

Package ‘mpaR’

July 24, 2026

Title Main Path Analysis for Citation and Directed Networks

Version 0.5.0

Description Implements Main Path Analysis (MPA) as introduced by Hummon and Doreian (1989) <doi:10.1016/0378-8733(89)90017-8>. Given a directed acyclic graph (DAG) representing a citation or precedence network, the package computes traversal weights (SPC, SPLC, SPNP) for each edge and extracts the global, local, and key-route main paths. Also provides tools for DAG validation, node role classification (source/terminal/user), per-component path extraction for disconnected networks, and scale-free network testing. Accepts 'igraph' objects or edge-list data frames as input. Includes readers for 'Pajek' (.net) and 'Gephi' export (.gexf, .graphml) files.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports igraph (>= 1.3.0), methods, rlang (>= 1.0.0), tools, xml2

Suggests openalexR (>= 2.0.0), testthat (>= 3.0.0)

Config/testthat/edition 3

URL <https://github.com/resendeph/mpaR>

BugReports <https://github.com/resendeph/mpaR/issues>

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Paulo H Resende [aut, cre]

Maintainer Paulo H Resende <paulo.resende@ttu.edu>

Repository CRAN

Date/Publication 2026-07-24 00:00:02 UTC

Contents

build_citation_network	2
check_dag	3

check_scale_free	4
classify_nodes	5
component_paths	7
edge_weights	8
fetch_openalex	9
key_route_sweep	11
main_path	12
mpa	15
node_weights	17
plot_mpa	18
read_gephi_export	21
read_pajek	22
traversal_weights	23

Index	25
--------------	-----------

build_citation_network

Build a citation network from fetched bibliographic data

Description

Converts a data frame of scholarly works (as returned by [fetch_openalex\(\)](#) or similar functions) into a directed igraph object representing the citation network.

An edge $A \rightarrow B$ means "paper A cites paper B". Under mpaR's citation-network convention (see [classify_nodes\(\)](#)):

- Old foundational papers accumulate in-edges and become "source" nodes.
- Recent papers not yet cited by others become "terminal" nodes.

After building the network, pass it to [check_dag\(\)](#) to verify acyclicity, then to [traversal_weights\(\)](#) and [mpa\(\)](#) to run Main Path Analysis.

Usage

```
build_citation_network(data, closed = TRUE)
```

Arguments

data	A data frame as returned by fetch_openalex() , containing at least the columns <code>id</code> and <code>referenced_works</code> .
closed	Logical. If TRUE (default), only citation links between papers present in data are included, producing a self-contained network. If FALSE, all referenced works become nodes even if they do not appear in data.

Value

A directed igraph object. Vertex attributes include `name` (the paper ID), `title`, `doi`, `year`, `first_author`, `journal`, and `cited_by_count` where available.

See Also

[fetch_openalex\(\)](#), [check_dag\(\)](#), [traversal_weights\(\)](#), [mpa\(\)](#)

Examples

```
## Not run:
papers <- fetch_openalex("main path analysis", n = 200)
g_raw <- build_citation_network(papers)

# Check for cycles and remove if needed
check_dag(g_raw)
g_dag <- igraph::delete_edges(g_raw, mpaR:::find_back_edges(g_raw))

# Run MPA
g <- traversal_weights(g_dag)
result <- mpa(g)
plot_mpa(g, result$global, scope = "full")

## End(Not run)
```

check_dag

Check DAG structure

Description

Validates that a directed graph is a directed acyclic graph (DAG). If cycles are present, the function identifies the back-edges responsible and reports them so the user can correct the underlying network data.

Cycles in citation or patent networks typically indicate data-entry errors, self-citations, or mutual-citation pairs. Because silently dropping edges would distort the network topology, it is necessary to review and correct the source data before proceeding.

Usage

```
check_dag(x, verbose = TRUE)
```

Arguments

x	An igraph object or a data frame / matrix with edge-list columns (from, to).
verbose	Logical. If TRUE (default), print a summary of the check result to the console.

Value

A list with elements:

`is_dag` Logical; TRUE if the graph is a valid DAG.

`cycle_edges` A data frame with columns `from` and `to` listing the back-edges that create cycles, or NULL if the graph is a DAG.

`n_cycle_edges` Integer; number of cycle-creating edges.

Examples

```
library(igraph)

# Clean DAG - should pass
el_clean <- data.frame(from = c(1, 2, 3), to = c(2, 3, 4))
check_dag(el_clean)

# Graph with a cycle: 1->2->3->1
el_cycle <- data.frame(
  from = c(1, 2, 3, 3),
  to   = c(2, 3, 1, 4)
)
result <- check_dag(el_cycle)
result$is_dag      # FALSE
result$cycle_edges # the offending edge(s) - fix these in your source data
```

check_scale_free	<i>Test whether a network has a scale-free degree distribution</i>
------------------	--

Description

Fits a discrete power-law distribution to the degree sequence of the graph object and reports whether the data are consistent with scale-free behaviour. Scale-free networks are characterised by a degree distribution that follows $P(k) \propto k^{-\gamma}$, where γ typically lies between 2 and 3 for empirical citation networks (this is used as a criterion for power-law, but the interpretation can be flexible).

