

# Package ‘netOmics’

April 12, 2022

**Title** Multi-Omics (time-course) network-based integration and interpretation

**Version** 1.0.0

**Description** netOmics is a multi-omics networks builder and explorer.

It uses a combination of network inference algorithms and knowledge-based graphs to build multi-layered networks.

The package can be combined with timeOmics to incorporate time-course expression data and build sub-networks from multi-omics kinetic clusters.

Finally, from the generated multi-omics networks, propagation analyses allow the identification of missing biological functions (1), multi-omics mechanisms (2) and molecules between kinetic clusters (3). This helps to resolve complex regulatory mechanisms.

**License** GPL-3

**Encoding** UTF-8

**LazyData** false

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.2

**VignetteBuilder** knitr

**Depends** R (>= 4.1)

**Imports** dplyr, ggplot2, igraph, magrittr, minet, purrr, tibble, tidyr, AnnotationDbi, GO.db, RandomWalkRestartMH, gprofiler2, methods, stats

**Suggests** mixOmics, timeOmics, tidyverse, BiocStyle, testthat, covr, rmarkdown, knitr

**biocViews** GraphAndNetwork, Software, TimeCourse, WorkflowStep, SystemsBiology, NetworkInference, Network

**URL** <https://github.com/abodein/netOmics>

**BugReports** <https://github.com/abodein/netOmics/issues>

**git\_url** <https://git.bioconductor.org/packages/netOmics>

**git\_branch** RELEASE\_3\_14

**git\_last\_commit** 00eaf6b

**git\_last\_commit\_date** 2021-10-26

**Date/Publication** 2022-04-12

**Author** Antoine Bodein [aut, cre]

**Maintainer** Antoine Bodein <antoine.bodein.1@ulaval.ca>

## R topics documented:

|                                             |           |
|---------------------------------------------|-----------|
| combine_layers . . . . .                    | 2         |
| get_go_info . . . . .                       | 3         |
| get_graph_stats . . . . .                   | 4         |
| get_grn . . . . .                           | 5         |
| get_interaction_from_correlation . . . . .  | 6         |
| get_interaction_from_database . . . . .     | 7         |
| get_interaction_from_ORA . . . . .          | 8         |
| get_ORA . . . . .                           | 9         |
| hmp_T2D . . . . .                           | 9         |
| netOmics . . . . .                          | 10        |
| plot_rwr_subnetwork . . . . .               | 11        |
| random_walk_restart . . . . .               | 12        |
| rwr_find_closest_type . . . . .             | 13        |
| rwr_find_seeds_between_attributes . . . . . | 14        |
| summary_plot_rwr_attributes . . . . .       | 15        |
| <b>Index</b>                                | <b>18</b> |

---

|                |                       |
|----------------|-----------------------|
| combine_layers | <i>Combine layers</i> |
|----------------|-----------------------|

---

### Description

Return a merged graph from two graph layers.

### Usage

```
combine_layers(graph1, graph2 = NULL, interaction.df = NULL)
```

### Arguments

|                |                                                                                                      |
|----------------|------------------------------------------------------------------------------------------------------|
| graph1         | an igraph object or list of igraph ( <code>list.igraph</code> ).                                     |
| graph2         | an igraph object or list of igraph ( <code>list.igraph</code> ) with the same length as graph1.      |
| interaction.df | (optional) a 2 columns data.frame (from, to) describing the edges between vertices from both graphs. |

## Details

If graph2 is a single graph, it will be merged to each element of graph1 (igraph or list.igraph).

If graph2 is a list of graph (list.igraph), each element of graph1 and each element of graph2 are merged in pairs.

Optionally, interaction.df should be provide if any vertex are shared between graphs. It can also be used to extend the first graph.

In both scenarios, vertex attributes are kept. If a vertex attribute is missing from graph1 or graph2, NULL value is added. Otherwise, if there is an overlap between attribute values for the same vertex, attribute from graph2 is dropped.

