Package ‘dce’

February 25, 2024

**Type** Package

**Title** Pathway Enrichment Based on Differential Causal Effects

**Version** 1.10.0

**Description** Compute differential causal effects (dce) on (biological) networks.

Given observational samples from a control experiment and non-control (e.g., cancer) for two genes A and B, we can compute differential causal effects with a (generalized) linear regression.

If the causal effect of gene A on gene B in the control samples is different from the causal effect in the non-control samples the dce will differ from zero.

We regularize the dce computation by the inclusion of prior network information from pathway databases such as KEGG.

**URL** [https://github.com/cbg-ethz/dce](https://github.com/cbg-ethz/dce)

**BugReports** [https://github.com/cbg-ethz/dce/issues](https://github.com/cbg-ethz/dce/issues)

**biocViews** Software, StatisticalMethod, GraphAndNetwork, Regression, GeneExpression, DifferentialExpression, NetworkEnrichment, Network, KEGG

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 4.1)

**Suggests** knitr, rmarkdown, testthat (>= 2.1.0), BiocStyle, formatR, cowplot, ggplotify, dagitty, lmttest, sandwich, devtools, curatedTCGAData, TCGAutils, SummarizedExperiment, RcppParallel, docopt, CARNIVAL

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2

**Imports** stats, methods, assertthat, graph, pcalg, purrr, tidyverse, Matrix, ggraph, tidygraph, ggplot2, rlang, expm, MASS, edgeR, epiNEM, igraph, metap, mnem, naturalsort, ppcor, glm2, graphite, reshape2, dplyr, magrittr, glue, Rgraphviz, harmonicmeanp, org.Hs.eg.db, logger, shadowtext
git_url  https://git.bioconductor.org/packages/dce

R topics documented:

as.data.frame.dce ............................................. 3
as_adjmat .................................................. 3
create_random_DAG .......................................... 4
dce ........................................................... 5
dce_nb .......................................................... 7
df_pathway_statistics ........................................ 8
estimate_latent_count ....................................... 9
g2dag .......................................................... 10
get_pathways ................................................ 10
get_pathway_info ........................................... 11
get_prediction_counts ....................................... 12
graph2df ..................................................... 13
graph_union .................................................. 13
pcor ............................................................. 14
permutation_test ............................................. 14
plot.dce ...................................................... 15
plot_network ................................................ 16
propagate_gene_edges ....................................... 17
resample_edge_weights .................................... 18
rlm_dce ....................................................... 19
simulate_data ............................................... 19
summary.rlm_dce ............................................ 21
topologically_ordering .................................... 22
trueEffects ................................................... 22

Index 24
## as.data.frame.dce

### Description

Turn dce object into data frame

### Usage

```r
## S3 method for class 'dce'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

### Arguments

- `x`: dce object
- `row.names`: optional character vector of rownames
- `optional`: logical; allow optional arguments
- `...`: additional arguments

### Value

Data frame containing the dce output

### Examples

```r
dag <- create_random_DAG(30, 0.2)
X_wt <- simulate_data(dag)
dag_mt <- resample_edge_weights(dag)
X_mt <- simulate_data(dag_mt)
dce_list <- dce(dag, X_wt, X_mt)
```

---

## as_adjmat

### Description

From graphNEL with 0 edge weights to proper adjacency matrix

### Usage