The function uses `igraph::fit_power_law()` (a maximum-likelihood estimator) and optionally produces a log-log plot of the complementary cumulative degree distribution (CCDF) with the fitted power-law overlaid.

Usage

```
check_scale_free(x, mode = c("all", "in", "out"), xmin = NULL, plot = TRUE)
```

Arguments

x	An igraph graph or a data frame / matrix edge list.
mode	Degree type to test: "all" (default, total degree), "in", or "out". For citation networks "in" (number of citations received) is usually of most interest.
xmin	Numeric. Minimum degree value from which the power law is fitted. If NULL (default), the optimal $x_{\min}$ is estimated automatically.
plot	Logical. If TRUE (default), draw a log-log plot of the degree CCDF with the fitted power-law line.

Value

A list (invisibly) with elements:

alpha Estimated power-law exponent γ .

xmin The $x_{\min}$ used for fitting.

KS_stat Kolmogorov–Smirnov statistic.

KS_p KS p-value. A large p-value (e.g. > 0.05) means the power-law cannot be rejected.

is_scale_free Logical; TRUE if $\text{KS_p} > 0.05$ and $2 \leq \gamma \leq 5$.

degree_sequence Integer vector of degrees used.

References

Barabási, A.-L. (2016). *Network Science*. Cambridge University Press.

Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). Power-law distributions in empirical data. *SIAM Review*, **51**(4), 661–703. doi:10.1137/070710111

Examples

```
library(igraph)

# Barabási-Albert scale-free graph
g_sf <- igraph::sample_pa(500, directed = FALSE)
result <- check_scale_free(g_sf)
result$alpha
result$is_scale_free

# Erdős-Rényi random graph (not scale-free)
g_er <- igraph::sample_gnp(500, p = 0.02)
check_scale_free(g_er, plot = FALSE)
```

classify_nodes

Classify vertices by their role in the citation network

Description

Labels every vertex in a directed graph as one of four roles based on its in-degree and out-degree. There are two main nomenclatures:

Citation-network convention (default, convention = "citation"):

"source" Pioneer / precursor patents: receive citations but cite no one ($k^{in} > 0$, $k^{out} = 0$).

"terminal" Cutting-edge / most-recent patents: cite others but have not yet been cited ($k^{in} = 0$, $k^{out} > 0$).

"user" Intermediate patents: both cite and are cited ($k^{in} > 0$, $k^{out} > 0$).

"isolated" No citations in either direction ($k^{in} = 0$, $k^{out} = 0$).

Graph-theory convention (convention = "graph"):

"source" In-degree zero ($k^{in} = 0$).

"sink" Out-degree zero ($k^{out} = 0$).

"user" Both degrees positive.

"isolated" Both degrees zero.

Usage

```
classify_nodes(x, convention = c("citation", "graph"), as_data_frame = TRUE)
```

Arguments

x	An igraph directed graph or a data frame / matrix edge list.
convention	Character; "citation" (default) or "graph". See Description for the difference.
as_data_frame	Logical. If TRUE (default), return a data frame with one row per vertex. If FALSE, return a named character vector of labels indexed by vertex name.

Value

A data frame with columns name (vertex name), in_degree, out_degree, and role; or a named character vector if as_data_frame = FALSE.

Note

The two conventions assign *opposite* meanings to "source":

- In a patent citation network, edge direction is $A \rightarrow B$ meaning "A cites B". The oldest, foundational patent B accumulates in-citations but cites nothing, since it is the *source* of the knowledge flow, or the preceding node (timewise), hence called "source" in the citation convention even though it is a *sink* in graph-theory terms.
- Use convention = "graph" if you are working with a DAG where edge direction represents precedence or causality rather than citation.

References

Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:10.1016/03788733(89)900178

Liu, J. S., Lu, L. Y. Y., Lu, W. M., & Lin, B. J. Y. (2012). A survey of DEA applications. *Omega*, **41**(5), 893–902.

Examples

```
library(igraph)

el <- data.frame(
  from = c(1, 2, 2, 3, 4),
  to   = c(2, 3, 4, 5, 5)
)
```

```
# Citation-network convention (default)
classify_nodes(el)

# Graph-theory convention
classify_nodes(el, convention = "graph")

# As a named vector
classify_nodes(el, as_data_frame = FALSE)
```

component_paths	<i>Extract main paths from every component of a disconnected network</i>
-----------------	--

Description

Real-world citation networks (e.g. cross-sector patent networks) often consist of many weakly connected components rather than a single giant component. `component_paths()` decomposes the network into its weakly connected components, runs `traversal_weights()` and `main_path()` on each one independently, and returns the combined result.

