")
gn <- graph::addEdge("A", "B", gn)
gn <- graph::addEdge("B", "C", gn)
cg_gn <- as_caugi(gn, class = "DAG")

# adjacency matrix
m <- matrix(0L, 3, 3, dimnames = list(LETTERS[1:3], LETTERS[1:3]))
m["A", "B"] <- 1L
m["B", "C"] <- 1L
cg_adj <- as_caugi(m, class = "DAG")

# bnlearn
bn <- bnlearn::model2network("[A][B|A][C|B]")
cg_bn <- as_caugi(bn, class = "DAG")

# dagitty
dg <- dagitty::dagitty("dag {
 A -> B
 B -> C
 }")
cg_dg <- as_caugi(dg, class = "DAG")

cg <- caugi(A %-->% B %-->% C, class = "DAG")

# check that all nodes are equal in all graph objects
for (cg_converted in list(cg_ig, cg_gn, cg_adj, cg_bn, cg_dg)) {
  stopifnot(identical(nodes(cg), nodes(cg_converted)))
  stopifnot(identical(edges(cg), edges(cg_converted)))
}

# collapse mutual edges
ig2 <- igraph::graph_from_literal(A - +B, B - +A, C - +D)
cg2 <- as_caugi(ig2, class = "PDAG", collapse = TRUE, collapse_to = "---")

# coded integer matrix for PAGs (pcalg style)
nm <- c("A", "B", "C", "D")
M <- matrix(0L, 4, 4, dimnames = list(nm, nm))
```

```
# A --> B
M["A", "B"] <- 2L # mark at B end
M["B", "A"] <- 3L # mark at A end

# A --- C
M["A", "C"] <- 3L
M["C", "A"] <- 3L

# B o-> C
M["B", "C"] <- 2L
M["C", "B"] <- 1L

# C o-o D
M["C", "D"] <- 1L
M["D", "C"] <- 1L

cg <- as_caugi(M, class = "PAG")
```

`as_dagitty` *Convert a caugi to a dagitty graph*

Description

Convert a caugi to a dagitty graph

Usage

```
as_dagitty(x)
```

Arguments

`x`	A `caugi` object.

Value

A `dagitty` object.

See Also

Other conversions: `as_adjacency()`, `as_bnlearn()`, `as_caugi()`, `as_igraph()`

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)
g_dg <- as_dagitty(cg)
```

as_igraph *Convert a caugi to an igraph object*

Description

Convert a caugi to an igraph object

Usage

```
as_igraph(x, ...)
```

Arguments

x	A caugi object.
...	Additional arguments passed to igraph::graph_from_data_frame().

Value

An igraph object representing the same graph structure.

See Also

Other conversions: as_adjacency(), as_bnlearn(), as_caugi(), as_dagitty()

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)
ig <- as_igraph(cg)
```

build *Build the graph now*

Description

If a caugi has been modified (nodes or edges added or removed), it is marked as *not built*, i.e cg@built = FALSE. This function builds the graph using the Rust backend and updates the internal pointer to the graph. If the graph is already built, it is returned.

Usage

```
build(cg, ...)
```

Arguments

cg	A caugi object.
...	Not used.

Value

The built caugi object.

See Also

Other verbs: caugi_verbs

Examples

```
# initialize empty graph and build slowly
cg <- caugi(class = "PDAG")

cg <- cg |>
  add_nodes(c("A", "B", "C", "D", "E")) |> # A, B, C, D, E
  add_edges(A %-->% B %-->% C) |> # A --> B --> C, D, E
  set_edges(B %---% C) # A --> B --- C, D, E

cg <- remove_edges(cg, B %---% C) |> # A --> B, C, D, E
  remove_nodes(c("C", "D", "E")) # A --> B

# verbs do not build the Rust backend
cg@built # FALSE
build(cg)
cg@built # TRUE
```

---

caugi *Create a* caugi *from edge expressions.*

Description

Create a caugi from a series of edge expressions using infix operators. Nodes can be specified as symbols, strings, or numbers.

The following edge operators are supported by default:

- %-->% for directed edges (A –> B)
- %---% for undirected edges (A — B)
- %<->% for bidirected edges (A <-> B)
- %o->% for partially directed edges (A o-> B)
- %--o% for partially undirected edges (A –o B)
- %o-o% for partial edges (A o-o B)

You can register additional edge types using register_caugi_edge().

Usage

