

Package: causalDisco (via r-universe)

July 20, 2026

Title Tools for Causal Discovery on Observational Data

Version 1.2.1

Description Tools for causal structure learning from observational data, with emphasis on temporally ordered variables. The package implements the Temporal Peter–Clark (TPC) algorithm (Petersen, Osler & Ekstrøm, 2021; <[doi:10.1093/aje/kwab087](https://doi.org/10.1093/aje/kwab087)>), the Temporal Greedy Equivalence Search (TGES) algorithm (Larsen, Ekstrøm & Petersen, 2025; <[doi:10.48550/arXiv.2502.06232](https://doi.org/10.48550/arXiv.2502.06232)>) and Temporal Fast Causal Inference (TFCI). It provides a unified framework for specifying background knowledge, which can be incorporated into the implemented algorithms from the R packages 'bnlearn' (Scutari, 2010; <[doi:10.18637/jss.v035.i03](https://doi.org/10.18637/jss.v035.i03)>) and 'pcalg' (Kalish et al., 2012; <[doi:10.18637/jss.v047.i11](https://doi.org/10.18637/jss.v047.i11)>), as well as the Java library 'Tetrad' (Scheines et al., 1998; <[doi:10.1207/s15327906mbr3301_3](https://doi.org/10.1207/s15327906mbr3301_3)>). The package further includes utilities for visualization, comparison, and evaluation of graph structures, facilitating performance evaluation and methodological studies.

License GPL-2

URL <https://github.com/disco-coders/causalDisco>,
<https://disco-coders.github.io/causalDisco/>

BugReports <https://github.com/disco-coders/causalDisco/issues>

Depends R (>= 4.2.0)

Imports bnlearn, caugi (>= 1.0.0), checkmate, cli, digest, dplyr, lifecycle, methods, pcalg, purrr, R6, rlang, S7, tibble, tidyr, tidyselect

Suggests knitr, mockery, quarto, RhpcBLASctl, rJava, rmarkdown, spelling, testthat (>= 3.0.0), withr, mice, micd

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Config/Needs/website quarto

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Bjarke Hautop Kristensen [aut, cre], Frederik Fabricius-Bjerre
[aut], Anne Helby Petersen [aut], Claus Thorn Ekstrøm [aut],
Tobias Ellegaard Larsen [ctb]

Maintainer Bjarke Hautop Kristensen <bjarke.kristensen@sund.ku.dk>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-07-20 18:00:02 UTC

RemoteUrl <https://github.com/cran/causalDisco>

RemoteRef HEAD

RemoteSha 095e4efb651d5f5922c8427b78d6742e5da30ef3

Contents

causalDisco-package	4
+Knowledge	5
add_exogenous	6
add_tier	8
add_to_tier	9
add_vars	11
as_bnlearn_knowledge	12
as_disco	14
as_pcalg_constraints	14
as_tetrad_knowledge	16
BnlearnSearch	17
boss	22
boss_fci	24
cat_data	26
cat_data_mcar	27
cat_ord_data	28
CausalDiscoSearch	29
confusion	34
convert_tiers_to_forbidden	36
cor_test	37
deparse_knowledge	37
disco	38
distribute_engine_args	40
evaluate	41
f1_score	42
false_omission_rate	43
fci	44

fdr	46
forbid_edge	47
gl_score	48
ges	50
get_tiers	52
gfcf	53
grasp	55
grasp_fci	57
gs	60
iamb-family	61
install_tetrad	64
knowledge	65
knowledge_to_caugi	70
list_registered_engines	71
list_registered_tetrad_algorithms	71
make_method	72
make_runner	73
make_tikz	73
mix_data	76
new_disco_method	77
nlsy97	78
npv	79
num_data	80
num_data_latent	81
pc	82
PcalgSearch	84
plot	89
plot.Disco	90
plot.Knowledge	92
precision	95
print.Disco	96
print.Knowledge	97
recall	98
reg_test	99
register_engine	100
register_tetrad_algorithm	101
remove_edge	101
remove_tiers	102
remove_vars	103
reorder_tiers	105
reposition_tier	106
require_edge	107
reset_engine_registry	108
reset_tetrad_alg_registry	109
rfci	109
seq_tiers	111
set_knowledge	112
sim_dag	113

sim_data	114
sp_fci	115
specificity	117
summary.Disco	118
summary.Knowledge	119
TetradSearch	119
tfci	134
tfci_run	135
tges	138
tges_run	139
tpc	141
tpc_example	143
tpc_run	143
unfreeze	146
verify_tetrad	147

Index	148
--------------	------------

causalDisco-package	<i>causalDisco: Causal Discovery in R</i>
---------------------	---

Description

Tools for causal structure learning from observational data, with emphasis on temporally ordered variables. The package implements the Temporal Peter–Clark (TPC) algorithm (Petersen, Osler & Ekstrøm, 2021; [doi:10.1093/aje/kwab087](https://doi.org/10.1093/aje/kwab087)), the Temporal Greedy Equivalence Search (TGES) algorithm (Larsen, Ekstrøm & Petersen, 2025; [doi:10.48550/arXiv.2502.06232](https://doi.org/10.48550/arXiv.2502.06232)) and Temporal Fast Causal Inference (TFCI). It provides a unified framework for specifying background knowledge, which can be incorporated into the implemented algorithms from the R packages 'bnlearn' (Scutari, 2010; [doi:10.18637/jss.v035.i03](https://doi.org/10.18637/jss.v035.i03)) and 'pcalg' (Kalish et al., 2012; [doi:10.18637/jss.v047.i11](https://doi.org/10.18637/jss.v047.i11)), as well as the Java library 'Tetrad' (Scheines et al., 1998; [doi:10.1207/s15327906mbr3301_3](https://doi.org/10.1207/s15327906mbr3301_3)). The package further includes utilities for visualization, comparison, and evaluation of graph structures, facilitating performance evaluation and methodological studies.

System requirements (optional)

If you want to use algorithms from the Java library Tetrad, a Java JDK (≥ 21) is required. The Tetrad .jar file can be downloaded using `install_tetrad()`.

Author(s)

Maintainer: Bjarke Hautop Kristensen <bjarke.kristensen@sund.ku.dk>

Authors:

- Bjarke Hautop Kristensen <bjarke.kristensen@sund.ku.dk>
- Frederik Fabricius-Bjerre <frederik@fabriciusbjerre.dk>
- Anne Helby Petersen <ahpe@sund.ku.dk>

- Claus Thorn Ekstrøm <ekstrom@sund.ku.dk>

Other contributors:

- Tobias Ellegaard Larsen <tobias.ellegaard@sund.ku.dk> [contributor]

See Also

Useful links:

- <https://github.com/disco-coders/causalDisco>
- <https://disco-coders.github.io/causalDisco/>
- Report bugs at <https://github.com/disco-coders/causalDisco/issues>

+.Knowledge

Merge Knowledge Objects

Description

Merge Knowledge Objects

Usage

```
## S3 method for class 'Knowledge'
kn1 + kn2
```

Arguments

kn1	A Knowledge object.
kn2	Another Knowledge object.

See Also

Other knowledge helpers: [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```
# Create two Knowledge objects
kn1 <- knowledge(
  tier(
    1 ~ V1,
    2 ~ V2
  ),
  V1 %-->% V2
)
```

```

kn2 <- knowledge(
  tier(3 ~ V3),
  V2 %!-->% V3
)

kn_merged <- kn1 + kn2

# Error paths
# Merging with conflicting tier information

kn1 <- knowledge(
  tier(
    1 ~ V1,
    2 ~ V2
  )
)

kn2 <- knowledge(
  tier(3 ~ V2)
)

try(kn1 + kn2)

kn2 <- knowledge(
  tier(1 ~ V1 + V2)
)

try(kn1 + kn2)

# Required / forbidden violations

kn1 <- knowledge(
  V1 %!-->% V2
)

kn2 <- knowledge(
  V1 %-->% V2
)

try(kn1 + kn2)

```

add_exogenous

Add Exogenous Variables to Knowledge

Description

Adds variables that cannot have incoming edges (exogenous nodes). Every possible incoming edge to these nodes is automatically forbidden. This is equivalent to writing `forbidden(everything() ~ vars)`.

Usage

```
add_exogenous(kn, vars)
```

```
add_exo(kn, vars)
```

Arguments

kn	A Knowledge object.
vars	Tidysselect specification or character vector of variables.

Value

Updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
kn1$frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  )
)
```

```

    ),
    child_x1 %-->% youth_x3,
    child_x2 %!-->% youth_x4,
    exo(child_x1) # synonym: exogenous()
  )

  print(identical(kn1, kn2))

  # cannot require an edge against tier direction
  try(
    kn1 |> require_edge(oldage_x6 ~ child_x1)
  )

  # cannot forbid and require same edge
  try(
    kn1 |> forbid_edge(child_x1 ~ youth_x3)
  )

```

add_tier

Add a Tier to Knowledge

Description

Adds a new tier to the Knowledge object, either at the start, end, or before/after an existing tier.

Usage

```
add_tier(kn, tier, before = NULL, after = NULL)
```

Arguments

kn	A Knowledge object.
tier	Bare symbol / character (label) or numeric literal.
before, after	Optional anchor relative to an existing tier label, tier index, or variable. Once the Knowledge object already has ≥ 1 tier, you must supply exactly one of these.

Value

The updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```

data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
kn1$frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!->% youth_x4,
  exo(child_x1) # synonym: exogenous()
)

print(identical(kn1, kn2))

# cannot require an edge against tier direction
try(
  kn1 |> require_edge(oldage_x6 ~ child_x1)
)

# cannot forbid and require same edge
try(
  kn1 |> forbid_edge(child_x1 ~ youth_x3)
)

```

Description

Add Variables to a Tier in Knowledge

Usage

```
add_to_tier(kn, ...)
```

Arguments

kn	A Knowledge object.
...	One or more two-sided formulas <code>tier ~ vars</code> .

Value

The updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
kn1$frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
```

```

    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!-->% youth_x4,
  exo(child_x1) # synonym: exogenous()
)

print(identical(kn1, kn2))

# cannot require an edge against tier direction
try(
  kn1 |> require_edge(oldage_x6 ~ child_x1)
)

# cannot forbid and require same edge
try(
  kn1 |> forbid_edge(child_x1 ~ youth_x3)
)

```

add_vars

Add Variables to Knowledge

Description

Adds variables to the Knowledge object. If the object is frozen, an error is thrown if any of the variables are not present in the data frame provided to the object.

Usage

```
add_vars(kn, vars)
```

Arguments

kn	A Knowledge object.
vars	A character vector of variable names to add.

Value

The updated Knowledge object.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```

data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
kn1$frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!->% youth_x4,
  exo(child_x1) # synonym: exogenous()
)

print(identical(kn1, kn2))

# cannot require an edge against tier direction
try(
  kn1 |> require_edge(oldage_x6 ~ child_x1)
)

# cannot forbid and require same edge
try(
  kn1 |> forbid_edge(child_x1 ~ youth_x3)
)

```

Description

Converts a Knowledge object to a list of two data frames, namely `whitelist` and `blacklist`, which can be used as arguments for **bnlearn** algorithms. The `whitelist` contains all required edges, and the `blacklist` contains all forbidden edges. Tiers will be made into forbidden edges before running the conversion.

Usage

```
as_bnlearn_knowledge(kn)
```

Arguments

`kn` A Knowledge object. Must have no tier information.

Value

A list with two elements, `whitelist` and `blacklist`, each a data frame containing the edges in a from, to format.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# produce whitelist/blacklist data frame for bnlearn
data(tpc_example)

kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3
)

bnlearn_kn <- as_bnlearn_knowledge(kn)
print(bnlearn_kn)
```

as_disco

Wrap a Graph as a Disco Object

Description

Wraps a causal graph object (e.g. a [caugi::caugi](#) graph, or a graph object from **pcalg** or **bnlearn**) and a Knowledge object into a Disco object, which is the output object of causal discovery methods used in **causalDisco**.

Usage

```
as_disco(graph, kn = knowledge(), class = "PDAG")
```

Arguments

graph	A causal graph object
kn	A Knowledge object. Default is an empty Knowledge object.
class	A string describing the graph class.

Details

The conversion from any graph type to a [caugi::caugi](#) is handled by the **caugi** package.

Value

A Disco object containing a [caugi::caugi](#) and a Knowledge object in a list.

See Also

[caugi::caugi\(\)](#)

Other Extending causalDisco: [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_a](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

as_pcalg_constraints

Convert Knowledge to pcalg Knowledge

Description

pcalg only supports *undirected* (symmetric) background constraints:

- **fixed_gaps** - forbidding edges (zeros enforced)

Usage

```
as_pcalg_constraints(
  kn,
  labels = kn$vars$var,
  directed_as_undirected = lifecycle::deprecated()
)
```

Arguments

kn	A Knowledge object. Must have no tier information.
labels	Character vector of all variable names, in the exact order of your data columns. Every variable referenced by an edge in kn must appear here.
directed_as_undirected	[Deprecated] This argument no longer has any effect and will be removed in a future release. Specify directed edges in both directions in <code>knowledge()</code> instead.

Details

This function takes a Knowledge object (with only forbidden edges, no tiers) and returns the logical constraint matrix in the exact variable order you supply.

Required edges in a Knowledge object are directed statements. **pcalg** constraints are adjacency-level only, so a required edge cannot be honored; any required edges are dropped with a warning.

Value

A list with one element, `fixed_gaps`: an $n \times n$ logical matrix corresponding to the **pcalg** `fixedGaps` argument.

Errors

- If the Knowledge object contains tiered knowledge.
- If any forbidden edge lacks its symmetrical counterpart.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# pcalg supports undirected constraints; build a tierless knowledge and convert
data(tpc_example)

kn <- knowledge(
```

```

    tpc_example,
    child_x1 %!-->% youth_x3,
    youth_x3 %!-->% child_x1
  )

pc_constraints <- as_pcalg_constraints(kn)
print(pc_constraints)

# error paths
# using tiers
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %!-->% youth_x3
)

try(as_pcalg_constraints(kn), silent = TRUE) # fails due to tiers

# using directed knowledge
kn <- knowledge(
  tpc_example,
  child_x1 %!-->% youth_x3
)

try(as_pcalg_constraints(kn), silent = TRUE) # fails due to directed knowledge

```

as_tetrad_knowledge *Convert Knowledge to Tetrad Knowledge*

Description

Converts a Knowledge object to a Tetrad `edu.cmu.tetrad.data.Knowledge`. This requires `rJava`. This is used internally, when setting knowledge with `set_knowledge()` for methods using the Tetrad engine. `set_knowledge()` is used internally, when using the `disco()` function with knowledge given.

Usage

```
as_tetrad_knowledge(kn)
```

Arguments

kn A Knowledge object.

Value

A Java `edu.cmu.tetrad.data.Knowledge` object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# convert to Tetrad Knowledge via rJava
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3
)

jk <- try(as_tetrad_knowledge(kn)) # will run only if rJava/JVM available
try(print(jk)) # prints a Java reference if successful
```

BnlearnSearch

*R6 Interface to bnlearn Search Algorithms***Description**

A wrapper that lets you drive **bnlearn** algorithms within the **causalDisco** framework. For arguments to the test, score, and algorithm, see the **bnlearn** documentation.

Value

An R6 object with the methods documented below.

Public fields

data A `data.frame` holding the data set currently attached to the search object. Can be set with `set_data()`.

score Character scalar naming the score function used in **bnlearn**. Can be set with `$set_score()`. Kebab-case score names (as used in **bnlearn**, e.g. "pred-loglik") are also accepted and automatically translated to snake_case. Recognised values are:

Continuous - Gaussian

- "aic_g", "bic_g", "ebic_g", "loglik_g", "pred_loglik_g" - gaussian versions of the respective scores for discrete data.

- "bge" - Gaussian posterior density.
- "nal_g" - node-average log-likelihood.
- "pnal_g" - penalised node-average log-likelihood.

Discrete – categorical

- "aic" - Akaike Information Criterion.
- "bdla" - locally averaged BDE.
- "bde" - Bayesian Dirichlet equivalent (uniform).
- "bds" - Bayesian Dirichlet score.
- "bic" - Bayesian Information Criterion.
- "ebic" - Extended BIC.
- "fnml" - factorised NML.
- "k2" - K2 score.
- "loglik" - log-likelihood.
- "mbde" - modified BDE.
- "nal" - node-average log-likelihood.
- "pnal" - penalised node-average log-likelihood.
- "pred_loglik" - predictive log-likelihood.
- "qnml" - quotient NML.

Mixed Discrete/Gaussian

- "aic_cg", "bic_cg", "ebic_cg", "loglik_cg", "nal_cg", "pnal_cg", "pred_loglik_cg"
- conditional Gaussian versions of the respective scores for discrete data.

`test` Character scalar naming the conditional-independence test passed to **bnlearn**. Can be set with `$set_score()`. Kebab-case test names (as used in **bnlearn**, e.g. "mi-adf") are also accepted and automatically translated to snake_case. Recognised values are:

Continuous - Gaussian

- "cor" – Pearson correlation
- "fisher_z" / "zf" – Fisher Z test
- "mc_cor" – Monte Carlo Pearson correlation
- "mc_mi_g" – Monte Carlo mutual information (Gaussian)
- "mc_zf" – Monte Carlo Fisher Z
- "mi_g" – mutual information (Gaussian)
- "mi_g_sh" – mutual information (Gaussian, shrinkage)
- "smc_cor" – sequential Monte Carlo Pearson correlation
- "smc_mi_g" – sequential Monte Carlo mutual information (Gaussian)
- "smc_zf" – sequential Monte Carlo Fisher Z

Discrete – categorical

- "mc_mi" – Monte Carlo mutual information
- "mc_x2" – Monte Carlo chi-squared
- "mi" – mutual information
- "mi_adf" – mutual information with adjusted d.f.
- "mi_sh" – mutual information (shrinkage)

- "smc_mi" – sequential Monte Carlo mutual information
- "smc_x2" – sequential Monte Carlo chi-squared
- "sp_mi" – semi-parametric mutual information
- "sp_x2" – semi-parametric chi-squared
- "x2" – chi-squared
- "x2_adf" – chi-squared with adjusted d.f.

Discrete – ordered factors

- "jt" – Jonckheere–Terpstra
- "mc_jt" – Monte Carlo Jonckheere–Terpstra
- "smc_jt" – sequential Monte Carlo Jonckheere–Terpstra

Mixed Discrete/Gaussian

- "mi_cg" / "conditional_gaussian" – mutual information (conditional Gaussian)

For Monte Carlo tests, set the number of permutations using the `B` argument.

`alg` Function generated by `$set_alg()` that runs a structure-learning algorithm from **bnlearn**. Periodic case alg names (as used in **bnlearn**, e.g. "fast.iamb") are also accepted and automatically translated to snake_case. Recognised values are:

Constraint-based

- "fast_iamb" – Fast-IAMB algorithm. See `fast_iamb()` and the underlying `bnlearn::fast.iamb()`.
- "gs" – Grow-Shrink algorithm. See `gs()` and the underlying `bnlearn::gs()`.
- "iamb" – Incremental Association Markov Blanket algorithm. See `iamb()` and the underlying `bnlearn::iamb()`.
- "iamb_fdr" – IAMB with FDR control algorithm. See `iamb_fdr()` and the underlying `bnlearn::iamb.fdr()`.
- "inter_iamb" – Interleaved-IAMB algorithm. See `inter_iamb()` and the underlying `bnlearn::inter.iamb()`.
- "pc" – PC-stable algorithm. See `pc()` and the underlying `bnlearn::pc.stable()`.

`params` A list of extra tuning parameters stored by `set_params()` and spliced into the learner call.

`knowledge` A list with elements `whitelist` and `blacklist` containing prior-knowledge constraints added via `set_knowledge()`.

Methods

Public methods:

- `BnlearnSearch$new()`
- `BnlearnSearch$set_params()`
- `BnlearnSearch$set_data()`
- `BnlearnSearch$set_test()`
- `BnlearnSearch$set_score()`
- `BnlearnSearch$set_alg()`
- `BnlearnSearch$set_knowledge()`
- `BnlearnSearch$run_search()`
- `BnlearnSearch$clone()`

`BnlearnSearch$new()`: Constructor for the `BnlearnSearch` class.

Usage:

```
BnlearnSearch$new()
```

`BnlearnSearch$set_params()`: Set the parameters for the search algorithm.

Usage:

```
BnlearnSearch$set_params(params)
```

Arguments:

`params` A parameter to set.

`BnlearnSearch$set_data()`: Set the data for the search algorithm.

Usage:

```
BnlearnSearch$set_data(data)
```

Arguments:

`data` A data frame containing the data to use for the search.

`BnlearnSearch$set_test()`: Set the conditional-independence test to use in the search algorithm.

Usage:

```
BnlearnSearch$set_test(method, alpha = 0.05)
```

Arguments:

`method` **[Experimental]**

A string specifying the type of test to use. Can also be a user-defined function with signature `function(x, y, z, data, args)`, where `x` and `y` are the variables being tested for independence, `z` is the conditioning set, `data` is the dataset, and `args` is a list of additional arguments. The function should return the test statistic and the p-value. See [bnlearn::ci.test\(\)](#) for more details.

EXPERIMENTAL: user-defined tests syntax are subject to change.

`alpha` Significance level for the test.

`BnlearnSearch$set_score()`: Set the score function for the search algorithm.

Usage:

```
BnlearnSearch$set_score(method)
```

Arguments:

`method` Character naming the score function to use.

`BnlearnSearch$set_alg()`: Set the causal discovery algorithm to use.

Usage:

```
BnlearnSearch$set_alg(method, args = NULL)
```

Arguments:

`method` Character naming the algorithm to use.

`args` A list of additional arguments to pass to the algorithm.

`BnlearnSearch$set_knowledge()`: Set the prior knowledge for the search algorithm using a Knowledge object.

Usage:

```
BnlearnSearch$set_knowledge(knowledge_obj)
```

Arguments:

`knowledge_obj` A Knowledge object containing prior knowledge.

`BnlearnSearch$run_search()`: Run the search algorithm on the currently set data.

Usage:

```
BnlearnSearch$run_search(data = NULL)
```

Arguments:

`data` A data frame containing the data to use for the search. If NULL, the currently set data will be used, i.e. `self$data`.

`BnlearnSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
BnlearnSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### bnlearn_search R6 class examples ###

# Generally, we do not recommend using the R6 classes directly, but rather
# use the disco() or any method function, for example pc(), instead.

# Load data
data(num_data)

# Recommended:
my_pc <- pc(engine = "bnlearn", test = "fisher_z", alpha = 0.05)
result <- my_pc(num_data)

# or
result <- disco(data = num_data, method = my_pc)

plot(result)

# Example with detailed settings:
my_pc2 <- pc(
  engine = "bnlearn",
  test = "mi_g",
  alpha = 0.01
)
disco(data = num_data, method = my_pc2)

# With knowledge
```

```

kn <- knowledge(
  num_data,
  starts_with("X") %-->% Y
)

disco(data = num_data, method = my_pc2, knowledge = kn)

# Using additional test args (bootstrap samples)

my_iamb <- iamb(
  engine = "bnlearn",
  test = "mc_zf",
  alpha = 0.05,
  B = 100
)

disco(data = num_data, method = my_iamb)

# Using R6 class:
s <- BnlearnSearch$new()

s$set_data(num_data)
s$set_test(method = "fisher_z", alpha = 0.05)
s$set_alg("pc")

g <- s$run_search()

plot(g)

```

boss

BOSS Algorithm for Causal Discovery

Description

Run the BOSS (Best Order Score Search) algorithm for causal discovery using one of several engines.

Usage

```
boss(engine = "tetrad", score, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
score	Character; name of the scoring function to use.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- `caugi` A `caugi::caugi` object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Andrews, B., Ramsey, J., Sánchez-Romero, R., Camchong, J., & Kummerfeld, E. (2023, December). Fast scalable and accurate discovery of DAGs using the Best Order Score Search and Grow-Shrink Trees. *Advances in Neural Information Processing Systems*, 36, 63945-63956. Epub 2024 May 30. PMID: 39280091; PMCID: PMC11393735.