Components smaller than `min_size` vertices are silently dropped, keeping only technologically significant sub-networks, similarly to the key-route selection strategy of Liu et al. (2019).

Usage

```
component_paths(
  x,
  type = c("global", "local", "key_route"),
  weight = c("SPC", "SPLC", "SPNP"),
  k = 1L,
  min_size = 3L
)
```

Arguments

<code>x</code>	An igraph DAG or a data frame / matrix edge list.
<code>type</code>	Main-path extraction strategy passed to <code>main_path()</code> : "global" (default), "local", or "key_route".
<code>weight</code>	Traversal weight: "SPC" (default), "SPLC", or "SPNP".
<code>k</code>	Number of key-route seed edges per component. Only used when <code>type = "key_route"</code> . Defaults to 1L.
<code>min_size</code>	Integer. Components with fewer than <code>min_size</code> vertices are excluded. Defaults to 3L.

Value

A named list of igraph subgraphs, one per retained component, named "component_1", "component_2", etc. (ordered by decreasing component size). Each subgraph contains only the main-path vertices and edges for that component, with the traversal weight as an edge attribute.

The full decomposition summary is attached as attribute "component_summary": a data frame with columns component, n_vertices, n_edges, and retained.

References

Liu, J. S., Chen, H. H., Ho, M. H. C., & Li, Y. C. (2012). Citations networks as a research tool. *Journal of the American Society for Information Science and Technology*, **63**(9).

Liu, P., Guo, Q., Yet, F., & Chen, X. (2019). Few-shot learning with key-route main paths. *Scientometrics*, **121**(3), 1437–1451.

Examples

```
library(igraph)

# Two disconnected chains: 1->2->3 and 4->5->6->7
el <- data.frame(
  from = c(1, 2, 4, 5, 6),
  to   = c(2, 3, 5, 6, 7)
)

paths <- component_paths(el, type = "global", weight = "SPC")
length(paths)      # 2 components
attr(paths, "component_summary")

# Inspect the larger component
igraph::as_edgelist(paths[["component_1"]])
```

edge_weights

Tabulate edge-level traversal weights

Description

Computes the SPC, SPLC, and/or SPNP traversal weights for every edge in a directed acyclic graph (via [traversal_weights\(\)](#)) and returns them as a tidy data frame.

Usage

```
edge_weights(x, method = "all")
```

Arguments

x	An igraph directed acyclic graph or a data frame / matrix whose first two columns are the edge endpoints (from, to).
method	Character vector; one or more of "SPC", "SPLC", "SPNP", or "all" (default). Case-insensitive.

Value

A data frame with columns from, to, and one column per requested weight (SPC, SPLC, SPNP).

References

Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:[10.1016/03788733\(89\)900178](https://doi.org/10.1016/03788733(89)900178)

See Also

[traversal_weights\(\)](#) to attach weights as edge attributes on the graph itself, [node_weights\(\)](#) for the node-level analogue.

Examples

```
library(igraph)

el <- data.frame(
  from = c(1, 2, 2, 3, 4),
  to   = c(2, 3, 4, 5, 5)
)
edge_weights(el)
edge_weights(el, method = "SPC")
```

fetch_openalex

Fetch works from OpenAlex

Description

Queries the OpenAlex API for scholarly works matching a search term, ORCID, journal, or DOI. Returns a standardised data frame that can be passed directly to [build_citation_network\(\)](#).

OpenAlex is a free, open database of scholarly works requiring no API key. It covers approximately 250 million works and includes full reference lists, making it suitable for constructing citation networks for Main Path Analysis.

Usage

```

fetch_openalex(
  query = NULL,
  field = c("title_abstract", "title"),
  n = 200,
  orcid = NULL,
  doi = NULL,
  journal = NULL,
  from_year = NULL,
  to_year = NULL,
  verbose = TRUE
)

```

Arguments

query	Character string. The search term (keyword, phrase, etc.). Ignored when orcid or doi is supplied.
field	Character; which part of the record to search. One of: "title_abstract" Title and abstract (default). "title" Title only.
n	Integer. Maximum number of works to retrieve. Defaults to 200. Large values may result in slow API calls.
orcid	Character. ORCID identifier (e.g. "0000-0002-1234-5678"). If supplied, retrieves all works by that author regardless of query.
doi	Character vector. One or more DOIs to retrieve directly.
journal	Character. Journal name to filter by (combined with query if both are supplied).
from_year	Integer. Restrict results to works published from this year onwards.
to_year	Integer. Restrict results to works published up to and including this year.
verbose	Logical. If TRUE (default), print progress messages.

