

Package: netkit (via r-universe)

October 7, 2026

Type Package

Title Network Analysis and Visualization Toolkit

Version 0.1.0

Description A collection of tools for analyzing and visualizing networks. Summarizes global topology and per-node centrality, identifies hubs, bottlenecks and community structure, and classifies nodes into connectivity roles following Guimera and Amaral (2005) <[doi:10.1038/nature03288](https://doi.org/10.1038/nature03288)>. Extracts interpretable subnetworks around a set of nodes, including approximate Steiner trees after Kou, Markowsky and Berman (1981) <[doi:10.1007/BF00288961](https://doi.org/10.1007/BF00288961)>. Propagates influence from seed nodes using Laplacian smoothing, heat-kernel and random-walk-with-restart diffusion, with degree-matched permutation nulls, and simulates network robustness under progressive node removal. Observed metrics can be tested against degree-preserving random ensembles. Edge weights are supported throughout, with an explicit declaration of whether a weight means strength or distance. Matrix operations use sparse representations throughout so that the methods remain usable on large graphs.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports igraph (>= 2.1.0), ineq, dplyr, expm, ggplot2 (>= 3.5.0), ggExtra, ggrepel, graphics, grDevices, tibble, scales, Matrix, utils, progress, pracma, rlang, future, future.apply, RSpectra

Suggests testthat (>= 3.0.0), knitr, rmarkdown, spelling

VignetteBuilder knitr

Config/testthat/edition 3

URL <https://agallinat.github.io/netkit>
BugReports <https://github.com/agallinat/netkit/issues>
Config/pak/sysreqs cmake libglpk-dev make libuv1-dev libxml2-dev zlib1g-dev
Repository <https://agallinat.r-universe.dev>
Date/Publication 2026-10-05 18:54:06 UTC
RemoteUrl <https://github.com/agallinat/netkit>
RemoteRef HEAD
RemoteSha be4182ef3bfb3b290e4b9504983a794121c531ef

Contents

assign_attributes	2
calculate_roles	3
compare_networks	6
extract_subnetwork	8
find_bottlenecks	11
find_hubs	14
find_modules	16
greedy_seed_selection	18
highlight_nodes	20
layout_horizontal_tree	22
metric_significance	22
netkit-print	24
network_diffusion	25
network_diffusion_with_pvalues	28
node_metrics	31
null_model	35
plot_CCDF	37
plot_Net	39
robustness_analysis	40
small_worldness	43
summarize_graph_metrics	44
Index	47

assign_attributes	<i>Assign Vertex and Edge Attributes to an igraph Graph</i>
-------------------	---

Description

Adds or updates vertex and edge attributes in an igraph object using user-provided metadata tables. Vertex attributes are matched by the first column of nodes_table, and edge attributes are matched using the first two columns of edge_table, taking graph direction into account. Only matching nodes and edges are updated. Warnings are issued when there are unmatched entries.

Usage

```
assign_attributes(  
  graph,  
  nodes_table = NULL,  
  edge_table = NULL,  
  overwrite = TRUE  
)
```

Arguments

graph	An igraph object or a data frame containing a symbolic edge list.
nodes_table	Optional. A data.frame whose first column corresponds to vertex names.
edge_table	Optional. A data.frame whose first two columns correspond to source and target vertices.
overwrite	Logical. If TRUE, existing attributes are overwritten. If FALSE, existing attributes are preserved. Default is TRUE.

Value

An igraph object with added or updated attributes.

Examples

```
g <- igraph::make_ring(5)  
igraph::V(g)$name <- letters[1:5]  
  
nodes <- data.frame(node = letters[1:5], group = c("a", "a", "b", "b", "b"))  
edges <- data.frame(from = c("a", "b"), to = c("b", "c"), weight = c(10, 20))  
  
g <- assign_attributes(g, nodes_table = nodes, edge_table = edges)  
igraph::vertex_attr(g, "group")  
igraph::edge_attr(g, "weight")
```

calculate_roles	<i>Calculate Network Roles Based on Within-Module Z-Score and Participation Coefficient</i>
-----------------	---

Description

Implements the node role classification system of Guimerà & Amaral (2005) by calculating the within-module degree z-score and the participation coefficient for each node in a network. Nodes are assigned to one of seven role categories (R1–R7) based on their local modular connectivity.

Usage

```
calculate_roles(
  graph,
  communities = NULL,
  cluster.method = "spinglass",
  plot = TRUE,
  highlight_roles = TRUE,
  hub_z = 2.5,
  label_region = NULL,
  label.size = 12,
  thresholds = NULL,
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object representing the network, or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
communities	Optional. A community clustering object (as returned by an igraph clustering function), or a named membership vector. If NULL, community detection is performed using cluster.method.
cluster.method	Character. Clustering algorithm to use if communities is NULL. Default is "spinglass". Passed to find_modules().
plot	Logical. Whether to generate a 2D plot of participation coefficient (P) vs. within-module z-score (z). Default is TRUE.
highlight_roles	Logical. If TRUE, the role regions in the z-P plane are shaded for visual clarity. Default is TRUE.
hub_z	Numeric. Threshold for defining hubs in terms of within-module z-score. Default is 2.5.
label_region	Optional character vector of role labels (e.g., c("R4", "R7")) indicating which role regions should have their nodes labeled in the plot. Default is NULL.
label.size	Numeric. Base font size for plot text. Default is 12.
thresholds	Optional named numeric vector overriding one or more of the participation-coefficient boundaries between roles. Names must be drawn from R1_R2, R2_R3, R3_R4, R5_R6 and R6_R7; unnamed entries, unknown names, values outside [0, 1] and non-monotonic sets are rejected. NULL (default) uses the published values – see Details.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length igraph::ecount(graph). When supplied, the within-module z-score is computed from strengths and the participation coefficient from summed edge strengths per module, which is the weighted generalization given in Guimera & Amaral's supplementary material. See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Details

If no community structure is provided, modules are automatically detected using the specified clustering method. The function can optionally produce a 2D role plot (z vs. P) highlighting the canonical role regions.

When `communities` is `NULL`, community detection is delegated to `find_modules()` with `min_size = 1`, so no module is discarded for being small and every vertex receives a role. This matters for correctness as well as coverage: the participation coefficient of a node is computed from the module memberships of its neighbors, so dropping a neighbor's module silently distorts the coefficient of the node that remains.

The node roles are defined as follows, where `hub_z` defaults to 2.5 and the participation-coefficient boundaries are those published in Guimera & Amaral (2005):

- R1 Ultra-peripheral (non-hub): $z < 2.5, P \leq 0.05$
- R2 Peripheral (non-hub): $z < 2.5, 0.05 < P \leq 0.62$
- R3 Non-hub connector: $z < 2.5, 0.62 < P \leq 0.80$
- R4 Non-hub kinless: $z < 2.5, P > 0.80$
- R5 Provincial hub: $z \geq 2.5, P \leq 0.30$
- R6 Connector hub: $z \geq 2.5, 0.30 < P \leq 0.75$
- R7 Kinless hub: $z \geq 2.5, P > 0.75$

Those five numbers are held in one place internally and are used by the classifier, by the `roles_definitions` table and by the shaded bands of the diagnostic plot alike, so the three cannot disagree. Override them with thresholds if a different convention is wanted.

Value

A list with five elements:

`plot` A `ggplot2` object, or `NULL` when `plot = FALSE`. The element is always present, so the return shape does not depend on the arguments.

`result` A data frame with node-level information: node name, module, z-score, participation coefficient, and assigned role. It has one row per graph vertex. The exception is `cluster.method = "springlass"`, which can only be run on the largest connected component of a disconnected graph; vertices outside it have no module and are absent, which raises a warning. z , p and `role` are `NA` for any vertex whose module, or whose neighbors' modules, are unknown.

`graph` The input graph with `module`, `role_z`, `role_p` and `role` attached as vertex attributes, so that the classification can be passed straight to `plot_Net()` or `robustness_analysis()`.

`method` A human-readable description of the community detection used and the thresholds actually applied.

`roles_definitions` A data frame describing the seven role types and their conditions, generated from the same thresholds the classifier used.

Edge weights

Functions that can use edge weights take two arguments:

weights NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

weight_type "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves weights and weight_type once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Guimerà, R., & Amaral, L. A. N. (2005). Functional cartography of complex metabolic networks. *Nature*, 433(7028), 895–900. doi:[10.1038/nature03288](https://doi.org/10.1038/nature03288)

See Also

`find_modules()`, `igraph::cluster_spinglass()`, `igraph::membership()`

Examples

```
g <- igraph::sample_gnp(80, 0.08, directed = FALSE)
igraph::V(g)$name <- as.character(seq_len(igraph::vcount(g)))

# "louvain" is used here rather than the "spinglass" default. spinglass cannot
# run on a disconnected graph, so find_modules() falls back to the largest
# connected component and the remaining nodes receive no role.
result <- calculate_roles(g, cluster.method = "louvain", plot = FALSE)
head(result$result)
result$roles_definitions
```

Description

This function compares two networks using summary metrics, degree distributions, and topological similarity measures. It overlays the complementary cumulative frequency distributions (CCDFs) of degree and returns a combined report.

Usage

```
compare_networks(
  graph1,
  graph2,
  remove_singles = FALSE,
  show_PL = TRUE,
  PL_exponents = c(2, 3),
  colors = c("#e41a1c", "#000831", "#9c52f2", "#b8b8ff"),
  label.size = 12,
  weights1 = NULL,
  weights2 = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph1	An igraph object or data.frame (edge list).
graph2	An igraph object or data.frame (edge list).
remove_singles	Logical; remove single nodes before analysis.
show_PL	Logical; whether to fit and display power law exponents.
PL_exponents	Vector; power-law slopes to show.
colors	Optional vector of colors for the CCDF plot.
label.size	Labels' size in the CCDF plot.
weights1, weights2	Optional edge weights for each graph: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector. Supplied separately because the two graphs need not carry the same attribute. See netkit-weights .
weight_type	Either "strength" (default) or "distance", applied to both graphs. See netkit-weights .