```
caugi(
  ...,
  from = NULL,
  edge = NULL,
  to = NULL,
  nodes = NULL,
  edges_df = NULL,
  simple = TRUE,
  build = TRUE,
  class = c("UNKNOWN", "DAG", "PDAG", "UG"),
  state = NULL
)
```

Arguments

`...`	Edge expressions using the supported infix operators, or nodes given by symbols or strings. Multiple edges can be combined using `+`: `A --> B + C`, indicating an edge from `A` to both `B` and `C`. Nodes can also be grouped using `c(...)` or parentheses.
`from`	Character vector of source node names. Optional; mutually exclusive with `....`
`edge`	Character vector of edge types. Optional; mutually exclusive with `....`
`to`	Character vector of target node names. Optional; mutually exclusive with `....`
`nodes`	Character vector of node names to declare as isolated nodes. An optional, but recommended, option is to provide all node names in the graph, including those that appear in edges. If `nodes` is provided, the order of nodes in the graph will follow the order in `nodes`.
`edges_df`	Optional data.frame or data.table with columns `from`, `edge`, and `to` to specify edges. Mutually exclusive with `...` and `from`, `edge`, `to`. Can be used to create graphs using `edges(cg)` from another `caugi` object, `cg`.
`simple`	Logical; if `TRUE` (default), the graph is a simple graph, and the function will throw an error if the input contains parallel edges or self-loops.
`build`	Logical; if `TRUE` (default), the graph will be built using the Rust backend. If `FALSE`, the graph will not be built, and the Rust backend cannot be used. The graph will build, when queries are made to the graph or if calling `build()`. **Note**: Even if `build = TRUE`, if no edges or nodes are provided, the graph will not be built and the pointer will be `NULL`.
`class`	Character; one of `"UNKNOWN"`, `"DAG"`, `"PDAG"`, or `"UG"`.
`state`	For internal use. Build a graph by supplying a pre-constructed state environment.

Value

A `caugi` S7 object containing the nodes, edges, and a pointer to the underlying Rust graph structure.

Examples

```
# create a simple DAG (using NSE)
cg <- caugi(
  A %-->% B + C,
  B %-->% D,
  class = "DAG"
)

# create a PDAG with undirected edges (using NSE)
cg2 <- caugi(
  A %-->% B + C,
  B %---% D,
  E, # no neighbors for this node
  class = "PDAG"
)

# create a DAG (using SE)
cg3 <- caugi(
  from = c("A", "A", "B"),
  edge = c("-->", "-->", "-->"),
  to = c("B", "C", "D"),
  nodes = c("A", "B", "C", "D", "E"),
  class = "DAG"
)

# create a non-simple graph
cg4 <- caugi(
  A %-->% B,
  B %-->% A,
  class = "UNKNOWN",
  simple = FALSE
)

cg4@simple # FALSE
cg4@built # TRUE
cg4@graph_class # "UNKNOWN"

# create graph, but don't built Rust object yet, which is needed for queries
cg5 <- caugi(
  A %-->% B + C,
  B %-->% D,
  class = "DAG",
  build = FALSE
)

cg@built # FALSE
```

---

caugi_verbs *Manipulate nodes and edges of a* caugi

Description

Add, remove, or and set nodes or edges to / from a caugi object. Edges can be specified using expressions with the infix operators. Alternatively, the edges to be added are specified using the from, edge, and to arguments.

Usage

```
add_edges(cg, ..., from = NULL, edge = NULL, to = NULL, inplace = FALSE)

remove_edges(cg, ..., from = NULL, edge = NULL, to = NULL, inplace = FALSE)

set_edges(cg, ..., from = NULL, edge = NULL, to = NULL, inplace = FALSE)

add_nodes(cg, ..., name = NULL, inplace = FALSE)

remove_nodes(cg, ..., name = NULL, inplace = FALSE)
```

Arguments

cg	A caugi object.
...	Expressions specifying edges to add using the infix operators, or nodes to add using unquoted names, vectors via c(), or + composition.
from	Character vector of source node names. Default is NULL.
edge	Character vector of edge types. Default is NULL.
to	Character vector of target node names. Default is NULL.
inplace	Logical, whether to modify the graph inplace or not. If FALSE (default), a copy of the caugi is made and modified.
name	Character vector of node names. Default is NULL.

Details

Caugi graph verbs

Value

The updated caugi.

Functions

- add_edges(): Add edges.
- remove_edges(): Remove edges.
- set_edges(): Set edge type for given pair(s).
- add_nodes(): Add nodes.
- remove_nodes(): Remove nodes.

See Also

Other verbs: build()

Examples