See Also

Other causal discovery algorithms: [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  boss_tetrad <- boss(engine = "tetrad", score = "sem_bic")
  disco(tpc_example, boss_tetrad)

  # or using boss_tetrad directly
  boss_tetrad(tpc_example)
}

#### With tier knowledge ####
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
```

```

    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    oldage ~ tidyselect::starts_with("oldage")
  )
)

# Recommended path using disco()
boss_tetrad <- boss(engine = "tetrad", score = "sem_bic")
disco(tpc_example, boss_tetrad, knowledge = kn)

# or using boss_tetrad directly
boss_tetrad <- boss_tetrad |> set_knowledge(kn)
boss_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  boss_tetrad <- boss(
    engine = "tetrad",
    score = "gic",
    num_starts = 2,
    use_bes = FALSE,
    use_data_order = FALSE,
    output_cpdag = FALSE
  )
  disco(tpc_example, boss_tetrad)
}

```

boss_fci

*BOSS-FCI Algorithm for Causal Discovery***Description**

Run the Best Order Score Search Fast Causal Inference algorithm for causal discovery using one of several engines.

Usage

```
boss_fci(engine = "tetrad", score, test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
score	Character; name of the scoring function to use.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- `caugi` A `caugi::caugi` object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

References

Ramsej, J., Andrews, B., Sprites, P. (2025). Efficient Latent Variable Causal Discovery: Combining Score Search and Targeted Testing. [doi:10.48550/arXiv.2510.04263](#).

See Also

Other causal discovery algorithms: [boss\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  boss_fci_tetrad <- boss_fci(
    engine = "tetrad",
    score = "sem_bic",
    test = "fisher_z"
  )
  disco(tpc_example, boss_fci_tetrad)

  # or using boss_fci_tetrad directly
  boss_fci_tetrad(tpc_example)
}

#### With tier knowledge ####
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
```

```

kn <- knowledge(
  tpc_example,
  tier(
    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    oldage ~ tidyselect::starts_with("oldage")
  )
)

# Recommended path using disco()
boss_fci_tetrad <- boss_fci(
  engine = "tetrad",
  score = "sem_bic",
  test = "fisher_z"
)
disco(tpc_example, boss_fci_tetrad, knowledge = kn)

# or using boss_fci_tetrad directly
boss_fci_tetrad <- boss_fci_tetrad |> set_knowledge(kn)
boss_fci_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  boss_fci_tetrad <- boss_fci(
    engine = "tetrad",
    score = "poisson_prior",
    test = "rank_independence",
    depth = 3,
    max_disc_path_length = 5,
    use_bes = FALSE,
    use_heuristic = FALSE,
    complete_rule_set_used = FALSE,
    guarantee_pag = TRUE
  )
  disco(tpc_example, boss_fci_tetrad)
}

```

cat_data

Simulated Categorical Data

Description

A dataset created by discretizing the continuous num_data into 5 categorical levels per variable.

Usage

```
cat_data
```

Format

A data.frame with 1000 rows and 5 variables.

X1 Categorical version of num_data\$X1, with 5 levels a–e.

X2 Categorical version of num_data\$X2, with 5 levels a–e.

X3 Categorical version of num_data\$X3, with 5 levels a–e.

Z Categorical version of num_data\$Z, with 5 levels a–e.

Y Categorical version of num_data\$Y, with 5 levels a–e.

Details

The R code used to generate this dataset is as follows:

```
data(num_data)
cat_data <- as.data.frame(
  lapply(num_data, function(x) cut(x, breaks = 5, labels = letters[1:5]))
)
```

See Also

[num_data](#)

Examples

```
data(cat_data)
head(cat_data)
```

cat_data_mcar

Simulated Categorical Data with MCAR

Description

A dataset based on cat_data where some values are randomly removed to simulate MCAR.

Usage

```
cat_data_mcar
```

Format

A data.frame with 1000 rows and 5 variables.

X1 Categorical, 100 values set to NA (MCAR).

X2 Categorical, 50 values set to NA (MCAR).

X3 Categorical, 200 values set to NA (MCAR).

Z Categorical, no missing values.

Y Categorical, no missing values.

Details

The R code used to generate this dataset is as follows:

```
data(cat_data)
cat_data_mcar <- cat_data
n <- nrow(cat_data_mcar)
set.seed(1405)
cat_data_mcar$X1[sample(1:n, 100)] <- NA
cat_data_mcar$X2[sample(1:n, 50)] <- NA
cat_data_mcar$X3[sample(1:n, 200)] <- NA
```

See Also

[cat_data](#)

Examples

```
data(cat_data_mcar)
head(cat_data_mcar)
```

cat_ord_data	<i>Simulated Ordered Categorical Data</i>
--------------	---

Description

A dataset created by discretizing the continuous num_data into 5 ordered categorical levels per variable.

Usage

cat_ord_data

Format

- A data.frame with 1000 rows and 5 variables.
- X1** Categorical version of num_data\$X1, with 5 ordered levels a–e.
 - X2** Categorical version of num_data\$X2, with 5 ordered levels a–e.
 - X3** Categorical version of num_data\$X3, with 5 ordered levels a–e.
 - Z** Categorical version of num_data\$Z, with 5 ordered levels a–e.
 - Y** Categorical version of num_data\$Y, with 5 ordered levels a–e.

Details

The R code used to generate this dataset is as follows:

```
data(num_data)
cat_ord_data <- as.data.frame(
  lapply(num_data, function(x) cut(x, breaks = 5, labels = letters[1:5], ordered_result = TRUE))
)
```

See Also

[num_data](#)

Examples

```
data(cat_ord_data)
head(cat_ord_data)
```

CausalDiscoSearch

R6 Interface to causalDisco Search Algorithms

Description

This class implements the search algorithms from the **causalDisco** package, which wraps and adds temporal order to **pcalg** algorithms. It allows to set the data, sufficient statistics, test, score, and algorithm.

Public fields

data A `data.frame` holding the data set currently attached to the search object. Can be set with `set_data()`.

score A function that will be used to build the score, when data is set. Can be set with `$set_score()`. Recognized values are:

- "tbic" - Temporal BIC score for Gaussian data. See [TemporalBIC](#).
- "tbdeu" - Temporal BDeu score for discrete data. See [TemporalBDeu](#).

test A function that will be used to test independence. Can be set with `$set_test()`. Recognized values are:

- "fisher_z" - Fisher Z test for Gaussian data. See [cor_test\(\)](#).
- "fisher_z_twd" - Fisher Z test for Gaussian data with test-wise deletion. See [micd::gaussCItd\(\)](#).
- "fisher_z_mi" - Fisher Z test for Gaussian data with multiple imputation. See [micd::gaussCItestMI\(\)](#).
- "reg" - Regression test for discrete or binary data. See [reg_test\(\)](#).
- "g_square" - G square test for discrete data. See [pcalg::binCItest\(\)](#) and [pcalg::disCItest\(\)](#).
- "g_square_twd" - G square test for discrete data with test-wise deletion. See [micd::disCItd\(\)](#).
- "g_square_mi" - G square test for discrete data with multiple imputation. See [micd::disMItest\(\)](#).

- "conditional_gaussian" - Test for conditional independence in mixed data. See `micd::mixCIttest()`.
- "conditional_gaussian_twd" - Test for conditional independence in mixed data with test-wise deletion. See `micd::mixCItwd()`.
- "conditional_gaussian_mi" - Test for conditional independence in mixed data with multiple imputation. See `micd::mixMItest()`.

`alg` A function that will be used to run the search algorithm. Can be set with `$set_alg()`. Recognized values are:

- "tfci" - TFCI algorithm. See `tfci()`.
- "tges" - TGES algorithm. See `tges()`.
- "tpc" - TPC algorithm. See `tpc()`.

Can also be a user-defined function; see `$set_alg()` for details.

`params` A list of parameters for the test and algorithm. Can be set with `$set_params()`. TODO: not secure yet in terms of distributing arguments. Use with caution.

`suff_stat` Sufficient statistic. The format and contents of the sufficient statistic depends on which test is being used.

`knowledge` A Knowledge object holding background knowledge.

Methods

Public methods:

- `CausalDiscoSearch$new()`
- `CausalDiscoSearch$set_params()`
- `CausalDiscoSearch$set_data()`
- `CausalDiscoSearch$set_suff_stat()`
- `CausalDiscoSearch$set_test()`
- `CausalDiscoSearch$set_score()`
- `CausalDiscoSearch$set_alg()`
- `CausalDiscoSearch$set_knowledge()`
- `CausalDiscoSearch$run_search()`
- `CausalDiscoSearch$clone()`

`CausalDiscoSearch$new()`: Constructor for the `CausalDiscoSearch` class.

Usage:

```
CausalDiscoSearch$new()
```

`CausalDiscoSearch$set_params()`: Sets the parameters for the test and algorithm.

Usage:

```
CausalDiscoSearch$set_params(params)
```

Arguments:

`params` A list of parameters to set.

`CausalDiscoSearch$set_data()`: Sets the data for the search algorithm.

Usage:

```
CausalDiscoSearch$set_data(data, set_suff_stat = TRUE)
```

Arguments:

`data` A `data.frame` or a matrix containing the data.

`set_suff_stat` Logical; whether to set the sufficient statistic.

`CausalDiscoSearch$set_suff_stat()`: Sets the sufficient statistic for the data.

Usage:

```
CausalDiscoSearch$set_suff_stat()
```

`CausalDiscoSearch$set_test()`: Sets the test for the search algorithm.

Usage:

```
CausalDiscoSearch$set_test(
  method,
  alpha = 0.05,
  suff_stat_fun = NULL,
  args = NULL
)
```

Arguments:

`method` **[Experimental]**

A string specifying the type of test to use.

Can also be a user-defined function with signature `function(x, y, conditioning_set, suff_stat)`, where `x` and `y` are the variables being tested for independence, `conditioning_set` is the conditioning set, and `suff_stat` is the sufficient statistic for the test. If a user-defined function is provided, then `suff_stat_fun` must also be provided, which is a function that should take the data as input and returns a sufficient statistic for the test. Optionally, the signature of the user-defined test function can also include an `args` parameter, which is a list of additional arguments to pass to the test function. If `args` is provided, then the test function should have the signature `function(x, y, conditioning_set, suff_stat, args)`, and the `args` parameter will be passed to the test function.

EXPERIMENTAL: user-defined tests syntax are subject to change.

`alpha` Significance level for the test.

`suff_stat_fun` A function that takes the data as input and returns a sufficient statistic for the test. Only needed if `method` is a user-defined function.

`args` A list of additional arguments to pass to the test. Only needed if `method` is a user-defined function with an `args` parameter in its signature.

`CausalDiscoSearch$set_score()`: Sets the score for the search algorithm.

Usage:

```
CausalDiscoSearch$set_score(method, params = list())
```

Arguments:

`method` A string specifying the type of score to use.

`params` A list of parameters to pass to the score function.

`CausalDiscoSearch$set_alg()`: Sets the algorithm for the search.

Usage:

`CausalDiscoSearch$set_alg(method, type = c("constraint", "score"), args = NULL)`

Arguments:

`method` **[Experimental]**

A string specifying the type of algorithm to use.

Can also be a user-defined function implementing a full search algorithm. Its required signature depends on type:

- `type = "constraint"` (default) - signature `function(data, knowledge, suff_stat, ...)`, analogous to `tpc_run()` and `tfci_run()`.
- `type = "score"` - signature `function(score, ...)`, analogous to `tges_run()`.

Optionally, the function's signature can also include an `args` parameter, which is a list of additional arguments to pass to the algorithm function. If `args` is provided, the function should have the signature `function(data, knowledge, suff_stat, args)` (or `function(score, args)` for `type = "score"`), and the `args` parameter will be passed to the algorithm function.

EXPERIMENTAL: user-defined algorithm syntax is subject to change.

`type` Only used if `method` is a user-defined function. One of `"constraint"` (default) or `"score"`, specifying whether the algorithm is constraint-based (called with `data/knowledge/suff_stat`) or score-based (called with `score`).

`args` A list of additional arguments to pass to the algorithm. Only needed if `method` is a user-defined function with an `args` parameter in its signature.

`CausalDiscoSearch$set_knowledge()`: Sets the background knowledge for the search with a `Knowledge` object.

Usage:

```
CausalDiscoSearch$set_knowledge(
  kn,
  directed_as_undirected = lifecycle::deprecated()
)
```

Arguments:

`kn` A `Knowledge` object.

`directed_as_undirected` **[Deprecated]** This argument no longer has any effect and will be removed in a future release. Directed forbidden edges are now resolved using tier information, or must be specified in both directions.

`CausalDiscoSearch$run_search()`: Runs the search algorithm on the data.

Usage:

```
CausalDiscoSearch$run_search(data = NULL, set_suff_stat = TRUE)
```

Arguments:

`data` A `data.frame` or a matrix containing the data.

`set_suff_stat` Logical; whether to set the sufficient statistic

`CausalDiscoSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CausalDiscoSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[knowledge\(\)](#).

Examples

Generally, we do not recommend using the R6 classes directly, but rather
use the disco() or any method function, for example pc(), instead.

```
data(tpc_example)
```

```
# background knowledge (tiered knowledge)
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  )
)
```

```
# Recommended (TPC example):
my_tpc <- tpc(engine = "causalDisco", test = "fisher_z", alpha = 0.05)
result <- disco(data = tpc_example, method = my_tpc, knowledge = kn)
plot(result)
```

```
# or
my_tpc <- my_tpc |>
  set_knowledge(kn)
result <- my_tpc(tpc_example)
plot(result)
```

Using R6 class:

```
# --- Constraint-based: TPC -----
s_tpc <- CausalDiscoSearch$new()
s_tpc$set_params(list(verbose = FALSE))
s_tpc$set_test("fisher_z", alpha = 0.2)
s_tpc$set_alg("tpc")
s_tpc$set_knowledge(kn)
s_tpc$set_data(tpc_example)
res_tpc <- s_tpc$run_search()
print(res_tpc)
```

```
# Switch to TFCI on the same object (reuses suff_stat/test)
s_tpc$set_alg("tfci")
res_tfci <- s_tpc$run_search()
print(res_tfci)
```

```
# --- Score-based: TGES -----
s_tges <- CausalDiscoSearch$new()
s_tges$set_score("tbic") # Gaussian temporal score
s_tges$set_alg("tges")
```

```

s_tges$set_data(tpc_example, set_suff_stat = FALSE) # suff stat not used for TGES
s_tges$set_knowledge(kn)
res_tges <- s_tges$run_search()
print(res_tges)

# --- Custom (user-defined) algorithm: constraint-based -----
my_alg <- function(data, knowledge, suff_stat) {
  tpc_run(
    data = data,
    knowledge = knowledge,
    suff_stat = suff_stat,
    test = reg_test
  )
}

s_custom <- CausalDiscoSearch$new()
s_custom$set_test("reg")
s_custom$set_knowledge(kn)
s_custom$set_alg(my_alg) # type = "constraint" by default
s_custom$set_data(tpc_example)
res_custom <- s_custom$run_search()
print(res_custom)

# --- Intentional error demonstrations -----

# run_search() without setting an algorithm
try(CausalDiscoSearch$new()$run_search(tpc_example))

# set_suff_stat() requires data and test first
s_err <- CausalDiscoSearch$new()
try(s_err$set_suff_stat()) # no data & no test
s_err$set_data(tpc_example, set_suff_stat = FALSE)
try(s_err$set_suff_stat()) # no test

# unknown test / score / algorithm
try(CausalDiscoSearch$new()$set_test("not_a_test"))
try(CausalDiscoSearch$new()$set_score("not_a_score"))
try(CausalDiscoSearch$new()$set_alg("not_an_alg"))

# set_knowledge() requires a `Knowledge` object
try(CausalDiscoSearch$new()$set_knowledge(list(not = "Knowledge")))
```

confusion

Confusion Matrix

Description

Compute confusion matrix for two PDAG [caugi::caugi](#) graphs.

Usage

```
confusion(truth, est, type = c("adj", "dir"))
```

Arguments

<code>truth</code>	A <code>caugi::caugi</code> object representing the truth graph.
<code>est</code>	A <code>caugi::caugi</code> object representing the estimated graph.
<code>type</code>	Character string specifying the comparison type: • <code>"adj"</code>: adjacency comparison. • <code>"dir"</code>: orientation comparison conditional on shared adjacencies.

Details

Adjacency comparison: The confusion matrix is a cross-tabulation of adjacencies. Hence, a true positive means that the two inputs agree on the presence of an adjacency. A true negative means that the two inputs agree on no adjacency. A false positive means that the estimated graph places an adjacency where there should be none. A false negative means that the estimated graph does not place an adjacency where there should have been one.

Orientation comparison: The orientation confusion matrix is conditional on agreement on adjacency. This means that only adjacencies that are shared in both input matrices are considered, and agreement wrt. orientation is then computed only among these edges that occur in both matrices. A true positive is a correctly placed arrowhead (1), a false positive marks placement of arrowhead (1) where there should have been a tail (0), a false negative marks placement of tail (0) where there should have been an arrowhead (1), and a truth negative marks correct placement of a tail (0).

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Value

A list with entries `tp` (true positives), `tn` (true negatives), `fp` (false positives), and `fn` (false negatives).

See Also

Other metrics: `evaluate()`, `f1_score()`, `false_omission_rate()`, `fdr()`, `g1_score()`, `npv()`, `precision()`, `recall()`, `reexports`, `specificity()`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
confusion(cg1, cg2)
confusion(cg1, cg2, type = "dir")
```

`convert_tiers_to_forbidden`*Convert Tiered Knowledge to Forbidden Knowledge*

Description

Converts tier assignments into forbidden edges, and drops tiers in the output.

Usage

```
convert_tiers_to_forbidden(kn)
```

Arguments

`kn` A Knowledge object.

Value

A Knowledge object with forbidden edges added, tiers removed.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)
kn_converted <- convert_tiers_to_forbidden(kn)
print(kn_converted)
plot(kn_converted)
```

cor_test	<i>Test for Vanishing Partial Correlations</i>
----------	--

Description

This function simply calls the `pcalg::gaussCitest()` function from the **pcalg** package.

Usage

```
cor_test(x, y, conditioning_set, suff_stat)
```

Arguments

x	Index of x variable.
y	Index of y variable.
conditioning_set	Index vector of conditioning variable(s), possibly NULL.
suff_stat	Sufficient statistic; A list with two elements, "C" and "n", corresponding to the correlation matrix and number of observations.

Value

A numeric, which is the p-value of the test.

deparse_knowledge	<i>Deparse a Knowledge Object into Knowledge DSL Code</i>
-------------------	---

Description

Given a Knowledge object, return a single string containing the R code (using `knowledge()`, `tier()`, `%-->%`, and `%!-->%`. that would rebuild that same object.

Usage

```
deparse_knowledge(kn, df_name = NULL)
```

Arguments

kn	A Knowledge object.
df_name	Optional name of the data frame you used (used as the first argument to <code>knowledge()</code>). If NULL, <code>knowledge()</code> is called with no data frame.

Value

A single string (with newlines) of R code.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# turn a Knowledge object back into DSL code
data(tpc_example)

kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3,
  oldage_x6 %!->% child_x1
)

code <- deparse_knowledge(kn, df_name = "tpc_example")
cat(code)

# Explicitly add all forbidden edges implied by tiers
kn <- convert_tiers_to_forbidden(kn)
code <- deparse_knowledge(kn, df_name = "tpc_example")
cat(code)
```

disco

*Perform Causal Discovery***Description**

Apply a causal discovery method to a data frame to infer causal relationships on observational data. Supports multiple algorithms and optionally incorporates prior knowledge.

Usage

```
disco(data, method, knowledge = NULL)
```

Arguments

data	A data frame.
method	A <code>disco_method</code> object representing a causal discovery algorithm. Available methods are

	• boss() - BOSS algorithm, • boss_fci() - BOSS-FCI algorithm, • fci() - FCI algorithm, • gfcf() - GFCE algorithm, • ges() - GES algorithm, • grasp() - GRASP algorithm, • grasp_fci() - GRASP-FCI algorithm, • gs() - GS algorithm, • iamb(), iamb_fdr(), fast_iamb(), inter_iamb() - IAMB algorithms, • pc() - PC algorithm, • sp_fci() - SP-FCI algorithm, • tfci() - TFCI algorithm, • tges() - TGES algorithm, • tpc() - TPC algorithm.
knowledge	A Knowledge object to be incorporated into the causal discovery algorithm. If NULL (default), the causal discovery algorithm is run without background knowledge. See knowledge() for how to create a Knowledge object.

Details

For specific details on the supported algorithms, scores, tests, and parameters for each engine, see:

- [BnlearnSearch](#) for **bnlearn**,
- [CausalDiscoSearch](#) for **causalDisco**,
- [PcalgSearch](#) for **pcalg**,
- [TetradSearch](#) for **Tetrad**.

Value

A Disco object (a list) containing the following components:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm.
- caugi A [caugi::caugi](#) object representing the learned causal graph from the causal discovery algorithm.
- graph_type A string with the semantic class of the learned graph (e.g. "CPDAG", "MPDAG", or "PAG").

Constraint-based algorithms may output graphs that are not valid CPDAGs/MPDAGs due to statistical errors in finite samples, violations of faithfulness, or latent confounding. In that case `disco()` emits a message and downgrades `graph_type`.

Examples

```

data(tpc_example)

# use pc with engine bnlearn and test fisher_z
my_pc <- pc(engine = "bnlearn", test = "fisher_z", alpha = 0.01)
pc_bnlearn <- disco(data = tpc_example, method = my_pc)
plot(pc_bnlearn)

# define tiered background knowledge
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)

# use gs with engine bnlearn and test cor and tiered background knowledge
my_pc_tiered <- pc(engine = "bnlearn", test = "cor", alpha = 0.01)
pc_tiered_bnlearn <- disco(
  data = tpc_example,
  method = my_pc_tiered,
  knowledge = kn
)
plot(pc_tiered_bnlearn)

```

distribute_engine_args

Distribute and Validate Engine Arguments

Description

This function checks the provided arguments against the expected arguments for the specified engine and algorithm, and distributes them appropriately to the search object. It ensures that the arguments are valid for the given engine and algorithm, and then sets them on the search object.

Usage

```
distribute_engine_args(search, args, engine, alg)
```

Arguments

search	R6 object, either TetradSearch, BnlearnSearch, PcalgSearch, or CausalDiscoSearch.
args	List of arguments to distribute
engine	Engine identifier, either "tetrad", "bnlearn", "pcalg", or "causalDisco"
alg	Algorithm name

See Also

Other Extending causalDisco: [as_disco\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

evaluate

*Evaluate Causal Graph Estimates***Description**

Computes various metrics to evaluate the difference between the estimated and true causal graph. Designed primarily for assessing the performance of causal discovery algorithms.

Metrics are supplied as a list with three slots: \$adj, \$dir, and \$other.

\$adj Metrics applied to the adjacency confusion matrix (see [confusion\(\)](#)).

\$dir Metrics applied to the conditional orientation confusion matrix (see [confusion\(\)](#)).

\$other Metrics applied directly to the adjacency matrices without computing confusion matrices.

Adjacency confusion matrix and conditional orientation confusion matrix only supports [caugi::caugi](#) objects whose edges are restricted to -->, <->, ---, or absence of an edge.

Usage

```
evaluate(truth, est, metrics = "all")
```

Arguments

truth	truth caugi::caugi object.
est	Estimated caugi::caugi object.
metrics	List of metrics, see details. If metrics = "all", all available metrics are computed.

Value

A data.frame with one column for each computed metric. Adjacency metrics are prefixed with "adj_", orientation metrics are prefixed with "dir_", other metrics do not get a prefix.

See Also

Other metrics: [confusion\(\)](#), [f1_score\(\)](#), [false_omission_rate\(\)](#), [fdr\(\)](#), [g1_score\(\)](#), [npv\(\)](#), [precision\(\)](#), [recall\(\)](#), [reexports](#), [specificity\(\)](#)

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
evaluate(cg1, cg2)
evaluate(
  cg1,
  cg2,
  metrics = list(
    adj = c("precision", "recall"),
    dir = c("f1_score"),
    other = c("shd")
  )
)
```

f1_score

F1 score

Description

Computes F1 score from two [caugi::caugi](#) objects. It converts the [caugi::caugi](#) objects to adjacency matrices and computes F1 score as $2 \cdot TP / (2 \cdot TP + FP + FN)$, where TP are true positives, FP are false positives, and FN are false negatives. If $TP + FP + FN = 0$, 1 is returned.

Only supports [caugi::caugi](#) objects whose edges are restricted to -->, <-->, ---, or absence of an edge.

Usage

```
f1_score(truth, est, type = c("adj", "dir"))
```

Arguments

- | | |
|-------|---|
| truth | A caugi::caugi object representing the truth graph. |
| est | A caugi::caugi object representing the estimated graph. |
| type | Character string specifying the comparison type: <ul style="list-style-type: none"> • "adj": adjacency comparison. • "dir": orientation comparison conditional on shared adjacencies. |

Value

A numeric in [0,1].

See Also

Other metrics: [confusion\(\)](#), [evaluate\(\)](#), [false_omission_rate\(\)](#), [fdr\(\)](#), [g1_score\(\)](#), [npv\(\)](#), [precision\(\)](#), [recall\(\)](#), [reexports](#), [specificity\(\)](#)

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
f1_score(cg1, cg2, type = "adj")
f1_score(cg1, cg2, type = "dir")
```

false_omission_rate	<i>False Omission Rate</i>
---------------------	----------------------------

Description

Computes false omission rate from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes false omission rate as $FN / (FN + TN)$, where FN are false negatives and TN are true negatives. If $FN + TN = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
false_omission_rate(truth, est, type = c("adj", "dir"))
```

Arguments

<code>truth</code>	A <code>caugi::caugi</code> object representing the truth graph.
<code>est</code>	A <code>caugi::caugi</code> object representing the estimated graph.
<code>type</code>	Character string specifying the comparison type: • <code>"adj"</code>: adjacency comparison. • <code>"dir"</code>: orientation comparison conditional on shared adjacencies.

Value

A numeric in $[0,1]$.

See Also

Other metrics: `confusion()`, `evaluate()`, `f1_score()`, `fdr()`, `g1_score()`, `npv()`, `precision()`, `recall()`, `reexports`, `specificity()`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
false_omission_rate(cg1, cg2, type = "adj")
false_omission_rate(cg1, cg2, type = "dir")
```

fci

*FCI Algorithm for Causal Discovery***Description**

Run the Fast Causal Inference algorithm for causal discovery using one of several engines.