Value

A data frame with columns:

id OpenAlex work ID (short form, e.g. "W2741809807").

title Paper title.

doi DOI.

year Publication year (integer).

first_author Display name of the first author.

journal Journal or source name.

abstract Abstract text.

cited_by_count Total citations received.

referenced_works List column of character vectors: OpenAlex IDs of works cited by each paper.
Used by [build_citation_network\(\)](#) to construct edges.

References

Priem, J., Piwowar, H., & Orr, R. (2022). OpenAlex: A fully-open index of the world's research works. <https://arxiv.org/abs/2205.01833>

See Also

`build_citation_network()`

Examples

```
## Not run:
# Keyword search (title + abstract)
papers <- fetch_openalex("main path analysis")

# Title-only search, limit to 100 results
papers <- fetch_openalex("Hayek", field = "title", n = 100)

# All works by a specific author (ORCID)
papers <- fetch_openalex(orcid = "0000-0002-1825-0097")

# Restrict by year range
papers <- fetch_openalex("citation network",
                        from_year = 2010, to_year = 2023)

# Full pipeline: fetch -> network -> MPA
papers <- fetch_openalex("main path analysis", n = 300)
g      <- build_citation_network(papers)
g      <- traversal_weights(g)
result <- mpa(g)
plot_mpa(g, result$global)

## End(Not run)
```

key_route_sweep

Sweep the key-route main path over multiple k values in one pass

Description

Computes the key-route main path for several values of k while sharing a single forward/backward dynamic-programming pass. The DP does not depend on k , so this is dramatically faster than calling `main_path()` in a loop when exploring how the path grows with k — the standard way to choose k is to look for a plateau in the number of nodes and arcs as k increases.

Usage

```
key_route_sweep(
  g,
  weight = NULL,
```

```

k_values = c(1, 10, 20, 30, 40, 50),
threshold = 1,
return_graphs = FALSE
)

```

Arguments

<code>g</code>	An igraph DAG carrying a traversal weight (see traversal_weights()).
<code>weight</code>	Character; which weight attribute to use. Defaults to the first of SPC, SPLC, SPNP present on <code>g</code> .
<code>k_values</code>	Integer vector of <code>k</code> values to evaluate.
<code>threshold</code>	Numeric in $(0, 1]$; broadens the seed set exactly as in main_path() .
<code>return_graphs</code>	Logical; if TRUE, the key-route subgraph for each <code>k</code> is attached as <code>attr(result, "graphs")</code> (a list aligned to the rows of the returned data frame).

Value

A data frame with columns `k`, `n_nodes`, `n_arcs`. For each `k`, these match [main_path\(type = "key_route", k = k\)](#).

See Also

[main_path\(\)](#)

Examples

```

library(igraph)
e1 <- data.frame(from = c(1,1,2,3,2,4), to = c(2,3,4,4,5,5))
g <- traversal_weights(e1, method = "SPC")
key_route_sweep(g, weight = "SPC", k_values = c(1, 2, 3))

```

main_path

Extract the main path(s) from a traversal-weighted DAG

Description

Given a DAG that already carries edge-traversal weights (see [traversal_weights\(\)](#)), extracts the main path(s) using one of three strategies:

- "global" – the single source-to-sink path whose cumulative edge weight is highest (longest-path dynamic programming on the DAG).
- "local" – a greedy forward search from *every* source: at each node follow the highest-weight outgoing edge until a sink is reached. Returns the union of all such paths.
- "key_route" – selects the top-*k* highest-weight edges (the *key routes*), then, for each key-route edge, finds the best incoming path from a source and the best outgoing path to a sink; the union of all these extended paths forms the key-route main path.

Broadening the path with threshold:

The classic algorithms above produce a single, narrow path. The threshold argument relaxes this to include near-optimal edges, producing a wider subgraph that captures alternative routes through the network. Set threshold to a value in $(0, 1)$:

- "global" – runs both a forward and a backward dynamic-programming pass. An edge $(u \rightarrow v, w)$ is included if the best source-to-sink path *through* that edge has total weight $\geq \text{threshold} \times dp^*$, where dp^* is the optimal total weight. At threshold = 0.8, all edges on paths within 80% optimal are included.
- "local" – at each node, follows every outgoing edge whose weight is $\geq \text{threshold} \times \max(\text{outgoing weights})$, instead of only the single best. Branches are explored via BFS until all reachable sinks are reached.
- "key_route" – expands the seed set to all edges with weight $\geq \text{threshold} \times w_k$, where w_k is the k -th highest weight. Useful when you want to pick k seeds but also catch edges of similar importance.