```r
as_adjmat(g)
```

### Arguments

- `g`: graphNEL object
Value

graph as adjacency matrix

Examples

dag <- create_random_DAG(30, 0.2)
adj <- as_adjmat(dag)

create_random_DAG  Create random DAG (topologically ordered)

Description

Creates a DAG according to given parameters.

Usage

create_random_DAG(
  node_num,
  prob,
  eff_min = -1,
  eff_max = 1,
  node_labels = paste0("n", as.character(seq_len(node_num))),
  max_par = 3
)

Arguments

node_num  Number of nodes
prob  Probability of creating an edge
eff_min  Lower bound for edge weights
eff_max  Upper bound for edge weights
node_labels  Node labels
max_par  Maximal number of parents

Value

graph

Author(s)

Martin Pirkl

Examples

dag <- create_random_DAG(30, 0.2)
dce  Differential Causal Effects - main function

Description

Main function to compute differential causal effects and the pathway enrichment

Usage

dce(
  graph,
  df_expr_wt,
  df_expr_mt,
  solver = "lm",
  solver_args = list(),
  adjustment_type = "parents",
  effect_type = "total",
  p_method = "hmp",
  test = "wald",
  lib_size = FALSE,
  deconfounding = FALSE,
  conservative = FALSE,
  log_level = logger::INFO
)

## S4 method for signature 'igraph'
dce(
  graph,
  df_expr_wt,
  df_expr_mt,
  solver = "lm",
  solver_args = list(),
  adjustment_type = "parents",
  effect_type = "total",
  p_method = "hmp",
  test = "wald",
  lib_size = FALSE,
  deconfounding = FALSE,
  conservative = FALSE,
  log_level = logger::INFO
)

## S4 method for signature 'graphNEL'
dce(
  graph,
  df_expr_wt,
  df_expr_mt,
solver = "lm",
solver_args = list(),
adjustment_type = "parents",
effect_type = "total",
p_method = "hmp",
test = "wald",
lib_size = FALSE,
deficiency = FALSE,
conservative = FALSE,
log_level = logger::INFO
)

## S4 method for signature 'matrix'
dce(
  graph,
  df_expr_wt,
  df_expr_mt,
  solver = "lm",
  solver_args = list(),
  adjustment_type = "parents",
  effect_type = "total",
p_method = "hmp",
test = "wald",
lib_size = FALSE,
deficiency = FALSE,
conservative = FALSE,
log_level = logger::INFO
)

Arguments

**graph** valid object defining a directed acyclic graph

**df_expr_wt** data frame with wild type expression values

**df_expr_mt** data from with mutation type expression values

**solver** character with name of solver function

**solver_args** additional arguments for the solver function. please adress this argument, if you use your own solver function. the default argument works with glm functions in the packages MASS, stats and glm2

**adjustment_type** character string for the method to define the adjustment set Z for the regression

**effect_type** method of computing causal effects

**p_method** character string. "mean", "sum" for standard summary functions, "hmp" for harmonic mean or any method from package 'metap', e.g., "meant" or "sump".

**test** either "wald" for testing significance with the wald test or "lr" for using a likelihood ratio test. Alternatively, "vcovHC" can improve results for zero-inflated date, i.e., from single cell RNAseq experiments.
**lib_size**

either a numeric vector of the same length as the sum of wild type and mutant samples or a logical. If TRUE, it is recommended that both data sets include not only the genes included in the graph but all genes available in the original data set.

**deconfounding**

indicates whether adjustment against latent confounding is used. If FALSE, no adjustment is used, if TRUE it adjusts for confounding by automatically estimating the number of latent confounders. The estimated number of latent confounders can be chosen manually by setting this variable to some number.

**conservative**

logical; if TRUE, does not use the indicator variable for the variables in the adjustment set

**log_level**

Control verbosity (logger::INFO, logger::DEBUG, ...)

**Value**

list of matrices with dces and corresponding p-value

**Examples**

```r
dag <- create_random_DAG(30, 0.2)
X.wt <- simulate_data(dag)
dag.mt <- resample_edge_weights(dag)
X.mt <- simulate_data(dag)
dce(dag, X.wt, X.mt)
```

---

**dce_nb**

*Differential Causal Effects for negative binomial data*

**Description**

Shortcut for the main function to analyse negative binomial data

**Usage**