Value

A list with:

plot A ggplot2 object overlaying both CCDF curves.

global_topology A data frame with one row per input graph, as produced by [summarize_graph_metrics\(\)](#).

similarity A one-row data frame with three distinct set statistics: `jaccard_similarity`, the Jaccard index of the edge sets (shared edges over their union); `node_overlap`, the Jaccard index of the vertex sets; and `edge_overlap`, the overlap coefficient of the edge sets (shared edges over the *smaller* of the two edge sets). The last is the informative companion to Jaccard when the two networks differ greatly in size – a small network nested inside a large one scores near 1 on overlap and near 0 on Jaccard. Each is NaN where its denominator is empty.

ks_test The Kolmogorov-Smirnov test comparing the two degree distributions, as returned by [stats::ks.test\(\)](#).

Edge weights

Functions that can use edge weights take two arguments:

`weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

Examples

```
g1 <- igraph::sample_pa(80, power = 1.5, directed = FALSE)
g2 <- igraph::sample_gnp(80, 0.05, directed = FALSE)
res <- compare_networks(g1, g2)
res$global_topology
res$ks_test
```

extract_subnetwork	<i>Extract an Interpretable Subnetwork Around a Set of Nodes</i>
--------------------	--

Description

Given a set of nodes of interest – a gene list, a set of disease genes, a group of drug targets – returns a connected, readable subnetwork around them. Five strategies are offered, from the plain induced subgraph through to an approximate Steiner tree and a diffusion-based expansion.

Usage

```
extract_subnetwork(
  graph,
  nodes,
  method = c("induced", "neighbors", "shortest_paths", "steiner", "diffusion"),
  order = 1,
  max_degree = NULL,
  top_n = 100,
```

```

diffusion_method = c("rwr", "laplacian", "heat"),
largest_component = FALSE,
weights = NULL,
weight_type = c("strength", "distance"),
plot = TRUE,
...
)

```

Arguments

graph	An igraph object representing the network to analyze, or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
nodes	Character vector of vertex names to build the subnetwork around (the "seeds", or in the Steiner case the "terminals").
method	How to choose the nodes to keep: "induced" Only nodes themselves, with whatever edges run between them. The baseline every other method should be compared against. "neighbors" nodes plus their order-step neighborhood. "shortest_paths" The union of all shortest paths between every pair of nodes. The classic "connect my gene list". "steiner" An approximate minimum Steiner tree over nodes: the smallest tree that connects them all. Usually the most readable figure, because it is a tree rather than a union of overlapping paths. "diffusion" Propagate from nodes with network_diffusion() and keep the top_n highest-scoring vertices.
order	Integer. Neighborhood radius when method = "neighbors". Default is 1.
max_degree	Integer or NULL. When method = "neighbors", exclude neighbors whose degree exceeds this. Without it, first-neighbor expansion on a hub-dominated network returns most of the network; see Details.
top_n	Integer. Number of vertices to keep when method = "diffusion". Default is 100.
diffusion_method	Passed to network_diffusion() as method when method = "diffusion". Default is "rwr".
largest_component	Logical. If TRUE, return only the largest connected component of the result. Default is FALSE, which keeps a forest when the seeds span several components.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . Path lengths and the Steiner tree use edge <i>costs</i> ; diffusion uses <i>strengths</i> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .
plot	Logical. If TRUE (default), draws the extracted subnetwork with plot_Net() , seeds highlighted.
...	Additional arguments passed to plot_Net() .

Details

"shortest_paths" and "steiner" both answer "how are these nodes connected to each other", but differently. The union of shortest paths keeps every equally short route, so it grows quickly and can be dense; the Steiner tree keeps one connecting structure of minimum total cost, so it is always a tree with $vcount - 1$ edges and is far easier to read. Use the union when you care about redundancy of connection, the tree when you want a figure.

The Steiner tree problem is NP-hard. This implementation uses the Kou-Markowsky-Berman heuristic: build the metric closure over the terminals (their pairwise shortest-path distances), take a minimum spanning tree of that closure, expand each closure edge back into the actual path it stands for, and prune non-terminal leaves. The result is guaranteed within a factor of $2 - 2/|terminals|$ of the true optimum, which in practice is close.

Because the problem is only solved approximately, the tree returned is one of possibly several near-minimal trees, and which one depends on how ties between equal-cost paths are broken. That tie-breaking is not the same in the weighted and unweighted code paths – **igraph** uses breadth-first search when no weights are given and Dijkstra when they are – so passing weights whose values happen to be all equal can return a different tree of slightly different total cost than omitting them. Both satisfy the approximation bound; neither is "the" Steiner tree. Treat the specific vertex set as one valid answer rather than a canonical one.

max_degree exists because first-neighbor expansion is dominated by hubs. In a protein interaction network a handful of promiscuous proteins are adjacent to a large fraction of the graph, so the one-step neighborhood of almost any gene list is most of the network. Capping degree removes them and leaves a subnetwork whose edges carry information.

Value

A list with:

result A tibble with one row per retained vertex: node, reason (why it was kept – "seed", "neighbor", "on_path", "steiner" or "diffused"), is_seed, and score (the diffusion score, or NA for the other methods).

graph The extracted subgraph, with is_seed and reason attached as vertex attributes. For method = "steiner" this is the Steiner tree itself – exactly the tree's edges. For every other method it is the subgraph *induced* on the selected vertices, so it carries all edges that run between them, which is more context but may contain cycles.

method A human-readable description of what was done.

Note there is no plot element: like `plot_Net()`, `find_modules()` and `highlight_nodes()`, this function renders through base graphics via `plot.igraph` and so has no plot object to hand back.

Edge weights

Functions that can use edge weights take two arguments:

weights NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

weight_type "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves weights and weight_type once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Kou, L., Markowsky, G., & Berman, L. (1981). A fast algorithm for Steiner trees. *Acta Informatica*, 15(2), 141-145. doi:10.1007/BF00288961

See Also

`network_diffusion()` for the scores behind method = "diffusion", `find_modules()` for unsupervised community structure.

Examples

```
g <- igraph::sample_pa(80, power = 1.5, directed = FALSE)
igraph::V(g)$name <- paste0("n", seq_len(igraph::vcount(g)))
seeds <- c("n5", "n20", "n40", "n60")

# The minimal tree connecting the seeds: always vcount - 1 edges.
st <- extract_subnetwork(g, seeds, method = "steiner", plot = FALSE)
igraph::vcount(st$graph)
st$result

# Every shortest route between them, which is a superset.
sp <- extract_subnetwork(g, seeds, method = "shortest_paths", plot = FALSE)
igraph::vcount(sp$graph) >= igraph::vcount(st$graph)

# `top_n` is small here to keep the example fast on an 80-node graph.
df <- extract_subnetwork(g, seeds, method = "diffusion", top_n = 15,
                        plot = FALSE)
head(df$result)
```

find_bottlenecks

Identify and Bottleneck Nodes in a Network

Description

Identifies bottleneck nodes in an igraph network as those with low degree and high betweenness centrality. The function supports both standardized (z-score) and quantile-based thresholding. Optionally, it produces a 2D scatter plot with bottlenecks highlighted.

