Package 'MetaPathNet'

November 25, 2026

Title KEGG-Based Metabolic and Signaling Network Analysis for Systems Biology

Version 1.0.0

Description Provides tools to construct KEGG-based metabolic and signaling networks as edge lists for single-organism or cross-species analyses. The package supports identifier mapping, shortest-path and topology analyses, community detection, permutation testing, pathway over-representation analysis, and node annotation for host-microbiome studies. It also provides network visualisation in R and Cytoscape and supports extension of KEGG-based networks through additional reaction resources and user-defined reactions.

Depends R (>= 4.6.0)

Imports igraph, httr, utils, RCurl, RCy3, tidygraph, ggraph, dplyr, KEGGREST, KEGGgraph, graph, mygene, ggplot2, rlang, curl, jsonlite, webchem, grDevices, grid, stats

Suggests testthat, knitr, rmarkdown, BiocStyle

SystemRequirements Cytoscape (>= 3.9.0) for Cytoscape-based visualisation functions

biocViews Network, Pathways, KEGG, SystemsBiology, Classification, Microbiome

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

URL https://github.com/zhaojie-wang/MetaPathNet

BugReports https://github.com/zhaojie-wang/MetaPathNet/issues

Config/roxygen2/version 8.0.0

git_url https://git.bioconductor.org/packages/MetaPathNet

git_branch RELEASE_3_24

git_last_commit a528a6d

git_last_commit_date 2026-11-23

Repository Bioconductor 3.24

Date/Publication 2026-11-24

Author Zhaojie Wang [aut, cre] (ORCID:
<https://orcid.org/0009-0009-7191-2107>),
Francesc Puig-Castellvi [aut],
Manyi Jia [aut],
Marc-Emmanuel Dumas [aut, ths],
Centre National de la Recherche Scientifique [fnd],
Imperial College London [fnd],
French National Research Agency [fnd]

Maintainer Zhaojie Wang <zhaojie.wang@cnrs.fr>

Contents

addCustomReaction . . . 3
annotateKoClass . . . 4
annotateOrigin . . . 6
buildCompoundEgoNetwork . . . 9
buildCrossSpeciesNetwork . . . 11
buildEgoNetwork . . . 12
buildKeggNetwork . . . 14
clusterNetwork . . . 16
combineNetworks . . . 18
convertGene . . . 19
enrichmentDotPlot . . . 20
enrichPathway . . . 22
findKeggIDs . . . 24
findMappedNodes . . . 25
findShortestPaths . . . 26
getPathIDs . . . 29
mapReaction . . . 30
MetaPathNet_example_network . . . 32
networkDistances . . . 33
networkSimilarity . . . 34
network_utils . . . 36
permutePaths . . . 37
removeDrugs . . . 39
removeNetworkNodes . . . 40
replaceNode . . . 41
suggestEntities . . . 42
viewClusterCy . . . 44
viewNetworkCy . . . 46
viewNetworkR . . . 48

Index **50**

addCustomReaction *Build MetaPathNet-style edges from user-defined reactions*

Description

Convert one or more manually curated reactions into the standard MetaPathNet 3-column edge list format (`source`, `target`, `interaction_type`).

Usage

```
addCustomReaction(
  reaction_table,
  substrate_col = "substrates",
  product_col = "products",
  ko_col = NULL,
  direction_col = NULL
)
```

Arguments

`reaction_table`	A data.frame in which each row corresponds to one custom reaction.
`substrate_col`	Character. Name of the column containing substrate node IDs. Multiple IDs must be separated by semicolons.
`product_col`	Character. Name of the column containing product node IDs. Multiple IDs must be separated by semicolons.
`ko_col`	Optional character. Name of the column containing KO IDs. Multiple IDs must be separated by semicolons. If `NULL`, no KO layer is added and direct substrate-product edges are returned.
`direction_col`	Optional character. Name of the column containing reaction direction. Accepted values are `"reversible"`, `"irreversible"`, and `"unknown"`. If `NULL`, all reactions are treated as `"unknown"`.

Details

Each input row represents one reaction. Substrates and products can contain one or multiple node IDs, separated by semicolons. Optional KO annotations can be provided to build compound-KO-compound edges; otherwise the function returns direct compound-compound edges. User-defined compounds or enzymes can first be mapped to KEGG identifiers with `findKeggIDs()` when possible.

Value

A character matrix with three columns:

source Source node ID.

target Target node ID.

interaction_type Reaction type, e.g. `"custom:reversible"`, `"custom:irreversible"`, or `"custom:unknown"`.

See Also

`findKeggIDs`, `mapReaction`, `combineNetworks`

Examples

```
## Example — Add a user-defined irreversible reaction

## 1) Build a custom reaction table
##    Multiple substrates, products, or KO terms must be separated by semicolons.
custom_df <- data.frame(
  reaction_id = "custom_reaction_1",
  substrates  = "cpd:C00078; cpd:C00001",
  products    = "cpd:C00328",
  ko          = "K00453",
  direction   = "irreversible",
  stringsAsFactors = FALSE
)

## 2) Convert the user-defined reaction into MetaPathNet-style edges
custom_edges <- addCustomReaction(
  reaction_table = custom_df,
  substrate_col  = "substrates",
  product_col    = "products",
  ko_col         = "ko",
  direction_col  = "direction"
)

custom_edges
```

`annotateKoClass` *Annotate KO functional class (metabolic vs signalling)*

Description

Annotates KEGG Orthology (KO) nodes in a MetaPathNet-style network by pathway context: `"metabolic"`, `"signaling"`, or `"metabolic&signaling"`. Classification is derived from KEGG BRITE pathway hierarchy and KO-to-pathway links. The function returns a KO-level annotation table (compound nodes excluded) and can optionally export a Cytoscape network with KO class colouring.

Usage

```
annotateKoClass(
  network_table,
  name = FALSE,
  export_cytoscape = FALSE,
  compound_color = "#9DC7DD",
  metabolic_color = "#9ED17B",
  signaling_color = "#C45745",
  metabolic_signaling_color = "#B57F60",
  other_color = "#D1C2C2",
  network_title = "",
  collection_title = ""
)
```

Arguments

network_table	A MetaPathNet-style network with at least two columns: source and target. If a third column is present, it is interpreted as interaction_type; otherwise it is set to NA_character_.
name	Logical. If TRUE, convert node IDs to KEGG-derived labels (KO: SYMBOL preferred, then NAME; non-KO: cleaned NAME). If FALSE (default), keep original node IDs.
export_cytoscape	Logical. If FALSE (default), return only the annotation data frame. If TRUE, also create and style a Cytoscape network using the annotated node table.
compound_color	Hex colour for compound nodes (used only when export_cytoscape = TRUE).
metabolic_color	Hex colour for KO nodes classified as "metabolic" (used only when export_cytoscape = TRUE).
signaling_color	Hex colour for KO nodes classified as "signaling" (used only when export_cytoscape = TRUE).
metabolic_signaling_color	Hex colour for KO nodes classified as "metabolic&signaling" (used only when export_cytoscape = TRUE).
other_color	Hex colour for KO nodes whose class cannot be determined and for non-compound, non-KO nodes (used only when export_cytoscape = TRUE).
network_title	Character. Cytoscape network title (used only when export_cytoscape = TRUE).
collection_title	Character. Cytoscape collection title (used only when export_cytoscape = TRUE).

Details

Cytoscape (version **>= 3.9.0**) must be open and reachable via **RCy3** only when export_cytoscape = TRUE.

Reverse duplicated edges are collapsed for Cytoscape display when they share the same metabolic interaction type KO functional class is derived from the KEGG BRITE pathway hierarchy (br:br08901). Pathways under the "Metabolism" branch are classified as metabolic, whereas all other linked pathways are treated as signaling for this annotation workflow.

Value

A data frame with columns id, type, name, and ko_class. Compound nodes are excluded from the returned table.

If export_cytoscape = TRUE, the same annotation is also applied to a Cytoscape network as a side effect, with node colours mapped to ko_class; the function invisibly returns the annotation data frame.

See Also

annotateOrigin for host vs bacteria origin, and viewNetworkCy for generic Cytoscape visualisation of MetaPathNet networks.

Examples

```
## 1) Prepare a mixed human-E. coli tryptophan-focused network
## Human: tryptophan metabolism + selected immune/inflammatory signaling pathways
## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Combined host-microbe network
net_trp_mixed <- MetaPathNet_example_network

## 2) Build a shortest-path subnetwork (used here as the input network_table)
source_nodes <- c("K00486", "K00453", "K00463", "K01667", "K01696", "K01695")
target_nodes <- c("cpd:C00078", "cpd:C00328", "cpd:C00780", "cpd:C00463", "cpd:C00108")

map_src <- findMappedNodes(source_nodes, net_trp_mixed)
map_tgt <- findMappedNodes(target_nodes, net_trp_mixed)

paths_trp_mixed <- findShortestPaths(
  network_table      = net_trp_mixed,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "network_matrix",
  name               = FALSE,
  distance_threshold = 6
)

## 3) KO class annotation table (data frame output)
ko_class_df <- annotateKoClass(
  network_table = paths_trp_mixed,
  name          = FALSE
)
```

`annotateOrigin` *Annotate KO Origin (Host vs Bacteria) and Optionally Export to Cytoscape*

Description

Annotates nodes in a MetaPathNet-style network with inferred biological origin, focusing on KEGG Orthology (KO) nodes. KO origin is classified as human-only (`"hsa"`), bacteria-only (`"bacteria"`), shared (`"hsa&bacteria"`), or `"other"` based on KEGG annotations. Compound nodes are assigned the separate class `"compound"`.

Usage

```
annotateOrigin(
  network_table,
  bacteria_codes = NULL,
  name = FALSE,
  export_cytoscape = FALSE,
  compound_color = "#9DC7DD",
  hsa_color = "#C45745",
```

```
  micro_color = "#9ED17B",
  hsa_micro_color = "#B57F60",
  other_color = "#D1C2C2",
  network_title = "",
  collection_title = ""
)
```

Arguments

network_table	A MetaPathNet-style network with at least two columns: source and target. If a third column is present, it is interpreted as interaction_type.
bacteria_codes	Optional character vector of KEGG bacterial organism codes. Restricts the "bacteria" and "hsa&bacteria" classes to this set. If NULL, all KEGG bacterial organism codes are used.
name	Logical. If TRUE, node display names are converted from KEGG entries. For KO nodes, SYMBOL is preferred and NAME is used as a fallback. For non-KO nodes, the KEGG NAME field is used. If FALSE, KEGG IDs are retained as labels.
export_cytoscape	Logical. If TRUE, the annotated network is exported to Cytoscape and styled by origin class. If FALSE, only the annotation table is returned.
compound_color	Hex colour for compound nodes.
hsa_color	Hex colour for KO nodes classified as "hsa".
micro_color	Hex colour for KO nodes classified as "bacteria".
hsa_micro_color	Hex colour for KO nodes classified as "hsa&bacteria".
other_color	Hex colour for KO nodes classified as "other".
network_title	Character string specifying the Cytoscape network title. Used only when export_cytoscape = TRUE.
collection_title	Character string specifying the Cytoscape collection title. Used only when export_cytoscape = TRUE.