```
# initialize empty graph and build slowly
cg <- caugi(class = "PDAG")

cg <- cg |>
  add_nodes(c("A", "B", "C", "D", "E")) |> # A, B, C, D, E
  add_edges(A %-->% B %-->% C) |> # A --> B --> C, D, E
  set_edges(B %---% C) # A --> B --- C, D, E

cg <- remove_edges(cg, B %---% C) |> # A --> B, C, D, E
  remove_nodes(c("C", "D", "E")) # A --> B

# verbs do not build the Rust backend
cg@built # FALSE
build(cg)
cg@built # TRUE
```

---

children *Get children of nodes in a* caugi

Description

Get children of nodes in a caugi

Usage

```
children(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
children(cg, "A") # "B"
children(cg, index = 2) # "C"
children(cg, "B") # "C"
children(cg, c("B", "C"))
#> $B
#> [1] "C"
#>
#> $C
#> NULL
```

---

descendants *Get descendants of nodes in a* caugi

Description

Get descendants of nodes in a caugi

Usage

```
descendants(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), children(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
descendants(cg, "A") # "B" "C"
descendants(cg, index = 2) # "C"
descendants(cg, "B") # "C"
descendants(cg, c("B", "C"))
#> $B
#> [1] "C"
#>
#> $C
#> NULL
```

---

d_separated *Are X and Y d-separated given Z?*

Description

Checks whether every node in X is d-separated from every node in Y given Z in a DAG.

Usage

```
d_separated(
  cg,
  X = NULL,
  Y = NULL,
  Z = NULL,
  X_index = NULL,
  Y_index = NULL,
  Z_index = NULL
)
```

Arguments

cg	A caugi object.
X, Y, Z	Node selectors: character vector of names, unquoted expression (supports + and c()), or NULL. Use *_index to pass 1-based indices. If Z is NULL or missing, no nodes are conditioned on.

X_index, Y_index, Z_index
: Optional numeric 1-based indices (exclusive with X,Y,Z respectively).

Value

TRUE if d-separated, FALSE otherwise.

See Also

Other adjustment: adjustment_set(), all_backdoor_sets(), is_valid_backdoor()

Examples

```
cg <- caugi(
  C %-->% X,
  X %-->% F,
  X %-->% D,
  A %-->% X,
  A %-->% K,
  K %-->% Y,
  D %-->% Y,
  D %-->% G,
  Y %-->% H,
  class = "DAG"
)

d_separated(cg, "X", "Y", Z = c("A", "D")) # TRUE
d_separated(cg, "X", "Y", Z = NULL) # FALSE
```

---

edges *Get edges of a* caugi.

Description

Get edges of a caugi.

Usage

```
edges(cg)

E(cg)
```

Arguments

cg
: A caugi object.

Value

A data.table with columns from, edge, and to.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  D,
  class = "DAG"
)
edges(cg) # returns the data.table with columns from, edge, to
```

---

edge_types *Get the edge types of a* caugi.

Description

Get the edge types of a caugi.

Usage

```
edge_types(cg)
```

Arguments

cg	A caugi object.

Value

A character vector of edge types.

See Also

Other queries: ancestors(), children(), descendants(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %--o% C,
  C %<->% D,
  D %---% E,
  A %o-o% E,
```

```
  class = "UNKNOWN"
)
edge_types(cg) # returns c("-->", "o-o", "--o", "<->", "---")
```

---

exogenous *Get all exogenous nodes in a* caugi

Description

Get all exogenous nodes (nodes with no parents) in a caugi.

Usage

```
exogenous(cg, undirected_as_parents = FALSE)
```

Arguments

cg	A caugi object.
undirected_as_parents	Logical; if TRUE, undirected edges are treated as (possible) parents, if FALSE (default), undirected edges are ignored.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
exogenous(cg) # "A"
```

generate_graph *Generate a* caugi *using Erdős-Rényi.*

Description

Sample a random DAG or CPDAG using Erdős-Rényi for random graph generation.