Usage

```
find_bottlenecks(
  graph,
  method = c("zscore", "quantile"),
  degree_threshold = -1,
  betweenness_threshold = 1,
  degree_quantile = 0.25,
  betweenness_quantile = 0.75,
  log_transform = TRUE,
  plot = TRUE,
  focus_color = "skyblue",
  bottleneck_names = TRUE,
  bottleneck_cex = 3,
  gg_extra = list(),
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
method	Character. Method to define bottlenecks: "zscore" or "quantile".
degree_threshold	Numeric. Upper threshold for standardized degree (only used if method = "zscore").
betweenness_threshold	Numeric. Lower threshold for standardized betweenness (used in both methods).
degree_quantile	Numeric between 0 and 1. Quantile threshold for degree (used if method = "quantile").
betweenness_quantile	Numeric between 0 and 1. Quantile threshold for betweenness (used if method = "quantile").
log_transform	Logical. If TRUE, applies log1p transformation to degree and betweenness.
plot	Logical. If TRUE, generates a plot of degree vs. betweenness highlighting bottlenecks.
focus_color	Character. Color to display in the focus area of the plot (bottlenecks region).
bottleneck_names	Logical. If TRUE, labels bottleneck nodes on the plot.
bottleneck_cex	Numeric. Font size scaling for bottleneck labels on the plot.
gg_extra	List. Additional user-defined layers for the returned ggplot. eg. list(ylim(-2,2), theme_bw(), theme(legend.position = "none"))

weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . When supplied, degree becomes vertex <i>strength</i> and betweenness is computed on edge <i>costs</i> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Value

A list with the following components:

- `plot` A scatter plot of degree vs. betweenness highlighting bottlenecks, or NULL when `plot = FALSE`. The element is always present, so the return shape does not depend on the arguments. Note this is a `ggExtraPlot` (a **grid** gtable), not a `ggplot`: the scatterplot is passed through [ggMarginal\(\)](#) to add the marginal histograms, which assembles it, so it can be printed but not extended with `+`. To customize it, pass layers through `gg_extra`; those are added before the assembly step.
- `method` A message describing the method and thresholds used.
- `result` A tibble with node name, degree, betweenness, transformed metrics, and bottleneck status.
- `graph` The original graph with a new vertex attribute `is_bottleneck` (logical).

Edge weights

Functions that can use edge weights take two arguments:

- `weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.
- `weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in [igraph::betweenness\(\)](#), [igraph::distances\(\)](#), [igraph::diameter\(\)](#) and [igraph::mean_distance\(\)](#), but a strength in [igraph::cluster_louvain\(\)](#) and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

`netkit` resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

Examples

```
g <- igraph::sample_pa(80, power = 1.5, directed = FALSE)
res <- find_bottlenecks(g, method = "quantile", plot = FALSE)
res$method
head(res$result)
```

find_hubs

*Identify Hub Nodes in a Network***Description**

This function identifies hub nodes in an igraph network based on degree and betweenness centrality using either z-score or quantile thresholds. Optionally, it visualizes the classification using a scatter plot with marginal histograms.

Usage

```
find_hubs(
  graph,
  method = c("zscore", "quantile"),
  degree_threshold = 3,
  betweenness_threshold = 1,
  degree_quantile = 0.95,
  betweenness_quantile = 0.95,
  log_transform = TRUE,
  plot = TRUE,
  focus_color = "darkgreen",
  label.size = 12,
  hub_names = TRUE,
  hub_cex = 3,
  gg_extra = list(),
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
method	Character. Method to identify hubs: "zscore" (standardized metrics) or "quantile" (empirical percentiles).
degree_threshold	Numeric. Threshold for standardized degree (only used if method = "zscore").
betweenness_threshold	Numeric. Threshold for standardized betweenness (only used if method = "zscore").
degree_quantile	Numeric between 0 and 1. Quantile threshold for degree (used if method = "quantile").
betweenness_quantile	Numeric between 0 and 1. Quantile threshold for betweenness (used if method = "quantile").
log_transform	Logical. If TRUE, applies log-transformation to degree and betweenness metrics.

plot	Logical. If TRUE, generates a plot of the degree vs. betweenness classification.
focus_color	Character. Color to display in the focus area of the plot (hubs region).
label.size	Numeric. Base font size for plot elements. Passed to <code>theme_classic</code> .
hub_names	Logical. If TRUE, adds node labels to identified hubs on the plot.
hub_cex	Numeric. Font size scaling factor for hub labels on the plot.
gg_extra	List. Additional user-defined layers for the returned ggplot. eg. <code>list(ylim(-2,2), theme_bw(), theme(legend.position = "none"))</code>
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . When supplied, degree becomes vertex <i>strength</i> and betweenness is computed on edge <i>costs</i> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Value

A list with the following components:

plot A scatter plot of degree vs. betweenness with hub nodes highlighted, or NULL when `plot = FALSE`. The element is always present, so the return shape does not depend on the arguments. Note this is a `ggExtraPlot` (a **grid** gtable), not a `ggplot`: the scatterplot is passed through `ggMarginal()` to add the marginal histograms, which assembles it, so it can be printed but not extended with `+`. To customize it, pass layers through `gg_extra`; those are added before the assembly step.

method Description of the method and thresholds used.

result A tibble with node name, degree, betweenness, transformed metrics, and hub status.

graph The original graph with a new vertex attribute `is_hub`.

Edge weights

Functions that can use edge weights take two arguments:

weights NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

weight_type "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

`netkit` resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

Examples

```
g <- igraph::sample_pa(80, power = 1.5, directed = FALSE)
res <- find_hubs(g, method = "quantile", plot = FALSE)
res$method
head(res$result)

# The returned graph carries an `is_hub` vertex attribute, so results chain.
table(igraph::V(res$graph)$is_hub)
```

find_modules

Detect and Visualize Network Modules (Communities)

Description

Identifies modules (communities) in a network using a variety of community detection algorithms from the **igraph** package. Optionally filters out small modules, visualizes the detected modules, and returns induced subgraphs for each module.

Usage

```
find_modules(
  graph,
  method = "louvain",
  min_size = 3,
  no.of.communities = NULL,
  return_subgraphs = FALSE,
  plot = TRUE,
  label = FALSE,
  weights = NULL,
  weight_type = c("strength", "distance"),
  ...
)
```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
method	Character. Community detection method. Options include: "louvain", "walktrap", "infomap", "edge_betweenness", "fluid_communities", "fast_greedy", "leading_eigen", "leiden", and "spinglass".
min_size	Integer. Minimum number of nodes required to retain a module. Modules smaller than this size are discarded. Default is 3.
no.of.communities	Integer. Required only when method = "fluid_communities". Specifies the number of communities to find.

return_subgraphs	Logical. If TRUE, returns a list of induced subgraphs for each detected module.
plot	Logical. If TRUE, generates a network plot colored by module.
label	Logical. If TRUE, displays node labels in the plot.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . Community detection reads a weight as a <i>strength</i> , except <code>method = "edge_betweenness"</code> , which needs costs and is given them. <code>method = "fluid_communities"</code> cannot use weights and warns. See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .
...	Additional parameters passed to the <code>plot_Net()</code> function for customizing the plot.

Details

This function is a wrapper around several **igraph** community detection algorithms, including Louvain (`cluster_louvain()`), Walktrap, Infomap, Fast Greedy, and others. It simplifies their application and offers optional filtering, visualization via `plot_Net()`, and module subgraph extraction.

Value

A list with the following components:

`result` A tibble mapping each node to its module assignment.
`module_table` Deprecated alias for `result`, kept for backward compatibility.
`n_modules` The number of modules that meet the `min_size` threshold.
`subgraphs` A named list of subgraphs for each module (only if `return_subgraphs = TRUE`).
`method` The community detection method used.
`graph` The input graph with assigned 'module' and 'color' as vertex attributes, if `plot = TRUE`.
If `plot = TRUE`, a network plot is displayed with nodes colored by module.

Edge weights

Functions that can use edge weights take two arguments:

`weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.
`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen

implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves weights and weight_type once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Csardi G, Nepusz T. The igraph software package for complex network research. InterJournal, Complex Systems. 2006;1695. <https://igraph.org>

Examples

```
g <- igraph::sample_pa(80, power = 1.5, directed = FALSE)
res <- find_modules(g, method = "louvain", plot = FALSE)
res$n_modules
head(res$module_table)
```

greedy_seed_selection	<i>Greedy Seed Node Selection to Maximize Diffusion Toward Target Nodes</i>
-----------------------	---

Description

This function implements a greedy algorithm to select a set of k seed nodes from a candidate list such that the resulting diffusion signal on a specified set of target nodes is maximized. It supports multiple diffusion models (Laplacian, Heat Kernel, Random Walk with Restart).