Usage

```
main_path(
    g,
    type = c("global", "local", "key_route"),
    weight = NULL,
    k = 1L,
    threshold = 1,
    k_range = NULL
)
```

Arguments

g	An igraph DAG with at least one of the edge attributes SPC, SPLC, or SPNP (produced by <code>traversal_weights()</code>).
type	Character; one of "global" (default), "local", or "key_route".
weight	Character; which traversal-weight edge attribute to use. Defaults to the first of SPC, SPLC, SPNP found on g.
k	Integer; number of key-route seed edges. Only used when type = "key_route". Defaults to 1L.
threshold	Numeric in $(0, 1]$. 1.0 (default) gives the classic algorithm. Values below 1.0 broaden the path by including near-optimal edges — see the Broadening section above.
k_range	Optional integer vector c(start, end); only used when type = "key_route". Seeds from the key routes ranked start to end by weight, instead of the top k. This mirrors Pajek's ability to compute a slice of key routes (e.g. c(5, 25)), which is useful for isolating the contribution of mid-tier routes or excluding the dominant trunk. Note that a slice which omits the top routes is a decomposition tool, not the canonical cumulative main path. When supplied, k_range takes precedence over k.

Value

An igraph subgraph containing only the vertices and edges that belong to the main path(s). The subgraph retains the vertex name attribute and the original traversal-weight edge attribute.

References

Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:[10.1016/03788733\(89\)900178](https://doi.org/10.1016/03788733(89)900178)

Garfield, E., Pudovkin, A. I., & Istomin, V. S. (2003). Why do we need algorithmic historiography? *Journal of the American Society for Information Science and Technology*, **54**(5), 400–412.

See Also

[mpa\(\)](#) for a one-call wrapper, [traversal_weights\(\)](#) to compute edge weights, [plot_mpa\(\)](#) to visualise the result.

Examples

```
library(igraph)

# Small diamond-shaped network (two competing routes from 1 to 5)
el <- data.frame(
  from = c(1, 1, 2, 3, 2, 4),
  to   = c(2, 3, 4, 4, 5, 5)
)
g_w <- traversal_weights(el, method = "SPC")

# --- Classic algorithms (threshold = 1.0, default) -----

# Global: single best source-to-sink path
mp_global <- main_path(g_w, type = "global", weight = "SPC")
igraph::as_edgelist(mp_global)
igraph::vcount(mp_global) # narrow – just the optimal chain

# Local: greedy from every source, union of all resulting paths
mp_local <- main_path(g_w, type = "local", weight = "SPC")
igraph::vcount(mp_local)

# Key-route: extend from the single highest-weight edge
mp_kr <- main_path(g_w, type = "key_route", weight = "SPC", k = 1L)

# Key-route with k = 3: seed from the top 3 edges
mp_kr3 <- main_path(g_w, type = "key_route", weight = "SPC", k = 3L)
igraph::vcount(mp_kr3)

# --- Broadened paths (threshold < 1.0) -----

# Global, broadened: include all edges on paths >= 80% of optimal weight
mp_broad <- main_path(g_w, type = "global", weight = "SPC", threshold = 0.8)
igraph::vcount(mp_broad) # more nodes than mp_global
```

```
# Local, broadened: at each node follow edges within 90% of the local max
mp_local_broad <- main_path(g_w, type = "local", weight = "SPC",
                           threshold = 0.9)

# Key-route, broadened: cast a wider seed net around k = 1
mp_kr_broad <- main_path(g_w, type = "key_route", weight = "SPC",
                        k = 1L, threshold = 0.7)

# Compare path sizes as threshold decreases
sizes <- sapply(c(1.0, 0.9, 0.8, 0.7, 0.5), function(t) {
  igraph::vcount(main_path(g_w, type = "global", weight = "SPC",
                          threshold = t))
})
names(sizes) <- c("1.0", "0.9", "0.8", "0.7", "0.5")
print(sizes)
```

mpa

Main Path Analysis

Description

A wrapper that coerces the input to an igraph DAG, computes the requested traversal weight, and extracts the main path(s). For finer control use [traversal_weights\(\)](#) and [main_path\(\)](#) separately.

Usage

```
mpa(
  x,
  type = c("global", "local", "key_route"),
  weight = c("SPC", "SPLC", "SPNP"),
  k = 1L,
  threshold = 1
)
```

Arguments

x	An igraph directed acyclic graph, or a data frame / matrix whose first two columns are the edge list (from, to).
type	Main-path extraction strategy: "global" (default), "local", or "key_route". See main_path() for details.
weight	Traversal weight to compute and use: "SPC" (default), "SPLC", or "SPNP".
k	Integer; number of key-route seed edges. Ignored unless type = "key_route". Defaults to 1L.
threshold	Numeric in (0, 1]. 1.0 (default) gives the classic single optimal path; lower values broaden the path to include near-optimal edges. See main_path() for a full description.

Value

An igraph subgraph of the main path(s), carrying the chosen traversal weight as an edge attribute.

References

Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:10.1016/03788733(89)900178

See Also

[main_path\(\)](#) for the lower-level function and full parameter documentation, [traversal_weights\(\)](#) to compute edge weights, [plot_mpa\(\)](#) to visualise results.