Details

The primary output is a node-level annotation table containing the node ID, type, display name, and origin. Cytoscape export is optional and can be enabled to visualise the annotated network with origin-based node colours.

KO origin is inferred from KEGG GENES annotations retrieved with KEGGREST::keggGet():

- "hsa" if hsa is present and no selected bacterial organism code is detected.
- "bacteria" if at least one selected bacterial organism code is detected and hsa is absent.
- "hsa&bacteria" if both hsa and at least one selected bacterial organism code are detected.
- "other" if none of the above can be established.

The bacteria_codes argument can be used to restrict the bacterial origin class to a user-defined organism panel, such as cohort-specific microbes. If bacteria_codes = NULL, all KEGG bacterial organism codes are used.

If the KEGG genome or bacterial genome-group information cannot be retrieved, the function issues a warning instead of stopping. In this case, KO nodes linked exclusively to hsa are classified as "hsa", while other KO nodes are classified as "other".

Cytoscape version 3.9.0 or later must be running when `export_cytoscape = TRUE`.

Reverse duplicated edges are collapsed for Cytoscape display when they share the same supported metabolic interaction type.

Value

A node-level data frame with columns:

- `id`: KEGG node identifier.
- `type`: coarse node type (`"KO"`, `"Compound"`, or `"Other"`).
- `name`: display label containing the KEGG ID or converted KEGG name or symbol.
- `origin`: inferred origin class (`"compound"`, `"hsa"`, `"bacteria"`, `"hsa&bacteria"`, or `"other"`).

If `export_cytoscape = TRUE`, the function also creates and styles a Cytoscape network as a side effect and returns the annotation table invisibly.

See Also

`annotateKoClass` for KO functional class annotation, and `viewNetworkCy` for generic Cytoscape rendering.

Examples

```
## 1) Load the precomputed mixed human-E. coli example network
data(MetaPathNet_example_network)
net_trp_mixed <- MetaPathNet_example_network

## 2) Build a shortest-path subnetwork for origin annotation
source_nodes <- c(
  "K00486", "K00453", "K00463", "K01667",
  "K01696", "K01610", "K01695"
)

target_nodes <- c(
  "cpd:C00078", "cpd:C00328", "cpd:C00780", "cpd:C00463",
  "cpd:C00108", "cpd:C00458", "cpd:C00336"
)

map_src <- findMappedNodes(source_nodes, net_trp_mixed)
map_tgt <- findMappedNodes(target_nodes, net_trp_mixed)

paths_trp_mixed <- findShortestPaths(
  network_table      = net_trp_mixed,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "network_matrix",
  name               = FALSE,
  distance_threshold = 6
)

## 3) Return the annotation table while retaining KEGG IDs as labels
origin_paths_trp_mixed <- annotateOrigin(
  network_table    = paths_trp_mixed,
  bacteria_codes   = NULL,
```

```
  name             = FALSE,
  export_cytoscape = FALSE
)

head(origin_paths_trp_mixed)
```

`buildCompoundEgoNetwork` *Build compound-centred ego subnetworks across KEGG organisms*

Description

Builds a compound-centred ego subnetwork around one or more KEGG compounds across one or several organisms. The resulting network is returned as a MetaPathNet-style edge list (3-column matrix) that can be used directly for visualisation or further network analysis.

Usage

```
buildCompoundEgoNetwork(
  compounds,
  organism_codes,
  path_type = c("metabolic", "signaling", "integrated"),
  order = 2,
  mode = c("out", "all", "in")
)
```

Arguments

`compounds`	Character vector of KEGG compound identifiers. Only IDs in the formats `"Cxxxxx"` or `"cpd:Cxxxxx"` are accepted; any other format results in an error.
`organism_codes`	Character vector of one or more KEGG organism codes (e.g. `"hsa"`, `"eco"`, `"bsu"`). For each organism, the function identifies pathways containing the input compounds and builds an ego subnetwork restricted to those pathways.
`path_type`	Character; type of pathways to include when building the organism-specific networks. One of: • `"metabolic"`: use only metabolic pathways. • `"signaling"`: use only signaling pathways. • `"integrated"`: use both metabolic and signaling pathways. The classification is derived from `getPathIDs`.
`order`	Integer (default: 2). Maximum graph distance (number of edges) used in `buildEgoNetwork` to define the ego neighbourhood around the seed compounds. Must be a single positive integer (≥ 1).
`mode`	Character; directionality of the ego search, passed to `buildEgoNetwork` and ultimately `igraph::ego`. One of: • `"out"` (default): nodes reachable downstream from the seeds. • `"in"`: nodes from which the seeds are reachable. • `"all"`: treat edges as undirected for distance calculations.

Details

Organism-specific ego subnetworks are merged, deduplicated, and returned as a single combined network.

Internally, compound–pathway mapping is performed via KEGG REST / KGML queries (using a non-exported helper), then routed through `buildKeggNetwork` and `buildEgoNetwork`. These details are handled automatically and do not require user intervention.

Value

A character matrix with three columns:

source Source node of each edge in the combined ego subnetwork.

target Target node of each edge in the combined ego subnetwork.

interaction_type Interaction or edge type, as returned by `buildKeggNetwork` (e.g. `"k_compound:reversible"`, `"k_activation"`, etc.).

If no ego subnetworks can be generated for the given inputs, the function stops with an error.

See Also

`getPathIDs`, `buildKeggNetwork`, `buildEgoNetwork`

Examples

```
## Not run:
## Requires extensive KEGG API querying; runtime depends on server response
## 1) Define Trp/kynurenine/serotonin/anthranilate seeds
seed_compounds <- c(
  "cpd:C00078",  # L-tryptophan
  "cpd:C00328",  # L-kynurenine
  "cpd:C00780",  # serotonin
  "cpd:C00108"   # anthranilate
)

## 2) Build ego network (order = 2, metabolic pathways, human only)
ego_trp_hsa <- buildCompoundEgoNetwork(
  compounds      = seed_compounds,
  organism_codes = "hsa",
  path_type      = "metabolic",
  order          = 2,
  mode           = "out"
)

## 3) (Optional) Visualise in Cytoscape
## Cytoscape must be open and running
viewNetworkCy(
  network_table    = ego_trp_hsa,
  name             = TRUE,
  category         = TRUE,
  network_title    = "Human metabolic ego network",
  collection_title = "MetaPathNet_Examples"
)

## End(Not run)
```

`buildCrossSpeciesNetwork` *Build Cross-Species MetaPathNet Network*

Description

Constructs and merges *MetaPathNet tables* (integrated metabolic and/or signaling networks) across one or multiple KEGG organisms.

Usage

```
buildCrossSpeciesNetwork(
  organism_codes,
  path_type = c("metabolic", "signaling", "integrated")
)
```

Arguments

`organism_codes`	Character vector of KEGG organism codes (e.g., "hsa" for *Homo sapiens*, "eco" for *E. coli*, "bsu" for *Bacillus subtilis*).
`path_type`	Character. One of: **"metabolic"** Build network from metabolic pathways only. **"signaling"** Build network from signaling pathways only. **"integrated"** Combine both metabolic and signaling pathways into a single network.

Details

- KEGG pathway lists are retrieved via `getPathIDs()` for all requested organisms.
- Each species is processed separately, generating a species-specific MetaPathNet table via `buildKeggNetwork()` according to `path_type`.
- Species-level tables are then merged and deduplicated into a single cross-species network.

Value

A data frame representing the merged *MetaPathNet network*, with three columns:

source Source node (KO, compound, or gene).

target Target node (KO, compound, or gene).

interaction_type Type of biological interaction or reaction.

See Also

`getPathIDs` for retrieving pathway lists, and `buildKeggNetwork` for building individual pathway networks.

Examples

```
## Cross-species metabolic network (human + E. coli strains)
## Note: Network construction can be time-consuming and depends on KEGG server response

host_microbe_code <- c("ece", "eco", "ecoo", "ecc", "ecs", "hsa")

## Not run:
## Requires extensive KEGG API querying; runtime depends on server response
net_hsa_ecoli <- buildCrossSpeciesNetwork(
  organism_codes = host_microbe_code,
  path_type      = "metabolic"   # metabolic layer only
)

## End(Not run)
```

`buildEgoNetwork` *Extract an Ego Subnetwork from a Metabolic / Signalling Network*

Description

Builds an ego (neighbourhood) subnetwork around one or more seed nodes in a directed network, up to a user-defined distance (`order`). In graph terminology, *ego* refers to the focal node(s) used as the starting point(s), and the function returns the local network around them. This is useful to inspect which nodes connect to a selected compound, KO, or other node(s) within a chosen radius in a pathway- or organism-specific network.

Usage

```
buildEgoNetwork(
  network_table,
  nodes,
  order = 2,
  mode = c("out", "all", "in"),
  mindist = 0,
  output = c("matrix", "vector")
)
```

Arguments

`network_table`	A three-column matrix or data.frame representing a directed network. The first three columns must correspond to `source`, `target`, and `interaction_type` (e.g. output of `buildKeggNetwork()`).
`nodes`	Character vector of one or more node IDs (e.g. `"cpd:C00078"`). Values must match the node names used in the network (case sensitive); leading/trailing whitespace is trimmed. Compound IDs supplied as `"Cxxxxx"` are internally harmonised to `"cpd:Cxxxxx"`.
`order`	Integer (default 2). Maximum graph distance (in number of edges) from the ego nodes to include in the neighbourhood. Must be a single positive integer ≥ 1.
`mode`	Character. Directionality of the search, passed to `igraph::ego()`:

- `"out"` (default): include nodes reachable by following edge direction outward from the ego node(s).
- `"in"`: include nodes that can reach the ego node(s) (against edge direction).
- `"all"`: ignore edge direction for neighbourhood expansion (treat the graph as undirected for this search).

`mindist` Integer (default `0`). Minimum graph distance from the ego nodes to include (also passed to `igraph::ego()`). Must satisfy $0 \leq mindist \leq order$.

`output` Character. Type of result:

- `"matrix"` (default): 3-column edge list (`source`, `target`, `interaction_type`) for the induced subgraph on the ego neighbourhood.
- `"vector"`: character vector of unique node IDs present in the ego subnetwork.

Details

The default output is a 3-column MetaPathNet-style edge list (`source`, `target`, `interaction_type`); alternatively, the function can return only the unique node vector of the ego neighbourhood.