## Value

a merged graph with both vertex attributes from graph1 and graph2.

## Examples

```
# with single graphs
graph1 <- igraph::graph_from_data_frame(list(from = c('A', 'B'),
                                             to = c('B', 'C')),
                                       directed = FALSE)
graph2 <- igraph::graph_from_data_frame(list(from = c(1),
                                             to = c(2)),
                                       directed = FALSE)

res <- combine_layers(graph1 = graph1,
                    graph2 = graph2)

# with list of graphs
graph1.list <- list(graph1, graph1)
graph2.list <- list(graph2, graph2)
class(graph1.list) <- class(graph2.list) <- 'list.igraph'

res <- combine_layers(graph1 = graph1.list,
                    graph2 = graph2)
res <- combine_layers(graph1 = graph1.list,
                    graph2 = graph2.list)

# with interaction dataframe
interaction.df1 <- as.data.frame(list(from = c('C', 'B'), to = c(1, 2)))
res <- combine_layers(graph1 = graph1.list,
                    graph2 = graph2,
                    interaction.df = interaction.df1)
```

**Description**

From a GO terms (GOID), return definition, ontology and term values from GO.db

**Usage**

```
get_go_info(go)
```

**Arguments**

go                    a character, GO term

**Value**

a data.frame with the following columns: 'GOID', 'DEFINITION', 'ONTOLOGY', 'TERM'

---

|                 |                             |
|-----------------|-----------------------------|
| get_graph_stats | <i>Get graph statistics</i> |
|-----------------|-----------------------------|

---

**Description**

For a given igraph or list of igraph objects, this function summarize the number of vertices/edges and other vertex attributes.

**Usage**

```
get_graph_stats(X)
```

**Arguments**

X                    an 'igraph' or 'list.igraph' object

**Value**

It returns a long data.frame with number of nodes/edges, and the count of the different attributes (if X is a list of graph, each row describes a graph)

**Examples**

```
graph1 <- igraph::graph_from_data_frame(
  list(from = c('A', 'B', 'A', 'D', 'C', 'A', 'C'),
       to = c('B', 'C', 'D', 'E', 'D', 'F', 'G')),
  directed = FALSE)
graph1 <- igraph::set_vertex_attr(graph = graph1,
  name = 'type',
  index = c('A','B','C'),
  value = '1')
graph1 <- igraph::set_vertex_attr(graph = graph1,
  name = 'type',
  index = c('D','E'),
```

```

                                value = '2')
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                name = 'type',
                                index = c('F', 'G'),
                                value = '-1')

get_graph_stats(graph1)

graph1.list <- list(graph1 = graph1,
                   graph2 = graph1)
get_graph_stats(graph1.list)

```

---

get\_grn

*Gene Regulatory Network*


---

### Description

Get Gene Regulatory Network (GRN) from a data.frame. Optionally, if the gene are clustered, sub\_network are build for each cluster.

### Usage

```
get_grn(X, cluster = NULL, method = c("aracne"), type = "gene")
```

### Arguments

|         |                                                                               |
|---------|-------------------------------------------------------------------------------|
| X       | a data.frame/matrix with gene expression (genes in columns, samples in rows). |
| cluster | (optional) clustering result from <a href="#">getCluster</a>                  |
| method  | network building method, one of c('aracne')                                   |
| type    | character added to node metadata                                              |

### Details

Methods of GRN reconstruction are as follows: 'aracne': use ARACNe algorithm on Mutual Information (MI) adjacency matrix to remove low MI edges in triangles.

### Value

An igraph object if no cluster informations are given. Otherwise, it returns a list of igraph object (list.igraph) with a subgraph for each cluster and a global graph with all the genes.