Usage

```
fci(engine = c("tetrad", "pcalg"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library. "pcalg" pcalg R package.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**,
- [PcalgSearch](#) for **pcalg**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

References

Spirtes, P., Meek, C., & Richardson, T. (1995, August). Causal inference in the presence of latent variables and selection bias. In Proceedings of the Eleventh conference on Uncertainty in artificial intelligence (pp. 499-506).

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Recommended path using disco()
fci_pcalg <- fci(engine = "pcalg", test = "fisher_z", alpha = 0.05)
disco(tpc_example, fci_pcalg)

# or using fci_pcalg directly
fci_pcalg(tpc_example)

# With all algorithm arguments specified
fci_pcalg <- fci(
  engine = "pcalg",
  test = "fisher_z",
  alpha = 0.05,
  skel.method = "original",
  type = "anytime",
  fixedGaps = NULL,
  fixedEdges = NULL,
  NAdelete = FALSE,
  m.max = 10,
  pdsep.max = 2,
  rules = c(rep(TRUE, 9), FALSE),
  doPdsep = FALSE,
  biCC = TRUE,
  conservative = TRUE,
  maj.rule = FALSE,
  numCores = 1,
  selectionBias = FALSE,
  jci = "1",
  verbose = FALSE
)
disco(tpc_example, fci_pcalg)

#### Using tetrad engine with tier knowledge ####
# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
```

```

    youth ~ tidyselect::starts_with("youth"),
    oldage ~ tidyselect::starts_with("oldage")
  )
)

# Recommended path using disco()
fci_tetrad <- fci(engine = "tetrad", test = "fisher_z", alpha = 0.05)
disco(tpc_example, fci_tetrad, knowledge = kn)

# or using fci_tetrad directly
fci_tetrad <- fci_tetrad |> set_knowledge(kn)
fci_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  fci_tetrad <- fci(
    engine = "tetrad",
    test = "fisher_z",
    alpha = 0.05,
    complete_rule_set_used = FALSE,
    max_disc_path_length = 4,
    depth = 10,
    stable_fas = FALSE,
    guarantee_pag = TRUE
  )
  disco(tpc_example, fci_tetrad)
}

```

fdr

*False Discovery Rate***Description**

Computes false discovery rate from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes false discovery rate as $FP/(FP + TP)$, where FP are false positives and TP are true positives. If $FP + TP = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
fdr(truth, est, type = c("adj", "dir"))
```

Arguments

<code>truth</code>	A <code>caugi::caugi</code> object representing the truth graph.
<code>est</code>	A <code>caugi::caugi</code> object representing the estimated graph.
<code>type</code>	Character string specifying the comparison type:

- "adj": adjacency comparison.
- "dir": orientation comparison conditional on shared adjacencies.

Value

A numeric in [0,1].

See Also

Other metrics: [confusion\(\)](#), [evaluate\(\)](#), [f1_score\(\)](#), [false_omission_rate\(\)](#), [g1_score\(\)](#), [npv\(\)](#), [precision\(\)](#), [recall\(\)](#), [reexports](#), [specificity\(\)](#)

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
fdr(cg1, cg2, type = "adj")
fdr(cg1, cg2, type = "dir")
```

forbid_edge

Add Forbidden Edges to Knowledge

Description

Forbid one or more directed edges. Each argument **must** be a two-sided formula, e.g. $X \sim Y$. Formulas can use tidy-select on either side, so `forbid_edge(kn, starts_with("X") ~ Y)` forbids every $X_i \rightarrow Y$.

Usage

```
forbid_edge(kn, ...)
```

Arguments

kn	A Knowledge object.
...	One or more two-sided formulas.

Value

The updated Knowledge object.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```

data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
kn1$frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!->% youth_x4,
  exo(child_x1) # synonym: exogenous()
)

print(identical(kn1, kn2))

# cannot require an edge against tier direction
try(
  kn1 |> require_edge(oldage_x6 ~ child_x1)
)

# cannot forbid and require same edge
try(
  kn1 |> forbid_edge(child_x1 ~ youth_x3)
)

```

Description

Computes G1 score from two `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes G1 score defined as $2 \cdot TN / (2 \cdot TN + FN + FP)$, where TN are true negatives, FP are false positives, and FN are false negatives. If $TN + FN + FP = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
g1_score(truth, est, type = c("adj", "dir"))
```

Arguments

- | | |
|-------|---|
| truth | A <code>caugi::caugi</code> object representing the truth graph. |
| est | A <code>caugi::caugi</code> object representing the estimated graph. |
| type | Character string specifying the comparison type: <ul style="list-style-type: none"> • "adj": adjacency comparison. • "dir": orientation comparison conditional on shared adjacencies. |

Value

A numeric in [0,1].

References

Petersen, Anne Helby, et al. "Causal discovery for observational sciences using supervised machine learning." arXiv preprint arXiv:2202.12813 (2022).

See Also

Other metrics: `confusion()`, `evaluate()`, `f1_score()`, `false_omission_rate()`, `fdr()`, `npv()`, `precision()`, `recall()`, `reexports`, `specificity()`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
g1_score(cg1, cg2, type = "adj")
g1_score(cg1, cg2, type = "dir")
```

ges

*GES Algorithm for Causal Discovery***Description**

Run the Greedy Equivalent Search algorithm for causal discovery using one of several engines.

Usage

```
ges(engine = c("tetrad", "pcalg"), score, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library. "pcalg" pcalg R package.
score	Character; name of the scoring function to use.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad** (note, Tetrad refers to it as "fges"),
- [PcalgSearch](#) for **pcalg**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Chickering, D. M. (2002). Optimal structure identification with greedy search. *Journal of Machine Learning Research* 3, 507-554.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

#### Using pcalg engine ####
# Recommended path using disco()
ges_pcalg <- ges(engine = "pcalg", score = "sem_bic")
disco(tpc_example, ges_pcalg)

# or using ges_pcalg directly
ges_pcalg(tpc_example)

# With all algorithm arguments specified
ges_pcalg <- ges(
  engine = "pcalg",
  score = "sem_bic",
  adaptive = "vstructures",
  phase = "forward",
  iterate = FALSE,
  maxDegree = 3,
  verbose = FALSE
)
disco(tpc_example, ges_pcalg)

#### Using tetrad engine with tier knowledge ####
# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )

  # Recommended path using disco()
  ges_tetrad <- ges(engine = "tetrad", score = "sem_bic")
  disco(tpc_example, ges_tetrad, knowledge = kn)

  # or using ges_tetrad directly
  ges_tetrad <- ges_tetrad |> set_knowledge(kn)
  ges_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
```

```

ges_tetrad <- ges(
  engine = "tetrad",
  score = "ebic",
  symmetric_first_step = TRUE,
  max_degree = 3,
  parallelized = TRUE,
  faithfulness_assumed = TRUE
)
disco(tpc_example, ges_tetrad)
}

```

get_tiers

Get Tiers from Knowledge

Description

Get tiers from a Knowledge object.

Usage

```
get_tiers(kn)
```

Arguments

kn A Knowledge object.

Value

A tibble with the tiers.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```

kn <- knowledge(
  tier(
    1 ~ V1 + V2,
    2 ~ V3
  )
)

get_tiers(kn)

```

gfc

*GFCI Algorithm for Causal Discovery***Description**

Run the Greedy Fast Causal Inference algorithm for causal discovery using one of several engines.

Usage

```
gfc(engine = "tetrad", score, test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
score	Character; name of the scoring function to use.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A `caugi::caugi` object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

References

Ogarrio, J. M., Spirtes, P., and Ramsey, J. (2016). A hybrid causal search algorithm for latent variable models. In Conference on probabilistic graphical models, pages 368–379. PMLR.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(num_data)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  gfc_tetrad <- gfc(
    engine = "tetrad",
    score = "sem_bic",
    test = "fisher_z"
  )
  disco(tpc_example, gfc_tetrad)

  # or using gfc_tetrad directly
  gfc_tetrad(tpc_example)
}

#### With tier knowledge ####
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )

  # Recommended path using disco()
  gfc_tetrad <- gfc(
    engine = "tetrad",
    score = "sem_bic",
    test = "fisher_z"
  )
  disco(tpc_example, gfc_tetrad, knowledge = kn)

  # or using gfc_tetrad directly
  gfc_tetrad <- gfc_tetrad |> set_knowledge(kn)
  gfc_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  gfc_tetrad <- gfc(
    engine = "tetrad",
    score = "poisson_prior",
    test = "rank_independence",
```

```

    depth = 3,
    max_degree = 2,
    max_disc_path_length = 5,
    use_heuristic = FALSE,
    complete_rule_set_used = FALSE,
    guarantee_pag = TRUE,
    start_complete = TRUE,
    num_threads = 2,
    verbose = TRUE
  )
  disco(num_data, gfc_tetrad)
}

```

grasp

GRaSP Algorithm for Causal Discovery

Description

Run the Greedy Relaxations of the Sparsest Permutation algorithm for causal discovery using one of several engines.

Usage

```
grasp(engine = "tetrad", score, test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
score	Character; name of the scoring function to use.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- `caugi` A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Lam, W.-Y., Andrews, B., & Ramsey, J. (2022). Greedy Relaxations of the Sparsest Permutation Algorithm. In The 38th Conference on Uncertainty in Artificial Intelligence.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfc_i\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  grasp_tetrad <- grasp(
    engine = "tetrad",
    test = "fisher_z",
    score = "sem_bic",
    alpha = 0.05
  )
  disco(tpc_example, grasp_tetrad)

  # or using grasp_tetrad directly
  grasp_tetrad(tpc_example)
}

#### With tier knowledge ####
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )

  # Recommended path using disco()
  grasp_tetrad <- grasp(
```

```

    engine = "tetrad",
    test = "fisher_z",
    score = "sem_bic",
    alpha = 0.05
  )
  disco(tpc_example, grasp_tetrad, knowledge = kn)

  # or using grasp_tetrad directly
  grasp_tetrad <- grasp_tetrad |> set_knowledge(kn)
  grasp_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  grasp_tetrad <- grasp_fci(
    engine = "tetrad",
    test = "poisson_prior",
    score = "rank_bic",
    alpha = 0.05,
    depth = 3,
    stable_fas = FALSE,
    max_disc_path_length = 5,
    covered_depth = 3,
    singular_depth = 2,
    nonsingular_depth = 2,
    ordered_alg = TRUE,
    raskutti_uhler = TRUE,
    use_data_order = FALSE,
    num_starts = 3
  )
  disco(tpc_example, grasp_tetrad)
}

```

grasp_fci

GRaSP-FCI Algorithm for Causal Discovery

Description

Run the Greedy Relaxations of the Sparsest Permutation Fast Causal Inference algorithm for causal discovery using one of several engines.

Usage

```
grasp_fci(engine = "tetrad", score, test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
--------	---

score	Character; name of the scoring function to use.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

References

Ramsey, J., Andrews, B., Sprites, P. (2025). Efficient Latent Variable Causal Discovery: Combining Score Search and Targeted Testing. [doi:10.48550/arXiv.2510.04263](#).

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  grasp_fci_tetrad <- grasp_fci(
    engine = "tetrad",
    test = "fisher_z",
    score = "sem_bic",
    alpha = 0.05
  )
}
```

```

    )
    disco(tpc_example, grasp_fci_tetrad)

    # or using grasp_fci_tetrad directly
    grasp_fci_tetrad(tpc_example)
}

#### With tier knowledge ####
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )

  # Recommended path using disco()
  grasp_fci_tetrad <- grasp_fci(
    engine = "tetrad",
    test = "fisher_z",
    score = "sem_bic",
    alpha = 0.05
  )
  disco(tpc_example, grasp_fci_tetrad, knowledge = kn)

  # or using grasp_fci_tetrad directly
  grasp_fci_tetrad <- grasp_fci_tetrad |> set_knowledge(kn)
  grasp_fci_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  grasp_fci_tetrad <- grasp_fci(
    engine = "tetrad",
    test = "poisson_prior",
    score = "rank_bic",
    alpha = 0.05,
    depth = 3,
    stable_fas = FALSE,
    max_disc_path_length = 5,
    covered_depth = 3,
    singular_depth = 2,
    nonsingular_depth = 2,
    ordered_alg = TRUE,
    raskutti_uhler = TRUE,
    use_data_order = FALSE,
    num_starts = 3,
    guarantee_pag = TRUE
  )
  disco(tpc_example, grasp_fci_tetrad)
}

```

gs

*GS Algorithm for Causal Discovery***Description**

Run the Grow-Shrink algorithm for causal discovery using one of several engines.

Usage

```
gs(engine = c("bnlearn"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "bnlearn" bnlearn R package.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests and parameters for each engine, see:

- [BnlearnSearch](#) for **bnlearn**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Margaritis, D., Thrun, S.: Bayesian network induction via local neighborhoods. Tech. rep., DTIC Document (2000).

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfcf\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

kn <- knowledge(
  tpc_example,
  starts_with("child") %-->% starts_with("youth")
)

# Recommended path using disco()
gs_bnlearn <- gs(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05
)
disco(tpc_example, gs_bnlearn, knowledge = kn)

# or using gs_bnlearn directly
gs_bnlearn <- gs_bnlearn |> set_knowledge(kn)
gs_bnlearn(tpc_example)

# With all algorithm arguments specified
gs_bnlearn <- gs(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05,
  max.sx = 2,
  debug = FALSE,
  undirected = TRUE
)

disco(tpc_example, gs_bnlearn)
```

Description

Functions for causal discovery using variants of the Incremental Association algorithm:

- `iamb`: Incremental Association (IAMB)
- `inter_iamb`: Interleaved Incremental Association (Inter-IAMB)
- `iamb_fdr`: Incremental Association with FDR (IAMB-FDR)
- `fast_iamb`: Fast Incremental Association (Fast-IAMB)

Usage

```
iamb(engine = c("bnlearn"), test, alpha = 0.05, ...)

iamb_fdr(engine = c("bnlearn"), test, alpha = 0.05, ...)

fast_iamb(engine = c("bnlearn"), test, alpha = 0.05, ...)

inter_iamb(engine = c("bnlearn"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "bnlearn" bnlearn R package.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g., test or algorithm parameters).

Details

Each function supports the same engines and parameters. For details on tests and parameters for each engine, see:

- [BnlearnSearch](#) for **bnlearn**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

References

I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale Markov blanket discovery. In Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference, pages 376-381. AAAI Press, 2003.

See Also

Other causal discovery algorithms: `boss()`, `boss_fci()`, `fci()`, `ges()`, `gfci()`, `grasp()`, `grasp_fci()`, `gs()`, `pc()`, `rfci()`, `sp_fci()`, `tfci()`, `tges()`, `tpc()`

Examples

```
data(tpc_example)

kn <- knowledge(
  tpc_example,
  starts_with("child") %-->% starts_with("youth")
)

##### iamb #####

# Recommended path using disco()
iamb_bnlearn <- iamb(engine = "bnlearn", test = "fisher_z", alpha = 0.05)
disco(tpc_example, iamb_bnlearn, knowledge = kn)

# or using iamb_bnlearn directly
iamb_bnlearn <- iamb_bnlearn |> set_knowledge(kn)
iamb_bnlearn(tpc_example)

# With all algorithm arguments specified
iamb_bnlearn <- iamb(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05,
  max.sx = 2,
  debug = FALSE,
  undirected = TRUE
)

disco(tpc_example, iamb_bnlearn)

##### iamb_fdr #####

iamb_fdr_bnlearn <- iamb_fdr(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05
)
disco(tpc_example, iamb_fdr_bnlearn, knowledge = kn)

##### fast_iamb #####

fast_iamb_bnlearn <- fast_iamb(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05
)
```

```

disco(tpc_example, fast_iamb_bnlearn, knowledge = kn)

#### inter_iamb ####

inter_iamb_bnlearn <- inter_iamb(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05
)
disco(tpc_example, inter_iamb_bnlearn, knowledge = kn)

```

install_tetrad

*Install Tetrad GUI***Description**

Downloads and installs the Tetrad GUI JAR file from [Maven Central](#). It downloads the specified version of the Tetrad GUI JAR and its corresponding SHA256 checksum file, and saves them in the specified directory (or cache). If the JAR already exists and `force = FALSE`, it will skip downloading.

Usage

```

install_tetrad(
  version = getOption("causalDisco.tetrad.version"),
  dir = NULL,
  force = FALSE,
  quiet = FALSE,
  temp_dir = FALSE
)

```

Arguments

<code>version</code>	Character; the Tetrad version to install. Default is <code>getOption("causalDisco.tetrad.version")</code> .
<code>dir</code>	Character; the directory where the JAR should be installed. If <code>NULL</code> (default), the function uses the cache directory defined by <code>getOption("causalDisco.tetrad_cache")</code> . The directory will be created if it does not exist.
<code>force</code>	Logical; if <code>TRUE</code> , forces re-download even if the JAR already exists. Default is <code>FALSE</code> .
<code>quiet</code>	Logical; if <code>FALSE</code> , shows progress and messages about downloading and checksum verification. Default is <code>FALSE</code> .
<code>temp_dir</code>	Logical; if <code>TRUE</code> , installs the JAR in a temporary directory instead of the cache. Default is <code>FALSE</code> .

Details

In line with [CRAN policies](#) this function will only return messages and not throw warnings/errors if the installation fails (e.g. due to no internet connection), and return `NULL`.

Value

Invisibly returns the full path to the installed Tetrad JAR.

Examples

```
## Not run:
# Install default version in cache directory
install_tetrad()

# Install a specific version and force re-download
install_tetrad(version = "7.6.10", force = TRUE)

# Install in a temporary directory
install_tetrad(temp_dir = TRUE)

# Install quietly (suppress messages)
install_tetrad(quiet = TRUE)

## End(Not run)
```

knowledge

Define Background Knowledge

Description

Constructs a Knowledge object optionally initialized with a data frame and extended with variable relationships expressed via formulas, selectors, or infix operators:

```
tier(1 ~ V1 + V2, exposure ~ E)
V1 %-->% V3    # infix syntax for required edge from V1 to V3
V2 %!-->% V3    # infix syntax for an edge from V2 to V3 that is forbidden
exogenous(V1, V2)
```

Usage

```
knowledge(...)
```

Arguments

```
...
```

Arguments to define the Knowledge object:

- Optionally, a single data frame (first argument) whose column names initialize and freeze the variable set.
- Zero or more mini-DSL calls: `tier()`, `exogenous()`, (shorthand `exo()`), or infix operators `%-->%`, `%!-->%`.
 - `tier()`: One or more two-sided formulas (`tier(1 ~ x + y)`), or a numeric vector.
 - `exogenous()` / `exo()`: Variable names or `tidyselect` selectors. Arguments are evaluated in order; only these calls are allowed.

Details

Create a Knowledge object using a concise mini-DSL with `tier()`, `exogenous()` and infix edge operators `%-->%` and `%!-->%`.

The first argument can be a data frame, which will be used to populate the Knowledge object with variable names. Supplying a data frame also gives **tidyselect** helpers (e.g. `tidyselect::starts_with()`), a set of column names to resolve against, so they can be used throughout the mini-DSL, for example `tier(1 ~ starts_with("V"))` or `everything() %-->% Y`. If you later add variables with `add_*` verbs, this will throw a warning, since the Knowledge object will be *frozen*. You can unfreeze a Knowledge object by using the function `unfreeze()`.

If no data frame is provided, the object is initially empty. Variables can then be added via `tier()`, `forbidden()`, `required()`, infix operators, or `add_vars()`.

- `tier()`: Assigns variables to tiers. Tiers may be numeric or string labels. The left-hand side (LHS) of the formula is the tier; the right-hand side (RHS) specifies variables. Variables can also be selected using tidyselect syntax: `tier(1 ~ starts_with("V"))`.
- `%-->%` and `%!-->%`: Infix operators to define required and forbidden edges, respectively. Both sides of the operator can use tidyselect syntax to select multiple variables.
- `exogenous()` / `exo()`: Mark variables as exogenous.
- Numeric vector shortcut for `tier()`: `tier(c(1, 2, 1))` assigns tiers by index to all existing variables.

Multiple calls or operators are additive: each call adds new edges to the Knowledge object. For example:

```
V1 %-->% V3
V2 %-->% V3
```

results in both edges being required - i.e., the union of all specified required edges.

Value

A populated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
data(tpc_example)

# Knowledge objects can contain tier information, forbidden and required edges
kn <- knowledge(
  tier(
```

```

      1 ~ V1 + V2,
      2 ~ V3
    ),
    V1 %-->% V2,
    V3 %!-->% V1
  )

# If a data frame is provided, variable names are checked against it
kn <- knowledge(
  tpc_example,
  tier(
    1 ~ child_x1 + child_x2,
    2 ~ youth_x3 + youth_x4,
    3 ~ oldage_x5 + oldage_x6
  )
)

# Throws error if variable not in data
try(
  knowledge(
    tpc_example,
    tier(
      1 ~ child_x1 + child_x2,
      2 ~ youth_x3 + youth_x4,
      3 ~ oldage_x5 + woops
    )
  )
)

# Using tidyselect helpers
kn <- knowledge(
  tpc_example,
  tier(
    1 ~ starts_with("child"),
    2 ~ ends_with(c("_x3", "_x4")),
    3 ~ starts_with("oldage")
  )
)

# Numeric vector shortcut
kn <- knowledge(
  tpc_example,
  tier(c(1, 1, 2, 2, 3, 3))
)

# Custom tier naming
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    elderly ~ starts_with("oldage")
  )
)
```

```

)

# There is also required and forbidden edges, which are specified like so
kn <- knowledge(
  tpc_example,
  child_x1 %-->% youth_x3,
  oldage_x6 %!-->% child_x1
)

# Both sides of an edge operator accept tidyselect helpers, so a set of edges
# can be specified at once. Here every "child" variable is forbidden from
# pointing to every "youth" variable.
kn <- knowledge(
  tpc_example,
  starts_with("child") %!-->% starts_with("youth")
)

# tidyselect set operations such as `!` (negation) and `&` (intersection)
# also work. Forbid edges from child_x1 to every variable that is not a
# "child" variable.
kn <- knowledge(
  tpc_example,
  child_x1 %!-->% !starts_with("child")
)

# You can also add exogenous variables
kn <- knowledge(
  tpc_example,
  exogenous(child_x1),
  exo(child_x2) # shorthand
)

# Mix different operators
kn <- knowledge(
  tpc_example,
  tier(
    1 ~ starts_with("child") + youth_x4,
    2 ~ youth_x3 + starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  oldage_x6 %!-->% oldage_x5,
  exo(child_x2)
)

# You can also build knowledge with a verb pipeline
kn <-
  knowledge() |>
  add_vars(c("A", "B", "C", "D")) |> # Knowledge now only takes these variables
  add_tier(One) |>
  add_to_tier(One ~ A + B) |>
  add_tier(2, after = One) |>
  add_to_tier(2 ~ C + D) |>
  forbid_edge(A ~ C) |>

```

```

require_edge(A ~ B)

# Mix DSL start + verb refinement
kn <-
  knowledge(
    tier(1 ~ V5, 2 ~ V6),
    V5 %!-->% V6
  ) |>
  add_tier(3, after = "2") |>
  add_to_tier(3 ~ V7) |>
  add_exo(V2) |>
  add_exogenous(V3)

# Using seq_tiers for larger datasets
large_data <- as.data.frame(
  matrix(
    runif(100),
    nrow = 1,
    ncol = 100,
    byrow = TRUE
  )
)

names(large_data) <- paste0("X_", 1:100)

kn <- knowledge(
  large_data,
  tier(
    seq_tiers(
      1:100,
      ends_with("_{i}")
    )
  ),
  X_1 %-->% X_2
)

small_data <- data.frame(
  X_1 = 1,
  X_2 = 2,
  tier3_A = 3,
  Y5_ok = 4,
  check.names = FALSE
)

kn <- knowledge(
  small_data,
  tier(
    seq_tiers(1:2, ends_with("_{i}")),
    seq_tiers(3, starts_with("tier{i}")),
    seq_tiers(5, matches("Y{i}_ok"))
  )
)

```

knowledge_to_caugi	<i>Convert Knowledge to Causi</i>
--------------------	-----------------------------------

Description

Converts a Knowledge object to a `caugi::caugi` object used for plotting.