Usage

```
greedy_seed_selection(
  graph,
  target_nodes,
  candidate_nodes = NULL,
  method = c("laplacian", "heat", "rwr"),
  alpha = 0.7,
  t = 1,
  restart_prob = 0.3,
  k = 5,
  normalize = TRUE,
  plot = TRUE,
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
target_nodes	A character vector of node names to prioritize for receiving the diffusion signal.
candidate_nodes	Optional character vector of eligible nodes to consider as seeds. If NULL (default), all non-target nodes are used.
method	Diffusion method to use. One of "laplacian", "heat", or "rwr".
alpha	Scaling parameter for the Laplacian diffusion method (default = 0.7).
t	Time parameter for the Heat Kernel diffusion (default = 1).
restart_prob	Restart probability for the Random Walk with Restart (default = 0.3).
k	Number of seed nodes to select (default = 5).
normalize	Logical; whether to normalize the cumulative diffusion score over the target nodes (default = TRUE).
plot	Logical; whether to plot the progression of target diffusion score as seeds are selected (default = TRUE).
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Details

This function uses a stepwise greedy heuristic. At each step, it evaluates the marginal gain in target score from adding each candidate node to the current seed set, and selects the one with the highest gain. This is repeated for k steps.

Note: The score of each candidate at every step is recomputed via a full diffusion run, making the function computationally intensive for large graphs or large k .

The target score can be interpreted as either the total or average diffusion signal received by the target nodes. Normalization helps scale results across networks of different sizes.

Value

A list with the following elements:

selected Character vector of selected seed node names.

final_target_score Final cumulative (or normalized) diffusion score over the target nodes.

scores_at_each_step Vector of target scores at each greedy selection step.

Edge weights

Functions that can use edge weights take two arguments:

`weights` `NULL` (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

See Also

[network_diffusion](#), [network_diffusion_with_pvalues](#)

Examples

```
g <- igraph::sample_gnp(60, 0.08, directed = FALSE)
igraph::V(g)$name <- as.character(seq_len(igraph::vcount(g)))
target <- c("1", "2", "3")

res <- greedy_seed_selection(g, target_nodes = target, k = 3, plot = FALSE)
res$selected_seeds
res$scores_at_each_step
```

highlight_nodes

Highlight Nodes in a Network Plot

Description

Highlights selected nodes in an `igraph` object by changing their label, fill color, and/or outline color. The function modifies node attributes and visualizes the result using `plot_Net()`.

Usage

```
highlight_nodes(
  graph,
  nodes,
  method = c("label", "fill", "outline"),
  label_color = "darkred",
  highlight_fill_color = "orange",
  highlight_frame_color = "darkblue",
  background_color = "gray",
  ...
)
```

Arguments

<code>graph</code>	An igraph object or a data.frame representing a symbolic edge list. If a data.frame, it should have at least two columns specifying source and target nodes.
<code>nodes</code>	A character vector of node names to highlight.
<code>method</code>	Character vector specifying how to highlight the nodes. One or more of: "label", "fill", "outline".
<code>label_color</code>	Color for highlighted node labels. Used only if "label" is in method. Default is "darkred".
<code>highlight_fill_color</code>	Fill color for highlighted nodes. Used only if "fill" is in method. Default is "orange".
<code>highlight_frame_color</code>	Outline color for highlighted nodes. Used only if "outline" is in method. Default is "darkblue".
<code>background_color</code>	Fill or frame color for non-highlighted nodes. Default is "gray".
<code>...</code>	Additional arguments passed to <code>plot_Net()</code> .

Value

Invisibly returns the igraph object with updated attributes.

Examples

```
g <- igraph::make_ring(10)
igraph::V(g)$name <- letters[1:10]
highlight_nodes(g, nodes = c("a", "j"), method = "fill")
```

 layout_horizontal_tree

Horizontal Tree Layout for Graph Visualization

Description

Rotates a tree layout by -90 degrees to produce a horizontal orientation. This is particularly useful for hierarchical visualizations where a left-to-right structure is preferred over the default top-to-bottom tree layout.

Usage

```
layout_horizontal_tree(graph)
```

Arguments

graph	An igraph object representing the input graph, or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
-------	--

Value

A numeric matrix with 2 columns representing x and y coordinates of each node in the layout. This matrix can be passed to `plot.igraph()` or other plotting functions.

Examples

```
g <- igraph::make_tree(15)
coords <- layout_horizontal_tree(g)
head(coords)
plot_Net(g, layout = coords)
```

 metric_significance

Score an Observed Graph Against a Null Ensemble

Description

Compares each of a graph's global metrics against the distribution that metric takes over a null ensemble, and reports the z-score and empirical p-value.

Usage

```
metric_significance(
  graph,
  null = NULL,
  metrics = NULL,
  n = 100,
  model = c("rewire", "configuration", "erdos_renyi"),
  weights = NULL,
  weight_type = c("strength", "distance"),
  seed = NULL,
  plot = TRUE,
  label.size = 12
)
```

Arguments

<code>graph</code>	An igraph object or a data frame edge list, as elsewhere in netkit.
<code>null</code>	A list of null graphs, as produced by <code>null_model()</code> . If NULL (default), an ensemble is generated internally with <code>n</code> graphs under <code>model</code> .
<code>metrics</code>	Character vector of metric names to test, matching columns of <code>summarize_graph_metrics()</code> . If NULL (default), every numeric metric that varies across the ensemble is used.
<code>n</code>	Integer. Number of null graphs to generate when <code>null</code> is NULL. Default is 100. Named <code>n</code> rather than <code>n_null</code> so that it matches <code>null_model()</code> and so that <code>n =</code> is an exact argument match – with a formal called <code>n_null</code> alongside <code>null</code> , <code>n =</code> is ambiguous and errors.
<code>model</code>	Passed to <code>null_model()</code> when <code>null</code> is NULL.
<code>weights, weight_type</code>	Passed to <code>summarize_graph_metrics()</code> . See <code>netkit-weights</code> .
<code>seed</code>	Integer or NULL. Seed for reproducibility.
<code>plot</code>	Logical. Whether to build a diagnostic plot. Default is TRUE.
<code>label.size</code>	Numeric. Base font size for plot text. Default is 12.

Details

`p_empirical` is two-sided and uses the $(r + 1) / (n + 1)$ convention, where `r` counts null values at least as extreme as the observed one. It is therefore never exactly zero: with `n` nulls the smallest attainable p-value is $1 / (n + 1)$, so testing against 20 nulls cannot produce evidence at $p < 0.05$ however large the effect. Choose `n` accordingly.

`z` is NA when the null distribution has zero variance – which happens legitimately for metrics the null model holds fixed, such as Nodes, Edges and Density under a degree-preserving model. Those metrics are excluded from the automatic `metrics` selection for that reason, but are reported as NA rather than dropped if you request them explicitly.

Value

A list with:

`plot` A faceted ggplot2 object showing each metric's null distribution with the observed value marked, or NULL when `plot = FALSE`. The element is always present.

`result` A tibble with one row per metric: `metric`, `observed`, `null_mean`, `null_sd`, `z`, `p_empirical`, `ci_lower` and `ci_upper` (the 2.5th and 97.5th percentiles of the null).

`graph` The input graph, unchanged, so the call still chains.

`method` A human-readable description of the test performed.

See Also

[null_model\(\)](#) for the ensembles, [summarize_graph_metrics\(\)](#) for the metrics themselves.

Examples

```
g <- igraph::sample_pa(60, power = 1.5, directed = FALSE)

# `n` is small here to keep the example fast. Note the floor this puts
# on the attainable p-value: 1 / (10 + 1).
res <- metric_significance(g, metrics = c("Clustering_coefficient",
                                         "Modularity"),
                          n = 10, seed = 1, plot = FALSE)

res$result
```

netkit-print

Printing netkit objects

Description

netkit's analysis functions return a list holding a result table, the annotated graph, a method string describing what was computed and, where there is one, a plot. Printing such a list with R's default method dumps the whole graph and every row of the table, which for a 100-graph [null_model\(\)](#) ensemble runs to well over a thousand lines. These methods print a summary instead.

Usage

```
## S3 method for class 'netkit_result'
print(x, n = 5, ...)

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

## S3 method for class 'netkit_null'
x[i]

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

Arguments

x	A netkit object: an analysis result, a <code>null_model()</code> ensemble or a <code>prepare_diffusion()</code> kernel.
n	Number of rows of <code>x\$result</code> to preview. Default 5.
...	Ignored, present for consistency with the generic.
i	Indices of the null graphs to keep.

Details

Only the printing changes. The objects are still plain lists: `x$result`, `x$graph`, `x$plot` and `x[[i]]` behave exactly as they did, and `unclass(x)` recovers the default printing.

Value

`x`, invisibly. Called for the side effect of printing.

Examples

```
g <- igraph::sample_gnp(40, 0.1, directed = FALSE)
igraph::V(g)$name <- as.character(seq_len(igraph::vcount(g)))

find_hubs(g, plot = FALSE)

# The elements are unchanged -- only the printing is summarized.
hubs <- find_hubs(g, plot = FALSE)
head(hubs$result, 3)

# n = 20 rather than the default 100 null graphs, to keep the example quick.
null_model(g, n = 20, seed = 1)

prepare_diffusion(g, method = "rwr")
```

network_diffusion	<i>Perform Network Diffusion from Seed Nodes</i>
-------------------	--

Description

Applies network diffusion techniques to propagate influence from a set of seed nodes across a graph. Supports Laplacian smoothing, heat diffusion, and random walk with restart (RWR).