Internally, the function:

1. checks and normalizes `network_table` and input `nodes`;
2. builds a directed `igraph` object from the first three columns;
3. maps the requested ego nodes to existing vertex names (with a message for nodes not found);
4. uses `igraph::ego()` to collect all vertices within `[mindist, order]` of the mapped egos;
5. returns either the induced edge list (`"matrix"`) or the node set (`"vector"`).

For large networks and high values of `order`, ego subnetworks can become large and slower to generate. It is often practical to start with `order = 1` or `2` and increase only if needed.

Value

If `output = "matrix"`, a character matrix with columns `source`, `target`, `interaction_type` representing the induced subnetwork on all nodes within `[mindist, order]` of the ego nodes (according to `mode`). If no edges are found, an empty 3-column matrix with these columns is returned.

If `output = "vector"`, a character vector of unique node IDs in the ego subnetwork is returned.

References

Ego-network (ego graph) concept is standard in social/network analysis; see Wasserman S, Faust K (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

The neighbourhood extraction in this function is implemented with `igraph::ego()` from the **igraph** package.

See Also

`ego`, `induced_subgraph`, `enrichPathway`, `viewNetworkR`

Examples

```
## Example — Ego subnetwork around tryptophan-related metabolites in a mixed hsa/eco network

## 1) Build a mixed host-microbe network for tryptophan-related metabolism
## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Combined host-microbe network
net_trp_mixed <- MetaPathNet_example_network

## 2) Extract an ego subnetwork around key metabolites used as seed nodes
## (tryptophan, serotonin, and indole are central compounds in the example storyline)
ego_trp_compounds <- buildEgoNetwork(
  network_table = net_trp_mixed,
  nodes         = c(
    "cpd:C00078",  # L-tryptophan
    "cpd:C00780",  # serotonin
    "cpd:C00463"   # indole
  ),
  order   = 2,
  mode    = "out",
  mindist = 0,
  output  = "matrix"
)
```

`buildKeggNetwork` *Build KEGG Metabolic and Signaling Network*

Description

Constructs a directed network (3-column edge list) from user-specified KEGG metabolic and/or signaling pathways. KEGG KGML files are downloaded and parsed, networks are merged, organism-specific gene IDs in signaling pathways are converted to KEGG Orthology (KO) IDs, self-loops are removed, and interaction types are annotated.

Usage

```
buildKeggNetwork(metabo_paths = NULL, signaling_paths = NULL)
```

Arguments

`metabo_paths`	Character vector of KEGG pathway IDs for metabolic pathways (e.g. `"hsa00380"`, `"eco00380"`). Use `NULL` to exclude the metabolic layer.
`signaling_paths`	Character vector of KEGG pathway IDs for signaling pathways (e.g. `"hsa04060"`, `"hsa04630"`). Use `NULL` to exclude the signaling layer.

Details

- All pathways supplied in `metabo_paths` and `signaling_paths` must belong to the same organism (same prefix, e.g. `"hsa"`, `"eco"`, `"bsu"`).

- If both metabolic and signaling pathways are provided, the output network is the union of both layers.
- Invalid or unavailable pathway IDs are skipped with a message, and the network is built from valid pathways only.

Value

A `data.frame` with 3 columns:

source Source node (compound or KO/gene).

target Target node (KO/gene or compound).

interaction_type Interaction type (e.g. metabolic reaction, activation, inhibition, phosphorylation).

See Also

`getPathIDs` for pathway ID retrieval.

Examples

```
## Not run:
## Requires extensive KEGG API querying; runtime depends on server response
## Example 1 — Human tryptophan metabolism + immune signaling

## Metabolic pathway: Tryptophan metabolism (human)
metabo_paths_hsa <- c("hsa00380")

## Selected immune / inflammatory signaling pathways (human)
signaling_paths_hsa <- c(
  "hsa04060",  # Cytokine-cytokine receptor interaction
  "hsa04630",  # JAK-STAT signaling pathway
  "hsa04064",  # NF-kappa B signaling pathway
  "hsa04660",  # T cell receptor signaling pathway
  "hsa04659"   # Th17 cell differentiation
)

net_trp_hsa <- buildKeggNetwork(
  metabo_paths    = metabo_paths_hsa,
  signaling_paths = signaling_paths_hsa
)

head(net_trp_hsa)

## Example 2 — E. coli tryptophan metabolism + biosynthesis

metabo_paths_eco <- c(
  "eco00380",  # Tryptophan metabolism (E. coli)
  "eco00400"   # Phenylalanine, tyrosine and tryptophan biosynthesis (E. coli)
)

net_trp_eco <- buildKeggNetwork(
  metabo_paths    = metabo_paths_eco,
  signaling_paths = NULL
)

head(net_trp_eco)
```

```
## End(Not run)
```

`clusterNetwork` *Community Detection in KEGG-Derived Networks*

Description

Applies graph-based community detection (Louvain or Leiden) to a KEGG-derived network and returns an annotated edge list with cluster membership for each source and target node. This supports module-based interpretation and visualisation of large metabolic / signalling networks.

Usage

```
clusterNetwork(
  network_table,
  method = c("louvain", "leiden"),
  resolution = 0.1
)
```

Arguments

`network_table`	A matrix or data.frame with at least 3 columns, representing a directed KEGG-based network (typically from `buildKeggNetwork()` or `buildCrossSpeciesNetwork()`). The first three columns are interpreted as `source`, `target`, and `interaction_type`.
`method`	Character. Community detection algorithm: • `"louvain"` — Louvain modularity optimisation. • `"leiden"` — Leiden algorithm (improved modularity-based clustering).
`resolution`	Numeric (default: `0.1`). Resolution parameter for modularity-based clustering. Smaller values give fewer, larger communities; larger values give more, smaller communities.

Details

Community detection is performed on the full graph (all nodes together), using an undirected collapsed version of the input network.

In the returned edge list, `source_cluster` and `target_cluster` are endpoint annotations: they indicate the community membership of the source node and target node of each edge, respectively.

This edge-level annotation format is used to keep the output compatible with downstream visualisation and filtering functions (e.g. `viewClusterCy()` and `viewNetworkR()`).

In MetaPathNet, these cluster assignments can also be used to define modules for downstream enrichment or ego-network extraction.

Value

A matrix with the same rows as `network_table` and the original first three columns:

source Source node identifier.

target Target node identifier.

interaction_type Interaction or edge annotation.

plus two additional columns:

source_cluster Community label of the source node.

target_cluster Community label of the target node.

References

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), P10008. doi:10.1088/17425468/2008/10/P10008

Traag, V. A., Waltman, L., and van Eck, N. J. (2019). From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9, 5233. doi:10.1038/s4159801941695z

See Also

`buildKeggNetwork`, `buildCrossSpeciesNetwork`, `viewClusterCy`, `viewNetworkR`

Examples

```
## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

net_trp <- MetaPathNet_example_network

## 2) Run community detection on the example network
net_trp_clusters <- clusterNetwork(
  network_table = net_trp,
  method        = "leiden",
  resolution    = 0.01
)

## 3) (Optional) Visualise one cluster in R
##    Here, source_cluster and target_cluster are node-cluster labels
##    attached to each edge endpoint.
subnet_cluster1 <- net_trp_clusters[
  net_trp_clusters[, "source_cluster"] == 1 |
    net_trp_clusters[, "target_cluster"] == 1,
]

p_cluster1 <- viewNetworkR(
  network_table = subnet_cluster1,
  category      = TRUE,
  name          = TRUE,
  layout        = "fr",
  node_size     = 3,
  network_title = "Cluster 1 subnetwork in example network"
)
```

```
print(p_cluster1)

## Not run:
## Requires a running Cytoscape session for network layout and rendering
## 4) Recommended: full cluster-aware visualisation in Cytoscape
viewClusterCy(
  cluster_matrix    = net_trp_clusters,
  label            = "all",   # convert KEGG IDs to biological names and label all nodes
  category          = TRUE,    # shape by node type (Compound / KO / Contextual)
  network_title     = "Example network clusters",
  collection_title  = "MetaPathNet_Examples"
)

## End(Not run)
```

`combineNetworks` *Merge Multiple MetaPathNet-Style Networks*

Description

Merges two or more MetaPathNet-style edge lists into a single network. All inputs must be 3-column matrices or data.frames corresponding to `source`, `target`, and `interaction_type`.

Usage

```
combineNetworks(...)
```

Arguments

...	Two or more MetaPathNet-style network tables.

Details

- All inputs must be MetaPathNet-style edge lists with at least 3 columns corresponding to `source`, `target`, and `interaction_type`.
- Only the first 3 columns of each input are used.
- Duplicated rows across input networks are removed after merging.

Value

A 3-column matrix containing the merged MetaPathNet-style edge list with duplicated rows removed.

See Also

`buildKeggNetwork`, `buildCrossSpeciesNetwork`

Examples

```
## Example 1 — Merge two example network subsets

## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Create two small example network subsets
example_network_1 <- MetaPathNet_example_network[seq_len(20), , drop = FALSE]
example_network_2 <- MetaPathNet_example_network[21:40, , drop = FALSE]

## Merge both subsets into one edge list
merged_network <- combineNetworks(example_network_1, example_network_2)

head(merged_network)
```

`convertGene` *Convert Ensembl or Entrez Gene IDs to KEGG Gene IDs and KO Terms*

Description

Converts Ensembl or Entrez gene identifiers to KEGG gene IDs and, optionally, to KEGG Orthology (KO) IDs using **mygene** and **KEGGREST**. The input gene type must be specified.

Usage

```
convertGene(genes, gene_type = c("ensembl", "entrez"), orthology = TRUE)
```

Arguments

`genes`	A character vector of gene IDs. All IDs must be of the same type, specified by `gene_type`.
`gene_type`	Type of gene ID supplied. One of `"ensembl"` or `"entrez"`.
`orthology`	Logical; if `TRUE` (default), KEGG gene IDs are further mapped to KO IDs.

Details

This function supports integration of commonly used gene identifier systems (Ensembl/Entrez) into KO-based and KEGG network workflows.

The returned data frame keeps all input IDs, including those that cannot be mapped; unmapped entries are labelled as `"unmapped"` in the corresponding columns. This is useful when building KO- or pathway-centric views of host genes, for example in tryptophan–kynurenine metabolism or other immune–metabolic axes.

- Only one gene type can be used per call; mixed Ensembl/Entrez input must be split beforehand.
- When `gene_type = "ensembl"`, Ensembl IDs are first converted to Entrez IDs using **mygene**, then mapped to KEGG gene IDs via **KEGGREST**.
- When `gene_type = "entrez"`, input IDs are checked for a numeric pattern; non-numeric entries are retained and marked as `"unmapped"`.