```r
dce_nb(
  graph,
  df_expr_wt,
  df_expr_mt,
  solver_args = list(method = "glm.dce.nb.fit", link = "identity"),
  adjustment_type = "parents",
  effect_type = "total",
  p_method = "hmp",
  test = "wald",
  lib_size = FALSE,
  deconfounding = FALSE,
  conservative = FALSE,
  log_level = logger::INFO
)
```
Arguments

- **graph**: valid object defining a directed acyclic graph
- **df_expr_wt**: data frame with wild type expression values
- **df_expr_mt**: data from with mutation type expression values
- **solver_args**: additional arguments for the solver function
- **adjustment_type**: character string for the method to define the adjustment set \( Z \) for the regression
- **effect_type**: method of computing causal effects
- **p_method**: character string. "mean", "sum" for standard summary functions, "hmp" for harmonic mean or any method from package 'metap', e.g., "meanp" or "sump".
- **test**: either "wald" for testing significance with the wald test or "lr" for using a likelihood ratio test
- **lib_size**: either a numeric vector of the same length as the sum of wild type and mutant samples or a logical. If TRUE, it is recommended that both data sets include not only the genes included in the graph but all genes available in the original data set.
- **deconfounding**: indicates whether adjustment against latent confounding is used. If FALSE, no adjustment is used, if TRUE it adjusts for confounding by automatically estimating the number of latent confounders. The estimated number of latent confounders can be chosen manually by setting this variable to some number.
- **conservative**: logical; if TRUE, does not use the indicator variable for the variables in the adjustment set
- **log_level**: Control verbosity (logger::INFO, logger::DEBUG, ...)

Value

- list of matrices with dces and corresponding p-value

Examples

```r
dag <- create_random_DAG(30, 0.2)
X.wt <- simulate_data(dag)
dag.mt <- resample_edge_weights(dag)
X.mt <- simulate_data(dag)
dce_nb(dag,X.wt,X.mt)
```

**df_pathway_statistics**  Biological pathway information.

Description

A dataset containing pathway statistics.
**Usage**

df_pathway_statistics

**Format**

A data frame with pathway statistics

- **database**: Pathway database
- **pathway_id**: Internal ID of pathway
- **pathway_name**: Canonical name of pathway
- **node_num**: Number of nodes in pathway
- **edge_num**: Number of edges in pathway

**estimate_latent_count**  
*Estimate number of latent confounders  Compute the true casual effects of a simulated dag*

**Description**

This function takes a DAG with edgeweights as input and computes the causal effects of all nodes on all direct and indirect children in the DAG. Alternatively see pcalg::causalEffect for pairwise computation.

**Usage**

estimate_latent_count(X1, X2, method = "auto")

**Arguments**

- **X1**: data matrix corresponding to the first condition
- **X2**: data matrix corresponding to the second condition
- **method**: a string indicating the method used for estimating the number of latent variables

**Value**

estimated number of latent variables

**Author(s)**

Domagoj Ćević

**Examples**

```r
graph1 <- create_random_DAG(node_num = 100, prob = .1)
graph2 <- resample_edge_weights(graph1, tp=0.15)
X1 <- simulate_data(graph1, n=200, latent = 3)
X2 <- simulate_data(graph2, n=200, latent = 3)
estimate_latent_count(X1, X2)
```
**g2dag**  
*Graph to DAG*

**Description**

Converts a general graph to a dag with minimum distance to the original graph. The general idea is to transitively close the graph to detect cycles and remove them based on the rule "the more outgoing edges a node has, the more likely it is that incoming edges from a cycle will be deleted, and vice versa. However, this is too rigorous and deletes too many edges, which do not lead to a cycle. These edges are added back in the final step.

**Usage**

```r
g2dag(g, tc = FALSE)
```

**Arguments**

- `g`  
  graph as adjacency matrix

- `tc`
  if TRUE computes the transitive closure

**Value**

dag as adjacency matrix

**Author(s)**

Ken Adams

**Examples**