Usage

```
knowledge_to_caugi(kn)
```

Arguments

`kn` A Knowledge object.

Value

A list with the `caugi::caugi` object alongside information about the knowledge (tiers, required and forbidden edges) that can be used for plotting.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
data(tpc_example)
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!-->% youth_x3
)
cg <- knowledge_to_caugi(kn)
```

`list_registered_engines`*List Registered Engines*

Description

Returns the names of all custom engines registered via [register_engine\(\)](#) (in addition to the built-in engines "bnlearn", "causalDisco", "pcalg", and "tetrad", which are always available).

Usage

```
list_registered_engines()
```

Value

Character vector of engine names.

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

`list_registered_tetrad_algorithms`*List Registered Tetrad Algorithms*

Description

Returns the names of all custom registered Tetrad algorithms.

Usage

```
list_registered_tetrad_algorithms()
```

Value

Character vector of algorithm names.

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

make_method

Create a Custom Causal Discovery Method

Description

[Experimental]

Constructs a new causal discovery method that can be used with the [disco\(\)](#) framework. Users can provide an engine, engine-specific functions, and optional test and alpha parameters.

Usage

```
make_method(
  method_name,
  engine,
  engine_fns,
  test = NULL,
  alpha = NULL,
  score = NULL,
  graph_class,
  ...
)
```

Arguments

method_name	Character. The name of the method to create.
engine	Character. The engine to use. Must be one of the names of engine_fns.
engine_fns	Named list of functions. Each element corresponds to an engine and is a function that implements the causal discovery algorithm.
test	Optional. A test statistic to pass to the engine function.
alpha	Optional. A significance level to pass to the engine function.
score	Optional. A score to pass to the engine function.
graph_class	Character. The graph class that this method produces.
...	Additional arguments passed to the engine function.

Value

A disco_method object with attributes engine and graph_class.

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

make_runner

Create a Custom Runner for a Causal Discovery Algorithm

Description

[Experimental]

Constructs a runner function for a specific causal discovery engine and algorithm. This allows users to support new algorithms.

Usage

```
make_runner(engine, alg, test = NULL, alpha = NULL, score = NULL, ...)
```

Arguments

engine	Character. The engine to use. Options include "causalDisco", "pcalg", "bnlearn", "tetrad", or the name of any engine added via register_engine() .
alg	Character. The algorithm name.
test	Optional. A test statistic to pass to the engine.
alpha	Optional. Significance level to pass to the engine.
score	Optional. A scoring function for score-based methods.
...	Additional arguments passed to the engine-specific runner.

Value

An object representing a configured runner for the chosen engine. The type depends on the engine.

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

make_tikz

Generate TikZ Code from a Causal Graph

Description

Generates LaTeX TikZ code from a Disco, Knowledge, or `caugi::caugi` object, preserving node positions, labels, and visual styles. Edges are rendered with arrows, line widths, and colors. The output is readable LaTeX code that can be directly compiled or modified.

Usage

```
make_tikz(
  x,
  ...,
  scale = 10,
  full_doc = TRUE,
  bend_edges = FALSE,
  bend_angle = 25,
  tier_label_pos = c("above", "below", "left", "right")
)
```

Arguments

<code>x</code>	A <code>Disco</code> , <code>Knowledge</code> , or <code>caugi::caugi</code> object.
<code>...</code>	Additional arguments passed to <code>plot()</code> and <code>caugi::plot()</code> .
<code>scale</code>	Numeric scalar. Scaling factor for node coordinates. Default is 10.
<code>full_doc</code>	Logical. If <code>TRUE</code> (default), generates a full standalone LaTeX document. If <code>FALSE</code> , returns only the <code>tikzpicture</code> environment.
<code>bend_edges</code>	Logical. If <code>TRUE</code> , edges are drawn with bent edges. Default is <code>FALSE</code> . Edges connecting the same pair of nodes in both directions (<code>A %-->% B</code> and <code>B %-->% A</code>) are automatically bent left and right to avoid overlap. Bend direction is automatically chosen to reduce overlap.
<code>bend_angle</code>	Numeric scalar. Angle in degrees for bending arrows when <code>bend_edges = TRUE</code> . Default is 25.
<code>tier_label_pos</code>	Character string specifying the position of tier labels relative to the tier rectangles. Must be one of "above", "below", "left", or "right". Default is "above".

Details

The function calls `plot()` to generate a `caugi::caugi_plot` object, then traverses the plot object's grob structure to extract nodes and edges. Supported features include:

- **Nodes**
 - Fill color and draw color (supports both named colors and custom RGB values)
 - Font size
 - Coordinates are scaled by the `scale` parameter
- **Edges**
 - Line color and width
 - Arrow scale
 - Optional bending to reduce overlapping arrows

The generated TikZ code uses global style settings, and edges are connected to nodes by name (as opposed to hard-coded coordinates), making it easy to modify the output further if needed.

Value

A character string containing LaTeX TikZ code. Depending on `full_doc`, this is either:

- a complete LaTeX document (`full_doc = TRUE`), or
- only the `tikzpicture` environment (`full_doc = FALSE`).

Examples

```
##### Convert Knowledge to Tikz #####

data(num_data)
kn <- knowledge(
  num_data,
  X1 %-->% X2,
  X2 %!-->% X3 + Y,
  Y %!-->% Z
)

# Full standalone document
tikz_kn <- make_tikz(kn, scale = 10, full_doc = TRUE)
cat(tikz_kn)

# Only the tikzpicture environment
tikz_kn_snippet <- make_tikz(kn, full_doc = FALSE)
cat(tikz_kn_snippet)

# With bent edges
tikz_bent <- make_tikz(
  kn,
  full_doc = FALSE,
  bend_edges = TRUE
)
cat(tikz_bent)

# With a color not supported by default TikZ colors; will fall back to RGB
tikz_darkblue <- make_tikz(
  kn,
  node_style = list(fill = "darkblue"),
  full_doc = FALSE
)
cat(tikz_darkblue)

# With tiered knowledge
data(tpc_example)
kn_tiered <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)
```

```

tikz_tiered_kn <- make_tikz(
  kn_tiered,
  full_doc = FALSE
)
cat(tikz_tiered_kn)

##### Convert Disco to Tikz #####

data(num_data)
kn <- knowledge(
  num_data,
  X1 %-->% X2,
  X2 %!-->% X3 + Y,
  Y %!-->% Z
)

pc_bnlearn <- pc(engine = "bnlearn", test = "fisher_z", alpha = 0.05)
disco_kn <- disco(data = num_data, method = pc_bnlearn, knowledge = kn)

tikz_snippet <- make_tikz(disco_kn, scale = 10, full_doc = FALSE)
cat(tikz_snippet)

##### Convert caugi objects to Tikz #####

cg <- caugi::caugi(A %-->% B + C)

tikz_snippet <- make_tikz(
  cg,
  node_style = list(fill = "red"),
  scale = 10,
  full_doc = FALSE
)
cat(tikz_snippet)

```

mix_data

Simulated Mixed Data

Description

A dataset combining continuous and categorical variables. The first three variables are replaced with categorical versions from `cat_data`.

Usage

```
mix_data
```

Format

A data.frame with 1000 rows and 5 variables.

X1 Categorical, from cat_data\$X1.

X2 Categorical, from cat_data\$X2.

X3 Categorical, from cat_data\$X3.

Z Numeric, same as num_data\$Z.

Y Numeric, same as num_data\$Y.

Details

The R code used to generate this dataset is as follows:

```
data(num_data)
data(cat_data)
mix_data <- num_data
mix_data$X1 <- cat_data$X1
mix_data$X2 <- cat_data$X2
mix_data$X3 <- cat_data$X3
```

See Also

[num_data](#), [cat_data](#)

Examples

```
data(mix_data)
head(mix_data)
```

new_disco_method

Add a New causalDisco Method

Description

This function allows you to create a new causal discovery method that can be used with the [disco\(\)](#) function. You provide a builder function that constructs a runner object, along with metadata about the algorithm, and it returns a closure that can be called with a data frame to perform causal discovery and return a [caugi::caugi](#) object.

Usage

```
new_disco_method(builder, name, engine, graph_class)
```

Arguments

builder	A function returning a runner
name	Algorithm name
engine	Engine identifier
graph_class	Output graph class

Value

A function of class "disco_method" that takes a single argument data (a data frame) and returns a `caugi::caugi` object.

See Also

Other Extending causalDisco: `as_disco()`, `distribute_engine_args()`, `list_registered_engines()`, `list_registered_tetrad_algorithms()`, `make_method()`, `make_runner()`, `register_engine()`, `register_tetrad_algorithm()`, `reset_engine_registry()`, `reset_tetrad_alg_registry()`

nlsy97

US National Longitudinal Survey of Youth 1997

Description

A subset of data from the US National Longitudinal Survey of Youth 1997 (NLSY97). Several ordinal variables were dichotomized for analysis purposes, and all survey-specific missing value codes were recoded to NA.

Usage

```
nlsy97
```

Format

A tibble with 8,974 rows and 9 columns:

r1_hhchildren Number of household members younger than 18. Measured in Round 1, 1997.

r1_mcollege Did biological mother attend college? Binary factor (levels: No, Yes). Measured in Round 1, 1997.

r1_rural_urban Residence in an urban or a rural area. Binary factor (levels: Rural, Urban). Measured in Round 1, 1997.

r6_depressed Depressive feelings during the last month. Binary factor derived from 4-point scale (levels: Any = 1–3, None = 4). Measured in Round 6, 2002.

r6_docvisits Number of doctor's visits during the last year. Measured in Round 6, 2002.

r6_exercise Days with 30+ minutes exercise / week. Measured in Round 6, 2002.

r12_depressed Depressive feelings during the last month. Binary factor derived from 4-point scale (levels: Any = 1–3, None = 4). Measured in Round 12, 2008.

r12_docvisits Number of doctor's visits during the last year. Measured in Round 12, 2008.

r12_health Perception of own health. Binary factor derived from 5-point scale (levels: Good = 1–3, Poor = 4–5). Measured in Round 12, 2008.

Details

See source code on GitHub in `data-raw/` for the raw data, the corresponding RNUM, and code showing how it has been cleaned.

Source

<https://www.nlsinfo.org/investigator/pages/search?s=NLSY97>

npv	<i>Negative Predictive Value</i>
-----	----------------------------------

Description

Computes negative predictive value from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes negative predictive value as $TN / (TN + FN)$, where TN are true negatives and FN are false negatives. If $TN + FN = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
npv(truth, est, type = c("adj", "dir"))
```

Arguments

- | | |
|-------|---|
| truth | A <code>caugi::caugi</code> object representing the truth graph. |
| est | A <code>caugi::caugi</code> object representing the estimated graph. |
| type | Character string specifying the comparison type: <ul style="list-style-type: none"> "adj": adjacency comparison. "dir": orientation comparison conditional on shared adjacencies. |

Value

A numeric in $[0,1]$.

See Also

Other metrics: `confusion()`, `evaluate()`, `f1_score()`, `false_omission_rate()`, `fdr()`, `g1_score()`, `precision()`, `recall()`, `reexports`, `specificity()`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
npv(cg1, cg2, type = "adj")
npv(cg1, cg2, type = "dir")
```

num_data

*Simulated Numerical Data***Description**

Simulated Numerical Data

Usage

```
num_data
```

Format

A data.frame with 1000 rows and 5 variables.

X1 Structural equation: $X_1 := \sqrt{Z} + \epsilon_1$ with $\epsilon_1 \sim \text{Unif}[0, 2]$

X2 Structural equation: $X_2 := 2 \cdot X_3 - \epsilon_2$ with $\epsilon_2 \sim N(5, 1)$

X3 Structural equation: $X_3 := \epsilon_3$ with $\epsilon_3 \sim \text{Unif}[5, 10]$

Z Structural equation: $Z := |\epsilon_4|$ with $\epsilon_4 \sim N(10, 1)$

Y Structural equation: $Y := X_1^2 + X_2 - X_3 - Z + \epsilon_5$ with $\epsilon_5 \sim N(10, 1)$

Details

The R code used to generate this dataset is as follows:

```
set.seed(1405)
n <- 1000
Z <- abs(rnorm(n, mean = 10))
X1 <- sqrt(Z) + runif(n, min = 0, max = 2)
X3 <- runif(n, min = 5, max = 10)
X2 <- 2 * X3 - rnorm(n, mean = 5)
Y <- X1^2 + X2 - X3 - Z + rnorm(n, mean = 10)
num_data <- data.frame(X1, X2, X3, Z, Y)
```

Examples

```
data(num_data)
head(num_data)
```

num_data_latent	<i>Simulated Numerical Data with Latent Variable</i>
-----------------	--

Description

A dataset similar to num_data but with the variable Z treated as a latent variable and thus omitted.

Usage

```
num_data_latent
```

Format

A data.frame with 1000 rows and 4 variables.

X1 Structural equation: $X_1 := \sqrt{Z} + \epsilon_1$ with $\epsilon_1 \sim \text{Unif}[0, 2]$

X2 Structural equation: $X_2 := 2 \cdot X_3 - \epsilon_2$ with $\epsilon_2 \sim N(5, 1)$

X3 Structural equation: $X_3 := \epsilon_3$ with $\epsilon_3 \sim \text{Unif}[5, 10]$

Z Structural equation: $Z := |\epsilon_4|$ with $\epsilon_4 \sim N(10, 1)$

Y Structural equation: $Y := X_1^2 + X_2 - X_3 - Z + \epsilon_5$ with $\epsilon_5 \sim N(10, 1)$

Details

The R code used to generate this dataset is as follows:

```
data(num_data)
num_data_latent <- num_data[, c("X1", "X2", "X3", "Y")]
```

See Also

[num_data](#)

Examples

```
data(num_data_latent)
head(num_data_latent)
```

pc

*PC Algorithm for Causal Discovery***Description**

Run the Peter-Clark algorithm for causal discovery using one of several engines.

Usage

```
pc(engine = c("tetrad", "pcalg", "bnlearn"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library. "pcalg" pcalg R package. "bnlearn" bnlearn R package.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**,
- [PcalgSearch](#) for **pcalg**,
- [BnlearnSearch](#) for **bnlearn**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Spirtes P, Glymour C, and Scheines R. Causation, Prediction, and Search. MIT Press, 2000.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [rfci\(\)](#), [sp_fci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

#### Using pcalg engine ####
# Recommended path using disco()
pc_pcalg <- pc(engine = "pcalg", test = "fisher_z", alpha = 0.05)
disco(tpc_example, pc_pcalg)

# or using pc_pcalg directly
pc_pcalg(tpc_example)

# With all algorithm arguments specified
pc_pcalg <- pc(
  engine = "pcalg",
  test = "fisher_z",
  alpha = 0.05,
  fixedGaps = NULL,
  fixedEdges = NULL,
  NAdelete = FALSE,
  m.max = 10,
  u2pd = "relaxed",
  skel.method = "original",
  conservative = TRUE,
  maj.rule = FALSE,
  solve.confl = TRUE,
  numCores = 1,
  verbose = FALSE
)

#### Using bnlearn engine with required knowledge ####
kn <- knowledge(
  tpc_example,
  starts_with("child") %-->% starts_with("youth")
)

# Recommended path using disco()
pc_bnlearn <- pc(engine = "bnlearn", test = "fisher_z", alpha = 0.05)
disco(tpc_example, pc_bnlearn, knowledge = kn)

# or using pc_bnlearn directly
pc_bnlearn <- pc_bnlearn |> set_knowledge(kn)
pc_bnlearn(tpc_example)
```

```

# With all algorithm arguments specified
pc_bnlearn <- pc(
  engine = "bnlearn",
  test = "fisher_z",
  alpha = 0.05,
  max.sx = 2,
  debug = FALSE,
  undirected = TRUE
)

disco(tpc_example, pc_bnlearn)

#### Using tetrad engine with tier knowledge ####
# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )
}

# Recommended path using disco()
pc_tetrad <- pc(engine = "tetrad", test = "fisher_z", alpha = 0.05)
disco(tpc_example, pc_tetrad, knowledge = kn)

# or using pc_tetrad directly
pc_tetrad <- pc_tetrad |> set_knowledge(kn)
pc_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  pc_tetrad <- pc(
    engine = "tetrad",
    test = "fisher_z",
    alpha = 0.05,
    conflict_rule = 2,
    depth = 10,
    stable_fas = FALSE,
    guarantee_cpdag = TRUE
  )
  disco(tpc_example, pc_tetrad)
}

```

Description

A wrapper that lets you drive **pcalg** algorithms within the **causalDisco** framework. For arguments to the test, score, and algorithm, see the **pcalg** documentation, which we link to in the respective sections below.

Public fields

data A `data.frame` holding the data set currently attached to the search object. Can be set with `set_data()`.

score A function that will be used to build the score, when data is set. Can be set with `$set_score()`. Recognized values are:

- "sem_bic" - BIC score for Gaussian observed data. See `pcalg::GaussLOpenObsScore`.
- "sem_bic_int" - BIC score for Gaussian data from jointly interventional and observational Gaussian data. See `pcalg::GaussLOpenIntScore`.

test A function that will be used to test independence. Can be set with `$set_test()`. Recognized values are:

- "fisher_z" - Fisher Z test for Gaussian data. See `pcalg::gaussCItest()`.
- "fisher_z_twd" - Fisher Z test for Gaussian data with test-wise deletion. See `micd::gaussCItwd()`.
- "fisher_z_mi" - Fisher Z test for Gaussian data with multiple imputation. See `micd::gaussCItestMI()`.
- "reg" - Regression test for discrete or binary data. See `reg_test()`.
- "g_square" - G square test for discrete data. See `pcalg::binCItest()` and `pcalg::disCItest()`.
- "g_square_twd" - G square test for discrete data with test-wise deletion. See `micd::disCItwd()`.
- "g_square_mi" - G square test for discrete data with multiple imputation. See `micd::disMItest()`.
- "conditional_gaussian" - Test for conditional independence in mixed data. See `micd::mixCItest()`.
- "conditional_gaussian_twd" - Test for conditional independence in mixed data with test-wise deletion. See `micd::mixCItwd()`.
- "conditional_gaussian_mi" - Test for conditional independence in mixed data with multiple imputation. See `micd::mixMItest()`.

alg A function that will be used to run the search algorithm. Can be set with `$set_alg()`. Recognized values are:

- "fci" - FCI algorithm. See `fci()` and the underlying `pcalg::fci()`.
- "ges" - GES algorithm. See `ges()` and the underlying `pcalg::ges()`.
- "pc" - PC algorithm. See `pc()` and the underlying `pcalg::pc()`.
- "rfci" - RFCI algorithm. See `rfci()` and the underlying `pcalg::rfci()`.

params A list of parameters for the test and algorithm. Can be set with `$set_params()`. The parameters are passed to the test and algorithm functions.

suff_stat Sufficient statistic. The format and contents of the sufficient statistic depends on which test is being used.

continuous Logical; whether the sufficient statistic is for a continuous test. If both continuous and discrete are TRUE, the sufficient statistic is build for a mixed test.

discrete Logical; whether the sufficient statistic is for a discrete test. If both continuous and discrete are TRUE, the sufficient statistic is build for a mixed test.

knowledge A list of fixed constraints for the search algorithm. Note, that pcalg only works with symmetric knowledge. Thus, the only allowed types of knowledge is forbidden edges in both directions.

adapt_df Logical; whether to adapt the degrees of freedom for discrete tests.

Methods

Public methods:

- `PcalgSearch$new()`
- `PcalgSearch$set_params()`
- `PcalgSearch$set_data()`
- `PcalgSearch$set_suff_stat()`
- `PcalgSearch$set_test()`
- `PcalgSearch$set_score()`
- `PcalgSearch$set_alg()`
- `PcalgSearch$set_knowledge()`
- `PcalgSearch$run_search()`
- `PcalgSearch$clone()`

`PcalgSearch$new()`: Constructor for the PcalgSearch class.

Usage:

```
PcalgSearch$new()
```

`PcalgSearch$set_params()`: Sets the parameters for the test and algorithm.

Usage:

```
PcalgSearch$set_params(params)
```

Arguments:

params A list of parameters to set.

`PcalgSearch$set_data()`: Sets the data for the search algorithm.

Usage:

```
PcalgSearch$set_data(data, set_suff_stat = TRUE)
```

Arguments:

data A data.frame or a matrix containing the data.

set_suff_stat Logical; whether to set the sufficient statistic. for the data.

`PcalgSearch$set_suff_stat()`: Sets the sufficient statistic for the data.

Usage:

```
PcalgSearch$set_suff_stat()
```

`PcalgSearch$set_test()`: Sets the test for the search algorithm.

Usage:

```
PcalgSearch$set_test(method, alpha = 0.05, suff_stat_fun = NULL, args = NULL)
```

*Arguments:***method** **[Experimental]**

A string specifying the type of test to use.

Can also be a user-defined function with signature `function(x, y, conditioning_set, suff_stat)`, where `x` and `y` are the variables being tested for independence, `conditioning_set` is the conditioning set, and `suff_stat` is the sufficient statistic for the test. If a user-defined function is provided, then `suff_stat_fun` must also be provided, which is a function that should take the data as input and returns a sufficient statistic for the test. Optionally, the signature of the user-defined test function can also include an `args` parameter, which is a list of additional arguments to pass to the test function. If `args` is provided, then the test function should have the signature `function(x, y, conditioning_set, suff_stat, args)`, and the `args` parameter will be passed to the test function.

EXPERIMENTAL: user-defined tests syntax are subject to change.

`alpha` Significance level for the test.

`suff_stat_fun` A function that takes the data as input and returns a sufficient statistic for the test. Only needed if `method` is a user-defined function.

`args` A list of additional arguments to pass to the test. Only needed if `method` is a user-defined function with an `args` parameter in its signature.

`PcalgSearch$set_score()`: Sets the score for the search algorithm.

Usage:

```
PcalgSearch$set_score(method, params = list())
```

Arguments:

`method` A string specifying the type of score to use.

`params` A list of parameters to pass to the score function.

`PcalgSearch$set_alg()`: Sets the algorithm for the search.

Usage:

```
PcalgSearch$set_alg(method)
```

Arguments:

`method` A string specifying the type of algorithm to use.

`PcalgSearch$set_knowledge()`: Sets the knowledge for the search algorithm. Due to the nature of `pcalg`, we cannot set knowledge before we run it on data. So we set the function that will be used to build the fixed constraints, but it can first be done when data is provided.

Usage:

```
PcalgSearch$set_knowledge(
  knowledge_obj,
  directed_as_undirected = lifecycle::deprecated()
)
```

Arguments:

`knowledge_obj` A Knowledge object that contains the fixed constraints.

`directed_as_undirected` **[Deprecated]** This argument no longer has any effect and will be removed in a future release. Specify directed edges in both directions in `knowledge()` instead.

`PcalgSearch$run_search()`: Runs the search algorithm on the data.

Usage:

```
PcalgSearch$run_search(data = NULL, set_suff_stat = TRUE)
```

Arguments:

`data` A `data.frame` or a `matrix` containing the data.

`set_suff_stat` Logical; whether to set the sufficient statistic

`PcalgSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PcalgSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
### pcalg_search R6 class examples ###

# Generally, we do not recommend using the R6 classes directly, but rather
# use the disco() or any method function, for example pc(), instead.

# Load data
data(num_data)

# Recommended:
my_pc <- pc(engine = "pcalg", test = "fisher_z")
my_pc(num_data)

# or
disco(data = num_data, method = my_pc)

# Example with detailed settings:
my_pc2 <- pc(
  engine = "pcalg",
  test = "fisher_z",
  alpha = 0.01,
  m.max = 4,
  skel.method = "original"
)

disco(data = num_data, method = my_pc2)

# With knowledge
kn <- knowledge(
  num_data,
  X1 %!-->% X2,
  X2 %!-->% X1
)
```

```
disco(data = num_data, method = my_pc2, knowledge = kn)

# Using R6 class:
s <- PcalgSearch$new()

s$set_test(method = "fisher_z", alpha = 0.05)
s$set_data(tpc_example)
s$set_alg("pc")

g <- s$run_search()

print(g)
```

plot

Plot Method for causalDisco Objects

Description

This is the generic `plot()` function for objects of class `Knowledge` or `Disco`. It dispatches to the class-specific plotting methods `plot.Knowledge()` and `plot.Disco()`.