- KEGG gene IDs are optionally mapped to KO terms using keggLink when orthology = TRUE.
- If any step fails (no Entrez, no KEGG gene, or no KO), the corresponding field is set to "unmapped"; no rows are dropped.

Value

A data.frame with columns:

- Input_ID: original gene ID (Ensembl or Entrez).
- Entrez_ID: mapped Entrez ID, or "unmapped".
- KEGG_ID: mapped KEGG gene ID, or "unmapped".
- KO_ID: mapped KEGG Orthology ID, or "unmapped" (present only when orthology = TRUE).

See Also

findKeggIDs

Examples

```
## Example 1 — Ensembl IDs for human tryptophan-kynurenine genes
trp_kynurenine_ens <- c("ENSG00000151790", "ENSG00000131203")

convertGene(
  genes      = trp_kynurenine_ens,
  gene_type  = "ensembl",
  orthology  = TRUE
)

## Example 2 — Entrez IDs linked to downstream MetaPathNet network use
## Human enzymes TDO2 (6999) and IDO1 (3620), plus tnaA (948221; a typical microbial enzyme),
## are included here to illustrate how converted KO terms can be checked in a human network.
trp_kynurenine_genes <- c("6999", "3620", "948221")

trp_kyn_conversion <- convertGene(
  genes      = trp_kynurenine_genes,
  gene_type  = "entrez",
  orthology  = TRUE
)
```

---

enrichmentDotPlot *Create a Dot Plot of Pathway Enrichment Results*

Description

Creates a dot plot from pathway enrichment results, such as those returned by enrichPathway. Pathways are ranked by adjusted p-value and displayed according to -log10(adjusted p-value).

Usage

```
enrichmentDotPlot(
  enrichment_result,
  top_n = 10,
  alpha = 0.05,
  title = "Pathway Enrichment",
  significant_colour = "#C45745",
  nonsignificant_colour = "#D1C2C2"
)
```

Arguments

enrichment_result	A data frame containing pathway enrichment results. It must include the columns Description and p_adj.
top_n	Positive integer specifying the maximum number of pathways to display. Pathways are selected according to the smallest adjusted p-values. Default 10.
alpha	Numeric significance threshold applied to adjusted p-values. Pathways with p_adj < alpha are classified as significant. Default 0.05.
title	Character string specifying the plot title. Default "Pathway Enrichment". Use NULL to omit the title.
significant_colour	Character string specifying the colour used for significant pathways. Default "#C45745".
nonsignificant_colour	Character string specifying the colour used for nonsignificant pathways. Default "#D1C2C2".

Details

The top pathways are selected according to adjusted p-value, regardless of significance status. Therefore, nonsignificant pathways may be displayed.

Adjusted p-values equal to zero are replaced by the smallest positive adjusted p-value in the enrichment results before the -log10 transformation. If all adjusted p-values are zero, .Machine$double.eps is used.

Rows with missing pathway descriptions or adjusted p-values are excluded.

Value

A ggplot object that can be further customised using **ggplot2**.

See Also

enrichPathway

Examples

```
enrichment_example <- data.frame(
  Description = c(
    "Tryptophan metabolism",
    "Phenylalanine metabolism",
    "Tyrosine metabolism"
```

```
  ),
  p_adj = c(0.001, 0.02, 0.15)
)

enrichmentDotPlot(
  enrichment_result = enrichment_example,
  top_n = 3
)

## Use custom colours
enrichmentDotPlot(
  enrichment_result = enrichment_example,
  significant_colour = "#B2182B",
  nonsignificant_colour = "#BDBDBD"
)
```

`enrichPathway` *Pathway Enrichment Analysis on KEGG-Based Node Sets*

Description

Performs pathway over-representation analysis using the hypergeometric test with Benjamini–Hochberg correction on nodes from KEGG-derived networks. Supports KO-level, compound-level, and integrated KO–compound analyses.

Usage

```
enrichPathway(
  test_set,
  background_set,
  pathway_set,
  entity = c("ko", "compound", "integrated"),
  pvalueCutoff = 0.05,
  add_hits = FALSE
)
```

Arguments

`test_set`	Character vector of node identifiers, such as KEGG compound or KO identifiers. A matrix or data frame may also be supplied, in which case unique nodes from the first two columns are used.
`background_set`	Character vector of node identifiers defining the background universe. A network matrix or data frame may also be supplied.
`pathway_set`	Either a character vector of KEGG reference pathway identifiers or a named list in which each element contains the node members of one pathway.
`entity`	Character string specifying the entity type to analyse: `"ko"` for KOs, `"compound"` for compounds, or `"integrated"` for both. Default `"ko"`.
`pvalueCutoff`	Numeric significance threshold applied to adjusted p-values after Benjamini–Hochberg correction. Default `0.05`.
`add_hits`	Logical. If `TRUE`, adds a `Test_Hits` column containing the test nodes found in each pathway. Default `FALSE`.

Details

If `test_set` or `background_set` is supplied as a matrix or data frame, unique nodes are extracted from the first two columns.

Before enrichment analysis, the test set is restricted to nodes present in the background set. Enrichment is then calculated using the hypergeometric distribution with `phyper`, followed by Benjamini–Hochberg adjustment using `p.adjust`.

Pathways with no test-set overlap are excluded from the returned results.

Value

A data frame containing pathway enrichment results with columns:

Pathway_ID KEGG pathway identifier, such as `"map00380"`.

Description Pathway description retrieved from KEGG.

bg_overlap Number of background nodes belonging to the pathway.

test_overlap Number of test nodes belonging to the pathway.

Test_Hits Optional semicolon-separated test nodes belonging to the pathway. Included only when `add_hits = TRUE`.

p_value Raw hypergeometric p-value.

p_adj Adjusted p-value obtained using the Benjamini–Hochberg method.

Enrichment_Ratio Ratio between the test-set pathway proportion and the background pathway proportion.

significance `"significant"` or `"NS"`, determined using `pvalueCutoff`.

The returned data frame can be visualised using `enrichmentDotPlot`.

Note

Very small test sets, or test sets that are nearly identical to the background set, may produce uninformative enrichment results.

See Also

`enrichmentDotPlot`, `getPathIDs`

Examples

```
## Load the precomputed example network
## See buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Define a small test set from the example network
test_nodes <- c(
  "K00463",
  "K00453",
  "cpd:C00078",
  "cpd:C02700"
)

## Run integrated pathway over-representation analysis
enrich_res <- enrichPathway(
  test_set = test_nodes,
```

```
  background_set = MetaPathNet_example_network,
  pathway_set = c("map00380", "map00400"),
  entity = "integrated",
  pvalueCutoff = 1,
  add_hits = TRUE
)

## Preview the enrichment results
head(enrich_res)

## Visualise the enrichment results
if (nrow(enrich_res) > 0) {
  enrichmentDotPlot(enrich_res)
}
```

`findKeggIDs` *Find KEGG Identifiers for Compounds, Organisms, and KO Terms*

Description

Map user-supplied compound, organism, or KO queries to KEGG identifiers.

Usage

```
findKeggIDs(
  KEGG_database = c("compound", "organism", "ko"),
  searchBy = NULL,
  query = NULL
)
```

Arguments

`KEGG_database`	Character string specifying the KEGG entry type: `"compound"`, `"organism"`, or `"ko"`.
`searchBy`	Character string specifying the query type. For `KEGG_database = "compound"`: one of `"name"`, `"sid"`, `"cid"`, `"SMILES"`, `"InChI"`, or `"InChIKey"`. For `KEGG_database = "organism"`: one of `"name"` or `"taxon_id"`, where `"taxon_id"` refers to the KEGG genome T number. For `KEGG_database = "ko"`: one of `"name"`, `"symbol"`, or `"ECnumber"`.
`query`	Vector of queries to be matched. Non-character input is coerced to `character`.

Details

For compounds, name queries are first matched against the live KEGG compound table. Unresolved names and structure-based inputs are queried through PubChem, and supported PubChem identifiers are converted to KEGG compound IDs when possible.

For organisms, matching is performed against the live KEGG genome table. Scientific and common names are matched exactly without regard to case.

For KO terms, the current KEGG KO table is retrieved through the KEGG REST API. Symbol queries may return multiple KO IDs combined with `"/"`.

Value

A `data.frame` with the input values and matched KEGG identifiers:

- for `KEGG_database = "compound"`: columns `Input`, `KEGG_ID`;
- for `KEGG_database = "organism"`: columns `Input`, `Organism_Code`;
- for `KEGG_database = "ko"`: columns `Input`, `KO_ID`.

If no match is found, the corresponding identifier is `NA`. Some inputs may return multiple rows when multiple matches are found.

Examples

```
## 1) Map compounds to KEGG compound IDs
findKeggIDs(
  KEGG_database = "compound",
  searchBy      = "name",
  query         = c("tryptophan", "indole", "serotonin")
)

## 2) Map organisms to KEGG organism codes by name
## This example requires live KEGG genome access.

findKeggIDs(
  KEGG_database = "organism",
  searchBy      = "name",
  query         = c("human", "horse", "dragon", "Escherichia coli W")
)


## 3) Map enzyme EC numbers to KEGG KO identifiers
findKeggIDs(
  KEGG_database = "ko",
  searchBy      = "ECnumber",
  query         = c("4.1.99.1", "1.13.11.27")
)
```

`findMappedNodes` *Map User-Supplied Nodes to a Network*

Description

Check which nodes from a user-supplied vector are present in a given network edge list and which are absent.

Usage

```
findMappedNodes(nodes, network_table)
```

Arguments

`nodes`	Character vector of node IDs (e.g. KEGG compounds, KO IDs, reaction IDs) to check.
`network_table`	Matrix or data frame (2 or 3 columns) representing a network, typically generated by `buildKeggNetwork` or related functions.

Details

This function is a small utility to verify that user-specified compounds, genes/KOs, or reactions are present in a network before running downstream analyses (e.g. distance calculations, shortest paths, ego networks).

Value

A list with two components:

mapped_nodes Character vector of nodes found in the network.

unmapped_nodes Character vector of nodes not found in the network.

Examples

```
## Example — Check presence of Trp-pathway nodes in a MetaPathNet example network

## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Define a small set of candidate nodes (compounds + KO + fake ID)
candidate_nodes <- c(
  "C00078",       # tryptophan      (will be normalised to "cpd:C00078")
  "cpd:C00328",   # L-kynurenine
  "K00463",       # tryptophan synthase beta chain
  "NotID"         # fake node, should be unmapped
)

## Check which of these nodes are present in the network
res <- findMappedNodes(
  nodes         = candidate_nodes,
  network_table = MetaPathNet_example_network
)

res$mapped_nodes    # present in MetaPathNet_example_network
res$unmapped_nodes  # absent from MetaPathNet_example_network
```

---

findShortestPaths *Identify Shortest Paths Between Node Sets in a KEGG-Derived Network*

Description

Computes shortest paths between specified source and target node sets in a KEGG-based network. Source nodes are often KEGG KOs/genes (upstream biological functions) and target nodes are often KEGG compounds (downstream metabolites), but this is a recommended usage pattern rather than a strict requirement.