```r
g <- matrix(c(1,0,1,0, 1,1,0,0, 0,1,1,0, 1,1,0,1), 4, 4)  
rownames(g) <- colnames(g) <- LETTERS[seq_len(4)]  
dag <- g2dag(g)
```

---

**get_pathways**  
*Easy pathway network access*

**Description**

Easy pathway network access
get_pathway_info

Usage

get_pathways(
    query_species = "hsapiens",
    database_list = NULL,
    remove_empty_pathways = TRUE,
    pathway_list = NULL
)

Arguments

query_species For which species
database_list Which databases to query. Query all if ‘NULL’. remove_empty_pathways Discard pathways without nodes pathway_list List mapping database name to vector of pathway names to download

Value

list of pathways

Examples

pathways <- get_pathways(
    pathway_list = list(kegg = c(
        "Protein processing in endoplasmic reticulum"
    ))
)
plot_network(as(pathways[[1]]$graph, "matrix"))

get_pathway_info

Dataframe containing meta-information of pathways in database

Description

Dataframe containing meta-information of pathways in database

Usage

get_pathway_info(
    query_species = "hsapiens",
    database_list = NULL,
    include_network_statistics = FALSE
)
Arguments

query_species For which species
database_list Which databases to query. Query all if ‘NULL’.
include_network_statistics Compute some useful statistics per pathway. Takes longer!

Value
data frame with pathway meta information

Examples

head(get_pathway_info(database_list = c("kegg")))

generate_counts
Compute true positive/... counts

Description
Useful for performance evaluations

Usage
get_prediction_counts(truth, inferred, cutoff = 0.5)

Arguments

truth Ground truth
inferred Computed results
cutoff Threshold for classification

Value
data.frame

Author(s)
Hans Wurst

Examples
get_prediction_counts(c(1,0), c(1,1))
**graph2df**  
*Graph to dataframe*

**Description**  
Convert graph object to dataframe with source and target columns

**Usage**  
```r  
graph2df(graph)  
```

**Arguments**  
- `graph`  
  Network

**Value**  
data frame

**Examples**  
```r  
dag <- create_random_DAG(30, 0.2)  
graph2df(dag)  
```

---

**graph_union**  
*Graph union*

**Description**  
Create union of multiple graphs

**Usage**  
```r  
graph_union(graph_list)  
```

**Arguments**  
- `graph_list`  
  List of graphs

**Value**  
graph union

**Examples**  
```r  
dag <- create_random_DAG(30, 0.2)  
dag2 <- create_random_DAG(30, 0.2)  
graph_union(list(g1=dag, g2=dag2))  
```
**pcor**

Partial correlation

**Description**

Robust partial correlation of column variables of a numeric matrix

**Usage**

```r
pcor(x, g = NULL, adjustment_type = "parents", ...)```

**Arguments**

- `x`: matrix
- `g`: related graph as adjacency matrix (optional)
- `adjustment_type`: character string for the method to define the adjustment set Z for the regression
- `...`: additional arguments for function 'cor'

**Value**

matrix of partial correlations

**Examples**

```r
x <- matrix(rnorm(100),10,10)
pkor(x)
```

**permutation_test**

Permutation test for (partial) correlation on non-Gaussian data

**Description**

Computes the significance of (partial) correlation based on permutations of the observations

**Usage**

```r
permutation_test(x, y, iter = 1000, fun = pcor, mode = 1, ...)```
**Arguments**

- **x**: wild type data set
- **y**: mutant data set
- **iter**: number of iterations (permutations)
- **fun**: function to compute the statistic, e.g., cor or pcor
- **mode**: either 1 for a function that takes a single data set and produces an output of class matrix, and 2, if the function takes two data sets
- **...**: additional arguments for function ‘fun’

**Value**

matrix of p-values

**Examples**

```r
x <- matrix(rnorm(100),10,10)
y <- matrix(rnorm(100),10,10)
permutation_test(x,y,iter=10)
```

---

**Description**

This function takes a differential causal effects object and plots the dag with the dces

**Usage**