Value

Invisibly returns the input object. The primary effect is the generated plot.

See Also

Other plot: `plot.Disco()`, `plot.Knowledge()`

Examples

```
data(tpc_example)
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)
plot(kn)

cd_tges <- tges(engine = "causalDisco", score = "tbic")
disco_cd_tges <- disco(data = tpc_example, method = cd_tges, knowledge = kn)
plot(disco_cd_tges)
```

plot.Disco

Plot a Disco Object

Description

Visualize a causal graph stored within a Disco object. This function extends [plot.Knowledge\(\)](#) by combining the causal graph from a `caugi` object with background knowledge.

Usage

```
## S3 method for class 'Disco'
plot(x, required_col = "blue", ...)
```

Arguments

<code>x</code>	A Disco object containing both the causal graph and the associated knowledge.
<code>required_col</code>	Character(1). Color for edges marked as "required". Default "blue".
<code>...</code>	Additional arguments passed to caugi::plot() and plot.Knowledge() .

Details

- **Required edges** are drawn in **blue** by default (`required_col`), can be changed.
- **Forbidden edges** are not drawn.
- If tiered knowledge is provided, nodes are arranged according to their tiers.
- Other edge styling (line width, arrow size, etc.) can be supplied via `edge_style`. To override the color of a specific edge, specify it in `edge_style$by_edge[[from]][[to]]$col`.

This function combines the causal graph and the Knowledge object into a single plotting structure. If the knowledge contains tiers, nodes are laid out accordingly; otherwise, the default `caugi` layout is used. Edges marked as required are automatically colored (or can be overridden per edge using `edge_style$by_edge`).

Value

Invisibly returns the underlying `caugi` object. The main effect is the plot.

See Also

Other plot: [plot](#), [plot.Knowledge\(\)](#)

Examples

```
data(tpc_example)

# Define tiered knowledge
kn <- knowledge(
  tpc_example,
```

```

    tier(
      child ~ starts_with("child"),
      youth ~ starts_with("youth"),
      old ~ starts_with("old")
    )
  )

# Fit a causal discovery model
cd_tges <- tges(engine = "causalDisco", score = "tbic")
disco_cd_tges <- disco(data = tpc_example, method = cd_tges, knowledge = kn)

# Plot with default column orientation
plot(disco_cd_tges)

# Plot with row orientation
plot(disco_cd_tges, orientation = "rows")

# Plot with custom node and edge styling
plot(
  disco_cd_tges,
  node_style = list(
    fill = "lightblue", # Fill color
    col = "darkblue", # Border color
    lwd = 2, # Border width
    padding = 4, # Text padding (mm)
    size = 1.2 # Size multiplier
  ),
  edge_style = list(
    lwd = 1.5, # Edge width
    arrow_size = 4, # Arrow size (mm)
    col = "darkgreen", # Edge color
    fill = "black", # Arrow fill color
    lty = "dashed" # Edge line type
  )
)

# To override a specific edge style which is required you need to target that individual node:
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  ),
  child_x1 %-->% child_x2 + youth_x4 # required edges
)

bnlearn_pc <- pc(engine = "bnlearn", test = "fisher_z")
disco_bnlearn_pc <- disco(data = tpc_example, method = bnlearn_pc, knowledge = kn)

# Edge from child_x1 to child_x2 will be orange, but edge from child_x1 to youth_x4
# will be required_col (blue) since we only override the child_x1 to child_x2 edge.
plot(
  disco_bnlearn_pc,

```

```

    edge_style = list(
      by_edge = list(
        child_x1 = list(
          child_x2 = list(col = "orange", fill = "orange")
        )
      )
    ),
    required_col = "blue"
  )

  # Plot without tiers
  data(num_data)
  kn_untiered <- knowledge(
    num_data,
    X1 %-->% X2 + X3,
    Z %!->% Y
  )

  bnlearn_pc <- pc(engine = "bnlearn", test = "fisher_z")
  res_untiered <- disco(data = num_data, method = bnlearn_pc, knowledge = kn_untiered)
  plot(res_untiered)

  # With a custom defined layout
  custom_layout <- data.frame(
    name = c("X1", "X2", "X3", "Z", "Y"),
    x = c(0, 1, 2, 2, 3),
    y = c(0, 1, 0.25, -1, 0)
  )
  plot(res_untiered, layout = custom_layout)

```

plot.Knowledge

Plot a Knowledge Object

Description

Visualize a Knowledge object as a directed graph using `caugi::plot()`.

Usage

```

## S3 method for class 'Knowledge'
plot(x, required_col = "blue", forbidden_col = "red", ...)

```

Arguments

<code>x</code>	A Knowledge object, created using <code>knowledge()</code> .
<code>required_col</code>	Character(1). Color for edges marked as "required". Default "blue".
<code>forbidden_col</code>	Character(1). Color for edges marked as "forbidden". Default "red".
<code>...</code>	Additional arguments passed to <code>caugi::plot()</code> , e.g., <code>node_style</code> , <code>edge_style</code> .

Details

- **Required edges** are drawn in **blue** by default (can be changed via `required_col`).
- **Forbidden edges** are drawn in **red** by default (can be changed via `forbidden_col`). If A to B and B to A is forbidden, an edge `<->` is drawn.
- If tiered knowledge is provided, nodes are arranged according to their tiers.
- Users can override other edge styling (e.g., line width, arrow size) via the `edge_style` argument. To override the color of a specific edge, use `edge_style$by_edge[[from]][[to]]$col`.
- Nodes are arranged by tiers if tier information is provided in the Knowledge object.
- If some nodes are missing tier assignments, a warning is issued and the plot falls back to untiered plotting.
- The function automatically handles edges marked as "required" or "forbidden" in the Knowledge object.
- Other edge styling (line width, arrow size, etc.) can be supplied via `edge_style`. The only way to override edge colors for specific edges is to specify them directly in `edge_style$by_edge[[from]][[to]]$col`.

Value

Invisibly returns the `caugi::caugi` object used for plotting. The main effect is the plot.

See Also

Other plot: `plot`, `plot.Disco()`

Examples

```
data(tpc_example)

# Define a `Knowledge` object with tiers
kn_tiered <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)

# Simple plot (default column orientation)
plot(kn_tiered)

# Plot with row orientation
plot(kn_tiered, orientation = "rows")

# Plot with custom node styling, edge width/arrow size and edge colors
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
```

```

    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  ),
  child_x1 %-->% child_x2, # required edge
  youth_x4 %!->% youth_x3 # forbidden edge
)
plot(
  kn,
  node_style = list(
    fill = "lightblue", # Fill color
    col = "darkblue", # Border color
    lwd = 2, # Border width
    padding = 4, # Text padding (mm)
    size = 1.2 # Size multiplier
  ),
  edge_style = list(
    lwd = 1.5, # Edge width
    arrow_size = 4 # Arrow size (mm)
  ),
  required_col = "darkgreen",
  forbidden_col = "darkorange"
)

# To override a specific edge style which is required/forbidden
# you need to target that individual node:
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  ),
  child_x1 %-->% child_x2 + youth_x4, # required edges
  youth_x4 %!->% youth_x3 + oldage_x5 # forbidden edges
)

# Edge from child_x1 to child_x2 will be orange, but edge from child_x1 to youth_x4
# will be required_col (blue) since we only override the child_x1 to child_x2 edge.
# Similarly, edge from youth_x4 to youth_x3 will be yellow, but edge from youth_x4
# to oldage_x5 will be forbidden_col (red).
plot(
  kn,
  edge_style = list(
    by_edge = list(
      child_x1 = list(
        child_x2 = list(col = "orange", fill = "orange")
      ),
      youth_x4 = list(
        youth_x3 = list(col = "yellow", fill = "yellow")
      )
    )
  ),
  required_col = "blue",

```

```

    forbidden_col = "red"
  )

  # Define a `Knowledge` object without tiers
  kn_untiered <- knowledge(
    tpc_example,
    child_x1 %-->% child_x2 + youth_x3,
    youth_x4 %!-->% oldage_x5
  )
  # Plot with default layout
  plot(kn_untiered)

  # With a custom defined layout
  custom_layout <- data.frame(
    name = c("child_x1", "child_x2", "youth_x3", "youth_x4", "oldage_x5", "oldage_x6"),
    x = c(0, 1, 2, 2, 3, 4),
    y = c(0, 1, 0, -1, 0, 1)
  )
  plot(kn_untiered, layout = custom_layout)

```

precision

Precision

Description

Computes precision from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes precision as $TP / (TP + FP)$, where TP are true positives and FP are false positives. If $TP + FP = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
precision(truth, est, type = c("adj", "dir"))
```

Arguments

<code>truth</code>	A <code>caugi::caugi</code> object representing the truth graph.
<code>est</code>	A <code>caugi::caugi</code> object representing the estimated graph.
<code>type</code>	Character string specifying the comparison type: • <code>"adj"</code>: adjacency comparison. • <code>"dir"</code>: orientation comparison conditional on shared adjacencies.

Value

A numeric in $[0, 1]$.

See Also

Other metrics: [confusion\(\)](#), [evaluate\(\)](#), [f1_score\(\)](#), [false_omission_rate\(\)](#), [fdr\(\)](#), [g1_score\(\)](#), [npv\(\)](#), [recall\(\)](#), [reexports](#), [specificity\(\)](#)

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
precision(cg1, cg2, type = "adj")
precision(cg1, cg2, type = "dir")
```

<code>print.Disco</code>	<i>Print a Disco Object</i>
--------------------------	-----------------------------

Description

Print a Disco Object

Usage

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

Arguments

<code>x</code>	A Disco object.
<code>...</code>	Additional arguments (not used).

Value

Invisibly returns the Disco object.

See Also

Other print: [print.Knowledge\(\)](#)

Examples

```
data(tpc_example)
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)
```

```
cd_tges <- tpc(engine = "causalDisco", test = "fisher_z")
disco_cd_tges <- disco(data = tpc_example, method = cd_tges, knowledge = kn)
print(disco_cd_tges)
```

print.Knowledge	<i>Print a Knowledge Object</i>
-----------------	---------------------------------

Description

Print a Knowledge Object

Usage

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

Arguments

x	A Knowledge object.
...	Additional arguments (not used).

Value

Invisibly returns the Knowledge object.

See Also

Other print: [print.Disco\(\)](#)

Examples

```
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)
print(kn)
```

recall

*Recall***Description**

Computes recall from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes recall as $TP / (TP + FN)$, where TP are true positives and FN are false negatives. If $TP + FN = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
recall(truth, est, type = c("adj", "dir"))
```

Arguments

<code>truth</code>	A <code>caugi::caugi</code> object representing the truth graph.
<code>est</code>	A <code>caugi::caugi</code> object representing the estimated graph.
<code>type</code>	Character string specifying the comparison type: • <code>"adj"</code>: adjacency comparison. • <code>"dir"</code>: orientation comparison conditional on shared adjacencies.

Value

A numeric in $[0, 1]$.

See Also

Other metrics: `confusion()`, `evaluate()`, `f1_score()`, `false_omission_rate()`, `fdr()`, `g1_score()`, `npv()`, `precision()`, `reexports`, `specificity()`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
recall(cg1, cg2, type = "adj")
recall(cg1, cg2, type = "dir")
```

reg_test	<i>Regression-based Information Loss Test</i>
----------	---

Description

Test whether x and y are associated, given `conditioning_set` using a generalized linear model.

Usage

```
reg_test(x, y, conditioning_set, suff_stat)
```

Arguments

<code>x</code>	Index of x variable.
<code>y</code>	Index of y variable.
<code>conditioning_set</code>	Index vector of conditioning variable(s), possibly NULL.
<code>suff_stat</code>	List with data, binary variables and order.

Details

All included variables should be either numeric or binary. If y is binary, a logistic regression model is fitted. If y is numeric, a linear regression model is fitted. x and `conditioning_set` are included as explanatory variables. Any numeric variables among x and `conditioning_set` are modeled with spline expansions (natural splines, 3 df). This model is tested against a numeric where x (including a possible spline expansion) has been left out using a likelihood ratio test. The model is fitted in both directions (interchanging the roles of x and y). The final p-value is the maximum of the two obtained p-values.

Value

A numeric, which is the p-value of the test.

Examples

```
# Use via disco() and method functions (recommended):
data(tpc_example)
my_tpc <- tpc(engine = "causalDisco", test = "reg", alpha = 0.05)
result <- disco(data = tpc_example, method = my_tpc)
```

register_engine	<i>Register a New Engine</i>
-----------------	------------------------------

Description

The built-in engines ("bnlearn", "causalDisco", "pcalg", "tetrad") are backed by R6 search classes ([BnlearnSearch](#), [CausalDiscoSearch](#), [PcalgSearch](#), [TetradSearch](#)) that [make_runner\(\)](#) knows how to build and configure. `register_engine()` lets you plug in a backend for a package that isn't one of the built-ins, so that it can be selected by name from [make_runner\(\)](#) the same way as a built-in engine.

Usage

```
register_engine(name, make_runner_fn, pkgs = character(0))
```

Arguments

name	Engine name (string). Cannot be one of the built-in engine names ("bnlearn", "causalDisco", "pcalg", "tetrad").
make_runner_fn	A function implementing the engine. See Details.
pkgs	Character vector of package names required by the engine. Checked, with an informative error if missing, before <code>make_runner_fn</code> is called.

Details

`make_runner_fn` must accept `alg` and `...`, and return a runner: a list with two elements,

- `set_knowledge`, a function that takes a single Knowledge object and configures background knowledge on the underlying search, and
- `run`, a function that takes a single data frame, runs the search, and returns the algorithm's result using [as_disco\(\)](#).

[make_runner\(\)](#) calls `make_runner_fn` with `alg`, `test`, `alpha`, `score`, and any additional arguments it was itself called with. Pick up whichever of these your engine needs by naming them explicitly, and let `...` absorb the rest.

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

register_tetrad_algorithm

Register a New Tetrad Algorithm

Description

Registers a new Tetrad algorithm by adding it to the internal registry. The `setup_fun()` should be a function that takes the same arguments as the runner function for the algorithm and sets up the Tetrad search object accordingly. This allows you to extend the set of Tetrad algorithms that can be used with `causalDisco`.

Usage

```
register_tetrad_algorithm(name, setup_fun)
```

Arguments

name	Algorithm name (string)
setup_fun	A function that sets up the Tetrad search object for the algorithm. It should take the same arguments as the runner function for the algorithm.

See Also

Other Extending `causalDisco`: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [reset_engine_registry\(\)](#), [reset_tetrad_alg_registry\(\)](#)

remove_edge

Remove an Edge from Knowledge

Description

Drop a single directed edge specified by `from` and `to`. Errors if the edge does not exist.

Usage

```
remove_edge(kn, from, to)
```

Arguments

kn	A Knowledge object.
from	The source node (unquoted or character).
to	The target node (unquoted or character).

Value

The updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_causi()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# remove variables and their incident edges
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3
)
print(kn)

kn <- remove_edge(kn, child_x1, youth_x3)
print(kn)

kn <- remove_vars(kn, starts_with("child_"))
print(kn)

kn <- remove_tiers(kn, "child")
print(kn)
```

remove_tiers

Remove Tiers from Knowledge

Description

Drops tier definitions (and un-tiers any vars assigned to them).

Usage

```
remove_tiers(kn, ...)
```

Arguments

kn A Knowledge object.
 ... Tier labels (unquoted or character) or numeric indices.

Value

An updated Knowledge object.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```
# remove variables and their incident edges
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3
)
print(kn)

kn <- remove_edge(kn, child_x1, youth_x3)
print(kn)

kn <- remove_vars(kn, starts_with("child_"))
print(kn)

kn <- remove_tiers(kn, "child")
print(kn)
```

remove_vars

Remove Variables from Knowledge

Description

Drops the given variables from `kn$vars`, and automatically removes any edges that mention them.

Usage

```
remove_vars(kn, ...)
```

Arguments

<code>kn</code>	A Knowledge object.
<code>...</code>	Unquoted variable names or tidy-select helpers.

Value

An updated Knowledge object.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```
# remove variables and their incident edges
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  ),
  child_x1 %-->% youth_x3
)
print(kn)

kn <- remove_edge(kn, child_x1, youth_x3)
print(kn)

kn <- remove_vars(kn, starts_with("child_"))
print(kn)

kn <- remove_tiers(kn, "child")
print(kn)
```

reorder_tiers	<i>Reorder Tiers in Knowledge</i>
---------------	-----------------------------------

Description

Reorder Tiers in Knowledge

Usage

```
reorder_tiers(kn, order, by_index = FALSE)
```

Arguments

kn	A Knowledge object.
order	A vector that lists <i>every</i> tier exactly once, either by label (default) or by numeric index (by_index = TRUE). Be careful if you have numeric tier labels.
by_index	If TRUE, treat order as the positions instead of labels. Defaults to FALSE.

Value

The same Knowledge object with tiers rearranged.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [set_knowledge\(\)](#), [unfreeze\(\)](#)

Examples

```
# Move one tier relative to another
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  )
)

kn <- reorder_tiers(kn, c("youth", "child", "oldage"))
print(kn)
```

reposition_tier	<i>Move a Tier Relative to Another in Knowledge</i>
-----------------	---

Description

Move a Tier Relative to Another in Knowledge

Usage

```
reposition_tier(kn, tier, before = NULL, after = NULL, by_index = FALSE)
```

Arguments

kn	A Knowledge object.
tier	The tier to move (label or index, honouring by_index).
before	Exactly one of these must be supplied and must identify another existing tier.
after	Exactly one of these must be supplied and must identify another existing tier.
by_index	If TRUE, treat order as the positions instead of labels. Defaults to FALSE.

Value

The updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
# Move one tier relative to another
data(tpc_example)

kn <- knowledge(
  head(tpc_example),
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    oldage ~ starts_with("old")
  )
)

kn <- reorder_tiers(kn, c("youth", "child", "oldage"))
print(kn)
```

require_edge

Add Required Edges to Knowledge

Description

Require one or more directed edges. Arguments follow the same rules as `forbid_edge()` but a required edge may only be given in *one* direction ($X \sim Y$ **or** $Y \sim X$, not both).

Usage

```
require_edge(kn, ...)
```

Arguments

<code>kn</code>	A Knowledge object.
<code>...</code>	One or more two-sided formulas.

Value

The updated Knowledge object.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `seq_tiers()`, `set_knowledge()`, `unfreeze()`

Examples

```
data(tpc_example)

# create Knowledge object using verbs
kn1 <- knowledge() |>
  add_vars(names(tpc_example)) |>
  add_tier(child) |>
  add_tier(old, after = child) |>
  add_tier(youth, before = old) |>
  add_to_tier(child ~ starts_with("child")) |>
  add_to_tier(youth ~ starts_with("youth")) |>
  add_to_tier(old ~ starts_with("oldage")) |>
  require_edge(child_x1 ~ youth_x3) |>
  forbid_edge(child_x2 ~ youth_x4) |>
  add_exogenous(child_x1) # synonym: add_exo()

# set kn1 to frozen
# (meaning you cannot add variables to the Knowledge object anymore)
# this is to get a true on the identical check
```

```

kn1$ frozen <- TRUE

# create identical Knowledge object using DSL
kn2 <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("oldage")
  ),
  child_x1 %-->% youth_x3,
  child_x2 %!->% youth_x4,
  exo(child_x1) # synonym: exogenous()
)

print(identical(kn1, kn2))

# cannot require an edge against tier direction
try(
  kn1 |> require_edge(oldage_x6 ~ child_x1)
)

# cannot forbid and require same edge
try(
  kn1 |> forbid_edge(child_x1 ~ youth_x3)
)

```

reset_engine_registry *Reset the Engine Registry*

Description

Clears all custom registered engines.

Usage

```
reset_engine_registry()
```

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_tetrad_alg_registry\(\)](#)

reset_tetrad_alg_registry

Reset the Tetrad Algorithm Registry

Description

Clears all custom registered algorithms.

Usage

```
reset_tetrad_alg_registry()
```

See Also

Other Extending causalDisco: [as_disco\(\)](#), [distribute_engine_args\(\)](#), [list_registered_engines\(\)](#), [list_registered_tetrad_algorithms\(\)](#), [make_method\(\)](#), [make_runner\(\)](#), [new_disco_method\(\)](#), [register_engine\(\)](#), [register_tetrad_algorithm\(\)](#), [reset_engine_registry\(\)](#)

rfci

RFCI Algorithm for Causal Discovery

Description

Run the Really Fast Causal Inference algorithm for causal discovery using one of several engines.