Usage

```
network_diffusion(
  graph,
  seed_nodes,
  method = c("laplacian", "heat", "rwr"),
```

```

alpha = 0.7,
t = 1,
restart_prob = 0.3,
normalize = TRUE,
precompute = NULL,
weights = NULL,
weight_type = c("strength", "distance"),
seed_weights = NULL
)

```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes. Must have named vertices.
seed_nodes	Character vector of seed node names (must match $V(\text{graph})\$name$).
method	Character. Diffusion method to use: "laplacian": Solves the linear system $(I + \alpha L)^{-1} f_0$, where L is the (normalized) graph Laplacian and α is a smoothing parameter. Internally, a sparse Cholesky decomposition is used for efficiency. "heat": Applies the heat diffusion model $e^{-tL} f_0$, where t controls diffusion time. A truncated Taylor expansion is used for approximation. "rwr": Random Walk with Restart. Iteratively solves $f = (1-r)Pf + rf_0$, where P is the transition matrix and r is the restart probability.
alpha	Damping factor for the Laplacian method. Default is 0.7.
t	Time parameter for the heat diffusion method. Default is 1.
restart_prob	Restart probability (usually between 0.3 and 0.7) for the RWR method. Default is 0.3.
normalize	Logical. Whether to normalize the adjacency matrix (symmetric normalization for undirected graphs). Default is TRUE.
precompute	Optional list of precompute diffusion matrices (e.g., Laplacian, Cholesky factor, or transition matrix). Use prepare_diffusion() to generate this object and avoid redundant computations when calling this function repeatedly (e.g., in greedy optimization). A kernel built with different edge weights than requested is rejected rather than reused.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length $\text{igraph::ecount}(\text{graph})$. Signal propagates along edge <i>strengths</i> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .
seed_weights	Optional numeric vector of initial values for the seeds, replacing the default binary indicator. Either named (matched to <code>seed_nodes</code> by name) or unnamed and parallel to <code>seed_nodes</code> . Use this to diffuse from a continuous signal – log fold changes, scores, prior probabilities – rather than from set membership, which is what the propagation literature generally assumes.

Details

This function allows flexible application of network diffusion strategies, useful in systems biology (e.g., gene prioritization, pathway propagation), network analysis, and disease gene discovery. The underlying matrix operations are based on well-established diffusion models from graph theory.

For "rwr" (random walk with restart), the algorithm iteratively propagates scores until convergence based on a row-normalized transition matrix. Recommended method for large networks.

For "laplacian" and "heat", the graph Laplacian is computed from the (optionally normalized) adjacency matrix. For efficiency in iterative applications, precompute Laplacian and Cholesky decomposition using [prepare_diffusion\(\)](#).

Value

A tibble with two columns, sorted by descending score:

node Node name

score Diffusion score representing influence from the seed nodes

Edge weights

Functions that can use edge weights take two arguments:

weights NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

weight_type "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves **weights** and **weight_type** once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Köhler S, Bauer S, Horn D, Robinson PN. Walking the interactome for prioritization of candidate disease genes. *Am J Hum Genet.* 2008;82(4):949–958. doi:10.1016/j.ajhg.2008.02.013

Vanunu O, Magger O, Ruppin E, Shlomi T, Sharan R. Associating genes and protein complexes with disease via network propagation. *PLoS Comput Biol.* 2010;6(1):e1000641. doi:10.1371/journal.pcbi.1000641

Examples

```

g <- igraph::sample_gnp(80, 0.06, directed = FALSE)
igraph::V(g)$name <- as.character(seq_len(igraph::vcount(g)))
seed_nodes <- igraph::V(g)$name[1:5]

network_diffusion(g, seed_nodes, method = "laplacian")

# Reuse a precomputed kernel across repeated calls.
kernel <- prepare_diffusion(g, method = "rwr")
network_diffusion(g, seed_nodes, method = "rwr", precompute = kernel)

# Diffuse along edge strengths rather than treating every edge alike.
igraph::E(g)$confidence <- runif(igraph::ecount(g), 0.1, 1)
network_diffusion(g, seed_nodes, method = "rwr", weights = "confidence")

# Start from a continuous signal instead of set membership.
network_diffusion(g, seed_nodes, method = "rwr",
  seed_weights = c(2.4, -1.1, 0.7, 3.0, 0.2))

```

network_diffusion_with_pvalues

Perform Network Diffusion from Seed Nodes

Description

Applies network diffusion techniques to propagate influence from a set of seed nodes across a graph. Supports Laplacian smoothing, heat diffusion, and random walk with restart (RWR).

Usage

```

network_diffusion_with_pvalues(
  graph,
  seed_nodes,
  method = c("laplacian", "heat", "rwr"),
  alpha = 0.7,
  t = 1,
  restart_prob = 0.3,
  normalize = TRUE,
  n_permutations = 1000,
  seed = NULL,
  verbose = TRUE,
  weights = NULL,
  weight_type = c("strength", "distance"),
  seed_weights = NULL,
  null = c("degree_matched", "uniform"),
  match_pool = 10
)

```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes. Must have named vertices.
seed_nodes	Character vector of seed node names (must match $V(\text{graph})\$name$).
method	Character. Diffusion method to use: • "laplacian": Solves the linear system $(I + \alpha L)^{-1} f_0$, where L is the (normalized) graph Laplacian and α is a smoothing parameter. Internally, a sparse Cholesky decomposition is used for efficiency. • "heat": Applies the heat diffusion model $e^{-tL} f_0$, where t controls diffusion time. If available, a sparse approximation method is used to avoid dense matrix exponential. • "rwr": Random Walk with Restart. Iteratively solves $f = (1-r)Pf + rf_0$, where P is the transition matrix and r is the restart probability.
alpha	Damping factor for the Laplacian method. Default is 0.7.
t	Time parameter for the heat diffusion method. Default is 1.
restart_prob	Restart probability (usually between 0.3 and 0.7) for the RWR method. Default is 0.3.
normalize	Logical. Whether to normalize the adjacency matrix (symmetric normalization for undirected graphs). Default is TRUE.
n_permutations	Integer. Number of permutations to run for empirical p-value estimation (default 1000).
seed	Optional integer for reproducible random number generation. If NULL (default), seed is not set.
verbose	Logical. If TRUE (default), displays a progress bar during permutations.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length $\text{igraph}::\text{ecount}(\text{graph})$. See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .
seed_weights	Optional numeric vector of initial seed values, passed to network_diffusion() , replacing the default binary indicator. Note that the permutation null re-uses these magnitudes on the permuted seed sets, so the null tests the <i>position</i> of the seeds rather than their values.
null	Which permutation null to compare against: "degree_matched" The default. Each real seed is replaced by a randomly chosen vertex of similar degree, so the permuted sets have the same degree profile as the real one. "uniform" Permuted seed sets are drawn uniformly from the non-seed vertices, matching only the seed count. See Details for why the default is degree-matched.
match_pool	Integer. For null = "degree_matched", how many nearest-degree candidates each real seed may be replaced by. Default is 10. Smaller values match degree more tightly but give the null less variance; Inf makes every candidate eligible and so reduces exactly to null = "uniform".

Details

This function allows flexible application of network diffusion strategies, useful in systems biology (e.g., gene prioritization, pathway propagation), network analysis, and disease gene discovery. The underlying matrix operations are based on well-established diffusion models from graph theory.

For "rwr" (random walk with restart), the algorithm iteratively propagates scores until convergence based on a row-normalized transition matrix. Recommended method for large networks.

For "laplacian" and "heat", the graph Laplacian is computed from the (optionally normalized) adjacency matrix.

Value

A data frame with two columns:

node Node name

score Diffusion score representing influence from the seed nodes

Parallel execution

The permutation null is evaluated with `future.apply::future_lapply()`, which runs under whichever **future** plan is currently active. This function does not set a plan itself, so by default permutations run **sequentially**. To parallelize, set a plan once in your own session before calling:

```
future::plan("multisession", workers = 4)
res <- network_diffusion_with_pvalues(g, seed_nodes, n_permutations = 1000)
future::plan("sequential") # release the workers when finished
```

Permutations are reproducible regardless of the plan: seed is passed to `future_lapply(future.seed = )`, which generates parallel-safe RNG streams.

Choice of null

Diffusion scores are strongly degree-dependent: a high-degree vertex receives signal from more directions, so it scores highly almost regardless of where the seeds are. A null that draws seed sets uniformly therefore compares a real seed set – which in most applications is enriched for well-studied, highly connected vertices – against a null composed mostly of poorly connected ones, and reports much of the degree difference as significance.

`null = "degree_matched"` removes that confound by replacing each real seed with a randomly chosen vertex drawn from the `match_pool` candidates closest to it in degree. The resulting p-values answer "are these seeds unusually well *placed*?" rather than "are these seeds unusually well *connected*?", which is almost always the intended question.

How closely the match can be made is limited by the graph. If the seed set is extreme – the six highest-degree vertices of a scale-free network, say – there are simply no other vertices of comparable degree to permute with, and no permutation null can fully correct for degree. The function warns when the closest available candidates are still far from the seeds, so the residual confound is visible rather than assumed away.