Usage

```
findShortestPaths(
  network_table,
  source_nodes,
  target_nodes,
  mode = c("out", "in", "all"),
  output = c("path_list", "network_matrix"),
  name = FALSE,
  distance_threshold = Inf,
  betweenness = FALSE,
  reactant_filter = FALSE
)
```

Arguments

network_table	A 3-column matrix or data.frame of edges with columns source, target, interaction_type, typically from buildKeggNetwork() or buildCrossSpeciesNetwork().
source_nodes	Character vector of KEGG node IDs to use as path sources.
target_nodes	Character vector of KEGG node IDs to use as path targets.
mode	Character; direction mode passed to igraph shortest-path and distance functions. One of: "out" Shortest-path search follows edge direction. "in" Shortest-path search is computed against edge direction (reverse direction). "all" Ignore edge direction during shortest-path search (graph treated as undirected). For output = "network_matrix", interaction types are still recovered from the original network edge annotation.
output	Character; output format: • "path_list" - list of shortest paths per source-target pair. • "network_matrix" - 3-column edge list (source, target, interaction_type).
name	Logical; if TRUE and output = "path_list", node IDs are converted to KEGG names or symbols via **KEGGREST**.
distance_threshold	Numeric; only source-target pairs with shortest-path distance $\leq$ this value are processed (default Inf).
betweenness	Logical (default FALSE). If TRUE, when more than one shortest path of the same length exists for a given source-target pair, a single path is selected by ranking candidate shortest paths by node betweenness.
reactant_filter	Logical (default FALSE). If TRUE, shortest paths are filtered to keep only paths whose consecutive compound-compound transitions are supported by KEGG reaction data. This step issues repeated KEGG API queries and may be slow for large path sets.

Details

Valid source-target pairs are first filtered by graph distance, and paths can be returned either as path lists or as a condensed edge list.

The function:

- Accepts `matrix` or `data.frame` input and internally standardizes it to a character edge table.
- Builds a directed **igraph** graph from `network_table` and computes pairwise shortest-path distances (breadth-first search on this unweighted graph).
- Stops if any node is present in both `source_nodes` and `target_nodes`, because the distance from a node to itself is 0 and is not allowed in this workflow.
- Keeps only mapped source-target pairs with finite distance within `distance_threshold`; unreachable pairs (`Inf`) and pairs longer than the threshold are filtered out and not processed further.
- If `mode = "all"`, shortest-path search ignores edge direction. However, for `output = "network_matrix"`, the returned `interaction_type` values are still recovered from the original directed edge annotation in `network_table`, including reverse matching when needed.
- If `reactant_filter = TRUE`, candidate shortest paths are further filtered by checking whether consecutive compound-compound transitions are supported by KEGG reaction data.

Value

Depending on `output`:

- `"path_list"` - named list; each element is one or more shortest paths (character vectors of node IDs or names).
- `"network_matrix"` - 3-column matrix of unique edges traversed by the identified shortest paths. Edges are returned in path order, and `interaction_type` is assigned from the original network edge annotation.

Note

Name/symbol conversion relies on the KEGG REST API and can be slow for large node sets. The internal 6-step cutoff is empirical and intended to prevent excessive enumeration of very long shortest paths. When `betweenness = TRUE`, enumerating all shortest paths for tie-breaking may increase runtime and memory use.

Examples

```
## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)
## Use the precomputed object as the example network_table
net_trp_mixed <- MetaPathNet_example_network

## 2) Define source and target nodes of interest in this network
## Source/target can be any KEGG node types, but they must be mapped in the network
## and should not overlap (same node cannot be both source and target).
source_nodes <- c("K00486", "K00453", "K00463",
                  "K01667", "K01696", "K01610", "K01695")
target_nodes <- c("cpd:C00078", "cpd:C00328", "cpd:C00780",
                  "cpd:C00463", "cpd:C00108", "cpd:C00458", "cpd:C00336")

map_src <- findMappedNodes(source_nodes, net_trp_mixed)
map_tgt <- findMappedNodes(target_nodes, net_trp_mixed)

## 3) Optional: inspect shortest-path distances between mapped node sets
## (useful for checking connectivity before extracting shortest paths)
dist_trp <- networkDistances(
  network_table = net_trp_mixed,
```

```
  source_nodes  = map_src$mapped_nodes,
  target_nodes  = map_tgt$mapped_nodes,
  name          = FALSE
)

## 4) Extract shortest paths as annotated path list
## Pairs that are unreachable (Inf distance) or above the distance threshold
## are filtered internally.
paths_trp_mixed_list <- findShortestPaths(
  network_table      = net_trp_mixed,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "path_list",
  name               = FALSE,
  distance_threshold = 6
)

## 5) Extract a subnetwork containing all edges used in the selected shortest paths
paths_trp_mixed <- findShortestPaths(
  network_table      = net_trp_mixed,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "network_matrix",
  name               = FALSE
)
```

`getPathIDs` *Retrieve and Classify Organism-Specific KEGG Pathways*

Description

Retrieves KEGG pathway IDs for one or more organisms and assigns each to a functional category and type based on KEGG BRITE hierarchy.

Usage

```
getPathIDs(organism_code)
```

Arguments

`organism_code`	Character vector of KEGG organism codes.

Details

Pathway classification uses the KEGG BRITE hierarchy `br08901`. Pathways under the top-level category `"Metabolism"` are typed as `"metabolic"`; all others are typed as `"signaling"`. Global and overview maps are excluded from the output.

Value

A `data.frame` with columns:

Path_id Full KEGG pathway ID.

Path_description Full KEGG pathway name.

Path_category Top-level and subcategory from KEGG BRITE hierarchy.

Path_type `"metabolic"` or `"signaling"`, based on top-level category.

Species KEGG organism code.

See Also

`buildKeggNetwork` for building networks from these pathway sets.

Examples

```
## 1) Retrieve all human pathways and filter for tryptophan metabolism
path_hsa <- getPathIDs("hsa")
trp_hsa <- subset(path_hsa, grepl("Tryptophan metabolism", Path_description))
head(trp_hsa)

## 2) Retrieve pathways for multiple organisms
org_vec <- c("hsa", "eco", "bsu")
path_multi <- getPathIDs(org_vec)

## 3) Show pathways shared across the selected organisms
##    (appear once per species, so shared ones have count == length(org_vec))
shared_counts <- table(path_multi$Path_description)
head(shared_counts[shared_counts == length(org_vec)])

## 4) Filter tryptophan metabolism pathways and show which species contain them
trp_multi <- subset(path_multi, grepl("Tryptophan metabolism", Path_description))
head(trp_multi)
table(trp_multi$Species)
```

`mapReaction` *Map reaction IDs to MetaPathNet-style networks*

Description

`mapReaction()` converts reaction identifiers into the standard MetaPathNet 3-column edge list format (`source`, `target`, `interaction_type`).

Usage

```
mapReaction(reaction_ids, source = c("rhea", "metacyc", "kegg"))
```

Arguments

reaction_ids	Character vector of reaction identifiers.
source	Character string indicating the origin of the reaction IDs. Supported values are: • "rhea" - Rhea reaction IDs (digits only, no prefix). • "metacyc" - MetaCyc reaction IDs. • "kegg" - KEGG reaction IDs (e.g. "R01302" or "rn:R01302").

Details

The function supports direct KEGG reaction IDs, as well as external reaction identifiers from Rhea and MetaCyc. For external sources, it uses the MetaNetX SPARQL endpoint as a neutral layer to translate reaction IDs into KEGG reaction IDs where possible. KEGG-mapped reactions are then converted into MetaPathNet-style KO–compound edges. If no KEGG mapping is available, MetaNetX reaction equations are used to generate fallback compound-to-compound edges.

For source = "kegg", reaction IDs are converted directly using KEGG reaction entries. Reaction equations are parsed to infer substrates, products, and directionality. KEGG Orthology (KO) identifiers are retrieved preferentially from the reaction ORTHOLOGY field, and if unavailable, the function falls back to reaction-to-enzyme and enzyme-to-KO mapping.

For source = "rhea" and source = "metacyc", the function first queries MetaNetX to identify corresponding MetaNetX reactions, then attempts to recover KEGG reaction IDs. KEGG-mapped reactions follow the same semantics as the metabolic layer of MetaPathNet: compounds are coded as "cpd:Cxxxxx", KOs as "Kxxxxx", and directionality (reversible vs irreversible) is derived from the KEGG EQUATION field.

Reactions without KEGG mapping are represented as compound-to-compound relations, using human-readable compound names as node IDs and "external" as interaction type. This preserves reaction information while clearly marking the non-KEGG origin of these edges.

Value

A character matrix with three columns:

source Source node ID (compound, KO, or compound name for non-KEGG reactions).

target Target node ID.

interaction_type For KEGG-mapped reactions, values such as "k_compound:reversible" or "k_compound:irreversible"; for reactions without KEGG mapping, "external".

If no valid reactions can be converted, an empty 3-column matrix is returned and a warning is issued.

See Also

buildKeggNetwork, buildCrossSpeciesNetwork

Examples

```
## NOTE: The following examples require live access to KEGG REST and/or the
## MetaNetX SPARQL endpoint.

## 1) Map a MetaCyc reaction and inspect the returned MetaPathNet-style edges
trp_metacyc_edges <- mapReaction(
  reaction_ids = "TRYPTOPHAN-RXN",
  source       = "metacyc"
```

```
)

## 2) Map a Rhea reaction
rhea16505_edges <- mapReaction(
  reaction_ids = "16505",
  source       = "rhea"
)

## 3) Map KEGG reaction IDs directly
tma_edges <- mapReaction(
  reaction_ids = c("R05623", "R10285"),
  source       = "kegg"
)
```

`MetaPathNet_example_network` *Example tryptophan-related MetaPathNet network*

Description

A precomputed host-microbe example network constructed from human and Escherichia coli KEGG pathways related to tryptophan metabolism and immune/signaling context. This dataset is intended for runnable examples, tests, and package demonstrations without requiring live KEGG queries.

Usage

```
data(MetaPathNet_example_network)
```

Format

A character matrix with 1769 rows and 3 columns:

source Source node identifier.

target Target node identifier.

interaction_type Interaction type between source and target.

Value

A character matrix containing source nodes, target nodes, and interaction types.