Usage

```
rfci(engine = c("tetrad", "pcalg"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library. "pcalg" pcalg R package.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**,
- [PcalgSearch](#) for **pcalg**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using `disco()`. This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See `knowledge()` for how to construct it.
- `caugi` A `caugi::caugi` object representing the learned causal graph. This graph is an RFCI-PAG (RFCI Partial Ancestral Graph), but since RFCI-PAGs are not yet natively supported in `caugi`, it is currently stored with class UNKNOWN.

Please see the definition 3.2 of the paper referenced for definition of an RFCI-PAG, and it's differences from a standard PAG.

References

Colombo, D., Maathuis, M. H., Kalisch, M., & Richardson, T. S. (2012). Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, 294-321.

See Also

Other causal discovery algorithms: `boss()`, `boss_fci()`, `fci()`, `ges()`, `gfci()`, `grasp()`, `grasp_fci()`, `gs()`, `iamb-family`, `pc()`, `sp_fci()`, `tfci()`, `tges()`, `tpc()`

Examples

```
data(tpc_example)

# Recommended path using disco()
rfci_pcalg <- rfci(engine = "pcalg", test = "fisher_z", alpha = 0.05)
disco(tpc_example, rfci_pcalg)

# or using rfci_pcalg directly
rfci_pcalg(tpc_example)

# With all algorithm arguments specified
rfci_pcalg <- rfci(
  engine = "pcalg",
  test = "fisher_z",
  alpha = 0.05,
  skel.method = "original",
  fixedGaps = NULL,
  fixedEdges = NULL,
  NAdelete = FALSE,
  m.max = 10,
  rules = c(rep(TRUE, 9), FALSE),
  conservative = TRUE,
```

```

    maj.rule = FALSE,
    numCores = 1,
    verbose = FALSE
  )
  disco(tpc_example, rfci_pcalg)

#### Using tetrad engine with tier knowledge ####
# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  kn <- knowledge(
    tpc_example,
    tier(
      child ~ tidyselect::starts_with("child"),
      youth ~ tidyselect::starts_with("youth"),
      oldage ~ tidyselect::starts_with("oldage")
    )
  )

  # Recommended path using disco()
  rfci_tetrad <- rfci(engine = "tetrad", test = "fisher_z", alpha = 0.05)
  disco(tpc_example, rfci_tetrad, knowledge = kn)

  # or using rfci_tetrad directly
  rfci_tetrad <- rfci_tetrad |> set_knowledge(kn)
  rfci_tetrad(tpc_example)
}

# With all algorithm arguments specified
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  rfci_tetrad <- rfci(
    engine = "tetrad",
    test = "fisher_z",
    alpha = 0.05,
    depth = 10,
    stable_fas = FALSE,
    max_disc_path_length = 2,
    complete_rule_set_used = TRUE
  )
  disco(tpc_example, rfci_tetrad)
}

```

seq_tiers

*Generate a Bundle of Tier-Variable Formulas***Description**

Quickly create a series of two-sided formulas for use with `tier()`, where each formula maps a numeric tier index to a `tidyselect` specification that contains the placeholder `i`. The placeholder `i` is replaced by each element of `tiers` in turn, allowing you to write a single template rather than many nearly identical formulas.

Usage

```
seq_tiers(tiers, vars)
```

Arguments

tiers	An integer vector of tier indices (each ≥ 1). These will appear as the left-hand sides of the generated formulas.
vars	A tidyselect specification (unevaluated) that <i>must</i> contain the special placeholder <code>i</code> , either as the symbol <code>i</code> or inside a string like <code>"...{i}..."</code> . For each value of <code>i</code> in <code>tiers</code> , that placeholder will be substituted and the resulting call used as the right-hand side of a formula.

Value

A list of two-sided formulas, each of class "tier_bundle". You can pass this list directly to `tier()` (which will expand it automatically).

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `set_knowledge()`, `unfreeze()`

Examples

```
# generate a bundle of tier formulas using a pattern with {i}
# here we create: 1 ~ matches("^child_x1$"), 2 ~ matches("^child_x2$")
data(tpc_example)

kn <- knowledge(
  tpc_example,
  tier(seq_tiers(1:2, matches("^child_x{i}$"))))
)
print(kn)
```

set_knowledge

Set Background Knowledge to Disco Method

Description

Set Background Knowledge to Disco Method

Usage

```
set_knowledge(method, knowledge)

## S3 method for class 'disco_method'
set_knowledge(method, knowledge)
```

Arguments

method A "disco_method" function.

knowledge A Knowledge object appropriate for the engine.

See Also

Other knowledge helpers: [+.Knowledge\(\)](#), [add_exogenous\(\)](#), [add_tier\(\)](#), [add_to_tier\(\)](#), [add_vars\(\)](#), [as_bnlearn_knowledge\(\)](#), [as_pcalg_constraints\(\)](#), [as_tetrad_knowledge\(\)](#), [convert_tiers_to_forbidden\(\)](#), [deparse_knowledge\(\)](#), [forbid_edge\(\)](#), [get_tiers\(\)](#), [knowledge\(\)](#), [knowledge_to_caugi\(\)](#), [remove_edge\(\)](#), [remove_tiers\(\)](#), [remove_vars\(\)](#), [reorder_tiers\(\)](#), [reposition_tier\(\)](#), [require_edge\(\)](#), [seq_tiers\(\)](#), [unfreeze\(\)](#)

sim_dag

Simulate a Random DAG

Description

Simulates a random directed acyclic graph adjacency (DAG) matrix with n nodes and either m edges, edge creation probability p , or edge creation probability range p_range .

Usage

```
sim_dag(n, m = NULL, p = NULL)
```

Arguments

n The number of nodes.

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.

Value

The sampled `caugi` object.

See Also

[caugi::generate_graph\(\)](#)

Examples

```
# Simulate a DAG with 5 nodes and 3 edges
sim_dag(n = 5, m = 3)

# Simulate a DAG with 5 nodes and edge creation probability of 0.2
sim_dag(n = 5, p = 0.2)
```

sim_data

Generate Synthetic Data from a Linear Gaussian DAG

Description

Generates synthetic data from a directed acyclic graph (DAG) specified as a `caugi` graph object. Each node is modeled as a linear combination of its parents plus additive Gaussian noise. Coefficients are randomly signed with a minimum absolute value, and noise standard deviations are sampled log-uniformly from a specified range. Custom node equations can override automatic linear generation.

Usage

```
sim_data(
  cg,
  n,
  ...,
  standardize = TRUE,
  coef_range = c(0.1, 0.9),
  error_sd = c(0.3, 2),
  seed = NULL
)
```

Arguments

<code>cg</code>	A <code>caugi</code> graph object representing a DAG.
<code>n</code>	Integer. Number of observations to simulate.
<code>...</code>	Optional named node equations to override automatic linear generation. Each should be an expression referencing all parent nodes.
<code>standardize</code>	Logical. If TRUE, each column of the output is standardized to mean 0 and standard deviation 1.
<code>coef_range</code>	Numeric vector of length 2 specifying the minimum and maximum absolute value of edge coefficients. For each edge, an absolute value is sampled uniformly from this range and then assigned a positive or negative sign with equal probability. Must satisfy <code>coef_range[1] > 0</code> and <code>coef_range[2] >= coef_range[1]</code> .
<code>error_sd</code>	Numeric vector of length 2 specifying the minimum and maximum standard deviation of the additive Gaussian noise at each node. For each node, a standard deviation is sampled from a log-uniform distribution over this range. Must satisfy <code>error_sd[1] > 0</code> and <code>error_sd[2] >= error_sd[1]</code> .

seed Optional integer. Sets the random seed for reproducibility.

Value

A tibble of simulated data with one column per node in the DAG, ordered according to the graph's node order. Standardization is applied if `standardize = TRUE`.

The returned tibble has an attribute `generating_model`, which is a list containing:

- `sd`: Named numeric vector of node-specific noise standard deviations.
- `coef`: Named list of numeric vectors, where each element corresponds to a child node. For a child node, the vector stores the coefficients of its parent nodes in the linear structural equation. That is: `generating_model$coef[[child]][parent]` gives the coefficient of parent in the equation for child.

Examples

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

# Simulate 1000 observations
sim_data <- sim_data(
  cg,
  n = 1000,
  coef_range = c(0.2, 0.8),
  error_sd = c(0.5, 1.5)
)

head(sim_data)
attr(sim_data, "generating_model")

# Simulate with custom equation for node C
sim_data_custom <- sim_data(
  cg,
  n = 1000,
  C = A^2 + B + rnorm(n, sd = 0.7),
  seed = 1405
)
head(sim_data_custom)
attr(sim_data_custom, "generating_model")
```

sp_fci

SP-FCI Algorithm for Causal Discovery

Description

Run the Sparsest Permutation-based Fast Causal Inference algorithm for causal discovery using one of several engines. Can be computationally intensive.

Usage

```
sp_fci(engine = "tetrad", score, test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "tetrad" Tetrad Java library.
score	Character; name of the scoring function to use.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. score and algorithm parameters).

Details

For specific details on the supported scores, and parameters for each engine, see:

- [TetradSearch](#) for **Tetrad**.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [tfci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  # Recommended path using disco()
  boss_fci_tetrad <- boss_fci(
    engine = "tetrad",
    score = "sem_bic",
    test = "fisher_z"
  )
  disco(tpc_example, boss_fci_tetrad)
```

```

    # or using boss_fci_tetrad directly
    boss_fci_tetrad(tpc_example)
  }

  ##### With tier knowledge #####
  if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
    kn <- knowledge(
      tpc_example,
      tier(
        child ~ tidyselect::starts_with("child"),
        youth ~ tidyselect::starts_with("youth"),
        oldage ~ tidyselect::starts_with("oldage")
      )
    )

    # Recommended path using disco()
    boss_fci_tetrad <- boss_fci(
      engine = "tetrad",
      score = "sem_bic",
      test = "fisher_z"
    )
    disco(tpc_example, boss_fci_tetrad, knowledge = kn)

    # or using boss_fci_tetrad directly
    boss_fci_tetrad <- boss_fci_tetrad |> set_knowledge(kn)
    boss_fci_tetrad(tpc_example)
  }

  # With all algorithm arguments specified
  if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
    boss_fci_tetrad <- boss_fci(
      engine = "tetrad",
      score = "poisson_prior",
      test = "rank_independence",
      depth = 3,
      max_disc_path_length = 5,
      use_bes = FALSE,
      use_heuristic = FALSE,
      complete_rule_set_used = FALSE,
      guarantee_pag = TRUE
    )
    disco(tpc_example, boss_fci_tetrad)
  }

```

Description

Computes specificity from two PDAG `caugi::caugi` objects. It converts the `caugi::caugi` objects to adjacency matrices and computes specificity as $TN / (TN + FP)$, where TN are true negatives and FP are false positives. If $TN + FP = 0$, 1 is returned.

Only supports `caugi::caugi` objects whose edges are restricted to `-->`, `<-->`, `---`, or absence of an edge.

Usage

```
specificity(truth, est, type = c("adj", "dir"))
```

Arguments

- | | |
|-------|---|
| truth | A <code>caugi::caugi</code> object representing the truth graph. |
| est | A <code>caugi::caugi</code> object representing the estimated graph. |
| type | Character string specifying the comparison type: <ul style="list-style-type: none"> • "adj": adjacency comparison. • "dir": orientation comparison conditional on shared adjacencies. |

Value

A numeric in [0,1].

See Also

Other metrics: `confusion()`, `evaluate()`, `f1_score()`, `false_omission_rate()`, `fdr()`, `g1_score()`, `npv()`, `precision()`, `recall()`, `reexports`

Examples

```
cg1 <- caugi::caugi(A %-->% B + C)
cg2 <- caugi::caugi(B %-->% A + C)
specificity(cg1, cg2, type = "adj")
specificity(cg1, cg2, type = "dir")
```

summary.Disco

Summarize a Disco Object

Description**[Deprecated]**

`summary()` for Disco objects is deprecated. Use `print()` instead.

Usage

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

Arguments

- object A Disco object.
- ... Additional arguments (not used).

summary.Knowledge	<i>Summarize a Knowledge Object</i>
-------------------	-------------------------------------

Description

[Deprecated]
summary() for Knowledge objects is deprecated. Use print() instead.

Usage

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

Arguments

- object A Disco object.
- ... Additional arguments (not used).

TetradSearch	<i>R6 Interface to Tetrad Search Algorithms</i>
--------------	---

Description

High-level wrapper around the Java-based **Tetrad** causal-discovery library. The class lets you choose independence tests, scores, and search algorithms from **Tetrad**, run them on an R data set, and retrieve the resulting graph or statistics.

Public fields

data Java object that stores the (possibly converted) data set used by **Tetrad**.

rdata Original **R** data.frame supplied by the user.

score Java object holding the scoring function selected with `set_score()`. Supply one of the method strings for `set_score()`. Recognised values are:

Continuous - Gaussian

- "ebic" - Extended BIC score.
- "gic" - Generalized Information Criterion (GIC) score.
- "poisson_prior" - Poisson prior score.
- "rank_bic" - Rank-based BIC score.
- "sem_bic" - SEM BIC score.
- "zhang_shen_bound" - Zhang and Shen bound score.

Discrete - categorical

- "bdeu" - Bayes Dirichlet Equivalent score with uniform priors.
- "discrete_bic" - BIC score for discrete data.

Mixed Discrete/Gaussian

- "basis_function_bic" - BIC score for basis-function models. This is a generalization of the Degenerate Gaussian score.
- "basis_function_blocks_bic" - BIC score for mixed data using basis-function models.
- "basis_function_sem_bic" - SEM BIC score for basis-function models.
- "conditional_gaussian" - Conditional Gaussian BIC score.
- "degenerate_gaussian" - Degenerate Gaussian BIC score.
- "mag_degenerate_gaussian_bic" - MAG Degenerate Gaussian BIC Score.

test Java object holding the independence test selected with `set_test()`. Supply one of the method strings for `set_test()`. Recognised values are:

Continuous - Gaussian

- "fisher_z" - Fisher Z (partial correlation) test.
- "poisson_prior" - Poisson prior test.
- "rank_independence" - Rank-based independence test.
- "sem_bic" - SEM BIC test.

Discrete - categorical

- "chi_square" - chi-squared test
- "g_square" - likelihood-ratio G^2 test.
- "probabilistic" - Uses BCInference by Cooper and Bui to calculate probabilistic conditional independence judgments.

General

- "gin" - Generalized Independence Noise test.
- "kci" - Kernel Conditional Independence Test (KCI) by Kun Zhang.
- "rcit" - Randomized Conditional Independence Test (RCIT).

Mixed Discrete/Gaussian

- "basis_function_blocks" - Basis-function blocks test.
- "basis_function_lrt" - basis-function likelihood-ratio.
- "conditional_gaussian" - Conditional Gaussian test as a likelihood ratio test.
- "degenerate_gaussian" - Degenerate Gaussian test as a likelihood ratio test.

`alg` Java object representing the search algorithm. Supply one of the method strings for `set_alg()`.
Recognised values are:

Constraint-based

- "fci" - FCI algorithm. See [fci\(\)](#).
- "pc" - Peter-Clark (PC) algorithm. See [pc\(\)](#).
- "rfci" - Restricted FCI algorithm. See [rfci\(\)](#).

Hybrid

- "boss_fci" - BOSS-FCI algorithm. See [boss_fci\(\)](#).
- "gfcf" - GFCI algorithm. See [gfcf\(\)](#).
- "grasp_fci" - GRaSP-FCI algorithm. See [grasp_fci\(\)](#).
- "sp_fci" - Sparsest Permutation using FCI. See [sp_fci\(\)](#).

Score-based

- "boss" - BOSS algorithm. See [boss\(\)](#).
- "ges" ("fges") - (Fast) Greedy Equivalence Search (GES) algorithm. See [ges\(\)](#).
- "grasp" - GRaSP (Greedy Relations of Sparsest Permutation) algorithm. See [grasp\(\)](#).

`mc_test` Java independence-test object used by the Markov checker.

`java` Java object returned by the search (typically a graph).

`result` Convenience alias for `java`; may store additional metadata depending on the search type.

`knowledge` Java Knowledge object carrying background constraints (required/forbidden edges).

`params` Java Parameters object holding algorithm settings.

`bootstrap_graphs` Java List of graphs produced by bootstrap resampling, if that feature was requested.

`mc_ind_results` Java List with Markov-checker test results.

Methods**Public methods:**

- [TetradSearch\\$new\(\)](#)
- [TetradSearch\\$set_test\(\)](#)
- [TetradSearch\\$set_score\(\)](#)
- [TetradSearch\\$set_alg\(\)](#)
- [TetradSearch\\$set_knowledge\(\)](#)
- [TetradSearch\\$set_params\(\)](#)
- [TetradSearch\\$get_parameters_for_function\(\)](#)
- [TetradSearch\\$run_search\(\)](#)
- [TetradSearch\\$set_bootstrapping\(\)](#)

- `TetradSearch$set_data()`
- `TetradSearch$set_verbose()`
- `TetradSearch$set_time_lag()`
- `TetradSearch$get_data()`
- `TetradSearch$get_knowledge()`
- `TetradSearch$get_java()`
- `TetradSearch$get_string()`
- `TetradSearch$get_dot()`
- `TetradSearch$get_amat()`
- `TetradSearch$clone()`

`TetradSearch$new()`: Initializes the TetradSearch object, creating new Java objects for knowledge and params.

Usage:

`TetradSearch$new()`

`TetradSearch$set_test()`: Sets the independence test to use in Tetrad.

Usage:

`TetradSearch$set_test(method, ..., use_for_mc = FALSE)`

Arguments:

`method` (character) Name of the test method (e.g., "chi_square", "fisher_z").

- "basis_function_blocks" - Basis-function blocks test
- "basis_function_lrt" - basis-function likelihood-ratio
- "chi_square" - chi-squared test
- "conditional_gaussian" - Mixed discrete/continuous test
- "degenerate_gaussian" - Degenerate Gaussian test as a likelihood ratio test
- "fisher_z" - Fisher $\lambda(Z)$ (partial correlation) test
- "gin" - Generalized Independence Noise test
- "kci" - Kernel Conditional Independence Test (KCI) by Kun Zhang
- "poisson_prior" - Poisson prior test
- "probabilistic" - Uses BCInference by Cooper and Bui to calculate probabilistic conditional independence judgments.
- "rcit" - Randomized Conditional Independence Test (RCIT)
- "rank_independence" - Rank-based independence test
- "sem_bic" - SEM BIC test

... Additional arguments passed to the private test-setting methods. For the following tests, the following parameters are available:

- "basis_function_blocks" - Basis-function blocks test.
 - `alpha = 0.05` - Significance level for the independence test,
 - `basis_type = "polynomial"` - The type of basis to use. Supported types are "polynomial", "legendre", "hermite", and "chebyshev",
 - `truncation_limit = 3` - Basis functions 1 through this number will be used. The Degenerate Gaussian category indicator variables for mixed data are also used.

- "basis_function_lrt" - basis-function likelihood-ratio
 - truncation_limit = 3 - Basis functions 1 through this number will be used. The Degenerate Gaussian category indicator variables for mixed data are also used,
 - alpha = 0.05 - Significance level for the likelihood-ratio test,
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse,
 - do_one_equation_only = FALSE - If TRUE, only one equation should be used when expanding the basis.
- "chi_square" - chi-squared test
 - min_count = 1 - Minimum count for the chi-squared test per cell. Increasing this can improve accuracy of the test estimates,
 - alpha = 0.05 - Significance level for the independence test,
 - cell_table_type = "ad" - The type of cell table to use for optimization. Available types are: "ad" - AD tree, "count" - Count sample.
- "conditional_gaussian" - Mixed discrete/continuous test
 - alpha = 0.05 - Significance level for the independence test,
 - discretize = TRUE - If TRUE for the conditional Gaussian likelihood, when scoring $X \rightarrow D$ where X is continuous and D discrete, one should simply discretize X for just those cases. If FALSE, the integration will be exact,
 - num_categories_to_discretize = 3 - In case the exact algorithm is not used for discrete children and continuous parents is not used, this parameter gives the number of categories to use for this second (discretized) backup copy of the continuous variables,
 - min_sample_size_per_cell = 4 - Minimum sample size per cell for the independence test.
- "degenerate_gaussian" - Degenerate Gaussian likelihood ratio test
 - alpha = 0.05 - Significance level for the independence test,
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "fisher_z" - Fisher $\sqrt{Z}$ (partial correlation) test
 - alpha = 0.05 - Significance level for the independence test,
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "gin" - Generalized Independence Noise test.
 - alpha = 0.05 - Significance level for the independence test,
 - gin_backend = "dcor" - Unconditional test for residual independence. Available types are "dcor" - Distance correlation (for non-linear) and "pearson" - Pearson correlation (for linear),
 - num_permutations = 200 - Number of permutations used for "dcor" backend. If "pearson" backend is used, this parameter is ignored.
 - gin_ridge = 1e-8 - Ridge parameter used when computing residuals. A small number ≥ 0 .
 - seed = -1 - Random seed for the independence test. If -1, no seed is set.
- "kci" - Kernel Conditional Independence Test (KCI) by Kun Zhang
 - alpha = 0.05 - Significance level for the independence test,

- `approximate = TRUE` - If TRUE, use the approximate Gamma approximation algorithm. If FALSE, use the exact,
- `scaling_factor = 1` - For Gaussian kernel: The scaling factor * Silverman bandwidth.
- `num_bootstraps = 1000` - Number of bootstrap samples to use for the KCI test. Only used if `approximate = FALSE`.
- `threshold = 1e-3` - Threshold for the KCI test. Threshold to determine how many eigenvalues to use – the lower the more (0 to 1).
- `kernel_type = "gaussian"` - The type of kernel to use. Available types are "gaussian", "linear", or "polynomial".
- `polyd = 5` - The degree of the polynomial kernel, if used.
- `polyc = 1` - The constant of the polynomial kernel, if used.
- `"poisson_prior"` - Poisson prior test
 - `poisson_lambda = 1` - Lambda parameter for the Poisson distribution (> 0),
 - `singularity_lambda = 0.0` - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- `"probabilistic"` - Uses BCInference by Cooper and Bui to calculate probabilistic conditional independence judgments.
 - `threshold = FALSE` - Set to TRUE if using the cutoff threshold for the independence test,
 - `cutoff = 0.5` - Cutoff for the independence test,
 - `prior_ess = 10` - Prior equivalent sample size for the independence test. This number is added to the sample size for each conditional probability table in the model and is divided equally among the cells in the table.
- `"rcit"` - Randomized Conditional Independence Test (RCIT).
 - `alpha = 0.05` - Significance level for the independence test,
 - `rcit_approx = "lpb4"` - Null approximation method. Recognized values are: "lpb4" - Lindsay-Pilla-Basak method with 4 support points, "hbe" - Hall-Buckley-Eagleson method, "gamma" - Gamma (Satterthwaite-Welch), "chi_square" - Chi-square (normalized), "permutation" - Permutation-based (computationally intensive),
 - `rcit_ridge = 1e-3` - Ridge parameter used when computing residuals. A small number ≥ 0 ,
 - `num_feat = 10` - Number of random features to use for the regression of X and Y on Z. Values between 5 and 20 often suffice.
 - `num_fourier_feat_xy = 5` - Number of random Fourier features to use for the tested variables X and Y. Small values often suffice (e.g., 3 to 10),
 - `num_fourier_feat_z = 100` - Number of random Fourier features to use for the conditioning set Z. Values between 50 and 300 often suffice,
 - `center_features = TRUE` - If TRUE, center the random features to have mean zero. Recommended for better numerical stability,
 - `use_rcit = TRUE` - If TRUE, use RCIT; if FALSE, use RCoT (Randomized Conditional Correlation Test),
 - `num_permutations = 500` - Number of permutations used for the independence test when `rcit_approx = "permutation"` is selected.
 - `seed = -1` - Random seed for the independence test. If -1, no seed is set.

- "rank_independence" - Rank-based independence test.
 - $\alpha = 0.05$ - Significance level for the independence test.
- "sem_bic" - SEM BIC test.
 - $\text{penalty_discount} = 2$ - Penalty discount factor used in $\text{BIC} = 2L - c_k \log N$, where c is the penalty. Higher c yield sparser graphs,
 - $\text{structure_prior} = 0$ - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution,
 - $\text{singularity_lambda} = 0.0$ - Small number ≥ 0 : Add λ to the diagonal, < 0 Pseudoinverse.

`use_for_mc` (logical) If TRUE, sets this test for the Markov checker `mc_test`.

Returns: Invisibly returns `self`, for chaining.

`TetradSearch$set_score()`: Sets the scoring function to use in Tetrad.

Usage:

`TetradSearch$set_score(method, ...)`

Arguments:

`method` (character) Name of the score (e.g., "sem_bic", "ebic", "bdeu").

- "bdeu" - Bayes Dirichlet Equivalent score with uniform priors.
- "basis_function_bic" - BIC score for basis-function models. This is a generalization of the Degenerate Gaussian score.
- "basis_function_blocks_bic" - BIC score for mixed data using basis-function models.
- "basis_function_sem_bic" - SEM BIC score for basis-function models.
- "conditional_gaussian" - Mixed discrete/continuous BIC score.
- "degenerate_gaussian" - Degenerate Gaussian BIC score.
- "discrete_bic" - BIC score for discrete data.
- "ebic" - Extended BIC score.
- "gic" - Generalized Information Criterion (GIC) score.
- "mag_degenerate_gaussian_bic" - MAG Degenerate Gaussian BIC Score.
- "poisson_prior" - Poisson prior score.
- "rank_bic" - Rank-based BIC score.
- "sem_bic" - SEM BIC score.
- "zhang_shen_bound" - Zhang and Shen bound score.

... Additional arguments passed to the private score-setting methods. For the following scores, the following parameters are available:

- "bdeu" - Bayes Dirichlet Equivalent score with uniform priors.
 - $\text{sample_prior} = 10$ - This sets the prior equivalent sample size. This number is added to the sample size for each conditional probability table in the model and is divided equally among the cells in the table,
 - $\text{singularity_lambda} = 0.0$ - Small number ≥ 0 : Add λ to the diagonal, < 0 Pseudoinverse.
- "basis_function_bic" - BIC score for basis-function models. This is a generalization of the Degenerate Gaussian score.

- `truncation_limit = 3` - Basis functions 1 through this number will be used. The Degenerate Gaussian category indicator variables for mixed data are also used,
- `penalty_discount = 2` - Penalty discount. Higher penalty yields sparser graphs,
- `singularity_lambda = 0.0` - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse,
- `do_one_equation_only = FALSE` - If TRUE, only one equation should be used when expanding the basis.
- `"basis_function_blocks_bic"` - BIC score for mixed data using basis-function models.
 - `basis_type = "polynomial"` - The type of basis to use. Supported types are "polynomial", "legendre", "hermite", and "chebyshev",
 - `penalty_discount = 2` - Penalty discount factor used in $BIC = 2L - ck \log N$, where c is the penalty. Higher c yield sparser graphs,
 - `truncation_limit = 3` - Basis functions 1 through this number will be used. The Degenerate Gaussian category indicator variables for mixed data are also used.
- `"basis_function_sem_bic"` - SEM BIC score for basis-function models.
 - `penalty_discount = 2` - Penalty discount factor used in $BIC = 2L - ck \log N$, where c is the penalty. Higher c yield sparser graphs,
 - `jitter = 1e-8` - Small non-negative constant added to the diagonal of covariance/correlation matrices for numerical stability,
 - `truncation_limit = 3` - Basis functions 1 through this number will be used. The Degenerate Gaussian category indicator variables for mixed data are also used.
- `"conditional_gaussian"` - Mixed discrete/continuous BIC score.
 - `penalty_discount = 1` - Penalty discount. Higher penalty yields sparser graphs,
 - `discretize = TRUE` - If TRUE for the conditional Gaussian likelihood, when scoring $X \rightarrow D$ where X is continuous and D discrete, one should simply discretize X for just those cases. If FALSE, the integration will be exact,
 - `num_categories_to_discretize = 3` - In case the exact algorithm is not used for discrete children and continuous parents is not used, this parameter gives the number of categories to use for this second (discretized) backup copy of the continuous variables,
 - `structure_prior = 0` - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution.
- `"degenerate_gaussian"` - Degenerate Gaussian BIC score.
 - `penalty_discount = 1` - Penalty discount. Higher penalty yields sparser graphs,
 - `structure_prior = 0` - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution,
 - `singularity_lambda = 0.0` - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- `"discrete_bic"` - BIC score for discrete data.
 - `penalty_discount = 2` - Penalty discount. Higher penalty yields sparser graphs,
 - `structure_prior = 0` - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution.