Usage

```
generate_graph(n, m = NULL, p = NULL, class = c("DAG", "CPDAG"))
```

Arguments

n	Integer >= 0. Number of nodes in the graph.
m	Integer in `0, n*(n-1)/2`. Number of edges in the graph. Exactly one of `m` or `p` must be supplied.
p	Numeric in `[0,1]`. Probability of edge creation. Exactly one of `m` or `p` must be supplied.
class	"DAG" or "CPDAG".

Value

The sampled `caugi` object.

Examples

```
# generate a random DAG with 5 nodes and 4 edges
dag <- generate_graph(n = 5, m = 4, class = "DAG")

# generate a random CPDAG with 5 nodes and edge probability 0.3
cpdag <- generate_graph(n = 5, p = 0.3, class = "CPDAG")
```

hd *Hamming Distance*

Description

Compute the Hamming Distance between two graphs.

Usage

```
hd(cg1, cg2, normalized = FALSE)
```

Arguments

cg1	A caugi object.
cg2	A caugi object.
normalized	Logical; if TRUE, returns the normalized Hamming Distance.

Value

An integer representing the Hamming Distance between the two graphs, if normalized = FALSE, or a numeric between 0 and 1 if normalized = TRUE.

See Also

Other metrics: aid(), shd()

Examples

```
cg1 <- caugi(A %-->% B %-->% C, D %-->% C, class = "DAG")
cg2 <- caugi(A %-->% B %-->% C, D %---% C, class = "PDAG")
hd(cg1, cg2) # 0
```

---

is_acyclic *Is the* caugi *acyclic?*

Description

Checks if the given caugi graph is acyclic.

Usage

```
is_acyclic(cg, force_check = FALSE)
```

Arguments

cg	A caugi object.
force_check	Logical; if TRUE, the function will test if the graph is acyclic, if FALSE (default), it will look at the graph class and match it, if possible.

Details

Logically, it should not be possible to have a graph class of "DAG" or "PDAG" that has cycles, but in case the user modified the graph after creation in some unforeseen way that could have introduced cycles, this function allows to force a check of acyclicity, if needed.

Value

A logical value indicating whether the graph is acyclic.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg_acyclic <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
is_acyclic(cg_acyclic) # TRUE
cg_cyclic <- caugi(
  A %-->% B,
  B %-->% C,
  C %-->% A,
  class = "UNKNOWN"
)
is_acyclic(cg_cyclic) # FALSE
```

---

is_caugi *Is it a* caugi *graph?*

Description

Checks if the given object is a caugi. Mostly used internally to validate inputs.

Usage

```
is_caugi(x, throw_error = FALSE)
```

Arguments

x	An object to check.
throw_error	Logical; if TRUE, throws an error if x is not a caugi.

Value

A logical value indicating whether the object is a caugi.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)

is_caugi(cg) # TRUE
```

`is_cpdag` *Is the* `caugi` *graph a CPDAG?*

Description

Checks if the given `caugi` graph is a Complete Partially Directed Acyclic Graph (CPDAG).

Usage

```
is_cpdag(cg)
```

Arguments

`cg`	A `caugi` object.

Value

A logical value indicating whether the graph is a CPDAG.

See Also

Other queries: `ancestors()`, `children()`, `descendants()`, `edge_types()`, `edges()`, `exogenous()`, `is_acyclic()`, `is_caugi()`, `is_dag()`, `is_empty_caugi()`, `is_pdag()`, `is_ug()`, `markov_blanket()`, `neighbors()`, `nodes()`, `parents()`, `same_nodes()`, `subgraph()`

Examples

```
cg_cpdag <- caugi(
  A %---% B,
  A %-->% C,
  B %-->% C,
  class = "PDAG"
)
is_cpdag(cg_cpdag) # TRUE

cg_not_cpdag <- caugi(
  A %---% B,
  A %---% C,
  B %-->% C,
  class = "PDAG"
```

```
)
is_cpdag(cg_not_cpdag) # FALSE
```

`is_dag` *Is the* `caugi` *graph a DAG?*

Description

Checks if the given `caugi` graph is a Directed Acyclic Graph (DAG).