Edge weights

Functions that can use edge weights take two arguments:

`weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Köhler S, Bauer S, Horn D, Robinson PN. Walking the interactome for prioritization of candidate disease genes. *Am J Hum Genet.* 2008;82(4):949–958. doi:10.1016/j.ajhg.2008.02.013

Vanunu O, Mager O, Ruppin E, Shlomi T, Sharan R. Associating genes and protein complexes with disease via network propagation. *PLoS Comput Biol.* 2010;6(1):e1000641. doi:10.1371/journal.pcbi.1000641

Examples

```
g <- igraph::sample_gnp(60, 0.08, directed = FALSE)
igraph::V(g)$name <- as.character(seq_len(igraph::vcount(g)))
seed_nodes <- igraph::V(g)$name[1:5]

# n_permutations is reduced from its default of 1000 to keep the example fast;
# use the default or higher for real analyses.
network_diffusion_with_pvalues(g, seed_nodes, method = "laplacian",
                               n_permutations = 50, seed = 1, verbose = FALSE)
```

node_metrics

Compute a Table of Node-Level Centrality Metrics

Description

Computes several per-node centrality and position metrics in one call and returns them as a tibble, with every metric also attached to the graph as a vertex attribute.

Usage

```
node_metrics(
  graph,
  metrics = c("degree", "strength", "betweenness", "harmonic", "eigenvector", "pagerank",
    "coreness", "clustering", "constraint", "eccentricity"),
  weights = NULL,
  weight_type = c("strength", "distance"),
  normalized = TRUE,
  mode = c("all", "in", "out"),
  max_nodes = 5000,
  plot = TRUE,
  plot_type = c("correlation", "ranking"),
  top_n = 15,
  label.size = 12
)
```

Arguments

graph	An igraph object representing the network to analyze, or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
metrics	Character vector of metrics to compute. Any subset of "degree", "strength", "betweenness", "closeness", "harmonic", "eigenvector", "pagerank", "coreness", "clustering", "constraint" and "eccentricity". Defaults to all but "closeness" – see Details.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .
normalized	Logical. If TRUE (default), metrics that igraph can normalize to <code>[0, 1]</code> are normalized, which makes them comparable across graphs of different size.
mode	For directed graphs, whether to count "all" (default), "in" or "out" edges. Ignored for undirected graphs.
max_nodes	Integer. Above this vertex count, the metrics whose cost is $O(VE)$ – "betweenness", "closeness", "harmonic" and "eccentricity" – are skipped with a warning and returned as NA. Set to Inf to compute them regardless. Default is 5000, matching the threshold summarize_graph_metrics() already uses for betweenness.
plot	Logical. Whether to build a diagnostic plot. Default is TRUE.
plot_type	Either "correlation" (default), a Spearman correlation heatmap of the computed metrics, or "ranking", a faceted bar chart of the top_n highest-scoring nodes per metric.
top_n	Integer. Number of nodes shown per facet when <code>plot_type = "ranking"</code> . Default is 15.
label.size	Numeric. Base font size for plot text. Default is 12.

Details

This is the node-level counterpart to `summarize_graph_metrics()`, which describes the graph as a whole. Having both means a ranking of nodes no longer has to be stitched together from individual **igraph** calls, and because the returned graph carries the metrics as vertex attributes, the result feeds directly into `plot_Net()` (`color = "pagerank"`) and into `robustness_analysis()` (`removal_strategy = "pagerank"`), which already accepts any numeric vertex attribute as a removal priority.

The metrics are:

degree Number of incident edges. Always the unweighted count.

strength Sum of incident edge weights; equal to degree when unweighted.

betweenness Fraction of shortest paths passing through the node. Uses edge *costs* when weighted.

closeness Reciprocal of the mean distance to all *reachable* nodes. Not comparable across components – see below.

harmonic Sum of the reciprocal distances. Unlike **closeness** this is comparable across components, because an unreachable pair contributes $1/\infty = 0$ rather than being excluded from the average.

eigenvector Leading eigenvector of the adjacency matrix: a node is central if its neighbors are.

pagerank Stationary distribution of a random surfer. Sums to 1.

coreness Largest *k* for which the node belongs to the *k*-core.

clustering Local transitivity: how interconnected the node's neighborhood is. NaN for nodes of degree below 2, which have no neighbor pairs.

constraint Burt's constraint: how much the node's connections are concentrated within a single group. Low constraint marks a broker.

eccentricity Distance to the furthest reachable node.

"closeness" is omitted from the defaults because it is not comparable across components, and most real networks are disconnected. **igraph** averages the distance over reachable vertices only, so a node in a *small* isolated component scores *higher* than an equally well-placed node in the giant component – everything in its component is nearby. Ranking by closeness on a fragmented graph therefore puts the periphery on top. "harmonic" answers the same question without that failure mode, because unreachable pairs contribute zero rather than being dropped from the denominator. Request "closeness" explicitly if the graph is connected or you want the per-component reading.

The default plot is a correlation heatmap rather than a ranking because the most common mistake with a table like this is to treat the metrics as independent evidence. On many networks betweenness, eigenvector centrality and PageRank correlate with degree above 0.9, so a node that looks important by four measures may be important by one. The heatmap makes that visible before the ranking is interpreted.

Value

A list with:

plot A ggplot2 object, or NULL when `plot = FALSE`. The element is always present, so the return shape does not depend on the arguments.

result A tibble with one row per vertex: node followed by one column per requested metric, in the order given by **metrics**.

graph The input graph with each metric attached as a vertex attribute of the same name.

method A human-readable description of what was computed.

Edge weights

Functions that can use edge weights take two arguments:

weights NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

weight_type "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves **weights** and **weight_type** once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Freeman, L. C. (1978). Centrality in social networks: conceptual clarification. *Social Networks*, 1(3), 215-239. doi:10.1016/03788733(78)900217

Burt, R. S. (1992). *Structural Holes: The Social Structure of Competition*. Harvard University Press.

Marchiori, M., & Latora, V. (2000). Harmony in the small-world. *Physica A*, 285(3-4), 539-546. doi:10.1016/S03784371(00)003113

See Also

`summarize_graph_metrics()` for the graph-level counterpart, `find_hubs()` for thresholded classification.

Examples

```
g <- igraph::sample_pa(60, power = 1.5, directed = FALSE)
igraph::V(g)$name <- paste0("n", seq_len(igraph::vcount(g)))

res <- node_metrics(g, plot = FALSE)
head(res$result)
```

```
# Every metric is on the returned graph, so it chains straight onward.
igraph::vertex_attr_names(res$graph)
rob <- robustness_analysis(res$graph, removal_strategy = "pagerank",
                           steps = 10, plot = FALSE)

rob$auc$lcc_size

# The default plot shows how far the metrics actually disagree.
p <- node_metrics(g, metrics = c("degree", "betweenness", "pagerank"))$plot
```

null_model

Generate Null-Model Graphs for Significance Testing

Description

Produces an ensemble of random graphs matched to an observed graph in some respect, so that an observed metric can be compared against what that respect alone would predict.

Usage

```
null_model(
  graph,
  model = c("rewire", "configuration", "erdos_renyi"),
  n = 100,
  shuffle_weights = TRUE,
  seed = NULL,
  niter = NULL
)
```

Arguments

graph	An igraph object representing the network to analyze, or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
model	Which aspect of the observed graph to preserve: "rewire" Degree-preserving edge rewiring (the default). Preserves the degree sequence exactly while staying close to the observed graph. "configuration" Draws a fresh graph with the same degree sequence via igraph::sample_degseq() . Preserves the degree sequence exactly but is otherwise unconstrained. "erdos_renyi" Matches only the vertex and edge counts. The weakest null, included as the deliberate contrast: comparing against it tells you whether an effect needs anything beyond density.
n	Integer. Number of null graphs to generate. Default is 100.

shuffle_weights	Logical. If TRUE (default) and the observed graph carries a weight edge attribute, the observed weights are permuted across the null graph's edges, so the weight <i>distribution</i> is preserved while its placement is randomized. If FALSE, the null graphs carry no weights at all – including under model = "rewire", which would otherwise inherit the observed attribute bound to edges that have since been rewired.
seed	Integer or NULL. Seed for reproducibility. When NULL (default) the caller's random stream is used and left alone.
niter	Integer or NULL. Number of rewiring trials for model = "rewire". When NULL (default), $10 * \text{ecount}(\text{graph})$, which is the usual rule of thumb for mixing.

Details

This is what turns `summarize_graph_metrics()` from descriptive into inferential. On its own, a modularity of 0.42 says nothing: random graphs with the same degree sequence routinely reach 0.3. Pass an ensemble from this function to `metric_significance()` to get the z-score and empirical p-value.

"rewire" and "configuration" both preserve the degree sequence, which is almost always the right null for a topological claim: nearly every network metric is partly determined by the degree sequence, so a null that does not hold it fixed mostly rediscovers that the graph is heavy-tailed.

They differ in how far they travel from the observed graph. "rewire" makes local double-edge swaps, so the ensemble is centered on the observed graph and is the more conservative choice. "configuration" resamples from scratch.