### See Also

[build.mim](#), [aracne](#), [getCluster](#)

## Examples

```
data(hmp_T2D)
# grn only on gene
cluster.mRNA <- timeOmics::getCluster(hmp_T2D$getCluster.res,
                                     user.block = 'RNA')

X <- hmp_T2D$raw$RNA
grn.res <- get_grn(X = hmp_T2D$raw$RNA,
                  cluster = cluster.mRNA,
                  method = 'aracne')
```

---

get\_interaction\_from\_correlation

*Interaction\_from\_correlation*

---

## Description

Compute correlation between two dataframe X and Y (or list of data.frame). An incidence graph is returned. A link between two features is produced if their correlation (absolute value) is above the threshold.

## Usage

```
get_interaction_from_correlation(X, Y, threshold = 0.5)
```

## Arguments

|           |                                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------------|
| X         | a data.frame or list of data.frame (with a similar number of row).                                                     |
| Y         | a data.frame or list of data.frame (with a similar number of row).                                                     |
| threshold | a threshold to cut the correlation matrix above which a link is created between a feature from X and a feature from Y. |

## Value

an 'igraph' object

## Examples

```
X <- matrix(rexp(200, rate=.1), ncol=20)
Y <- matrix(rexp(200, rate=.1), ncol=20)
get_interaction_from_correlation(X,Y)

X <- list(matrix(rexp(200, rate=.1), ncol=20),
          matrix(rexp(200, rate=.1), ncol=20))
Y <- matrix(rexp(200, rate=.1), ncol=20)
get_interaction_from_correlation(X,Y)
```

---

get\_interaction\_from\_database  
*Get interaction from database*

---

## Description

Returns an interaction graph from a vector of nodes (or a list of vectors) and an interaction database (data.frame or igraph)

## Usage

```
get_interaction_from_database(X, db = NULL, type = "db", user.ego = FALSE)
```

## Arguments

|          |                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------|
| X        | vector of nodes or list of vectors                                                                              |
| db       | data.frame (with two columns: from, to) or igraph                                                               |
| type     | character added to node metadata                                                                                |
| user.ego | logical, if user.ego == TRUE looks for first degree neighbors in db and add 'mode' metadata ('core'/'extended') |

## Value

a subset graph of db from X list of nodes

## Examples

```
X <- letters[1:4]
db <- as.data.frame(list(from = sample(letters[1:10], replace = TRUE),
                        to = sample(letters[1:10], replace = TRUE)))

sub <- get_interaction_from_database(X,
                                   db)

db.graph <- igraph::graph_from_data_frame(db,
                                         directed=FALSE)

sub <- get_interaction_from_database(X,
                                   db)
```

---

`get_interaction_from_ORA`*Get interaction from ORA enrichment analysis*

---

**Description**

Returns results of an ORA analysis as an interaction graph

**Usage**

```
get_interaction_from_ORA(  
  query,  
  sources = "GO",  
  organism = "hsapiens",  
  signif.value = TRUE  
)
```

**Arguments**

|                           |                                                                           |
|---------------------------|---------------------------------------------------------------------------|
| <code>query</code>        | a vector (or a list) of character with the ID to perform the ORA analysis |
| <code>sources</code>      | (optional) a character in (GO, KEGG, REAC, TF, MIRNA, CORUM, HP, HPA, WP) |
| <code>organism</code>     | (optional) a character (default = 'hsapiens')                             |
| <code>signif.value</code> | (optional) a logical, default = ""                                        |

**Value**

a graph object (or list of graph) containing the interaction between the query and the target terms.

**See Also**

[gost gconvert](#)

**Examples**

```
query <- c('IL15', 'CDHR5', 'TGFA', 'C4B')  
get_interaction_from_ORA(query,  
  sources = 'GO')  
  
query <- list('All' = c('IL15', 'CDHR5', 'TGFA', 'C4B'),  
  'c1' = c('IL15', 'CDHR5', 'TGFA'))  
get_interaction_from_ORA(query,  
  sources = 'GO')
```



---

|         |                                |
|---------|--------------------------------|
| get_ORA | <i>ORA enrichment analysis</i> |
|---------|--------------------------------|