Usage

```
is_dag(cg, force_check = FALSE)
```

Arguments

`cg`	A `caugi` object.
`force_check`	Logical; if `TRUE`, the function will test if the graph is a DAG, if `FALSE` (default), it will look at the graph class and match it, if possible.

Value

A logical value indicating whether the graph is a DAG.

See Also

Other queries: `ancestors()`, `children()`, `descendants()`, `edge_types()`, `edges()`, `exogenous()`, `is_acyclic()`, `is_caugi()`, `is_cpdag()`, `is_empty_caugi()`, `is_pdag()`, `is_ug()`, `markov_blanket()`, `neighbors()`, `nodes()`, `parents()`, `same_nodes()`, `subgraph()`

Examples

```
cg_dag_class <- caugi(
  A %-->% B,
  class = "DAG"
)
is_dag(cg_dag_class) # TRUE
cg_dag_but_pdag_class <- caugi(
  A %-->% B,
  class = "PDAG"
)
is_dag(cg_dag_but_pdag_class) # TRUE
cg_cyclic <- caugi(
  A %-->% B,
  B %-->% C,
  C %-->% A,
  class = "UNKNOWN",
  simple = FALSE
```

```
)
is_dag(cg_cyclic) # FALSE

cg_undirected <- caugi(
  A %---% B,
  class = "UNKNOWN"
)
is_dag(cg_undirected) # FALSE
```

`is_empty_caugi` *Is the* `caugi` *graph empty?*

Description

Checks if the given `caugi` graph is empty (has no nodes).

Usage

```
is_empty_caugi(cg)
```

Arguments

`cg`	A `caugi` object.

Value

A logical value indicating whether the graph is empty.

See Also

Other queries: `ancestors()`, `children()`, `descendants()`, `edge_types()`, `edges()`, `exogenous()`, `is_acyclic()`, `is_caugi()`, `is_cpdag()`, `is_dag()`, `is_pdag()`, `is_ug()`, `markov_blanket()`, `neighbors()`, `nodes()`, `parents()`, `same_nodes()`, `subgraph()`

Examples

```
cg_empty <- caugi(class = "DAG")
is_empty_caugi(cg_empty) # TRUE
cg_non_empty <- caugi(
  A %-->% B,
  class = "DAG"
)
is_empty_caugi(cg_non_empty) # FALSE

cg_no_edges_but_has_nodes <- caugi(
  A, B,
  class = "DAG"
)
is_empty_caugi(cg_no_edges_but_has_nodes) # FALSE
```

is_pdag *Is the* caugi *graph a PDAG?*

Description

Checks if the given caugi graph is a Partially Directed Acyclic Graph (PDAG).

Usage

```
is_pdag(cg, force_check = FALSE)
```

Arguments

cg	A caugi object.
force_check	Logical; if TRUE, the function will test if the graph is a PDAG, if FALSE (default), it will look at the graph class and match it, if possible.

Value

A logical value indicating whether the graph is a PDAG.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg_dag_class <- caugi(
  A %-->% B,
  class = "DAG"
)
is_pdag(cg_dag_class) # TRUE
cg_dag_but_pdag_class <- caugi(
  A %-->% B,
  class = "PDAG"
)
is_pdag(cg_dag_but_pdag_class) # TRUE
cg_cyclic <- caugi(
  A %-->% B,
  B %-->% C,
  C %-->% A,
  D %---% A,
  class = "UNKNOWN",
  simple = FALSE
)
is_pdag(cg_cyclic) # FALSE
```

```
cg_undirected <- caugi(
  A %---% B,
  class = "UNKNOWN"
)
is_pdag(cg_undirected) # TRUE

cg_pag <- caugi(
  A %o->% B,
  class = "UNKNOWN"
)
is_pdag(cg_pag) # FALSE
```

`is_ug` *Is the* `caugi` *graph an UG?*

Description

Checks if the given `caugi` graph is an undirected graph (UG).

Usage