"configuration" is attempted with method = "v1", which produces simple connected graphs, and falls back to "configuration.simple" with a warning where that is not possible – "v1" requires a connected realization of the degree sequence to exist, which fails for example when the graph has isolated vertices.

Value

A list of `n` `igraph` objects, with class `c("netkit_null", "list")` and the model recorded in its "model" attribute.

References

- Maslov, S., & Sneppen, K. (2002). Specificity and stability in topology of protein networks. *Science*, 296(5569), 910-913. doi:10.1126/science.1065103
- Newman, M. E. J., Strogatz, S. H., & Watts, D. J. (2001). Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2), 026118. doi:10.1103/PhysRevE.64.026118

See Also

`metric_significance()` to score an observed graph against the ensemble, `small_worldness()` for the classic application.

Examples

```
g <- igraph::sample_pa(60, power = 1.5, directed = FALSE)

# `n` is small here to keep the example fast; use 100 or more in practice.
nulls <- null_model(g, model = "rewire", n = 10, seed = 1)
length(nulls)

# The degree sequence is preserved exactly, which is the point.
identical(sort(igraph::degree(nulls[[1]]), sort(igraph::degree(g)))
```

plot_CCDF

Plot Complementary Cumulative Degree Distribution (CCDF)

Description

This function plots the complementary cumulative distribution function (CCDF) of node degrees in a network and optionally overlays power-law reference curves.

Usage

```
plot_CCDF(
  graph,
  keep_direction = TRUE,
  remove_singles = FALSE,
  show_PL = TRUE,
  PL_exponents = c(2, 3),
  colors = c("#000831", "#e41a1c", "darkgreen", "#9c52f2", "#b8b8ff"),
  label.size = 12,
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object representing the network to analyze or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
keep_direction	Logical. Only for directed graphs. If TRUE, CCDF curves are drawn for 'in'-degree, 'out'-degree, and 'all'-degree distributions. FALSE to ignore directionality.
remove_singles	Logical. If TRUE, nodes with degree 0 are removed from the graph before computing the CCDF. Default is FALSE.
show_PL	Logical. If TRUE, overlays theoretical power-law reference lines of the form $P(K > k) \sim k^{-\gamma}$. Default is TRUE.
PL_exponents	Numeric vector. The γ exponents for the power-law curves. Default is c(2, 3).

colors	Optional character vector. Custom colors for the graph curve and power-law lines. If NULL, default colors are used.
label.size	Numeric. Font size for axis labels and theme. Passed to theme_minimal.
weights	Optional edge weights: NULL (default) to plot the degree distribution, or the name of an edge attribute / a numeric vector of length <code>igraph::ecount(graph)</code> to plot the vertex <i>strength</i> distribution instead. See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Value

A ggplot2 object showing the CCDF of node degrees on a log-log scale.

Edge weights

Functions that can use edge weights take two arguments:

`weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

Examples

```
g <- igraph::sample_pa(200, power = 1.5, directed = FALSE)
plot_CCDF(g)

# Compare against reference power-law slopes.
plot_CCDF(g, PL_exponents = c(2, 2.5, 3))
```

plot_Net	<i>Plot an igraph network with customizable node sizes and edge widths</i>
----------	--

Description

This function plots an igraph network graph with options to customize node size based on vertex degree or a numeric vertex attribute, set node and edge colors, label nodes, and adjust edge widths (optionally mapped to edge betweenness centrality).

Usage

```
plot_Net(
  graph,
  label = FALSE,
  color = "#006d77",
  color_ramp = c("blue", "white", "red"),
  NA_color = "gray",
  frame.color = NULL,
  node.size.factor = 1,
  node.degree.map = TRUE,
  edge.color = "#999999",
  edge.width.factor = 1,
  edge.bw.map = TRUE,
  label.color = "#e29578",
  label.size = 1,
  layout = NULL,
  ...
)
```

Arguments

graph	An igraph object representing the network to plot or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes.
label	Logical, or character vector. If FALSE, no node labels are shown. If TRUE, node labels are set to vertex names or existing labels. If a character vector of length equal to the number of vertices, used as custom node labels.
color	Character. Color for node fill, or existing vertex attribute name to map node fill color. Default is '#006d77'.
color_ramp	A character vector of colors (e.g., c("blue", "white", "red")) used to create a continuous color ramp for mapping numeric node attributes to colors. Ignored if a fixed color is provided or if node color is categorical. Passed to colorRampPalette to interpolate a gradient of colors.
NA_color	Node color for NAs in node color mapping.
frame.color	Character. Color for node frame, or existing vertex attribute name to map node frame color. Default is NULL.

<code>node.size.factor</code>	Numeric or character. If numeric, acts as a constant scaling factor for node size. If character, it should be the name of a numeric vertex attribute used to size nodes.
<code>node.degree.map</code>	Logical. If TRUE and <code>node.size.factor</code> is numeric, node size is mapped to vertex degree multiplied by <code>node.size.factor</code> . Ignored if <code>node.size.factor</code> is character.
<code>edge.color</code>	Character. Color of edges. Default is '#999999'.
<code>edge.width.factor</code>	Numeric. Factor to scale edge widths. Default is 1.
<code>edge.bw.map</code>	Logical. If TRUE, edge widths are mapped to the log-transformed edge betweenness centrality. Otherwise, edges have uniform width.
<code>label.color</code>	Character. Color of node labels. Default is '#e29578'.
<code>label.size</code>	Numeric. Character expansion factor for label size. Default is 1.
<code>layout</code>	Optional numeric matrix specifying vertex coordinates for layout.
<code>...</code>	Additional parameters passed to <code>plot.igraph</code> .

Value

Invisibly returns NULL. The function produces a plot.

Examples

```
g <- igraph::sample_gnp(50, 0.06, directed = FALSE)
plot_Net(g, label = TRUE, node.size.factor = 2)
```

robustness_analysis	<i>Network Robustness Analysis via Node Removal Simulation</i>
---------------------	--

Description

Simulates the removal of nodes from a network using various strategies and evaluates how the structure degrades using selected robustness metrics. Useful for assessing the vulnerability or resilience of a graph.

Usage

```
robustness_analysis(
  graph,
  removal_strategy = c("random", "degree", "betweenness", "strength"),
  steps = 50,
  metrics = c("lcc_size", "efficiency", "n_components"),
  n_reps = 50,
  plot = TRUE,
```

```

    seed = NULL,
    weights = NULL,
    weight_type = c("strength", "distance")
)

```

Arguments

graph	An igraph object representing the network to plot or a data frame containing a symbolic edge list in the first two columns. Additional columns are considered as edge attributes. Must be undirected; directed graphs will be converted.
removal_strategy	Character. Strategy used for node removal. Options are: "random", "degree", "betweenness", "strength", or the name of a numeric vertex attribute. Custom attributes are interpreted as priority scores (higher = removed first). "strength" requires weights and errors without them.
steps	Integer. Number of removal steps (default: 50).
metrics	Character vector. Structural metrics to compute at each step. Options include: "lcc_size", "efficiency", and "n_components".
n_reps	Integer. Number of simulation repetitions (only relevant if removal_strategy = "random").
plot	Logical. If TRUE, a robustness plot is generated.
seed	Integer or NULL. Random seed for reproducibility.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . When supplied, global efficiency is computed over weighted path lengths, removal_strategy = "betweenness" uses cost-based betweenness, and removal_strategy = "strength" becomes available. See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Details

This function builds on classic approaches in network science for evaluating structural robustness, simulating progressive node removal and quantifying the degradation of key topological features.

For deterministic strategies ("degree", "betweenness", or custom attributes), nodes are removed in a fixed priority order. For the "random" strategy, the process is repeated `n_reps` times, and the results are aggregated.

The available metrics are:

- **Largest Connected Component:** size of the largest remaining component (`lcc_size`).
- **Global Efficiency:** average inverse shortest path length among all pairs (`efficiency`).
- **Number of Components:** total number of disconnected components (`n_components`).

Additionally, Area Under the Curve (AUC) is calculated for each metric, providing a scalar summary of robustness. A higher AUC indicates greater resilience (i.e., slower degradation).

Value

A list with:

`plot` A `ggplot2` object showing the evolution of the selected metrics as nodes are progressively removed, or `NULL` when `plot = FALSE`. The element is always present, so the return shape does not depend on the arguments.

`result` A summarized data frame (mean and SD) if `n_reps > 1`, otherwise raw results.

`all_results` A data frame with simulation results across all steps and repetitions.

`summary` Deprecated alias for `result`, kept for backward compatibility.

`auc` Named list of AUC (area under the curve) values for each selected metric.

Edge weights

Functions that can use edge weights take two arguments:

`weights` `NULL` (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the `weight` attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

`netkit` resolves `weights` and `weight_type` once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

Albert R, Jeong H, Barabási AL. Error and attack tolerance of complex networks. *Nature*. 2000;406(6794):378–382. doi:10.1038/35019019

Examples

```
g <- igraph::sample_pa(80, power = 1.5, directed = FALSE)