Examples

```
library(igraph)

# Hummon & Doreian (1989) toy network
el <- data.frame(
  from = c(1, 1, 2, 3, 3, 4, 5, 6),
  to   = c(2, 3, 4, 4, 5, 6, 6, 7)
)

# --- Extraction types (classic, threshold = 1.0) -----

# Global: single best source-to-sink path by SPC weight
mp_global <- mpa(el, type = "global", weight = "SPC")
igraph::as_edgelist(mp_global)

# Local: greedy from every source, returns the union of all greedy paths
mp_local <- mpa(el, type = "local", weight = "SPC")
igraph::vcount(mp_local)

# Key-route: top-2 seed edges extended to sources/sinks, using SPLC weights
mp_kr <- mpa(el, type = "key_route", weight = "SPLC", k = 2L)

# --- Broadened paths (threshold < 1.0) -----

# Include all global edges on paths within 80% of the optimal SPC weight
mp_broad <- mpa(el, type = "global", weight = "SPC", threshold = 0.8)
igraph::vcount(mp_broad) # wider than mp_global

# Local broadened: at each node follow all edges within 90% of local max
mp_local_broad <- mpa(el, type = "local", weight = "SPC", threshold = 0.9)

# Inspect how path size grows as threshold decreases
thresholds <- c(1.0, 0.9, 0.8, 0.7, 0.5)
sizes <- sapply(thresholds, function(t)
  igraph::vcount(mpa(el, type = "global", weight = "SPC", threshold = t))
)
data.frame(threshold = thresholds, nodes = sizes)
```

node_weights

*Tabulate node-level traversal weights***Description**

Computes the node-level analogue of the SPC, SPLC, and SPNP traversal weights: the number of source-to-sink paths passing **through** each vertex, rather than across a single edge. Using the same forward/backward path counts described in `traversal_weights()` — $f(v)$, $f_a(v)$, $b(v)$, $b_a(v)$ — the node-level weights are:

$$SPC(v) = f(v) \cdot b(v)$$

$$SPLC(v) = f_a(v) \cdot b(v)$$

$$SPNP(v) = f_a(v) \cdot b_a(v)$$

This is the direct generalization of the edge formula $SPC(i \rightarrow j) = f(i) \cdot b(j)$ to a single vertex (setting $i = j = v$), and counts every source-to-sink path that visits v , regardless of which edge it uses to arrive or leave.

Usage

```
node_weights(x, method = "all")
```

Arguments

x	An igraph directed acyclic graph or a data frame / matrix whose first two columns are the edge endpoints (from, to).
method	Character vector; one or more of "SPC", "SPLC", "SPNP", or "all" (default). Case-insensitive.

Value

A data frame with columns name and one column per requested weight (SPC, SPLC, SPNP).

References

Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:[10.1016/03788733\(89\)900178](https://doi.org/10.1016/03788733(89)900178)

See Also

`edge_weights()` for the edge-level table, `traversal_weights()` for the underlying edge-level computation.

Examples

```
library(igraph)

el <- data.frame(
  from = c(1, 2, 2, 3, 4),
  to   = c(2, 3, 4, 5, 5)
)
node_weights(el)
node_weights(el, method = "SPNP")
```

plot_mpa

Plot a network with the main path highlighted

Description

Plots a DAG and overlays the main path in a contrasting colour. The scope argument controls which nodes are drawn:

- "full" (default) — the entire graph, with main-path nodes/edges highlighted and the rest shown in a muted style.
- "component" — only the weakly connected component(s) that contain at least one main-path node. Useful for large, multi-component networks where most components are irrelevant.
- "path" — only the main-path subgraph itself (no background nodes).

The layout defaults to `igraph::layout_with_sugiyama()`, which respects the topological order of the DAG and is therefore well suited to citation and precedence networks.

Usage

```
plot_mpa(
  graph,
  path,
  scope = c("full", "component", "path"),
  path_col = "#E63946",
  bg_col = "#AAAAAA",
  path_label_col = "white",
  bg_label_col = "#555555",
  layout = NULL,
  vertex_size = 20,
  vertex_label_cex = 0.7,
  edge_width_path = 3,
  edge_width_bg = 1,
  arrow_size = 0.4,
  vertex_label_dist = 0,
  ...
)
```