```
is_ug(cg, force_check = FALSE)
```

Arguments

`cg`	A `caugi` object.
`force_check`	Logical; if `TRUE`, the function will test if the graph is an UG, if `FALSE` (default), it will look at the graph class and match it, if possible.

Value

A logical value indicating whether the graph is an UG.

See Also

Other queries: `ancestors()`, `children()`, `descendants()`, `edge_types()`, `edges()`, `exogenous()`, `is_acyclic()`, `is_caugi()`, `is_cpdag()`, `is_dag()`, `is_empty_caugi()`, `is_pdag()`, `markov_blanket()`, `neighbors()`, `nodes()`, `parents()`, `same_nodes()`, `subgraph()`

Examples

```
cg_ug_class <- caugi(
  A %---% B,
  class = "UG"
)
is_ug(cg_ug_class) # TRUE
cg_not_ug <- caugi(
  A %-->% B,
```

```
  class = "DAG"
)
is_ug(cg_not_ug) # FALSE
```

`is_valid_backdoor` *Is a backdoor set valid?*

Description

Checks whether `Z` is a valid backdoor adjustment set for `X --> Y`.

Usage

```
is_valid_backdoor(
  cg,
  X = NULL,
  Y = NULL,
  Z = NULL,
  X_index = NULL,
  Y_index = NULL,
  Z_index = NULL
)
```

Arguments

`cg`	A `caugi` object.
`X`, `Y`	Single node names.
`Z`	Optional node set for conditioning
`X_index`, `Y_index`, `Z_index`	Optional 1-based indices.

Value

Logical value indicating if backdoor is valid or not.

See Also

Other adjustment: `adjustment_set()`, `all_backdoor_sets()`, `d_separated()`

Examples

```
cg <- caugi(
  C %-->% X,
  X %-->% F,
  X %-->% D,
  A %-->% X,
  A %-->% K,
```

```
  K %-->% Y,
  D %-->% Y,
  D %-->% G,
  Y %-->% H,
  class = "DAG"
)

is_valid_backdoor(cg, X = "X", Y = "Y", Z = NULL) # FALSE
is_valid_backdoor(cg, X = "X", Y = "Y", Z = "K") # TRUE
is_valid_backdoor(cg, X = "X", Y = "Y", Z = c("A", "C")) # TRUE
```

`length` *Length of a* `caugi`

Description

Returns the number of nodes in the graph.

Arguments

`x`	A `caugi` object.

Value

An integer representing the number of nodes.

See Also

Other caugi methods: `print()`

Examples

```
cg <- caugi(
  A %-->% B,
  class = "DAG"
)
length(cg) # 2

cg2 <- caugi(
  A %-->% B + C,
  nodes = LETTERS[1:5],
  class = "DAG"
)
length(cg2) # 5
```

markov_blanket *Get Markov blanket of nodes in a* caugi

Description

Get Markov blanket of nodes in a caugi

Usage

```
markov_blanket(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), neighbors(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
markov_blanket(cg, "A") # "B"
markov_blanket(cg, index = 2) # "A" "C"
markov_blanket(cg, "B") # "A" "C"
markov_blanket(cg, c("B", "C"))
#> $B
#> [1] "A" "C"
#>
#> $C
#> [1] "B"
```

moralize *Moralize a DAG*

Description

Moralizing a DAG involves connecting all parents of each node and then converting all directed edges into undirected edges.

Usage

```
moralize(cg)
```

Arguments

cg	A caugi object (DAG).

Details

This changes the graph from a Directed Acyclic Graph (DAG) to an Undirected Graph (UG), also known as a Markov Graph.

Value

A caugi object representing the moralized graph (UG).

See Also

Other operations: mutate_caugi(), skeleton()

Examples

```
cg <- caugi(A %-->% C, B %-->% C, class = "DAG")
moralize(cg) # A -- B, A -- C, B -- C
```

mutate_caugi *Mutate* caugi *class*

Description

Mutate the caugi class from one graph class to another, if possible. For example, convert a DAG to a PDAG, or a fully directed caugi of class UNKNOWN to a DAG. Throws an error if not possible.

Usage

```
mutate_caugi(cg, class)
```

Arguments

cg	A caugi object.
class	A character string specifying the new class.

Details

This function returns a copy of the object, and the original remains unchanged.

Value

A caugi object of the specified class.

See Also

Other operations: moralize(), skeleton()

Examples