# `steps` is reduced from its default of 50 to keep the example fast.
res <- robustness_analysis(g, removal_strategy = "degree", steps = 10,
                          metrics = c("lcc_size", "n_components"),
                          plot = FALSE, seed = 1)

res$auc
head(res$summary)
```

small_worldness	<i>Small-World Coefficients</i>
-----------------	---------------------------------

Description

Computes the small-world coefficient σ , which compares a graph's clustering and path length against a degree-matched random ensemble.

Usage

```
small_worldness(
  graph,
  n = 100,
  model = c("rewire", "configuration", "erdos_renyi"),
  weights = NULL,
  weight_type = c("strength", "distance"),
  seed = NULL
)
```

Arguments

graph	An igraph object or a data frame edge list.
n	Integer. Number of null graphs. Default is 100.
model	Passed to null_model() . Default is "rewire".
weights, weight_type	Passed to summarize_graph_metrics() . See netkit-weights .
seed	Integer or NULL. Seed for reproducibility.

Details

A network is "small-world" when it is much more clustered than a random graph with the same degree sequence while having a comparable average path length. σ expresses that as a single ratio: $(C/C_{\text{rand}}) / (L/L_{\text{rand}})$, where values appreciably above 1 indicate small-world organization.

Only σ is reported. The companion coefficient ω of Telesford et al. (2011) additionally requires a *lattice* reference, and **igraph** provides no degree-preserving laticization; implementing one approximately would make ω quietly dependent on how well that approximation worked, so it is omitted rather than shipped unreliable.

σ is known to grow with network size, so it is not comparable across graphs of different size – use it to ask whether one graph is small-world, not which of two is more so.

Value

A list with:

result A one-row tibble: σ , C , C_{rand} , L , L_{rand} and n_{null} .

graph The input graph, unchanged.
 method A human-readable description.

References

Humphries, M. D., & Gurney, K. (2008). Network "small-world-ness": a quantitative method for determining canonical network equivalence. *PLoS ONE*, 3(4), e0002051. doi:[10.1371/journal.pone.0002051](https://doi.org/10.1371/journal.pone.0002051)
 Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393(6684), 440-442. doi:[10.1038/30918](https://doi.org/10.1038/30918)

See Also

[null_model\(\)](#), [metric_significance\(\)](#)

Examples

```
g <- igraph::sample_smallworld(1, 60, 4, 0.05)

# `n` is small here to keep the example fast.
sw <- small_worldness(g, n = 10, seed = 1)
sw$result
```

summarize_graph_metrics

Summarize Topological Properties of a Graph

Description

Computes a comprehensive set of global topological metrics for an input graph, including basic structure, connectivity, spectral properties, and complexity. Supports both igraph objects and data frames representing edge lists.

Usage

```
summarize_graph_metrics(
  graph,
  weights = NULL,
  weight_type = c("strength", "distance")
)
```

Arguments

graph	An igraph object or a data frame with columns from and to representing an edge list.
weights	Optional edge weights: NULL (default) to ignore them, the name of an edge attribute, or a numeric vector of length <code>igraph::ecount(graph)</code> . See netkit-weights .
weight_type	Either "strength" (default) or "distance". See netkit-weights .

Details

Metrics computed:

- Number of nodes and edges
- Directed TRUE/FALSE
- Graph density
- Diameter and average path length of the largest connected component
- Clustering coefficient (transitivity)
- Degree assortativity
- Average degree and betweenness centrality
- Number of connected components and size of the largest connected component
- Number of single nodes
- Algebraic connectivity (second-smallest Laplacian eigenvalue)
- Degree entropy (Shannon entropy of the degree distribution)
- Gini coefficient of node degrees
- Modularity of the community structure (via Louvain algorithm)

When weights are supplied, the metrics switch to their weighted definitions: Avg_degree becomes mean vertex strength; Diameter, Average_path_length and Avg_betweenness are computed on edge *costs*; Clustering_coefficient uses Barrat's weighted transitivity; Degree_assortativity correlates strengths rather than degrees; Degree_entropy and Gini_degree describe the strength distribution; and the Laplacian behind Algebraic_connectivity and the Louvain run behind Modularity both use edge strengths. Two columns are added, Is_weighted and Avg_strength, so a weighted and an unweighted summary can be row-bound.

Value

A one-row data.frame, each column a graph-level metric.

Metrics that are mathematically undefined for the input are NaN rather than substituted values, as returned by the underlying **igraph** and **ineq** functions. On degenerate graphs this is expected: an edgeless graph has no paths (Average_path_length), no connected triples (Clustering_coefficient), no degree variance (Degree_assortativity) and a zero mean degree (Gini_degree).

Edge weights

Functions that can use edge weights take two arguments:

`weights` NULL (the default) to ignore edge weights; the name of an edge attribute, such as "weight"; or a numeric vector with one value per edge, in `igraph::E(graph)` order.

`weight_type` "strength" (the default) if a larger value means a more tightly connected pair – confidence scores, correlations, co-expression, read counts, interaction scores. "distance" if a larger value means further apart – costs, dissimilarities, reaction times.

Declaring which you have is not bookkeeping. **igraph** reads the weight attribute implicitly and gives it *opposite* meanings in different functions: a cost in `igraph::betweenness()`, `igraph::distances()`, `igraph::diameter()` and `igraph::mean_distance()`, but a strength in `igraph::cluster_louvain()` and the other community detection algorithms. Attaching a confidence score and letting that happen implicitly therefore inverts every path-based metric – a high-confidence interaction is treated as a long distance – while community detection reads the same numbers the way you intended.

netkit resolves weights and weight_type once per call and derives both a strength and a distance vector from them, so each metric receives the one it needs. A "strength" is converted to a distance by reciprocal ($1/w$); a "distance" is converted to a strength by reflection ($\max(w) - w + \min(w)$), which keeps a zero distance finite.

References

- Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press.
- Estrada, E. (2012). *The Structure of Complex Networks: Theory and Applications*. Oxford University Press.
- Latora, V., Nicosia, V., & Russo, G. (2017). *Complex Networks: Principles, Methods and Applications*. Cambridge University Press.
- Louvain modularity method: Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). *Fast unfolding of communities in large networks*. J. Stat. Mech., 2008(10), P10008.
- Barrat, A., Barthélemy, M., Pastor-Satorras, R., & Vespignani, A. (2004). *The architecture of complex weighted networks*. PNAS, 101(11), 3747-3752.

Examples

```
g <- igraph::sample_gnp(60, 0.08, directed = FALSE)
summarize_graph_metrics(g)

# Weighted: declare what the attribute means, because igraph alone would read
# it as a cost for path metrics and as a strength for modularity.
igraph::E(g)$confidence <- runif(igraph::ecount(g), 0.1, 1)
summarize_graph_metrics(g, weights = "confidence", weight_type = "strength")
```

Index

[.netkit_null (netkit-print), 24

assign_attributes, 2

calculate_roles, 3

colorRampPalette, 39

compare_networks, 6

extract_subnetwork, 8

find_bottlenecks, 11

find_hubs, 14

find_hubs(), 34

find_modules, 16

find_modules(), 5, 6, 10, 11

future.apply::future_lapply(), 30

ggMarginal, 13, 15

greedy_seed_selection, 18

highlight_nodes, 20

highlight_nodes(), 10

igraph::betweenness(), 6, 8, 11, 13, 15, 17, 20, 27, 31, 34, 38, 42, 46

igraph::cluster_louvain(), 6, 8, 11, 13, 15, 17, 20, 27, 31, 34, 38, 42, 46

igraph::cluster_spring(), 6

igraph::diameter(), 6, 8, 11, 13, 15, 17, 20, 27, 31, 34, 38, 42, 46

igraph::distances(), 6, 8, 11, 13, 15, 17, 20, 27, 31, 34, 38, 42, 46

igraph::mean_distance(), 6, 8, 11, 13, 15, 17, 20, 27, 31, 34, 38, 42, 46

igraph::membership(), 6

igraph::sample_degseq(), 35

layout_horizontal_tree, 22

metric_significance, 22

metric_significance(), 36, 44

netkit-print, 24

netkit_weights, 4, 7, 9, 13, 15, 17, 19, 23, 26, 29, 32, 38, 41, 43, 44

network_diffusion, 20, 25

network_diffusion(), 9, 11, 29

network_diffusion_with_pvalues, 20, 28

node_metrics, 31

null_model, 35

null_model(), 23–25, 43, 44

plot_CCDF, 37

plot_Net, 39

plot_Net(), 5, 9, 10, 33

prepare_diffusion, 26, 27

prepare_diffusion(), 25

print.netkit_kernel (netkit-print), 24

print.netkit_null (netkit-print), 24

print.netkit_result (netkit-print), 24

robustness_analysis, 40

robustness_analysis(), 5, 33

small_worldness, 43

small_worldness(), 36

stats::ks.test(), 7

summarize_graph_metrics, 44

summarize_graph_metrics(), 7, 23, 24, 32–34, 36, 43