Package ‘netrankr’

December 19, 2023

Type Package

Title Analyzing Partial Rankings in Networks

Version 1.2.3

Description Implements methods for centrality related analyses of networks. While the package includes the possibility to build more than 20 indices, its main focus lies on index-free assessment of centrality via partial rankings obtained by neighborhood-inclusion or positional dominance. These partial rankings can be analyzed with different methods, including probabilistic methods like computing expected node ranks and relative rank probabilities (how likely is it that a node is more central than another?). The methodology is described in depth in the vignettes and in Schoch (2018) <doi:10.1016/j.socnet.2017.12.003>.


BugReports https://github.com/schochastics/netrankr/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.0.1)

Imports igraph (>= 1.0.1), Rcpp (>= 0.12.8), Matrix

LinkingTo Rcpp,RcppArmadillo

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, magrittr, testthat, shiny (>= 0.13), miniUI (>= 0.1.1), rstudioapi (>= 0.5), covr

VignetteBuilder knitr

NeedsCompilation yes

Author David Schoch [aut, cre] (<https://orcid.org/0000-0003-2952-4812>), Julian Müller [ctb]

Maintainer David Schoch <david@schochastics.net>

Repository CRAN

Date/Publication 2023-12-19 13:10:02 UTC
**aggregate_positions**

**R topics documented:**

- aggregate_positions
- approx_rank_expected
- approx_rank_relative
- as.matrix.netrankr_full
- comparable_pairs
- compare_ranks
- dbces11
- dominance_graph
- exact_rank_prob
- florentine_m
- get_rankings
- hyperbolic_index
- incomparable_pairs
- index_builder
- indirect_relations
- is_preserved
- majorization_gap
- mcmc_rank_prob
- neighborhood_inclusion
- plot.netrankr_full
- plot.netrankr_interval
- plot.netrankr_mcmc
- plot_rank_intervals
- positional_dominance
- print.netrankr_full
- print.netrankr_interval
- print.netrankr_mcmc
- rank_intervals
- spectral_gap
- summary.netrankr_full
- threshold_graph
- transform_relations
- transitive_reduction

**Index**

<table>
<thead>
<tr>
<th>aggregate_positions</th>
<th>Quantification of (indirect) relations</th>
</tr>
</thead>
</table>

**Description**

Function to aggregate positions defined via indirect relations to construct centrality scores.

**Usage**

```r
aggregate_positions(tau_x, type = "sum")
```
aggregate_positions

Arguments

- **tau_x**: Numeric matrix containing indirect relations calculated with `indirect_relations`.
- **type**: String indicating the type of aggregation to be used. See Details for options.

Details

The predefined functions are mainly wrappers around base R functions. `type='sum'`, for instance, is equivalent to `rowSums()`. A non-base function is `type='invsum'` which calculates the inverse of `type='sum'`. `type='self'` is mostly useful for walk-based relations, e.g., to count closed walks. Other self-explanatory options are `type='mean'`, `type='min'`, `type='max'` and `type='prod'`.

Value

Scores for the index defined by the indirect relation `tau_x` and the used aggregation type.

Author(s)

David Schoch

See Also

`indirect_relations`, `transform_relations`

Examples

```r
library(igraph)
library(magrittr)

data("dbces11")
# degree
dbces11 %>%
  indirect_relations(type = "adjacency") %>%
  aggregate_positions(type = "sum")

# closeness centrality
dbces11 %>%
  indirect_relations(type = "dist_sp") %>%
  aggregate_positions(type = "invsum")

# betweenness centrality
dbces11 %>%
  indirect_relations(type = "depend_sp") %>%
  aggregate_positions(type = "sum")

# eigenvector centrality
dbces11 %>%
  indirect_relations(type = "walks", FUN = walks_limit_prop) %>%
  aggregate_positions(type = "sum")

# subgraph centrality
dbces11 %>%
```
approx_rank_expected

\[
\text{indirect_relations(type = "walks", FUN = walks_exp) \%>\%} \\
\text{aggregate_positions(type = "self")}
\]

approx_rank_expected  Approximation of expected ranks

Description

Implements a variety of functions to approximate expected ranks for partial rankings.

Usage

approx_rank_expected(P, method = "lpom")

Arguments

- **P**: A partial ranking as matrix object calculated with \textit{neighborhood\_inclusion} or \textit{positional\_dominance}.
- **method**: String indicating which method to be used. see Details.