```
cg <- caugi(A %-->% B, class = "UNKNOWN")
cg_dag <- mutate_caugi(cg, "DAG")
```

---

neighbors *Get neighbors of nodes in a* caugi

Description

Get neighbors of nodes in a caugi

Usage

```
neighbors(cg, nodes = NULL, index = NULL)

neighbours(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), nodes(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
neighbors(cg, "A") # "B"
neighbors(cg, index = 2) # "A" "C"
neighbors(cg, "B") # "A" "C"
neighbors(cg, c("B", "C"))
#> $B
#> [1] "A" "C"
#>
#> $C
#> [1] "B"
```

---

nodes *Get nodes or edges of a* caugi

Description

Get nodes or edges of a caugi

Usage

```
nodes(cg)

vertices(cg)

V(cg)
```

Arguments

cg	A caugi object.

Value

A data.table with a name column.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), parents(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  D,
  class = "DAG"
)
nodes(cg) # returns the data.table with nodes A, B, C, D
```

`parents` *Get parents of nodes in a* `caugi`

Description

Get parents of node in a graph. Note that not both nodes and index can be given.

Usage

```
parents(cg, nodes = NULL, index = NULL)
```

Arguments

`cg`	A `caugi` object.
`nodes`	A vector of node names, a vector of unquoted node names, or an expression combining these with + and `c()`.
`index`	A vector of node indexes.

Value

Either a character vector of node names (if a single node is requested) or a list of character vectors (if multiple nodes are requested).

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), same_nodes(), subgraph()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
parents(cg, "A") # NULL
parents(cg, index = 2) # "A"
parents(cg, "B") # "A"
parents(cg, c("B", "C"))
#> $B
#> [1] "A"
#>
#> $C
#> [1] "B"
```

`print` *Print a* `caugi`

Description

Print a `caugi`

Arguments

`x`	A `caugi` object.
`max_nodes`	Optional numeric; maximum number of node names to consider. If `NULL`, the method automatically prints as many as fit on one console line (plus a separate truncation line if needed).
`max_edges`	Optional numeric; maximum number of edges to consider. If `NULL`, the method automatically prints as many edges as fit on two console lines (plus a separate truncation line if needed).
`...`	Not used.

Value

The input `caugi` object, invisibly.

See Also

Other caugi methods: `length()`

Examples

```
cg <- caugi(A %-->% B, class = "DAG")
print(cg)
```

`register_caugi_edge` *Register a new edge type in the global registry.*

Description

Register a new edge type in the global registry.

Usage

```
register_caugi_edge(glyph, tail_mark, head_mark, class, symmetric = FALSE)
```

Arguments

`glyph`	A string representing the edge glyph (e.g., `"-->"`, `"<->"`).
`tail_mark`	One of "arrow", "tail", "circle", "other".
`head_mark`	One of "arrow", "tail", "circle", "other".
`class`	One of "directed","undirected","bidirected","partial".
`symmetric`	Logical.

Value

TRUE, invisibly.

See Also

Other registry: `registry`

Examples

```
# first, for reproducability, we reset the registry to default
reset_caugi_registry()

# create a new registry
reg <- caugi_registry()

# register an edge
register_caugi_edge(
  glyph = "<--",
  tail_mark = "arrow",
  head_mark = "tail",
  class = "directed",
  symmetric = FALSE
)

# now, this edge is available for caugi graphs:
cg <- caugi(A %-->% B, B %<--% C, class = "DAG")

# reset the registry to default
```

```
reset_caugi_registry()
```

`registry` `caugi` *edge registry*

Description

The `caugi` edge registry stores information about the different edge types that can be used in `caugi` graphs. It maps edge glyphs (e.g., `"-->"`, `"<->"`, `"o->"`, etc.) to their specifications, including tail and head marks, class, and symmetry. The registry allows for dynamic registration of new edge types, enabling users to extend the set of supported edges in `caugi`. It is implemented as a singleton, ensuring that there is a single global instance of the registry throughout the R session.

Usage