Source

Constructed using `buildKeggNetwork()` and `combineNetworks()` from KEGG pathways `hsa00380`, `hsa04060`, `hsa04630`, `hsa04064`, `hsa04660`, `hsa04659`, `eco00380`, and `eco00400`.

networkDistances *Calculate Shortest-Path Distances Between Node Sets in a KEGG-Based Network*

Description

Compute shortest-path distances between selected source and target nodes in a KEGG-based network produced by `buildKeggNetwork()` (or related functions). Source nodes are often KEGG KOs (upstream biological functions) and target nodes are often KEGG compounds (downstream metabolites), but this is a recommended usage pattern rather than a strict requirement.

Usage

```
networkDistances(
  network_table,
  mode = c("out", "in", "all"),
  source_nodes,
  target_nodes,
  name = FALSE
)
```

Arguments

`network_table`	A 3-column matrix or data.frame of edges (`source`, `target`, `interaction_type`), typically produced by `buildKeggNetwork()`. Entries should use KEGG-style node identifiers.
`mode`	Character, passed to `igraph::distances()`. Default `"out"`. One of: `"out"` Distances follow edge direction. `"in"` Distances are computed against edge direction (reverse direction). `"all"` Distances ignore edge direction (treat the graph as undirected).
`source_nodes`	Character vector of source nodes. Use `"all"` to include all non-compound nodes if `target_nodes = "all"`. Otherwise, `"all"` includes all nodes present in the network.
`target_nodes`	Character vector of target nodes. Use `"all"` to include all compound-like nodes (`cpd:`, `dr:`, `gl:`) if `source_nodes = "all"`. Otherwise, `"all"` includes all nodes present in the network.
`name`	Logical (default `FALSE`). If `TRUE`, row and column labels are converted to KEGG names via `KEGGREST::keggGet()`.

Details

The function builds a directed **igraph** object from the first two columns of `network_table` and computes pairwise shortest-path distances using `igraph::distances()`. For unweighted networks such as MetaPathNet edge lists, distance lengths are computed with a breadth-first search strategy.

Compound-like node IDs supplied as `"Cxxxxx"` or `"cpd:Cxxxxx"` are treated equivalently and internally harmonised to `"cpd:Cxxxxx"`.

If both `source_nodes = "all"` and `target_nodes = "all"`, the function automatically assigns non-compound nodes (e.g. KOs/genes) as sources and compound-like nodes (`cpd:`, `dr:`, `gl:`) as targets.

If any node is present in both the mapped source and target sets, the function stops and reports the overlapping node(s), because distance from a node to itself is 0 and is not allowed in this workflow.

Distances equal to `Inf` indicate unreachable source-target pairs under the selected `mode`, i.e. no valid path exists between those nodes in the current network direction setting.

If `name = TRUE`, `KEGGREST::keggGet()` is used to retrieve KEGG names. This may be slow for large node sets.

Value

A numeric matrix of shortest-path distances. Rows correspond to source nodes and columns to target nodes.

See Also

`buildKeggNetwork`, `distances`

Examples

```
## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## 2) Define node sets of interest in this network
##   - source_nodes: KOs representing enzymes/genes of interest in Trp/kynurenine biology
##   - target_nodes: KEGG compounds representing metabolites of interest in the same context
##   Note: networkDistances can accept any node types, but source/target must be disjoint:
##         if a node is in both sets, the function stops because self-distance is 0.
source_nodes <- c("K00486", "K00453", "K00463", "K09093", "K01667")
target_nodes <- c("cpd:C00078", "cpd:C00328", "cpd:C00780", "cpd:C00463")

## Keep only nodes that are present in the network (avoid "not mapped" errors)
map_src <- findMappedNodes(source_nodes, MetaPathNet_example_network)
map_tgt <- findMappedNodes(target_nodes, MetaPathNet_example_network)

## 3) Compute the shortest-path distance matrix
dist_trp <- networkDistances(
  network_table = MetaPathNet_example_network,
  source_nodes  = map_src$mapped_nodes,
  target_nodes  = map_tgt$mapped_nodes,
  name          = FALSE
)

dist_trp
```

---

networkSimilarity *Compare KEGG-Based Networks via Jaccard Similarity*

Description

Computes pairwise Jaccard similarity between two or more KEGG-derived networks, either at the node level (unique node sets) or edge level (directed source–target pairs, optionally including interaction type).

Usage

```
networkSimilarity(..., type = c("edge", "node"), match_interaction = FALSE)
```

Arguments

`...`	Two or more network matrices, typically generated by `buildKeggNetwork()`. Each must have at least two columns: `source` (column 1) and `target` (column 2). When `match_interaction = TRUE`, a third column `interaction_type` is required.
`type`	Character, `"edge"` or `"node"`. If `"node"`, similarity is computed on the set of unique nodes (union of sources and targets). If `"edge"`, similarity is computed on directed edges (source → target).
`match_interaction`	Logical. When `type = "edge"` and `match_interaction = FALSE` (default), edges are defined by `source -> target` only. If `TRUE`, edges must also share the same `interaction_type` to be considered identical. Ignored when `type = "node"`.

Details

This function is particularly useful for comparing pathways across organisms or conditions using networks built with `buildKeggNetwork()`.

Value

A numeric similarity matrix with dimensions `length(list(...))` x `length(list(...))`. Row and column names are derived from the argument names in `...`. Diagonal entries are 1; off-diagonal entries are Jaccard indices,

$$J = |A \cap B| / |A \cup B|,$$

where A and B are node sets (for `type = "node"`) or edge sets (for `type = "edge"`). If both sets are empty, the similarity is `NA_real_`.

References

Fuxman Bass JI, Diallo A, Nelson J, Soto JM, Myers CL, Walhout AJ. Using networks to measure similarity between genes: association index selection. Nat Methods. 2013;10(12):1169–1176. doi:10.1038/nmeth.2728.

See Also

`buildKeggNetwork` for network construction, `getPathIDs` for pathway ID retrieval.

Examples

```
## 1) Prepare example network subsets
data(MetaPathNet_example_network)

net_example_1 <- MetaPathNet_example_network[seq_len(50), , drop = FALSE]
net_example_2 <- MetaPathNet_example_network[51:100, , drop = FALSE]
net_example_3 <- MetaPathNet_example_network[101:150, , drop = FALSE]
net_example_4 <- MetaPathNet_example_network[151:200, , drop = FALSE]
net_example_5 <- MetaPathNet_example_network[201:250, , drop = FALSE]

## 2) Edge-level similarity (Jaccard on directed edges, ignoring interaction type)
```

```
##    This compares overlap in network structure (source-target pairs).
sim_edge_example <- networkSimilarity(
  net_example_1,
  net_example_2,
  net_example_3,
  net_example_4,
  net_example_5,
  type              = "edge",
  match_interaction = FALSE
)

## 3) (Optional) Node-level similarity (Jaccard on unique node sets)
##     This compares overlap in node composition, regardless of how nodes are connected.
sim_node_example <- networkSimilarity(
  net_example_1,
  net_example_2,
  net_example_3,
  net_example_4,
  net_example_5,
  type              = "node",
  match_interaction = FALSE
)
```

`network_utils` *Network Utility Functions for Edge-List Manipulation*

Description

A small set of helpers to inspect and edit network edge lists produced by `buildKeggNetwork()`, or related functions. These utilities make it easier to:

- check whether user-supplied nodes are present in a network,
- merge equivalent nodes under a single identifier,
- remove selected nodes (and all their incident edges),
- strip KEGG drug nodes (`"dr:"`) from a network.

Details

Each function has its own documentation and examples:

- `findMappedNodes` — test which nodes are mapped or unmapped.
- `replaceNode` — replace specified nodes with a new ID.
- `removeNetworkNodes` — remove specified nodes and their edges.
- `removeDrugs` — remove all KEGG drug nodes (`"dr:"`).

Value

Internal utility functions used by MetaPathNet.

See Also

`buildKeggNetwork`, `buildEgoNetwork`

`permutePaths` *Permutation Test of Shortest-Path Structure Between Node Sets*

Description

Performs a permutation-based test on the mean shortest-path length between a set of source nodes and target nodes within a KEGG-derived network. The observed mean shortest-path distance is compared to a null distribution obtained by repeatedly sampling random source and target sets of the same size and node-type composition from the same network.

Usage

```
permutePaths(
  network_table,
  source_nodes,
  target_nodes,
  n_perm = 10000,
  mode = c("out", "in", "all"),
  plot = FALSE,
  return_perm = FALSE
)
```

Arguments

`network_table`	A 3-column matrix or data.frame representing directed network edges. Columns must correspond to `source`, `target`, and `interaction_type`.
`source_nodes`	Character vector of KEGG node IDs used as *sources* in the permutation test.
`target_nodes`	Character vector of KEGG node IDs used as *targets* in the permutation test.
`n_perm`	Integer. Number of permutations used to build the null distribution of mean shortest-path lengths (default: `10000`).
`mode`	Character. Direction parameter passed to `igraph::distances()`. One of: `"out"` Distances follow edge direction. `"in"` Distances are computed against edge direction (reverse direction). `"all"` Distances ignore edge direction (graph treated as undirected).
`plot`	Logical (default: `FALSE`). If `TRUE`, a histogram and density curve of permuted mean shortest-path lengths is plotted, with a vertical line for the observed mean and an annotated p-value.
`return_perm`	Logical (default: `FALSE`). If `FALSE`, only summary statistics are returned. If `TRUE`, the raw permutation means are returned in addition to the summary.

Details

Internally, the function:

- identifies supported node types in `network_table`, currently KEGG Orthology nodes (`"K"` prefix) and KEGG compound nodes (`"cpd:"` prefix);
- maps the user-supplied source and target nodes onto the network;
- removes unreachable pairs (distance = `Inf`) when calculating observed and permuted mean distances;

- derives an empirical one-sided p-value from the proportion of permuted means that are smaller than the observed mean.

Source and target node sets are user-defined and may contain supported KEGG node types according to the biological question being addressed. A smaller mean shortest-path length indicates tighter connectivity. Therefore, the reported p-value tests whether random source-target sets tend to be *more tightly connected* than the observed set ($P(\text{random mean} < \text{observed mean})$).

Value

If `return_perm = FALSE`, a data.frame with one row:

observed_sp Observed mean shortest-path length (finite distances only; unreachable pairs are excluded).

mean_perm_sp Mean of permuted mean shortest-path lengths under the null (finite values only).

p_value Empirical one-sided p-value $P(\text{random mean} < \text{observed mean})$.

n_perm Number of permutations used.

If `return_perm = TRUE`, a list with:

stats The summary data.frame described above.

random_sp Numeric vector of permuted mean shortest-path lengths (finite values only).