```r
## S3 method for class 'dce'
plot(x, ...)
```

**Arguments**

- **x**: dce object
- **...**: Parameters passed to dce::plot_network

**Value**

plot of dag and dces

**Author(s)**

Martin Pirkl, Kim Philipp Jablonski
Examples

dag <- create_random_DAG(30, 0.2)
X.wt <- simulate_data(dag)
dag.mt <- resample_edge_weights(dag)
X.mt <- simulate_data(dag)
dce.list <- dce(dag,X.wt,X.mt)
plot(dce.list)

plot_network  

Plot network adjacency matrix

Description

Generic function which plots any adjacency matrix (assumes DAG)

Usage

plot_network(
  adja_matrix,
  nodename_map = NULL,
  edgescale_limits = NULL,
  nodesize = 17,
  labelsize = 3,
  node_color = "white",
  node_border_size = 0.5,
  arrow_size = 0.05,
  scale_edge_width_max = 1,
  show_edge_labels = FALSE,
  visualize_edge_weights = TRUE,
  use_symlog = FALSE,
  highlighted_nodes = c(),
  legend_title = "edge weight",
  value_matrix = NULL,
  shadowtext = FALSE,
  ...
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adja_matrix</td>
<td>Adjacency matrix of network</td>
</tr>
<tr>
<td>nodename_map</td>
<td>Node names</td>
</tr>
<tr>
<td>edgescale_limits</td>
<td>Limits for scale_edge_color_gradient2 (should contain 0). Useful to make plot comparable to others</td>
</tr>
<tr>
<td>nodesize</td>
<td>Node sizes</td>
</tr>
<tr>
<td>labelsize</td>
<td>Node label sizes</td>
</tr>
</tbody>
</table>
propagate_gene_edges

node_color Which color to plot nodes in
node_border_size Thickness of node’s border stroke
arrow_size Size of edge arrows
scale_edge_width_max Max range for ‘scale_edge_width’
show_edge_labels Whether to show edge labels (DCEs)
visualize_edge_weights Whether to change edge color/width/alpha relative to edge weight
use_symlog Scale edge colors using dce::symlog
highlighted_nodes List of nodes to highlight
legend_title Title of edge weight legend
value_matrix Optional matrix of edge weights if different from adjacency matrix
shadowtext Draw white outline around node labels
... additional parameters

Value

plot of dag and dces

Author(s)

Martin Pirkl, Kim Philipp Jablonski

Examples

```r
adj <- matrix(c(0,0,0,0,1,0,0,0,1,0,0,0,1,0),3,3)
plot_network(adj)
```

propagate_gene_edges  Remove non-gene nodes from pathway and reconnect nodes

Description

Remove non-gene nodes from pathway and reconnect nodes

Usage

`propagate_gene_edges(graph)`

Arguments

graph Biological pathway
Value

graph with only genes as nodes

Examples

dag <- create_random_DAG(30, 0.2)
propagate_gene_edges(dag)

graph.wt <- as(matrix(c(0,0,0,1,0,0,0,1,0), 3), "graphNEL")
graph.mt <- resample_edge_weights(graph.wt)

resample_edge_weights  Resample network edge weights

Description

Takes a graph and modifies edge weights.

Usage

resample_edge_weights(g, tp = 0.5, mineff = 1, maxeff = 2, method = "unif")

Arguments

g  original graph

  tp  fraction of edge weights which will be modified

  mineff  minimal differential effect size

  maxeff  maximum effect effect size or standard deviation, if method is "gauss"

  method  method for drawing the differential for the causal effects. Can be "unif", "exp" or "gauss".

Value

graph with new edge weights

Author(s)

Martin Pirkl

Examples

graph.wt <- as(matrix(c(0,0,0,1,0,0,0,1,0), 3), "graphNEL")
graph.mt <- resample_edge_weights(graph.wt)
**rlm_dce**  

**Description**  

custom rlm function  

**Usage**  

rlm_dce(...)  

**Arguments**  

... see ?MASS::rlm

---

**simulate_data**  

**Simulate data**  

**Description**  

Generate data for given DAG. The flexible framework allows for different distributions for source and child nodes. Default distributions are negative binomial (with mean = 100 and 1/dispersion = 100), and poisson, respectively.

**Usage**  