Arguments

graph	An igraph DAG — typically the output of <code>traversal_weights()</code> .
path	An igraph subgraph representing the main path — the output of <code>main_path()</code> or <code>mpa()</code> .
scope	Character; one of "full" (default), "component", or "path". Controls which portion of the graph is drawn.
path_col	Colour for main-path nodes and edges. Defaults to "#E63946" (red).
bg_col	Colour for background (non-path) nodes and edges. Defaults to "#AAAAAA" (grey).
path_label_col	Colour for vertex labels on main-path nodes. Defaults to "white". Pass NA to suppress labels on path nodes.
bg_label_col	Colour for vertex labels on background nodes. Defaults to "#555555". Only visible when scope = "full" or "component".
layout	A numeric matrix of vertex coordinates ($V \times 2$). If NULL (default), <code>igraph::layout_with_sugiyama()</code> is used. When scope = "component" or "path", the layout is automatically subsetting to the plotted vertices.
vertex_size	Size of all vertices. Defaults to 20.
vertex_label_cex	Font size multiplier for vertex labels. Defaults to 0.7. Set to 0 to hide all labels.
edge_width_path	Line width for main-path edges. Defaults to 3.
edge_width_bg	Line width for background edges. Defaults to 1.
arrow_size	Arrow size passed to <code>igraph::plot.igraph()</code> . Defaults to 0.4.
vertex_label_dist	Distance between vertex centre and label, passed to <code>igraph::plot.igraph()</code> as <code>vertex.label.dist</code> . Defaults to 0 (label centred inside the node). Set to a positive value (e.g. 1.5) to push labels outside the node circles, which is often more readable in dense graphs.
...	Additional arguments forwarded to <code>igraph::plot.igraph()</code> . Common uses: main for a plot title, <code>vertex.label</code> to supply custom label strings (e.g. short names instead of node IDs).

Value

Invisibly returns a list with:

`layout` The full-graph coordinate matrix used.

`path_vertices` Integer indices (in graph) of main-path vertices.

`plotted_graph` The igraph object that was actually rendered (may be a subgraph when scope != "full").

Examples

```

library(igraph)

el <- data.frame(
  from = c(1, 1, 2, 3, 3, 4, 5, 6),
  to   = c(2, 3, 4, 4, 5, 6, 6, 7)
)
g <- traversal_weights(el)
mp <- main_path(g, type = "global", weight = "SPC")

# Full graph: background nodes grey, path highlighted in red
plot_mpa(g, mp)

# Only the weakly connected component(s) containing the path
plot_mpa(g, mp, scope = "component")

# Path nodes only – clearest view for large networks
plot_mpa(g, mp, scope = "path")

# Custom colours with visible labels
plot_mpa(g, mp, scope = "path",
  path_col      = "#1D3557",
  path_label_col = "white",
  vertex_size    = 18,
  vertex_label_cex = 0.6)

# Supply a title and custom label strings via ...
plot_mpa(g, mp, scope = "path",
  main      = "Global MPA – SPC",
  vertex.label = paste0("N", igraph::V(mp)$name))

# Dense or large networks: shrink nodes and push labels outside the circles
# so they remain readable. Pass the layout explicitly to avoid recomputing it.
lay <- igraph::layout_with_sugiyama(g)$layout
plot_mpa(g, mp,
  scope      = "full",
  layout     = lay,
  vertex_size = 5,
  vertex_label_cex = 0.8,
  vertex_label_dist = 1.5,
  path_label_col = "red",
  arrow_size    = 0.5)

# Broadened path: threshold = 0.8 pulls in near-optimal routes
mp_broad <- main_path(g, type = "global", weight = "SPC", threshold = 0.8)
plot_mpa(g, mp_broad, scope = "path",
  path_col      = "#457B9D",
  path_label_col = "white",
  main          = "Broadened MPA (threshold = 0.8)")

```

read_gephi_export	<i>Read a Gephi export file (GEXF or GraphML)</i>
-------------------	---

Description

Reads a graph exported from Gephi in GEXF (.gexf) or GraphML (.graphml / .xml) format and returns an igraph object.

GEXF files are parsed directly via their XML structure using the **xml2** package. GraphML files are read with `igraph::read_graph()`.

Gephi's native project format (.gephi) uses a proprietary binary serialisation that cannot be parsed reliably outside of Gephi. Use **File** → **Export** → **Graph File** in Gephi and choose .gexf or .graphml before calling this function.

Usage

```
read_gephi_export(file, directed = NULL)
```

Arguments

file	Path to a .gexf or .graphml file exported from Gephi.
directed	Logical or NULL. If NULL (default), the directionality declared in the file is respected. Set to TRUE or FALSE to override.

Value

An igraph object. For GEXF files, vertex attributes include name (node id), label (if present), x, y, size, and any <attvalue> columns defined in the file. Edge attributes include weight and any additional attributes.

Examples

```
## Not run:
# After exporting from Gephi: File > Export > Graph File > .gexf
g <- read_gephi_export("my_project.gexf")
igraph::vcount(g)
igraph::ecount(g)
igraph::vertex_attr_names(g)

# GraphML export
g <- read_gephi_export("my_project.graphml")

## End(Not run)
```

read_pajek	<i>Read a Pajek network file (.net)</i>
------------	---

Description

Parses a Pajek .net file and returns a directed or undirected igraph object. The following Pajek sections are supported:

- **Vertices* — vertex ids, optional labels, and optional x/y/z coordinates stored as vertex attributes.
- **Arcs* — directed edges with optional weights.
- **Edges* — undirected edges with optional weights (converted to a directed graph by adding both orientations when `directed = TRUE`).
- **Arcslist* / **Edgeslist* — adjacency-list variants of the above.