Plotting

If `plot = TRUE`, a **ggplot2**-based histogram + density overlay of the permutation distribution is printed. For custom visualisation, set `return_perm = TRUE` and use the `random_sp` vector together with `stats` in your own plotting code.

See Also

`networkDistances` for shortest-path distance matrices, `findShortestPaths` for explicit shortest paths, `buildKeggNetwork`, `buildCrossSpeciesNetwork` for KEGG-based network construction.

Examples

```
## 1) Prepare a mixed human-E. coli network focused on tryptophan-related pathways
## Human: tryptophan metabolism + selected signaling pathways
## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Combine host and microbe networks into one edge list
net_trp_mixed <- MetaPathNet_example_network

## 2) Define source and target node sets for the Trp/kynurenine/serotonin context
source_nodes <- c(
  "K00486",  # kynurenine 3-monooxygenase
  "K00453",  # tryptophan 2,3-dioxygenase
  "K00463",  # indoleamine 2,3-dioxygenase
  "K01667",  # kynureninase
  "K01696",  # tryptophan synthase beta chain
  "K01695"   # tryptophan synthase alpha chain
)

target_nodes <- c(
```

```
  "cpd:C00078",  # L-tryptophan
  "cpd:C00328",  # L-kynurenine
  "cpd:C00780",  # serotonin
  "cpd:C00463",  # indole
  "cpd:C00108"   # anthranilate
)

## 3) Permutation test on mean shortest-path length between the two node sets
perm_trp <- permutePaths(
  network_table = net_trp_mixed, # mixed hsa/eco Trp + immune-related network
  source_nodes  = source_nodes,
  target_nodes  = target_nodes,
  n_perm        = 100,           # increase (e.g. 10000) for more stable inference
  mode          = "out",
  plot          = FALSE,         # show permutation distribution
  return_perm   = TRUE           # keep individual permuted means
)

## 4) View summary statistics
perm_trp$stats
```

`removeDrugs` *Remove Drug Nodes from a Network Matrix*

Description

Removes all edges from a network where either endpoint is annotated as a KEGG drug (prefix `"dr:"`).

Usage

```
removeDrugs(network_table)
```

Arguments

`network_table`	A two- or three-column matrix representing the network (edge list), typically as returned by `buildKeggNetwork()` or related functions.

Details

The function scans the first two columns (source and target) for node IDs containing the `"dr:"` prefix. All rows involving such nodes are removed. If no drug nodes are found, the original matrix is returned and a message is printed.

Value

A matrix of the same format as the input, with all drug nodes (prefix `"dr:"`) and their incident edges removed.

Examples

```
## 1) Prepare an example network containing KEGG drug nodes
data(MetaPathNet_example_network)

net_hsa <- rbind(
  MetaPathNet_example_network,
  c("dr:D00001", "K00463", "drug_example"),
  c("K00463", "dr:D00002", "drug_example")
)

## 2) Remove all KEGG drug nodes (dr:...) from the network
net_hsa_nodrug <- removeDrugs(net_hsa)

## 3) Compare dimensions before and after drug removal
dim(net_hsa)         # Original network dimensions
dim(net_hsa_nodrug)  # Network dimensions after drug removal
nrow(net_hsa) - nrow(net_hsa_nodrug)  # Number of drug-related edges removed
```

---

removeNetworkNodes *Remove Specified Nodes from a Network Matrix*

Description

Removes one or more nodes (compounds, KOs, reactions, etc.) from a network edge list. All edges involving any of the specified nodes are removed.

Usage

```
removeNetworkNodes(nodes_to_remove, network_table)
```

Arguments

nodes_to_remove	Character vector of node IDs to remove from the network (e.g. KEGG compound IDs, KO IDs). For compounds, both "Cxxxxx" and "cpd:Cxxxxx" formats are accepted; internally they are normalised to "cpd:Cxxxxx".
network_table	A matrix (typically 2 or 3 columns) representing the network edge list, as generated by buildKeggNetwork() or related functions.

Details

All occurrences of the supplied node IDs are set to NA, and all rows containing NA are then removed. This is useful to prune unwanted nodes before downstream analyses such as distance or shortest-path calculations.

Value

A matrix of the same format as network_table, with all edges involving the specified nodes removed and duplicate edges dropped.

Examples

```
## Example – Remove a selected node from a human integrated network

## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)
net_hsa <- MetaPathNet_example_network

## 2) Define node to remove
nodes_to_remove <- "cpd:C01213"   # (R)-Methylmalonyl-CoA

## 3) Remove all edges involving the selected node
net_hsa_filtered <- removeNetworkNodes(
  nodes_to_remove = nodes_to_remove,
  network_table   = net_hsa
)

## 4) Check that the node is no longer present
findMappedNodes(
  nodes         = nodes_to_remove,
  network_table = net_hsa_filtered
)$mapped_nodes

## 5) Compare network size before and after node removal
dim(net_hsa)           # Original network dimensions
dim(net_hsa_filtered)  # Network dimensions after node removal
nrow(net_hsa) - nrow(net_hsa_filtered)  # Number of removed edges
```

---

replaceNode *Replace Nodes in a Network Table*

Description

Replace one or more node IDs in a network matrix by a single new ID. This is typically used to cluster equivalent metabolites (e.g. isomers) under a common identifier before downstream analysis.

Usage

```
replaceNode(nodes_to_replace, replacement_node, network_table)
```

Arguments

`nodes_to_replace`	Character vector of node IDs to be replaced. Compound IDs can be given as `"Cxxxxx"` or `"cpd:Cxxxxx"`; both are accepted.
`replacement_node`	Character string giving the new node ID to use as replacement (same flexibility as `nodes_to_replace` for compound IDs).
`network_table`	A 2- or 3-column matrix representing the network (e.g. as returned by `buildKeggNetwork()`).

Details

All matches are performed on KEGG-style node labels. For KEGG compounds, the function internally normalises `"Cxxxxx"` and `"cpd:Cxxxxx"` to a consistent `"cpd:Cxxxxx"` form before replacement.

Value

A network matrix of the same structure as `network_table`, with all occurrences of `nodes_to_replace` replaced by `replacement_node`. Duplicate edges created by the replacement are removed.

See Also

`findMappedNodes`

Examples

```
## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)
net_hsa <- MetaPathNet_example_network

## 2) (R)- and (S)-methylmalonyl-CoA are stereoisomers and can be merged
##    into a single node
nodes_to_merge   <- c("cpd:C01213", "cpd:C00683")
replacement_node <- "cpd:C00683"

## 3) Replace the selected nodes with one common identifier
net_hsa_clustered <- replaceNode(
  nodes_to_replace = nodes_to_merge,
  replacement_node = replacement_node,
  network_table    = net_hsa
)

## 4) Check that the replacement was applied in the updated network
findMappedNodes(
  nodes         = c("cpd:C01213", "cpd:C00683"),
  network_table = net_hsa_clustered
)
```

---

suggestEntities *Suggest Candidate Prokaryotic Organisms or KOs from KEGG Identifiers*

Description

Suggest candidate prokaryotic organisms from KEGG KO IDs or candidate KEGG Orthology (KO) annotations from KEGG compound IDs.

Usage

```
suggestEntities(query, entity = c("ko", "compound"))
```

Arguments

query	Character vector of KEGG identifiers. For entity = "ko", KO IDs should follow the form "K#####". For entity = "compound", compound IDs should follow the form "C#####" or "cpd:C#####".
entity	Character string specifying the input type: "ko" or "compound". Only one entity type is accepted per call.

Details

The organism-suggestion workflow focuses on prokaryotic organisms to support host-microbiome network construction. *Homo sapiens* (hsa) is retained when present as the host reference.

For entity = "ko", each KO is queried with KEGGREST::keggLink() to retrieve genes carrying that orthology assignment. Organism codes are extracted from the linked KEGG gene IDs and matched to the live KEGG genome table.

Broad organism classification is reconstructed from the KEGG eukaryotic and prokaryotic genome groups. Candidate organisms are restricted to prokaryotes, while *Homo sapiens* (hsa) is retained as a host reference. Organisms are ranked according to the number of input KOs represented in each genome.

For entity = "compound", each compound is queried with KEGGREST::keggGet(). Linked KEGG reaction IDs are extracted, and the ORTHOLOGY field of each reaction is used to retrieve associated candidate KOs.

Input IDs are validated before querying KEGG. If none of the supplied IDs has a valid format, the function stops with an error.

Value

A data.frame whose structure depends on entity:

For entity = "ko"**:** Columns organism_name, organism_code, overlap_count, and overlap_hits. Candidate prokaryotic organisms and hsa, when present, are ranked by decreasing overlap_count.

For entity = "compound"**:** Columns compound_id, compound_name, KO_ID, and KO_name. Duplicate rows are removed.

An empty data frame with the appropriate columns is returned when no candidates are found or when the required KEGG genome classification cannot be retrieved.

See Also

findKeggIDs for converting compound names, organism names, or enzyme EC numbers to KEGG identifiers.

Examples

```
## 1) Retrieve candidate KOs associated with selected compounds
cpd_ids <- findKeggIDs(
  KEGG_database = "compound",
  searchBy      = "name",
  query         = c("tryptophan", "indole", "serotonin")
)

ko_results <- suggestEntities(
  query  = cpd_ids$KEGG_ID,
```

```
  entity = "compound"
)

head(ko_results)

## 2) Retrieve candidate prokaryotic organisms from selected KOs
org_results <- suggestEntities(
  query  = c("K18277", "K07811", "K00108", "K14156"),
  entity = "ko"
)

head(org_results)
```

`viewClusterCy` *Visualise Clustered MetaPathNet Networks in Cytoscape*

Description

Sends a cluster-annotated MetaPathNet network (typically the output of `clusterNetwork()`) to Cytoscape for interactive exploration. Nodes are coloured by community (cluster), optionally shaped by category (compound / KO / contextual), and can be labelled with KEGG-derived names.

Usage

```
viewClusterCy(
  cluster_matrix,
  label = c("none", "all", "non_background"),
  category = TRUE,
  network_title = "",
  collection_title = ""
)
```

Arguments

`cluster_matrix`	A matrix or data.frame with at least five columns: `source`, `target`, `interaction_type`, `source_cluster`, and `target_cluster`. Typically the direct output of `clusterNetwork()`.
`label`	Character; controls KEGG-based node labelling. One of: • `"none"` – no labels are shown on the Cytoscape network. • `"all"` – convert KEGG IDs to biological names and label all nodes. • `"non_background"` – convert and label only nodes in non-background clusters. The background cluster is defined as the largest community and is left unlabelled to reduce KEGG REST calls and visual clutter.
`category`	Logical. If `TRUE`, node shapes are mapped to basic categories: `"Compound"`, `"KO"`, `"Contextual"`. If `FALSE`, all nodes use the same shape (ellipse).
`network_title`	Character. Cytoscape network title. If empty or `NULL`, defaults to `"cluster_network"`.
`collection_title`	Character. Cytoscape collection title. If empty or `NULL`, defaults to `"cluster_collection"`.