- "ebic" - Extended BIC score.
 - gamma - The gamma parameter in the EBIC score.
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "gic" - Generalized Information Criterion (GIC) score.
 - penalty_discount = 1 - Penalty discount. Higher penalty yields sparser graphs,
 - sem_gic_rule = "bic" - The following rules are available: "bic" - $\ln n$, "gic2" - $pn^{1/3}$, "ric" - $2\ln(pn)$, "ricc" - $2(\ln(pn) + \ln \ln(pn))$, "gic6" - $\ln n \ln(pn)$.
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "mag_degenerate_gaussian_bic" - MAG Degenerate Gaussian BIC Score.
 - penalty_discount = 1 - Penalty discount. Higher penalty yields sparser graphs,
 - structure_prior = 0 - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution,
- "poisson_prior" - Poisson prior score.
 - poisson_lambda = 1 - Lambda parameter for the Poisson distribution (> 0),
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "sem_bic" - SEM BIC score.
 - penalty_discount = 2 - Penalty discount factor used in $BIC = 2L - ck \log N$, where c is the penalty. Higher c yield sparser graphs,
 - structure_prior = 0 - The default number of parents for any conditional probability table. Higher weight is accorded to tables with about that number of parents. The prior structure weights are distributed according to a binomial distribution,
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.
- "rank_bic" - Rank-based BIC score.
 - gamma = 0.8 - Gamma parameter for Extended BIC (Chen and Chen, 2008). Between 0 and 1,
 - penalty_discount = 2 - Penalty discount factor used in $BIC = 2L - ck \log N$, where c is the penalty. Higher c yield sparser graphs.
- "zhang_shen_bound" - Zhang and Shen bound score.
 - risk_bound = 0.2 - This is the probability of getting the true model if a correct model is discovered. Could underfit.
 - singularity_lambda = 0.0 - Small number ≥ 0 : Add lambda to the diagonal, < 0 Pseudoinverse.

Returns: Invisibly returns self.

`TetradSearch$set_alg()`: Sets the causal discovery algorithm to use in Tetrad.

Usage:

`TetradSearch$set_alg(method, ...)`

Arguments:

`method` (character) Name of the algorithm (e.g., "fges", "pc", "fci", etc.).

... Additional parameters passed to the private algorithm-setting methods. For the following algorithms, the following parameters are available:

- "boss" - BOSS algorithm.
 - num_starts = 1 - The number of times the algorithm should be started from different initializations. By default, the algorithm will be run through at least once using the initialized parameters,
 - use_bes = TRUE - If TRUE, the algorithm uses the backward equivalence search from the GES algorithm as one of its steps,
 - use_data_order = TRUE - If TRUE, the data variable order should be used for the first initial permutation,
 - output_cpdag = TRUE - If TRUE, the DAG output of the algorithm is converted to a CPDAG.
- "boss_fci" - BOSS-FCI algorithm.
 - depth = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
 - max_disc_path_length = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,
 - use_bes = TRUE - If TRUE, the algorithm uses the backward equivalence search from the GES algorithm as one of its steps,
 - use_heuristic = FALSE - If TRUE, use the max p heuristic version,
 - complete_rule_set_used = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness,
 - guarantee_pag = FALSE - Ensure the output is a legal PAG (where feasible).
- "fci" - FCI algorithm.
 - depth = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
 - stable_fas = TRUE - If TRUE, the "stable" version of the PC adjacency search is used, which for $k > 0$ fixes the graph for depth $k + 1$ to that of the previous depth k .
 - max_disc_path_length = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,
 - complete_rule_set_used = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness.
 - guarantee_pag = FALSE - Ensure the output is a legal PAG (where feasible).
- "ges" ("fges") - Fast Greedy Equivalence Search (FGES) algorithm.
 - symmetric_first_step = FALSE - If TRUE, scores for both $X \rightarrow Y$ and $X \leftarrow Y$ will be calculated and the higher score used.
 - max_degree = -1 - Maximum degree of any node in the graph. Set to -1 for unlimited,
 - parallelized = FALSE - If TRUE, the algorithm should be parallelized,
 - faithfulness_assumed = FALSE - If TRUE, assume that if $X \perp\!\!\!\perp Y$ (by an independence test) then $X \perp\!\!\!\perp Y \mid Z$ for nonempty Z .
- "gfci" - GFci algorithm. Combines FGES and FCI.
 - depth = -1 - Maximum size of conditioning set,
 - max_degree = -1 - Maximum degree of any node in the graph. Set to -1 for unlimited,
 - max_disc_path_length = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,

- `complete_rule_set_used` = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness,
- `guarantee_pag` = FALSE - Ensure the output is a legal PAG (where feasible),
- `use_heuristic` = FALSE - If TRUE, use the max p heuristic.
- `start_complete` = FALSE - If TRUE, start from a complete graph.
- "grasp" - GRaSP (Greedy Relations of Sparsest Permutation) algorithm.
 - `covered_depth` = 4 - The depth of recursion for first search,
 - `singular_depth` = 1 - Recursion depth for singular tucks,
 - `nonsingular_depth` = 1 - Recursion depth for nonsingular tucks,
 - `ordered_alg` = FALSE - If TRUE, earlier GRaSP stages should be performed before later stages,
 - `raskutti_uhler` = FALSE - If TRUE, use Raskutti and Uhler's DAG-building method (test); if FALSE, use Grow-Shrink (score).
 - `use_data_order` = TRUE - If TRUE, the data variable order should be used for the first initial permutation,
 - `num_starts` = 1 - The number of times the algorithm should be started from different initializations. By default, the algorithm will be run through at least once using the initialized parameters.
- "grasp_fci" - GRaSP-FCI algorithm. Combines GRaSP and FCI.
 - `depth` = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
 - `stable_fas` = TRUE - If TRUE, the "stable" version of the PC adjacency search is used, which for $k > 0$ fixes the graph for depth $k + 1$ to that of the previous depth k .
 - `max_disc_path_length` = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,
 - `complete_rule_set_used` = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness,
 - `covered_depth` = 4 - The depth of recursion for first search,
 - `singular_depth` = 1 - Recursion depth for singular tucks,
 - `nonsingular_depth` = 1 - Recursion depth for nonsingular tucks,
 - `ordered_alg` = FALSE - If TRUE, earlier GRaSP stages should be performed before later stages,
 - `raskutti_uhler` = FALSE - If TRUE, use Raskutti and Uhler's DAG-building method (test); if FALSE, use Grow-Shrink (score).
 - `use_data_order` = TRUE - If TRUE, the data variable order should be used for the first initial permutation,
 - `num_starts` = 1 - The number of times the algorithm should be started from different initializations. By default, the algorithm will be run through at least once using the initialized parameters,
 - `guarantee_pag` = FALSE - If TRUE, ensure the output is a legal PAG (where feasible).
- "pc" - Peter-Clark (PC) algorithm
 - `conflict_rule` = 1 - The value of `conflict_rule` determines how collider conflicts are handled. 1 corresponds to the "overwrite" rule as introduced in the **pcalg** package, see `pcalg::pc()`. 2 means that all collider conflicts using bidirected edges should

be prioritized, while 3 means that existing colliders should be prioritized, ignoring subsequent conflicting information.

- depth = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
- stable_fas = TRUE - If TRUE, the "stable" version of the PC adjacency search is used, which for $k > 0$ fixes the graph for depth $k + 1$ to that of the previous depth k .
- guarantee_cpdag = FALSE - If TRUE, ensure the output is a legal CPDAG.
- "rfci" - Restricted FCI algorithm
 - depth = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
 - stable_fas = TRUE - If TRUE, the "stable" version of the PC adjacency search is used, which for $k > 0$ fixes the graph for depth $k + 1$ to that of the previous depth k .
 - max_disc_path_length = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,
 - complete_rule_set_used = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness.
- "sp_fci" - Sparsest Permutation using FCI
 - depth = -1 - Maximum size of conditioning set. Set to -1 for unlimited,
 - max_disc_path_length = -1 - Maximum length for any discriminating path. Set to -1 for unlimited,
 - complete_rule_set_used = TRUE - FALSE if the (simpler) final orientation rules set due to P. Spirtes, guaranteeing arrow completeness, should be used; TRUE if the (fuller) set due to J. Zhang, should be used guaranteeing additional tail completeness,
 - guarantee_pag = FALSE - Ensure the output is a legal PAG (where feasible),
 - use_heuristic = FALSE - If TRUE, use the max p heuristic version.

Returns: Invisibly returns self.

`TetradSearch$set_knowledge()`: Sets the background Knowledge object.

Usage:

`TetradSearch$set_knowledge(knowledge_obj)`

Arguments:

knowledge_obj An object containing **Tetrad** knowledge.

`TetradSearch$set_params()`: Sets parameters for the Tetrad search.

Usage:

`TetradSearch$set_params(...)`

Arguments:

... Named arguments for the parameters to set.

`TetradSearch$get_parameters_for_function()`: Retrieves the argument names of a matching private function.

Usage:

```
TetradSearch$get_parameters_for_function(
  fn_pattern,
  score = FALSE,
  test = FALSE,
  alg = FALSE
)
```

Arguments:

`fn_pattern` (character) A pattern that should match a private method name.

`score` If TRUE, retrieves parameters for a scoring function.

`test` If TRUE, retrieves parameters for a test function.

`alg` If TRUE, retrieves parameters for an algorithm.

Returns: (character) The names of the parameters.

`TetradSearch$run_search()`: Runs the chosen Tetrad algorithm on the data.

Usage:

```
TetradSearch$run_search(
  data = NULL,
  bootstrap = FALSE,
  int_cols_as_cont = TRUE
)
```

Arguments:

`data` (optional) If provided, overrides the previously set data.

`bootstrap` (logical) If TRUE, bootstrapped graphs will be generated.

`int_cols_as_cont` (logical) If TRUE, integer columns are treated as continuous, since **Tetrad** does not support ordinal data, but only either continuous or nominal data. Default is TRUE.

Returns: A Disco object (a list with a `caugi` and a `Knowledge` object). Also populates `self$java`.

`TetradSearch$set_bootstrapping()`: Configures bootstrapping parameters for the Tetrad search.

Usage:

```
TetradSearch$set_bootstrapping(
  number_resampling = 0,
  percent_resample_size = 100,
  add_original = TRUE,
  with_replacement = TRUE,
  resampling_ensemble = 1,
  seed = -1
)
```

Arguments:

`number_resampling` (integer) Number of bootstrap samples.

`percent_resample_size` (numeric) Percentage of sample size for each bootstrap.

`add_original` (logical) If TRUE, add the original dataset to the bootstrap set.

`with_replacement` (logical) If TRUE, sampling is done with replacement.

resampling_ensemble (integer) How the resamples are used or aggregated.
 seed (integer) Random seed, or -1 for none.

TetradSearch\$set_data(): Sets or overrides the data used by Tetrad.

Usage:

```
TetradSearch$set_data(data, int_cols_as_cont = TRUE)
```

Arguments:

data (data.frame) The new data to load.
 int_cols_as_cont (logical) If TRUE, integer columns are treated as continuous, since Tetrad does not support ordinal data, but only either continuous or nominal data. Default is TRUE.

TetradSearch\$set_verbose(): Toggles the verbosity in Tetrad.

Usage:

```
TetradSearch$set_verbose(verbose)
```

Arguments:

verbose (logical) TRUE to enable verbose logging, FALSE otherwise.

TetradSearch\$set_time_lag(): Sets an integer time lag for time-series algorithms.

Usage:

```
TetradSearch$set_time_lag(time_lag = 0)
```

Arguments:

time_lag (integer) The time lag to set.

TetradSearch\$get_data(): Retrieves the current Java data object.

Usage:

```
TetradSearch$get_data()
```

Returns: (Java object) Tetrad dataset.

TetradSearch\$get_knowledge(): Returns the background Knowledge object.

Usage:

```
TetradSearch$get_knowledge()
```

Returns: (Java object) Tetrad Knowledge.

TetradSearch\$get_java(): Gets the main Java result object (usually a graph) from the last search.

Usage:

```
TetradSearch$get_java()
```

Returns: (Java object) The Tetrad result graph or model.

TetradSearch\$get_string(): Returns the string representation of a given Java object or self\$java.

Usage:

```
TetradSearch$get_string(java_obj = NULL)
```

Arguments:

java_obj (Java object, optional) If NULL, uses self\$java.

Returns: (character) The toString() of that Java object.

TetradSearch\$get_dot(): Produces a DOT (Graphviz) representation of the graph.

Usage:

TetradSearch\$get_dot(java_obj = NULL)

Arguments:

java_obj (Java object, optional) If NULL, uses self\$java.

Returns: (character) The DOT-format string.

TetradSearch\$get_amat(): Produces an amat representation of the graph.

Usage:

TetradSearch\$get_amat(java_obj = NULL)

Arguments:

java_obj (Java object, optional) If NULL, uses self\$java.

Returns: (character) The adjacency matrix.

TetradSearch\$clone(): The objects of this class are cloneable with this method.

Usage:

TetradSearch\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
### tetrad_search R6 class examples ###

# Generally, we do not recommend using the R6 classes directly, but rather
# use the disco() or any method function, for example pc(), instead.

# Requires Tetrad to be installed
if (verify_tetrad()$installed && verify_tetrad()$java_ok) {
  data(num_data)

  # Recommended:
  my_pc <- pc(engine = "tetrad", test = "fisher_z")
  my_pc(num_data)

  # or
  disco(data = num_data, method = my_pc)

  # Example with detailed settings:
  my_pc2 <- pc(
    engine = "tetrad",
    test = "sem_bic",
```

```

    penalty_discount = 1,
    structure_prior = 1,
    singularity_lambda = 0.1
  )
  disco(data = num_data, method = my_pc2)

  # Using R6 class:
  s <- TetradSearch$new()

  s$set_data(num_data)
  s$set_test(method = "fisher_z", alpha = 0.05)
  s$set_alg("pc")

  g <- s$run_search()

  print(g)
}

```

 tfci

TFCI Algorithm for Causal Discovery

Description

Run the temporal FCI algorithm for causal discovery using causalDisco.

Usage

```
tfci(engine = c("causalDisco"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "causalDisco" causalDisco library.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests, see [CausalDiscoSearch](#). For additional parameters passed via ..., see [tfci_run\(\)](#).

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- `caugi` A `caugi::caugi` object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in **caugi**, it is currently stored with class UNKNOWN.

See Also

Other causal discovery algorithms: [boss\(\)](#), [boss_fci\(\)](#), [fci\(\)](#), [ges\(\)](#), [gfci\(\)](#), [grasp\(\)](#), [grasp_fci\(\)](#), [gs\(\)](#), [iamb-family](#), [pc\(\)](#), [rfci\(\)](#), [sp_fci\(\)](#), [tges\(\)](#), [tpc\(\)](#)

Examples

```
data(tpc_example)

kn <- knowledge(
  tpc_example,
  tier(
    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    oldage ~ tidyselect::starts_with("oldage")
  )
)

# Recommended path using disco()
my_tfci <- tfci(engine = "causalDisco", test = "fisher_z", alpha = 0.05)

disco(tpc_example, my_tfci, knowledge = kn)

# or using my_tfci directly
my_tfci <- my_tfci |> set_knowledge(kn)
my_tfci(tpc_example)

# Also possible: using tfci_run()
tfci_run(tpc_example, test = cor_test, knowledge = kn)
```

 tfci_run

Run the TFCI Algorithm for Causal Discovery

Description

Use a modification of the FCI algorithm that makes use of background knowledge in the format of a partial ordering. This may, for instance, come about when variables can be assigned to distinct tiers or periods (i.e., a temporal ordering).

Usage

```
tfci_run(
  data = NULL,
  knowledge = NULL,
  alpha = 0.05,
  test = reg_test,
  suff_stat = NULL,
  method = "stable.fast",
  na_method = "none",
  orientation_method = "conservative",
  directed_as_undirected = lifecycle::deprecated(),
  varnames = NULL,
  num_cores = 1,
  ...
)
```

Arguments

data	A data frame with the observed variables.
knowledge	A Knowledge object describing tiers/periods and optional forbidden/required edges.
alpha	The alpha level used as the per-test significance threshold for conditional independence testing.
test	A conditional independence test. The default <code>reg_test()</code> uses a regression-based information-loss test. Another available option is <code>cor_test()</code> which tests for vanishing partial correlations. User-supplied functions may also be used; see details for the required interface.
suff_stat	A sufficient statistic. If supplied, it is passed directly to the test and no statistics are computed from data. Its structure depends on the chosen test.
method	Skeleton construction method, one of "stable", "original", or "stable.fast" (default). See <code>pcalg::skeleton()</code> for details.
na_method	Handling of missing values, one of "none" (default; error on any NA), "cc" (complete-case analysis), or "tw" (test-wise deletion).
orientation_method	Method for handling conflicting separating sets when orienting edges; must be one of "standard", "conservative" (the default) or "maj.rule". See <code>pcalg::pc()</code> for further details.
directed_as_undirected	[Deprecated] This argument no longer has any effect and will be removed in a future release. Directed forbidden edges are now resolved using tier information, or must be specified in both directions.
varnames	Character vector of variable names. Only needed when data is not supplied and all information is passed via <code>suff_stat</code> .
num_cores	Integer number of CPU cores to use for parallel skeleton learning.
...	Additional arguments passed to <code>pcalg::skeleton()</code> during skeleton construction.

Details

The temporal/tiered background information enters several places in the TFCI algorithm: (1) In the skeleton construction phase, when looking for separating sets Z between two variables X and Y , Z is not allowed to contain variables that are strictly after both X and Y in the temporal order (as specified by the knowledge tiers). (2) This also applies to the subsequent phase where the algorithm searches for possible D-SEP sets. (3) Prior to other orientation steps, any cross-tier edges get an arrowhead placed at their latest node.

After this, the usual FCI orientation rules are applied; see `pcalg::udag2pag()` for details.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using `disco()`. This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See `knowledge()` for how to construct it.
- `caugi` A `caugi::caugi` object representing the learned causal graph. This graph is a PAG (Partial Ancestral Graph), but since PAGs are not yet natively supported in `caugi`, it is currently stored with class UNKNOWN.

Examples

```
data(tpc_example)

kn <- knowledge(
  tpc_example,
  tier(
    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    oldage ~ tidyselect::starts_with("oldage")
  )
)

# Recommended path using disco()
my_tfci <- tfci(engine = "causalDisco", test = "fisher_z", alpha = 0.05)

disco(tpc_example, my_tfci, knowledge = kn)

# or using my_tfci directly
my_tfci <- my_tfci |> set_knowledge(kn)
my_tfci(tpc_example)

# Also possible: using tfci_run()
tfci_run(tpc_example, test = cor_test, knowledge = kn)
```

tges

TGES Algorithm for Causal Discovery

Description

Run the Temporal Greedy Equivalent Search algorithm for causal discovery using one of several engines.

Usage

```
tges(engine = c("causalDisco"), score, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "causalDisco" causalDisco library.
score	Character; name of the scoring function to use.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported scores, see [CausalDiscoSearch](#). For additional parameters passed via ..., see [tges_run\(\)](#).

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Larsen TE, Ekstrøm CT, and Petersen AH. Score-Based Causal Discovery with Temporal Background Information, 2025. [doi:10.48550/arXiv.2502.06232](#).

See Also

Other causal discovery algorithms: `boss()`, `boss_fci()`, `fci()`, `ges()`, `gfci()`, `grasp()`, `grasp_fci()`, `gs()`, `iamb-family`, `pc()`, `rfci()`, `sp_fci()`, `tfci()`, `tpc()`

Examples

```
# Recommended route using disco:
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)

my_tges <- tges(engine = "causalDisco", score = "tbic")

disco(tpc_example, my_tges, knowledge = kn)

# another way to run it

my_tges <- my_tges |>
  set_knowledge(kn)
my_tges(tpc_example)

# or you can run directly with tges_run()

data(tpc_example)

score_bic <- new(
  "TemporalBIC",
  data = tpc_example,
  nodes = colnames(tpc_example),
  knowledge = kn
)

res_bic <- tges_run(score_bic)
res_bic
```

tges_run

Run the TGES Algorithm for Causal Discovery

Description

Perform causal discovery using the temporal greedy equivalence search algorithm.

Usage

```
tges_run(score, verbose = FALSE)
```

Arguments

score	tiered scoring object to be used. At the moment only scores supported are • TemporalBIC and • TemporalBDeu.
verbose	indicates whether debug output should be printed.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument data (a data frame). When called, this function returns a list containing:

- knowledge A Knowledge object with the background knowledge used in the causal discovery algorithm. See [knowledge\(\)](#) for how to construct it.
- caugi A [caugi::caugi](#) object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

Author(s)

Tobias Ellegaard Larsen

Examples

```
# Recommended route using disco:
kn <- knowledge(
  tpc_example,
  tier(
    child ~ starts_with("child"),
    youth ~ starts_with("youth"),
    old ~ starts_with("old")
  )
)

my_tges <- tges(engine = "causalDisco", score = "tbic")

disco(tpc_example, my_tges, knowledge = kn)

# another way to run it

my_tges <- my_tges |>
  set_knowledge(kn)
my_tges(tpc_example)
```

```
# or you can run directly with tges_run()

data(tpc_example)

score_bic <- new(
  "TemporalBIC",
  data = tpc_example,
  nodes = colnames(tpc_example),
  knowledge = kn
)

res_bic <- tges_run(score_bic)
res_bic
```

tpc

TPC Algorithm for Causal Discovery

Description

Run the Temporal Peter-Clark algorithm for causal discovery using one of several engines.

Usage

```
tpc(engine = c("causalDisco"), test, alpha = 0.05, ...)
```

Arguments

engine	Character; which engine to use. Must be one of: "causalDisco" causalDisco library.
test	Character; name of the conditional-independence test.
alpha	Numeric; significance level for the CI tests.
...	Additional arguments passed to the chosen engine (e.g. test or algorithm parameters).

Details

For specific details on the supported tests, see [CausalDiscoSearch](#). For additional parameters passed via ..., see [tpc_run\(\)](#).

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using [disco\(\)](#). This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See `knowledge()` for how to construct it.
- `caugi` A `caugi::caugi` object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

References

Petersen AH, Osler M, and Ekstrøm CT. Data-Driven Model Building for Life-Course Epidemiology. American Journal of Epidemiology 2021 Mar; 190:1898–907, doi:10.1093/aje/kwab087.

See Also

Other causal discovery algorithms: `boss()`, `boss_fci()`, `fci()`, `ges()`, `gfci()`, `grasp()`, `grasp_fci()`, `gs()`, `iamb-family`, `pc()`, `rfci()`, `sp_fci()`, `tfci()`, `tges()`

Examples

```
# Load data
data(tpc_example)

# Build knowledge
kn <- knowledge(
  tpc_example,
  tier(
    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    old ~ tidyselect::starts_with("old")
  )
)

# Recommended route using disco
my_tpc <- tpc(engine = "causalDisco", test = "fisher_z", alpha = 0.05)

disco(tpc_example, my_tpc, knowledge = kn)

# or using my_tpc directly

my_tpc <- my_tpc |> set_knowledge(kn)
my_tpc(tpc_example)

# Using tpc_run() directly

tpc_run(tpc_example, knowledge = kn, alpha = 0.01)
```

tpc_example

*Simulated Life-Course Data***Description**

A small simulated data example intended to showcase the TPC algorithm. Note that the variable name prefixes defines which period they are related to ("child", "youth" or "oldage").

Usage

```
tpc_example
```

Format

A data.frame with 200 rows and 6 variables.

child_x1 Structural equation: $X_1 := \epsilon_1$ with $\epsilon_1 \sim \text{Unif}\{0, 1\}$

child_x2 Structural equation: $X_2 := 2 \cdot X_1 + \epsilon_2$ with $\epsilon_2 \sim N(0, 1)$

youth_x3 Structural equation: $X_3 := \epsilon_3$ with $\epsilon_3 \sim \text{Unif}\{0, 1\}$

youth_x4 Structural equation: $X_4 := X_2 + \epsilon_4$ with $\epsilon_4 \sim N(0, 1)$

oldage_x5 Structural equation: $X_5 := X_3^2 + X_3 - 3 \cdot X_2 + \epsilon_5$ with $\epsilon_5 \sim N(0, 1)$

oldage_x6 Structural equation: $X_6 := X_4^3 + X_4^2 + 2 \cdot X_5 + \epsilon_6$ with $\epsilon_6 \sim N(0, 1)$

References

Petersen, AH; Osler, M and Ekstrøm, CT (2021): Data-Driven Model Building for Life-Course Epidemiology, American Journal of Epidemiology.

Examples