Lines beginning with % are treated as comments and ignored.

Usage

```
read_pajek(file, directed = TRUE, weight_attr = "weight")
```

Arguments

<code>file</code>	Path to a .net file.
<code>directed</code>	Logical. If TRUE (default), the returned graph is directed. <i>*Arcs</i> become directed edges; <i>*Edges</i> are duplicated in both directions. Set to FALSE to force an undirected graph.
<code>weight_attr</code>	Character. Name of the edge attribute used to store edge weights (default "weight"). Set to NULL to drop weights.

Value

An igraph graph with:

Vertex attributes name (label from file, or integer id if absent); x, y, z (coordinates, NA when not provided).

Edge attribute weight (numeric; 1 when absent from the file).

Examples

```
net_file <- system.file("extdata", "sample_network_pajek.net",
                        package = "mpaR")
g <- read_pajek(net_file)
igraph::vcount(g)
igraph::ecount(g)
head(igraph::V(g)$name)
```

traversal_weights

*Compute traversal weights for edges in a citation DAG***Description**

Calculates one or more of the three traversal-weight measures introduced by Hummon & Doreian (1989) for every edge in a directed acyclic graph (DAG):

- **SPC** – *Search Path Count*: the number of source-to-sink paths that traverse each edge. Proposed by Batagelj (2003).
- **SPLC** – *Search Path Link Count*: for edge $(i \rightarrow j)$, the number of paths from any ancestor of i (including i itself) to any sink that traverse the edge. Proposed by Hummon & Doreian (1989).
- **SPNP** – *Search Path Node Pair*: for edge $(i \rightarrow j)$, the number of paths from any ancestor of i (including i) to any descendant of j (including j) that traverse the edge. Proposed by Hummon & Doreian (1989).

The formulas reduce to:

$$SPC(i \rightarrow j) = f(i) \cdot b(j)$$

$$SPLC(i \rightarrow j) = f_a(i) \cdot b(j)$$

$$SPNP(i \rightarrow j) = f_a(i) \cdot b_a(j)$$

where $f(i)$ = paths from global sources to i ; $f_a(i) = 1 + \sum_{u \rightarrow i} f_a(u)$ (paths from any ancestor, including i itself); $b(j)$ = paths from j to global sinks; $b_a(j) = 1 + \sum_{j \rightarrow w} b_a(w)$.

All three measures are computed in a single forward-backward pass over the topological ordering, so the function is $O(V + E)$.

Usage

```
traversal_weights(x, method = "all")
```

Arguments

x	An igraph directed acyclic graph or a data frame / matrix whose first two columns are the edge endpoints (from, to). Extra columns become edge attributes.
method	Character vector; one or more of "SPC", "SPLC", "SPNP", or "all" (default). Case-insensitive.

Value

The input graph (as an igraph object) with the requested weight(s) attached as edge attributes (SPC, SPLC, SPNP).

References

- Hummon, N. P., & Doreian, P. (1989). Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**(1), 39–63. doi:[10.1016/03788733\(89\)900178](https://doi.org/10.1016/03788733(89)900178)
- Batagelj, V. (2003). Efficient algorithms for citation network analysis. *arXiv preprint* cs/0309023.

Examples

```
library(igraph)

# Small citation chain: 1 -> 2 -> 3 -> 5
#                               \-> 4 -> 5
el <- data.frame(
  from = c(1, 2, 2, 3, 4),
  to   = c(2, 3, 4, 5, 5)
)
g_w <- traversal_weights(el)
igraph::E(g_w)$SPC
```

Index

`build_citation_network`, 2
`build_citation_network()`, 9–11

`check_dag`, 3
`check_dag()`, 2, 3
`check_scale_free`, 4
`classify_nodes`, 5
`classify_nodes()`, 2
`component_paths`, 7

`edge_weights`, 8
`edge_weights()`, 17

`fetch_openalex`, 9
`fetch_openalex()`, 2, 3

`igraph::fit_power_law()`, 4
`igraph::layout_with_sugiyama()`, 18, 19
`igraph::plot.igraph()`, 19

`key_route_sweep`, 11

`main_path`, 12
`main_path()`, 7, 11, 12, 15, 16, 19
`mpa`, 15
`mpa()`, 2, 3, 14, 19

`node_weights`, 17
`node_weights()`, 9

`plot_mpa`, 18
`plot_mpa()`, 14, 16

`read_gephi_export`, 21
`read_pajek`, 22

`traversal_weights`, 23
`traversal_weights()`, 2, 3, 7–9, 12–17, 19