---

**Description**

Returns results of an ORA analysis

**Usage**

```
get_ORA(query, sources = NULL, organism = "hsapiens")
```

**Arguments**

|          |                                     |
|----------|-------------------------------------|
| query    | a vector of character, a list of ID |
| sources  | a character or list of character    |
| organism | a character (default = 'hsapiens')  |

**Value**

a data.frame containing the enrichment result

**See Also**

[gost](#)

---

|         |                |
|---------|----------------|
| hmp_T2D | <i>hmp_T2D</i> |
|---------|----------------|

---

**Description**

This dataset contained a list of data.frames. Raw data is a subset of the data available at: [https://github.com/aametwally/ipop\\_s](https://github.com/aametwally/ipop_s). The package will be illustrated on longitudinal MO dataset to study the seasonality of MO expression in patients with diabetes (see netOmics vignette). In this subset we focused on a single individual with 7 timepoints. Briefly 6 different omics were sampled (RNA, proteins, cytokines, gut microbiome, metabolites and clinical variables).

**Usage**

```
hmp_T2D
```

**Format**

a list of data.frame

**raw** data.frame, raw data

**modelled** data.frame, modelled data

**getCluster.res** data.frame, clustering results from timeOmics

**getCluster.sparse.res** data.frame, sparse clustering results from timeOmics

**interaction.biogrid** data.frame, interactions from BioGRID database

**interaction.TF** data.frame, TFome interactions from TTrust and TF2DNA

**medlineranker.res.df** data.frame, medlineRanker enrichment results

**graph.gut** list of igraph, gut graph obtained with SparCC

---

netOmics

*netOmics: network-based multi-omics integration and interpretation*

---

**Description**

netOmics is a multi-omics networks builder and explorer. It uses a combination of network inference algorithms and knowledge-based graphs to build multi-layered networks.

The package can be combined with timeOmics to incorporate time-course expression data and build sub-networks from multi-omics kinetic clusters.

Finally, from the generated multi-omics networks, propagation analyses allow the identification of missing biological functions (1), multi-omics mechanisms (2) and molecules between kinetic clusters (3). This helps to resolve complex regulatory mechanisms. Here are the main functions.

**Network building**

**get\_grn** Based on expression matrix, this function build a gene gene regulatory network. Additionally, if clustering information is given, it builds cluster specific graph.

**get\_interaction\_from\_database** From a database (graph or data.frame with interactions between 2 molecules), this function build the induced graph based on a list of molecules . Alternatively, the function can build a graph with the first degree neighbors.

**get\_interaction\_from\_correlation** Compute correlation between two dataframe X and Y (or list of data.frame). An incidence graph is returned. A link between two features is produced if their correlation (absolute value) is above the threshold.

**combine\_layers** Combine 2 (or list of) graphs based on given intersections.

**Network exploration**

`random_walk_restart` This function performs a propagation analysis by random walk with restart in a multi-layered network from specific seeds.

`rwr_find_seeds_between_attributes` From rwr results, this function returns a subgraph if any vertex shares different attributes value. In biological context, this might be useful to identify vertex shared between clusters or omics types.

`rwr_find_closest_type` From a rwr results, this function returns the closest nodes from a seed with a given attribute and value. In biological context, it might be useful to get the closest Gene Ontology annotation nodes from unannotated seeds.

**Visualisation**

`summary_plot_rwr_attributes` #' Based on the results of `rwr_find_seeds_between_attributes` which identify the closest k neighbors from a seed, this function returns a barplot of the node types (layers) reached for each seed.

`plot_rwr_subnetwork` Display the subgraph from a RWR results. This function colors adds a specific color to each node based on their 'type' attribute. It also adds a legend including the number of vertices/edges and the number of nodes of specific type. Additionally, the function can display any igraph object.

---

`plot_rwr_subnetwork` *Plot RWR subnetwork*

---

**Description**

Display the subgraph from a RWR results. This function colors adds a specific color to each node based on their 'type' attribute. It also adds a legend including the number of vertices/edges and the number of nodes of specific type. Additionally, the function can display any igraph object.