```
caugi_registry()

reset_caugi_registry()

seal_caugi_registry()
```

Details

The intented use of the `caugi` registry is mostly for advanced users and developers. The registry enables users who need to define their own custom edge types in `caugi` directly. . It currently mostly supports the *representation* of new edges, but for users that might want to represent reverse edges, this preserves correctness of reason over these edges.

Value

An `edge_registry` external pointer.

Functions

- `caugi_registry()`: Access the global edge registry, creating it if needed.
- `reset_caugi_registry()`: Reset the global edge registry to its default state.
- `seal_caugi_registry()`: Seal the global edge registry to prevent further modifications.

See Also

Other registry: `register_caugi_edge()`

Examples

```
# first, for reproducability, we reset the registry to default
reset_caugi_registry()

# create a new registry
reg <- caugi_registry()

# register an edge
register_caugi_edge(
  glyph = "<--",
  tail_mark = "arrow",
  head_mark = "tail",
  class = "directed",
  symmetric = FALSE
)

# now, this edge is available for caugi graphs:
cg <- caugi(A %-->% B, B %<--% C, class = "DAG")

# reset the registry to default
reset_caugi_registry()
```

---

same_nodes *Same nodes?*

Description

Check if two `caugi` objects have the same nodes.

Usage

```
same_nodes(cg1, cg2, throw_error = FALSE)
```

Arguments

`cg1`	A `caugi` object.
`cg2`	A `caugi` object.
`throw_error`	Logical; if `TRUE`, throws an error if the graphs do not have the same nodes.

Value

A logical indicating if the two graphs have the same nodes.

See Also

Other queries: `ancestors()`, `children()`, `descendants()`, `edge_types()`, `edges()`, `exogenous()`, `is_acyclic()`, `is_caugi()`, `is_cpdag()`, `is_dag()`, `is_empty_caugi()`, `is_pdag()`, `is_ug()`, `markov_blanket()`, `neighbors()`, `nodes()`, `parents()`, `subgraph()`

Examples

```
cg1 <- caugi(
  A %-->% B,
  class = "DAG"
)
cg2 <- caugi(
  A %-->% B + C,
  class = "DAG"
)
same_nodes(cg1, cg2) # FALSE
```

---

shd *Structural Hamming Distance*

Description

Compute the Structural Hamming Distance (SHD) between two graphs.

Usage

```
shd(cg1, cg2, normalized = FALSE)
```

Arguments

cg1	A caugi object.
cg2	A caugi object.
normalized	Logical; if TRUE, returns the normalized SHD.

Value

An integer representing the Hamming Distance between the two graphs, if normalized = FALSE, or a numeric between 0 and 1 if normalized = TRUE.

See Also

Other metrics: aid(), hd()

Examples

```
cg1 <- caugi(A %-->% B %-->% C, D %-->% C, class = "DAG")
cg2 <- caugi(A %-->% B %-->% C, D %---% C, class = "PDAG")
shd(cg1, cg2) # 1
```

skeleton *Get the skeleton of a graph*

Description

The skeleton of a graph is obtained by replacing all directed edges with undirected edges.

Usage

```
skeleton(cg)
```

Arguments

cg	A caugi object. Either a DAG or PDAG.

Details

This changes the graph from any class to an Undirected Graph (UG), also known as a Markov Graph.

Value

A caugi object representing the skeleton of the graph (UG).

See Also

Other operations: moralize(), mutate_caugi()

Examples

```
cg <- caugi(A %-->% B, class = "DAG")
skeleton(cg) # A --- B
```

subgraph *Get the induced subgraph*

Description

Get the induced subgraph

Usage

```
subgraph(cg, nodes = NULL, index = NULL)
```

Arguments

cg	A caugi object.
nodes	A vector of node names, a vector of unquoted node names, or an expression combining these with + and c().
index	A vector of node indexes.

Value

A new caugi that is a subgraph of the selected nodes.

See Also

Other queries: ancestors(), children(), descendants(), edge_types(), edges(), exogenous(), is_acyclic(), is_caugi(), is_cpdag(), is_dag(), is_empty_caugi(), is_pdag(), is_ug(), markov_blanket(), neighbors(), nodes(), parents(), same_nodes()

Examples

```
cg <- caugi(
  A %-->% B,
  B %-->% C,
  class = "DAG"
)
sub_cg <- subgraph(cg, c("B", "C"))
cg2 <- caugi(B %-->% C, class = "DAG")
all(nodes(sub_cg) == nodes(cg2)) # TRUE
all(edges(sub_cg) == edges(cg2)) # TRUE
```

Index