```r  
simulate_data(  
  graph,  
  n = 100,  
  dist_fun = rbnbinom,  
  dist_args = list(mu = 1000, size = 100),  
  child_fun = rpois,  
  child_args = list(),  
  child_dep = "lambda",  
  link_fun = negative.binomial.special()$linkfun,  
  link_args = list(offset = 1),  
  pop_size = 0,  
  latent = 0,  
  latent_fun = "unif"  
)  
```

## S4 method for signature 'igraph'  
simulate_data(  
  graph,  
  n = 100,  
)
dist_fun = rnbinom,
dist_args = list(mu = 1000, size = 100),
child_fun = rpois,
child_args = list(),
child_dep = "lambda",
link_fun = negative.binomial.special()$linkfun,
link_args = list(offset = 1),
pop_size = 0,
latent = 0,
latent_fun = "unif"
)

## S4 method for signature 'graphNEL'
simulate_data(
  graph,
  n = 100,
  dist_fun = rnbinom,
dist_args = list(mu = 1000, size = 100),
child_fun = rpois,
child_args = list(),
child_dep = "lambda",
link_fun = negative.binomial.special()$linkfun,
link_args = list(offset = 1),
pop_size = 0,
latent = 0,
latent_fun = "unif"
)

## S4 method for signature 'matrix'
simulate_data(
  graph,
  n = 100,
  dist_fun = rnbinom,
dist_args = list(mu = 1000, size = 100),
child_fun = rpois,
child_args = list(),
child_dep = "lambda",
link_fun = negative.binomial.special()$linkfun,
link_args = list(offset = 1),
pop_size = 0,
latent = 0,
latent_fun = "unif"
)

Arguments

- **graph**: Graph to simulate on
- **n**: Number of samples
summary.rlm.dce

dist_fun  distribution function for nodes without parents
dist_args list of arguments for dist_fun
child_fun  distribution function for nodes with parents
child_args list of arguments for child_fun
child_dep  link_fun computes an output for the expression of nodes without parents. This
            output is than used as input for child_fun. child_dep defines the parameter (a a
            string) of child_fun, which is used for the input. E.g., the link_fun is the identity
            and the child_fun is rnorm, we usually set child_dep = "mean".
link_fun  special link function for the negative binomial distribution
link_args list of arguments for link_fun
pop_size numeric for the population size, e.g., pop_size=1000 adds 1000-n random genes
         not in the graph
latent  number of latent variables
latent_fun  uniform "unif" or exponential "exp" distribution of latent coefficients

Value

graph

Examples

dag <- create_random_DAG(30, 0.2)
X <- simulate_data(dag)

summary.rlm.dce  summary for rlm.dce

Description

summary for rlm.dce

Usage

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

Arguments

object  object of class 'rlm.dce'
...  see ?MASS::summary.rlm
topologically_ordering

Description
Order rows/columns of a adjacency matrix topologically

Usage
topologically_ordering(adja_mat, alt = FALSE)

Arguments
adja_mat Adjacency matrix of network
alt Use igraph implementation

Value
topologically ordered matrix

Examples
adj <- matrix(c(0,1,0,0,0,1,0,0,0),3,3)
topologically_ordering(adj)

tureEffects

Compute the true casual effects of a simulated dag

Description
This function takes a DAG with edgeweights as input and computes the causal effects of all nodes on all direct and indirect children in the DAG. Alternatively see pcalg::causalEffect for pairwise computation.

Usage
tureEffects(g, partial = FALSE)

Arguments
g graphNEL object
partial if FALSE computes the total causal effects and not just the partial edge effects
trueEffects

Value

matrix of causal effects

Author(s)

Martin Pirkl

Examples

```r
graph.wt <- as(matrix(c(0,0,0,1,0,0,0,1,0), 3), "graphNEL")
trueEffects(graph.wt)
```
Index

* datasets
  df_pathway_statistics, 8
  as.data.frame.dce, 3
  as_adjmat, 3
  create_random_DAG, 4
  dce, 5
  dce,graphNEL-method(dce), 5
  dce,igraph-method(dce), 5
  dce,matrix-method(dce), 5
  dce_nb, 7
  df_pathway_statistics, 8
  estimate_latent_count, 9
  g2dag, 10
  get_pathway_info, 11
  get_pathways, 10
  get_prediction_counts, 12
  graph2df, 13
  graph_union, 13
  pcor, 14
  permutation_test, 14
  plot.dce, 15
  plot_network, 16
  propagate_gene_edges, 17
  resample_edge_weights, 18
  rlm_dce, 19
  simulate_data, 19
  simulate_data,graphNEL-method
    (simulate_data), 19
  simulate_data,igraph-method
    (simulate_data), 19
  simulate_data,matrix-method
    (simulate_data), 19
  summary.rlm_dce, 21
  topologically_ordering, 22
  trueEffects, 22