```
data(tpc_example)
head(tpc_example)
```

tpc_run

*Run the TPC Algorithm for Causal Discovery***Description**

Run a tier-aware variant of the PC algorithm that respects background knowledge about a partial temporal order. Supply the temporal order via a Knowledge object.

Usage

```

tpc_run(
  data = NULL,
  knowledge = NULL,
  alpha = 0.05,
  test = reg_test,
  suff_stat = NULL,
  method = "stable.fast",
  na_method = "none",
  orientation_method = "conservative",
  directed_as_undirected = lifecycle::deprecated(),
  varnames = NULL,
  num_cores = 1,
  ...
)

```

Arguments

data	A data frame with the observed variables.
knowledge	A Knowledge object created with <code>knowledge()</code> , encoding tier assignments and optional forbidden/required edges. This is the preferred way to provide temporal background knowledge.
alpha	The alpha level used as the per-test significance threshold for conditional independence testing.
test	A conditional independence test. The default <code>reg_test()</code> uses a regression-based information-loss test. Another available option is <code>cor_test()</code> which tests for vanishing partial correlations. User-supplied functions may also be used; see details for the required interface.
suff_stat	A sufficient statistic. If supplied, it is passed directly to the test and no statistics are computed from data. Its structure depends on the chosen test.
method	Skeleton construction method, one of "stable", "original", or "stable.fast" (default). See <code>pcalg::skeleton()</code> for details.
na_method	Handling of missing values, one of "none" (default; error on any NA), "cc" (complete-case analysis), or "tw" (test-wise deletion).
orientation_method	Conflict-handling method when orienting edges. Currently only the conservative method is available.
directed_as_undirected	[Deprecated] This argument no longer has any effect and will be removed in a future release. Directed forbidden edges are now resolved using tier information, or must be specified in both directions.
varnames	Character vector of variable names. Only needed when data is not supplied and all information is passed via <code>suff_stat</code> .
num_cores	Integer number of CPU cores to use for parallel skeleton learning.
...	Additional arguments passed to <code>pcalg::skeleton()</code> during skeleton construction.

Details

Any independence test implemented in **pcalg** may be used; see `pcalg::pc()`. When `na_method = "twd"`, test-wise deletion is performed: for `cor_test()`, each pairwise correlation uses complete cases; for `reg_test()`, each conditional test performs its own deletion. If you supply a user-defined test, you must also provide `suff_stat`.

Temporal or tiered knowledge enters in two places:

- during skeleton estimation, candidate conditioning sets are pruned so they do not contain variables that are strictly after both endpoints;
- during orientation, any cross-tier edge is restricted to point forward in time.

Recommendation

While it is possible to call the function returned directly with a data frame, we recommend using `disco()`. This provides a consistent interface and handles knowledge integration.

Value

A function that takes a single argument `data` (a data frame). When called, this function returns a list containing:

- `knowledge` A Knowledge object with the background knowledge used in the causal discovery algorithm. See `knowledge()` for how to construct it.
- `caugi` A `caugi::caugi` object (of class PDAG) representing the learned causal graph from the causal discovery algorithm.

Examples

```
# Load data
data(tpc_example)

# Build knowledge
kn <- knowledge(
  tpc_example,
  tier(
    child ~ tidyselect::starts_with("child"),
    youth ~ tidyselect::starts_with("youth"),
    old ~ tidyselect::starts_with("old")
  )
)

# Recommended route using disco
my_tpc <- tpc(engine = "causalDisco", test = "fisher_z", alpha = 0.05)

disco(tpc_example, my_tpc, knowledge = kn)

# or using my_tpc directly
my_tpc <- my_tpc |> set_knowledge(kn)
my_tpc(tpc_example)
```

```
# Using tpc_run() directly

tpc_run(tpc_example, knowledge = kn, alpha = 0.01)
```

unfreeze

Unfreeze a Knowledge Object.

Description

This allows you to add new variables to the Knowledge object, even though it was frozen earlier by adding a data frame to the knowledge constructor `knowledge()`.

Usage

```
unfreeze(kn)
```

Arguments

`kn` A Knowledge object.

Value

The same Knowledge object with the `frozen` attribute set to `FALSE`.

See Also

Other knowledge helpers: `+.Knowledge()`, `add_exogenous()`, `add_tier()`, `add_to_tier()`, `add_vars()`, `as_bnlearn_knowledge()`, `as_pcalg_constraints()`, `as_tetrad_knowledge()`, `convert_tiers_to_forbidden()`, `deparse_knowledge()`, `forbid_edge()`, `get_tiers()`, `knowledge()`, `knowledge_to_caugi()`, `remove_edge()`, `remove_tiers()`, `remove_vars()`, `reorder_tiers()`, `reposition_tier()`, `require_edge()`, `seq_tiers()`, `set_knowledge()`

Examples

```
# unfreeze allows adding variables beyond the original data frame columns
data(tpc_example)

kn <- knowledge(tpc_example)

# this would error while frozen
try(add_vars(kn, "new_var"))

# unfreeze and add the new variable successfully
kn <- unfreeze(kn)
kn <- add_vars(kn, "new_var")

print(kn)
```

verify_tetrad	<i>Check Tetrad Installation</i>
---------------	----------------------------------

Description

Check Tetrad Installation

Usage

```
verify_tetrad(version = getOption("causalDisco.tetrad.version"))
```

Arguments

version Character. The version of Tetrad to check. Default is the value of `getOption("causalDisco.tetrad.version")`.

Value

A list with elements:

- installed: Logical, whether Tetrad is installed.
- version: Character or NULL, the installed version if found.
- java_ok: Logical, whether Java $\geq$ 21.
- java_version: Character, the installed Java version.
- message: Character, a message describing the status.

Examples

```
verify_tetrad()
```

Index

- * **Extending causalDisco**
 - as_disco, [14](#)
 - distribute_engine_args, [40](#)
 - list_registered_engines, [71](#)
 - list_registered_tetrad_algorithms, [71](#)
 - make_method, [72](#)
 - make_runner, [73](#)
 - new_disco_method, [77](#)
 - register_engine, [100](#)
 - register_tetrad_algorithm, [101](#)
 - reset_engine_registry, [108](#)
 - reset_tetrad_alg_registry, [109](#)
- * **causal discovery algorithms**
 - boss, [22](#)
 - boss_fci, [24](#)
 - fci, [44](#)
 - ges, [50](#)
 - gfci, [53](#)
 - grasp, [55](#)
 - grasp_fci, [57](#)
 - gs, [60](#)
 - iamb-family, [61](#)
 - pc, [82](#)
 - rfci, [109](#)
 - sp_fci, [115](#)
 - tfci, [134](#)
 - tges, [138](#)
 - tpc, [141](#)
- * **cd_algorithms**
 - boss, [22](#)
 - boss_fci, [24](#)
 - fci, [44](#)
 - ges, [50](#)
 - gfci, [53](#)
 - grasp, [55](#)
 - grasp_fci, [57](#)
 - gs, [60](#)
 - iamb-family, [61](#)
 - pc, [82](#)
 - rfci, [109](#)
 - sp_fci, [115](#)
 - tfci, [134](#)
 - tges, [138](#)
 - tpc, [141](#)
- * **datasets**
 - cat_data, [26](#)
 - cat_data_mcar, [27](#)
 - cat_ord_data, [28](#)
 - mix_data, [76](#)
 - nlsy97, [78](#)
 - num_data, [80](#)
 - num_data_latent, [81](#)
 - tpc_example, [143](#)
- * **dataset**
 - cat_data, [26](#)
 - cat_data_mcar, [27](#)
 - cat_ord_data, [28](#)
 - mix_data, [76](#)
 - nlsy97, [78](#)
 - num_data, [80](#)
 - num_data_latent, [81](#)
 - tpc_example, [143](#)
- * **extending_causalDisco**
 - as_disco, [14](#)
 - distribute_engine_args, [40](#)
 - list_registered_engines, [71](#)
 - list_registered_tetrad_algorithms, [71](#)
 - make_method, [72](#)
 - make_runner, [73](#)
 - new_disco_method, [77](#)
 - register_engine, [100](#)
 - register_tetrad_algorithm, [101](#)
 - reset_engine_registry, [108](#)
 - reset_tetrad_alg_registry, [109](#)
- * **knowledge helpers**
 - +.Knowledge, [5](#)

- add_exogenous, 6
- add_tier, 8
- add_to_tier, 9
- add_vars, 11
- as_bnlearn_knowledge, 12
- as_pcalg_constraints, 14
- as_tetrad_knowledge, 16
- convert_tiers_to_forbidden, 36
- deparse_knowledge, 37
- forbid_edge, 47
- get_tiers, 52
- knowledge, 65
- knowledge_to_caugi, 70
- remove_edge, 101
- remove_tiers, 102
- remove_vars, 103
- reorder_tiers, 105
- reposition_tier, 106
- require_edge, 107
- seq_tiers, 111
- set_knowledge, 112
- unfreeze, 146
- * knowledge-helper**
 - +.Knowledge, 5
 - add_exogenous, 6
 - add_tier, 8
 - add_to_tier, 9
 - add_vars, 11
 - as_bnlearn_knowledge, 12
 - as_pcalg_constraints, 14
 - as_tetrad_knowledge, 16
 - convert_tiers_to_forbidden, 36
 - deparse_knowledge, 37
 - forbid_edge, 47
 - get_tiers, 52
 - knowledge, 65
 - knowledge_to_caugi, 70
 - remove_edge, 101
 - remove_tiers, 102
 - remove_vars, 103
 - reorder_tiers, 105
 - reposition_tier, 106
 - require_edge, 107
 - seq_tiers, 111
 - set_knowledge, 112
 - unfreeze, 146
- * metrics**
 - confusion, 34
 - evaluate, 41
 - f1_score, 42
 - false_omission_rate, 43
 - fdr, 46
 - g1_score, 48
 - npv, 79
 - precision, 95
 - recall, 98
 - specificity, 117
- * plot**
 - plot, 89
 - plot.Disco, 90
 - plot.Knowledge, 92
- * print**
 - print.Disco, 96
 - print.Knowledge, 97
- +.Knowledge, 5
- +.Knowledge(), 7, 8, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- add_exo (add_exogenous), 6
- add_exogenous, 6
- add_exogenous(), 5, 8, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- add_tier, 8
- add_tier(), 5, 7, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- add_to_tier, 9
- add_to_tier(), 5, 7, 8, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- add_vars, 11
- add_vars(), 5, 7, 8, 10, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- as_bnlearn_knowledge, 12
- as_bnlearn_knowledge(), 5, 7, 8, 10, 11, 15, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- as_disco, 14
- as_disco(), 41, 71–73, 78, 100, 101, 108, 109
- as_pcalg_constraints, 14
- as_pcalg_constraints(), 5, 7, 8, 10, 11, 13, 17, 36, 38, 47, 52, 66, 70, 102–107, 112, 113, 146
- as_tetrad_knowledge, 16

- `as_tetrad_knowledge()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#),
[15](#), [36](#), [38](#), [47](#), [52](#), [66](#), [70](#), [102–107](#),
[112](#), [113](#), [146](#)
- `bnlearn::ci.test()`, [20](#)
- `bnlearn::fast.iamb()`, [19](#)
- `bnlearn::gs()`, [19](#)
- `bnlearn::iamb()`, [19](#)
- `bnlearn::iamb.fdr()`, [19](#)
- `bnlearn::inter.iamb()`, [19](#)
- `bnlearn::pc.stable()`, [19](#)
- `BnlearnSearch`, [17](#), [39](#), [60](#), [62](#), [82](#), [100](#)
- `boss`, [22](#)
- `boss()`, [25](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#), [63](#), [83](#),
[110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `boss_fci`, [24](#)
- `boss_fci()`, [23](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#), [63](#),
[83](#), [110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `cat_data`, [26](#), [28](#), [77](#)
- `cat_data_mcar`, [27](#)
- `cat_ord_data`, [28](#)
- `caugi::caugi`, [14](#), [23](#), [25](#), [34](#), [35](#), [39](#), [41–44](#),
[46](#), [49](#), [50](#), [53](#), [56](#), [58](#), [60](#), [62](#), [70](#),
[77–79](#), [82](#), [93](#), [95](#), [98](#), [110](#), [116](#), [118](#),
[135](#), [137](#), [138](#), [140](#), [142](#), [145](#)
- `caugi::caugi()`, [14](#)
- `caugi::generate_graph()`, [113](#)
- `caugi::plot()`, [74](#), [90](#), [92](#)
- `causalDisco` (causalDisco-package), [4](#)
- `causalDisco-package`, [4](#)
- `CausalDiscoSearch`, [29](#), [39](#), [100](#), [134](#), [138](#),
[141](#)
- `confusion`, [34](#)
- `confusion()`, [41–43](#), [47](#), [49](#), [79](#), [96](#), [98](#), [118](#)
- `convert_tiers_to_forbidden`, [36](#)
- `convert_tiers_to_forbidden()`, [5](#), [7](#), [8](#), [10](#),
[11](#), [13](#), [15](#), [17](#), [38](#), [47](#), [52](#), [66](#), [70](#),
[102–107](#), [112](#), [113](#), [146](#)
- `cor_test`, [37](#)
- `cor_test()`, [29](#), [136](#), [144](#), [145](#)
- `deparse_knowledge`, [37](#)
- `deparse_knowledge()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#),
[17](#), [36](#), [47](#), [52](#), [66](#), [70](#), [102–107](#), [112](#),
[113](#), [146](#)
- `disco`, [38](#)
- `disco()`, [16](#), [23](#), [25](#), [44](#), [50](#), [53](#), [55](#), [58](#), [60](#), [62](#),
[72](#), [77](#), [82](#), [110](#), [116](#), [134](#), [137](#), [138](#),
[140](#), [141](#), [145](#)
- `distribute_engine_args`, [40](#)
- `distribute_engine_args()`, [14](#), [71–73](#), [78](#),
[100](#), [101](#), [108](#), [109](#)
- `evaluate`, [41](#)
- `evaluate()`, [35](#), [42](#), [43](#), [47](#), [49](#), [79](#), [96](#), [98](#), [118](#)
- `f1_score`, [42](#)
- `f1_score()`, [35](#), [41](#), [43](#), [47](#), [49](#), [79](#), [96](#), [98](#), [118](#)
- `false_omission_rate`, [43](#)
- `false_omission_rate()`, [35](#), [41](#), [42](#), [47](#), [49](#),
[79](#), [96](#), [98](#), [118](#)
- `fast_iamb` (iamb-family), [61](#)
- `fast_iamb()`, [19](#), [39](#)
- `fci`, [44](#)
- `fci()`, [23](#), [25](#), [39](#), [51](#), [54](#), [56](#), [58](#), [61](#), [63](#), [83](#),
[85](#), [110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `fdr`, [46](#)
- `fdr()`, [35](#), [41–43](#), [49](#), [79](#), [96](#), [98](#), [118](#)
- `forbid_edge`, [47](#)
- `forbid_edge()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [36](#),
[38](#), [52](#), [66](#), [70](#), [102–107](#), [112](#), [113](#),
[146](#)
- `g1_score`, [48](#)
- `g1_score()`, [35](#), [41–43](#), [47](#), [79](#), [96](#), [98](#), [118](#)
- `ges`, [50](#)
- `ges()`, [23](#), [25](#), [39](#), [45](#), [54](#), [56](#), [58](#), [61](#), [63](#), [83](#),
[85](#), [110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `get_tiers`, [52](#)
- `get_tiers()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [36](#),
[38](#), [47](#), [66](#), [70](#), [102–107](#), [112](#), [113](#),
[146](#)
- `gfci`, [53](#)
- `gfci()`, [23](#), [25](#), [39](#), [45](#), [51](#), [56](#), [58](#), [61](#), [63](#), [83](#),
[110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `grasp`, [55](#)
- `grasp()`, [23](#), [25](#), [39](#), [45](#), [51](#), [54](#), [58](#), [61](#), [63](#), [83](#),
[110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `grasp_fci`, [57](#)
- `grasp_fci()`, [23](#), [25](#), [39](#), [45](#), [51](#), [54](#), [56](#), [61](#),
[63](#), [83](#), [110](#), [116](#), [121](#), [135](#), [139](#), [142](#)
- `gs`, [60](#)
- `gs()`, [19](#), [23](#), [25](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [63](#), [83](#),
[110](#), [116](#), [135](#), [139](#), [142](#)
- `iamb` (iamb-family), [61](#)
- `iamb()`, [19](#), [39](#)
- `iamb-family`, [61](#)

- `iamb_fdr(iamb-family)`, 61
- `iamb_fdr()`, 19, 39
- `install_tetrad`, 64
- `install_tetrad()`, 4
- `inter_iamb(iamb-family)`, 61
- `inter_iamb()`, 19, 39
- `knowledge`, 65
- `knowledge()`, 5, 7, 8, 10, 11, 13, 15, 17, 23, 25, 33, 36, 38, 39, 44, 47, 50, 52, 53, 56, 58, 60, 62, 70, 82, 87, 92, 102–107, 110, 112, 113, 116, 135, 137, 138, 140, 142, 144–146
- `knowledge_to_caugi`, 70
- `knowledge_to_caugi()`, 5, 7, 8, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 102–107, 112, 113, 146
- `list_registered_engines`, 71
- `list_registered_engines()`, 14, 41, 71–73, 78, 100, 101, 108, 109
- `list_registered_tetrad_algorithms`, 71
- `list_registered_tetrad_algorithms()`, 14, 41, 71–73, 78, 100, 101, 108, 109
- `make_method`, 72
- `make_method()`, 14, 41, 71, 73, 78, 100, 101, 108, 109
- `make_runner`, 73
- `make_runner()`, 14, 41, 71, 72, 78, 100, 101, 108, 109
- `make_tikz`, 73
- `micd::disCItd()`, 29, 85
- `micd::disMItest()`, 29, 85
- `micd::gaussCItestMI()`, 29, 85
- `micd::gaussCItd()`, 29, 85
- `micd::mixCItest()`, 30, 85
- `micd::mixCItd()`, 30, 85
- `micd::mixMItest()`, 30, 85
- `mix_data`, 76
- `new_disco_method`, 77
- `new_disco_method()`, 14, 41, 71–73, 100, 101, 108, 109
- `nlsy97`, 78
- `npv`, 79
- `npv()`, 35, 41–43, 47, 49, 96, 98, 118
- `num_data`, 27, 29, 77, 80, 81
- `num_data_latent`, 81
- `pc`, 82
- `pc()`, 19, 23, 25, 39, 45, 51, 54, 56, 58, 61, 63, 85, 110, 116, 121, 135, 139, 142
- `pcalg::binCItest()`, 29, 85
- `pcalg::disCItest()`, 29, 85
- `pcalg::fci()`, 85
- `pcalg::gaussCItest()`, 37, 85
- `pcalg::GaussL0penIntScore`, 85
- `pcalg::GaussL0penObsScore`, 85
- `pcalg::ges()`, 85
- `pcalg::pc()`, 85, 129, 136, 145
- `pcalg::rfci()`, 85
- `pcalg::skeleton()`, 136, 144
- `pcalg::udag2pag()`, 137
- `PcalgSearch`, 39, 44, 50, 82, 84, 100, 109
- `plot`, 89, 90, 93
- `plot()`, 74
- `plot.Disco`, 90
- `plot.Disco()`, 89, 93
- `plot.Knowledge`, 92
- `plot.Knowledge()`, 89, 90
- `precision`, 95
- `precision()`, 35, 41–43, 47, 49, 79, 98, 118
- `print.Disco`, 96
- `print.Disco()`, 97
- `print.Knowledge`, 97
- `print.Knowledge()`, 96
- `recall`, 98
- `recall()`, 35, 41–43, 47, 49, 79, 96, 118
- `reexports`, 35, 41–43, 47, 49, 79, 96, 98, 118
- `reg_test`, 99
- `reg_test()`, 29, 85, 136, 144, 145
- `register_engine`, 100
- `register_engine()`, 14, 41, 71–73, 78, 101, 108, 109
- `register_tetrad_algorithm`, 101
- `register_tetrad_algorithm()`, 14, 41, 71–73, 78, 100, 108, 109
- `remove_edge`, 101
- `remove_edge()`, 5, 7, 8, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 103–107, 112, 113, 146
- `remove_tiers`, 102
- `remove_tiers()`, 5, 7, 8, 10, 11, 13, 15, 17, 36, 38, 47, 52, 66, 70, 102, 104–107, 112, 113, 146
- `remove_vars`, 103

- `remove_vars()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [36](#),
[38](#), [47](#), [52](#), [66](#), [70](#), [102](#), [103](#),
[105–107](#), [112](#), [113](#), [146](#)
- `reorder_tiers`, [105](#)
- `reorder_tiers()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#),
[36](#), [38](#), [47](#), [52](#), [66](#), [70](#), [102–104](#), [106](#),
[107](#), [112](#), [113](#), [146](#)
- `reposition_tier`, [106](#)
- `reposition_tier()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#),
[17](#), [36](#), [38](#), [47](#), [52](#), [66](#), [70](#), [102–105](#),
[107](#), [112](#), [113](#), [146](#)
- `require_edge`, [107](#)
- `require_edge()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#),
[36](#), [38](#), [47](#), [52](#), [66](#), [70](#), [102–106](#), [112](#),
[113](#), [146](#)
- `reset_engine_registry`, [108](#)
- `reset_engine_registry()`, [14](#), [41](#), [71–73](#),
[78](#), [100](#), [101](#), [109](#)
- `reset_tetrad_alg_registry`, [109](#)
- `reset_tetrad_alg_registry()`, [14](#), [41](#),
[71–73](#), [78](#), [100](#), [101](#), [108](#)
- `rfci`, [109](#)
- `rfci()`, [23](#), [25](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#), [63](#), [83](#),
[85](#), [116](#), [121](#), [135](#), [139](#), [142](#)
-
- `seq_tiers`, [111](#)
- `seq_tiers()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [36](#),
[38](#), [47](#), [52](#), [66](#), [70](#), [102–107](#), [113](#), [146](#)
- `set_knowledge`, [112](#)
- `set_knowledge()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15–17](#),
[36](#), [38](#), [47](#), [52](#), [66](#), [70](#), [102–107](#), [112](#),
[146](#)
- `sim_dag`, [113](#)
- `sim_data`, [114](#)
- `sp_fci`, [115](#)
- `sp_fci()`, [23](#), [25](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#),
[63](#), [83](#), [110](#), [121](#), [135](#), [139](#), [142](#)
- `specificity`, [117](#)
- `specificity()`, [35](#), [41–43](#), [47](#), [49](#), [79](#), [96](#), [98](#)
- `summary.Disco`, [118](#)
- `summary.Knowledge`, [119](#)
-
- `TemporalBDeu`, [29](#), [140](#)
- `TemporalBIC`, [29](#), [140](#)
- `TetradSearch`, [23](#), [25](#), [39](#), [44](#), [50](#), [53](#), [55](#), [58](#),
[82](#), [100](#), [109](#), [116](#), [119](#)
- `tfci`, [134](#)
- `tfci()`, [23](#), [25](#), [30](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#),
[63](#), [83](#), [110](#), [116](#), [139](#), [142](#)
-
- `tfci_run`, [135](#)
- `tfci_run()`, [32](#), [134](#)
- `tges`, [138](#)
- `tges()`, [23](#), [25](#), [30](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#),
[63](#), [83](#), [110](#), [116](#), [135](#), [142](#)
- `tges_run`, [139](#)
- `tges_run()`, [32](#), [138](#)
- `tidyselect::starts_with()`, [66](#)
- `tpc`, [141](#)
- `tpc()`, [23](#), [25](#), [30](#), [39](#), [45](#), [51](#), [54](#), [56](#), [58](#), [61](#),
[63](#), [83](#), [110](#), [116](#), [135](#), [139](#)
- `tpc_example`, [143](#)
- `tpc_run`, [143](#)
- `tpc_run()`, [32](#), [141](#)
-
- `unfreeze`, [146](#)
- `unfreeze()`, [5](#), [7](#), [8](#), [10](#), [11](#), [13](#), [15](#), [17](#), [36](#), [38](#),
[47](#), [52](#), [66](#), [70](#), [102–107](#), [112](#), [113](#)
-
- `verify_tetrad`, [147](#)