**Usage**

```
plot_rwr_subnetwork(X, color = NULL, plot = TRUE, legend = TRUE, ...)
```

**Arguments**

|                     |                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>X</code>      | an igraph object                                                                                                                 |
| <code>color</code>  | (optional) a named character vector or list, list of color to apply to each type                                                 |
| <code>plot</code>   | logical, if TRUE then the plot is produced                                                                                       |
| <code>legend</code> | (optional) logical, if TRUE then the legend is displayed with number of vertices/edges and the number of nodes of specific type. |
| <code>...</code>    | Arguments to be passed to the plot method                                                                                        |

**Value**

`X` is returned with additional vertex attributes

**Examples**

```

graph1 <- igraph::graph_from_data_frame(
  list(from = c("A", "B", "A", "D", "C", "A", "C"),
       to = c("B", "C", "D", "E", "D", "F", "G")),
  directed = FALSE)
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c("A", "B", "C"),
                                 value = "1")
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c("D", "E"),
                                 value = "2")
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c("F", "G"),
                                 value = "3")

rwr_res <- random_walk_restart(X = graph1,
                              seed = c("A"))
rwr_res_type <- rwr_find_seeds_between_attributes(X = rwr_res,
                                                attribute = "type")

plot_rwr_subnetwork(rwr_res_type$A)

```

---

random\_walk\_restart     *Random Walk with Restart*

---

**Description**

This function performs a propagation analysis by random walk with restart in a multi-layered network from specific seeds.

**Usage**

```
random_walk_restart(X, seed = NULL, r = 0.7)
```

**Arguments**

|      |                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------|
| X    | an igraph or list.igraph object.                                                                       |
| seed | a character vector. Only seeds present in X are considered.                                            |
| r    | a numeric value between 0 and 1. It sets the probability of restarting to a seed node after each step. |

**Value**

Each element of *X* returns a list (class = 'rwr') containing the following elements:

*rwr*                    a data.frame, the RWR results for each valid seed.  
*seed*                    a character vector with the valid seeds  
*graph*                    igraph object from *X*

If *X* is a list.igraph, the returned object is a list.rwr.

**See Also**

[Random.Walk.Restart.Multiplex](#), [rwr\\_find\\_seeds\\_between\\_attributes](#), [rwr\\_find\\_closest\\_type](#)

**Examples**

```
graph1 <- igraph::graph_from_data_frame(
  list(from = c('A', 'B', 'A', 'D', 'C', 'A', 'C'),
       to = c('B', 'C', 'D', 'E', 'D', 'F', 'G')),
  directed = FALSE)
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c('A', 'B', 'C'),
                                 value = '1')
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c('D', 'E'),
                                 value = '2')
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                 name = 'type',
                                 index = c('F', 'G'),
                                 value = '3')

rwr_res <- random_walk_restart(X = graph1,
                               seed = c('A', 'B', 'C', 'D', 'E'))
```

---

rwr\_find\_closest\_type *RWR Find closest nodes*

---

**Description**

From a rwr results, this function returns the closest nodes from a seed with a given attribute and value. In biological context, it might be useful to get the closest Gene Ontology annotation nodes from unannotated seeds.

**Usage**

```
rwr_find_closest_type(X, seed = NULL, attribute = NULL, value = NULL, top = 1)
```

**Arguments**

|           |                                                                                         |
|-----------|-----------------------------------------------------------------------------------------|
| X         | a random walk result from random_walk_restart                                           |
| seed      | a character vector or NULL. If NULL, all the seeds from X are considered.               |
| attribute | a character value or NULL. If NULL, the closest node is returned.                       |
| value     | a character value or NULL. If NULL, the closest node for a given attribute is returned. |
| top       | a numeric value, the top closest nodes to extract                                       |

**Value**

A list of data.frame for each seed containing the closest nodes per seed and their vertex attributes. If X is list.rwr, the returned value is a list of list.