Details

Cytoscape (version **>= 3.9.0**) must be running and reachable via **RCy3** before calling this function.

The function reconstructs node-level cluster membership from the `source_cluster` and `target_cluster` columns, identifies the largest community as a background cluster (for `label = "non_background"`), optionally converts KEGG IDs to names via **KEGGREST**, and then creates a Cytoscape network via **RCy3** with:

- colour mapping by cluster,
- shape mapping by node type (if `category = TRUE`),
- node labels from the chosen `label` mode.

KEGG name conversion can be slow on large networks, especially with `label = "all"`. For big MetaPathNet networks, `label = "non_background"` is often a good compromise.

Value

Invisibly returns `NULL`. The function acts via side effects in Cytoscape, creating and styling a new network.

See Also

`clusterNetwork` for community detection, and `viewNetworkCy` for non-clustered KEGG network visualisation.

Examples

```
## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

net_trp <- MetaPathNet_example_network

## 2) Detect communities on the full network graph
net_trp_clusters <- clusterNetwork(
  network_table = net_trp,
  method        = "leiden",
  resolution    = 0.01
)

## Not run:
## Requires a running Cytoscape session for network layout and rendering
## 3) Recommended: full cluster-aware visualisation in Cytoscape
viewClusterCy(
  cluster_matrix    = net_trp_clusters,
  label            = "all",   # convert KEGG IDs to biological names and label all nodes
  category          = TRUE,    # shape by node type (Compound / KO / Contextual)
  network_title     = "Example network clusters",
  collection_title  = "MetaPathNet_Examples"
)

## 4) Faster option for larger networks: label only non-background clusters
viewClusterCy(
  cluster_matrix    = net_trp_clusters,
  label             = "non_background",
  category          = TRUE,
  network_title     = "Example network clusters (foreground labels)",
```

```
  collection_title  = "MetaPathNet_Examples"
)

## End(Not run)
```

`viewNetworkCy` *Visualize a KEGG-Derived Network in Cytoscape via RCy3*

Description

Imports a MetaPathNet-style network table (typically from `buildKeggNetwork()` or `findShortestPaths()`) into Cytoscape and applies basic visual styling. Nodes can be coloured by type (compound / KO / other). Node labels can be kept as KEGG IDs (default) or converted to KEGG-derived names.

Usage

```
viewNetworkCy(
  network_table,
  name = FALSE,
  category = TRUE,
  style_interaction = FALSE,
  node_compound = "#9DC7DD",
  node_gene = "#9ED17B",
  network_title = "",
  collection_title = ""
)
```

Arguments

`network_table`	A 3-column matrix or data.frame representing the network (columns: `source`, `target`, `interaction_type`).
`name`	Logical (default `FALSE`). If `TRUE`, node labels are converted from KEGG IDs to KEGG-derived names/symbols via **KEGGREST**. If `FALSE`, node labels remain KEGG IDs and no KEGG REST calls are made.
`category`	Logical. If `TRUE`, nodes are coloured by type (`"Compound"`, `"KO"`, `"Other"`). If `FALSE`, all nodes use a default colour.
`style_interaction`	Logical (default `FALSE`). If `TRUE`, basic edge styling is applied based on `interaction_type`: `"k_compound:reversible"` edges are drawn with parallel line style, and `"k_compound:irreversib` edges receive a target arrow. Other interaction types are displayed with default Cytoscape edge styling.
`node_compound`	Colour for compound nodes (default `"#9DC7DD"`).
`node_gene`	Colour for KO/gene nodes (default `"#9ED17B"`).
`network_title`	Character. Title of the network in Cytoscape.
`collection_title`	Character. Cytoscape collection name.

Details

Cytoscape (v>=3.9.0) must be running and reachable via **RCy3** before calling this function.

Reverse duplicated edges are collapsed for Cytoscape display when they share the same metabolic interaction type

When `name = TRUE`, KEGG IDs are resolved once per node. This can be slow for large networks and depends on KEGG API response time.

Value

Invisibly returns `NULL`. The function acts by side-effect in Cytoscape, creating a styled network ready for manual or scripted exploration.

See Also

`buildKeggNetwork`, `viewNetworkR`, `findShortestPaths`

Examples

```
## 1) Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

net_trp_mixed <- MetaPathNet_example_network

## 2) Extract a shortest-path subnetwork (paths_trp_mixed)
source_nodes <- c("K00486", "K00453", "K00463",
                  "K01667", "K01696", "K01610", "K01695")
target_nodes <- c("cpd:C00078", "cpd:C00328", "cpd:C00780",
                  "cpd:C00463", "cpd:C00108", "cpd:C00458", "cpd:C00336")

map_src <- findMappedNodes(source_nodes, net_trp_mixed)
map_tgt <- findMappedNodes(target_nodes, net_trp_mixed)

paths_trp_mixed <- findShortestPaths(
  network_table      = net_trp_mixed,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "network_matrix",
  name               = FALSE,
  distance_threshold = 6
)

## Not run:
## Requires a running Cytoscape session for network layout and rendering
## Example 1 — Shortest-path subnetwork with KEGG name conversion + interaction styling
## (k_compound:reversible shown as parallel lines; k_compound:irreversible with arrows)
viewNetworkCy(
  network_table     = paths_trp_mixed,
  name              = TRUE,
  category          = TRUE,
  style_interaction = TRUE,
  node_compound     = "#9DC7DD",
  node_gene         = "#9ED17B",
  network_title     = "Trp-immune shortest paths",
  collection_title  = "MetaPathNet_Examples"
```

```
)

## Example 2 — Full Trp network without name conversion (faster)
viewNetworkCy(
  network_table     = net_trp_mixed,
  name              = FALSE,
  category          = TRUE,
  style_interaction = FALSE,
  node_compound     = "#9DC7DD",
  node_gene         = "#9ED17B",
  network_title     = "Trp-immune network",
  collection_title  = "MetaPathNet_Examples"
)

## End(Not run)
```

`viewNetworkR` *Visualise a KEGG-Based Network Matrix in R*

Description

Plots a KEGG-derived network matrix (e.g. from `buildKeggNetwork()` or `findShortestPaths()`) directly in R using **ggraph**. Nodes can be coloured by type (compound, KO, contextual) and labelled with KEGG IDs or KEGG-derived names.

Usage

```
viewNetworkR(
  network_table,
  category = TRUE,
  name = FALSE,
  layout = "randomly",
  node_size = 2,
  node_compound = "#9DC7DD",
  node_gene = "#9ED17B",
  network_title = "MetaPathNet Network"
)
```

Arguments

`network_table`	A network matrix with at least three columns: source, target, interaction type (e.g. output of `buildKeggNetwork()`, `buildCrossSpeciesNetwork()`, or `findShortestPaths()` with `output = "network_matrix"`).
`category`	Logical. If `TRUE`, colour nodes by type (`"compound"`, `"KO"`, `"contextual"`); if `FALSE`, all nodes share the same colour.
`name`	Logical (default `FALSE`). If `TRUE`, node labels are retrieved from KEGG (compound names and KO symbols/names) via **KEGGREST**. If `FALSE`, node labels remain the raw KEGG IDs.
`layout`	Character. Layout algorithm for **ggraph** (e.g. `"randomly"`, `"fr"`, `"kk"`).
`node_size`	Numeric. Base size for nodes (and label text) in the plot.

node_compound	Colour for compound nodes (default "#9DC7DD").
node_gene	Colour for KO nodes (default "#9ED17B").
network_title	Character. Optional plot title.

Details

For large networks (> 2000 edges), only the largest connected component is displayed. If more than ~300 nodes are present, a message suggests using viewNetworkCy() for Cytoscape-based visualisation. When name = TRUE, KEGG REST queries are issued for each node, which can be slow for large graphs.

Value

A ggplot object.

See Also

viewNetworkCy, buildCrossSpeciesNetwork, buildKeggNetwork, findShortestPaths

Examples

```
## Load the precomputed example network; see buildKeggNetwork() for its construction
data(MetaPathNet_example_network)

## Define source and target node sets, then keep only nodes mapped in the network
source_nodes <- c("K00453", "K00463", "K01667")
target_nodes <- c("cpd:C00078", "cpd:C00328", "cpd:C02700")

map_src <- findMappedNodes(source_nodes, MetaPathNet_example_network)
map_tgt <- findMappedNodes(target_nodes, MetaPathNet_example_network)

## Extract a shortest-path subnetwork for visualisation
paths_trp <- findShortestPaths(
  network_table      = MetaPathNet_example_network,
  source_nodes       = map_src$mapped_nodes,
  target_nodes       = map_tgt$mapped_nodes,
  mode               = "out",
  output             = "network_matrix",
  name               = FALSE,
  distance_threshold = 6
)

## Visualise the shortest-path subnetwork in R
p_trp <- viewNetworkR(
  network_table = paths_trp,
  category      = TRUE,
  name          = FALSE,
  layout        = "fr",
  node_size     = 5,
  network_title = "Tryptophan shortest-path subnetwork"
)
print(p_trp)
```

Index

∗ **datasets**
MetaPathNet_example_network, 32
∗ **internal**
network_utils, 36

addCustomReaction, 3
annotateKoClass, 4, *8*
annotateOrigin, *5*, 6

buildCompoundEgoNetwork, 9
buildCrossSpeciesNetwork, 11, *17, 18*, *31*, *38*, *49*
buildEgoNetwork, *9, 10*, 12, *36*
buildKeggNetwork, *10, 11*, 14, *17, 18*, *25*, *30, 31*, *34–36*, *38*, *47*, *49*

clusterNetwork, 16, *45*
combineNetworks, *3*, 18
convertGene, 19

distances, *34*

ego, *13*
enrichmentDotPlot, 20, *23*
enrichPathway, *13*, *20, 21*, 22

findKeggIDs, *3*, *20*, 24, *43*
findMappedNodes, 25, *36*, *42*
findShortestPaths, 26, *38*, *47*, *49*

getPathIDs, *9–11*, *15*, *23*, 29, *35*

induced_subgraph, *13*

keggLink, *20*

mapReaction, *3*, 30
MetaPathNet_example_network, 32

network_utils, 36
networkDistances, 33, *38*
networkSimilarity, 34

p.adjust, *23*
permutePaths, 37
phyper, *23*

removeDrugs, *36*, 39
removeNetworkNodes, *36*, 40
replaceNode, *36*, 41

suggestEntities, 42

viewClusterCy, *17*, 44
viewNetworkCy, *5*, *8*, *45*, 46, *49*
viewNetworkR, *13*, *17*, *47*, 48