**Examples**

```
graph1 <- igraph::graph_from_data_frame(
  list(from = c("A", "B", "A", "D", "C", "A", "C"),
       to = c("B", "C", "D", "E", "D", "F", "G")),
  directed = FALSE)
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                name = 'type',
                                index = c("A", "B", "C"),
                                value = "1")
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                name = 'type',
                                index = c("D", "E"),
                                value = "2")
graph1 <- igraph::set_vertex_attr(graph = graph1,
                                name = 'type',
                                index = c("F", "G"),
                                value = "3")

rwr_res <- random_walk_restart(X = graph1,
                              seed = c("A", "B", "C", "D", "E"))
rwr_find_closest_type(X=rwr_res, attribute = "type",
                     seed = "A")
```

---

rwr\_find\_seeds\_between\_attributes

*RWR Find seeds between attributes*

---

**Description**

From rwr results, this function returns a subgraph if any vertex shares different attributes value. In biological context, this might be useful to identify vertex shared between clusters or omics types.

**Usage**

```
rwr_find_seeds_between_attributes(X, seed = NULL, k = 15, attribute = "type")
```

**Arguments**

**X** a random walk result from `random_walk_restart`

**seed** a character vector or `NULL`. If `NULL`, all the seeds from `X` are considered.

**k** an integer, `k` closest nodes to consider in the search

**attribute** a character value or `NULL`. If `NULL`, the closest node is returned.

**Value**

A list of igraph object for each seed. If `X` is a list, it returns a list of list of graph.

**Examples**

```
graph1 <- igraph::graph_from_data_frame(
  list(from = c("A", "B", "A", "D", "C", "A", "C"),
       to = c("B", "C", "D", "E", "D", "F", "G")),
  directed = FALSE)
graph1 <- igraph::set_vertex_attr(graph = graph1,
  name = 'type',
  index = c("A", "B", "C"),
  value = "1")
graph1 <- igraph::set_vertex_attr(graph = graph1,
  name = 'type',
  index = c("D", "E"),
  value = "2")
graph1 <- igraph::set_vertex_attr(graph = graph1,
  name = 'type',
  index = c("F", "G"),
  value = "3")

rwr_res <- random_walk_restart(X = graph1,
  seed = c("A", "B", "C", "D", "E"))
rwr_res_type <- rwr_find_seeds_between_attributes(X = rwr_res,
  attribute = "type",
  k = 3)
```

---

summary\_plot\_rwr\_attributes

*Summary Plot RWR attributes*

---

**Description**

Based on the results of `rwr_find_seeds_between_attributes` which identify the closest `k` neighbors from a seed, this function returns a barplot of the node types (layers) reached for each seed.





*summary\_plot\_rwr\_attributes*

17

`summary_plot_rwr_attributes(rwr_res_type)`

# Index

## \* datasets

- hmp\_T2D, [9](#)
  
- aracne, [5](#)
  
- build.mim, [5](#)
  
- combine\_layers, [2](#)
  
- gconvert, [8](#)
- get\_go\_info, [3](#)
- get\_graph\_stats, [4](#)
- get\_grn, [5](#)
- get\_interaction\_from\_correlation, [6](#)
- get\_interaction\_from\_database, [7](#)
- get\_interaction\_from\_ORA, [8](#)
- get\_ORA, [9](#)
- getCluster, [5](#)
- gost, [8](#), [9](#)
  
- hmp\_T2D, [9](#)
  
- netOmics, [10](#)
  
- plot\_rwr\_subnetwork, [11](#)
  
- Random.Walk.Restart.Multiplex, [13](#)
- random\_walk\_restart, [12](#), [16](#)
- rwr\_find\_closest\_type, [13](#), [13](#)
- rwr\_find\_seeds\_between\_attributes, [11](#),  
[13](#), [14](#), [15](#), [16](#)
  
- summary\_plot\_rwr\_attributes, [15](#)