Details

The \textit{method} parameter can be set to:

- **lpom**: local partial order model
- **glpom**: extension of the local partial order model.
- **loof1**: based on a connection with relative rank probabilities.
- **loof2**: extension of the previous method.

Which of the above methods performs best depends on the structure and size of the partial ranking. See vignette("benchmarks",package="netrankr") for more details.

Value

A vector containing approximated expected ranks.

Author(s)

David Schoch

References


approx_rank_relative

See Also

approx_rank_relative, exact_rank_prob, mcmc_rank_prob

Examples

```r
P <- matrix(c(0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, rep(0, 10)), 5, 5, byrow = TRUE)
# Exact result
exact_rank_prob(P)$expected.rank

approx_rank_expected(P, method = "lpom")
approx_rank_expected(P, method = "glpom")
```

Description

Approximate relative rank probabilities $P(rk(u) < rk(v))$. In a network context, $P(rk(u) < rk(v))$ is the probability that u is less central than v, given the partial ranking P.

Usage

```r
approx_rank_relative(P, iterative = TRUE, num.iter = 10)
```

Arguments

- **P**
  - A partial ranking as matrix object calculated with `neighborhood_inclusion` or `positional_dominance`.
- **iterative**
  - Logical scalar if iterative approximation should be used.
- **num.iter**
  - Number of iterations to be used. Defaults to 10 (see Details).

Details

The iterative approach generally gives better approximations than the non iterative, if only slightly. The default number of iterations is based on the observation, that the approximation does not improve significantly beyond this value. This observation, however, is based on very small networks such that increasing it for large network may yield better results. See vignette("benchmarks",package="netrankr") for more details.

Value

A matrix containing approximation of relative rank probabilities. `relative.rank[i,j]` is the probability that i is ranked lower than j.

Author(s)

David Schoch
References


See Also

approx_rank_expected, exact_rank_prob, mcmc_rank_prob

Examples

```R
P <- matrix(c(0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, rep(0, 10)), 5, 5, byrow = TRUE)
approx_rank_relative(P, iterative = FALSE)
approx_rank_relative(P, iterative = TRUE)
```

as.matrix.netrankr_full

*Extract probabilities from netrankr_full object*

Description

extract probabilities as matrices from the result of an object obtained from `exact_rank_prob`

Usage

```R
## S3 method for class 'netrankr_full'
as.matrix(x, type = "rank", ...)
```

Arguments

- `x` A netrankr_full object
- `type` which probabilities to return. "rank" for rank probabilities, "relative" for relative rank probabilities and "expected" for expected rank probabilities and their variants
- `...` additional parameters for as.matrix

Author(s)

David Schoch
comparable_pairs

Comparable pairs in a partial order

Description

Calculates the fraction of comparable pairs in a partial order.

Usage

comparable_pairs(P)

Arguments

P A partial order as matrix object, e.g. calculated with neighborhood_inclusion or positional_dominance.

Value

Fraction of comparable pairs in P.

Author(s)

David Schoch

See Also

incomparable_pairs

Examples

library(igraph)
g <- sample_gnp(100, 0.1)
P <- neighborhood_inclusion(g)
comparable_pairs(P)
# All pairs of vertices are comparable in a threshold graph
tg <- threshold_graph(100, 0.3)
P <- neighborhood_inclusion(g)
comparable_pairs(P)
compare_ranks  

Count occurrences of pairs in rankings

Description

Counts the number of concordant, discordant and (left/right) ties between two rankings.

Usage

```r
compare_ranks(x, y)
```

Arguments

- `x` A numeric vector.
- `y` A numeric vector with the same length as `x`.

Details

Explicitly calculating the number of occurring cases is more robust than using correlation indices as given in the `cor` function. Especially left and right ties can significantly alter correlations.

Value

A list containing

- `concordant` number of concordant pairs: \( x[i] > x[j] \) and \( y[i] > y[j] \)
- `discordant` number of discordant pairs: \( x[i] > x[j] \) and \( y[i] < y[j] \)
- `ties` number of tied pairs: \( x[i] == x[j] \) and \( y[i] == y[j] \)
- `left` number of left ties: \( x[i] == x[j] \) and \( y[i] != y[j] \)
- `right` number of right ties: \( x[i] != x[j] \) and \( y[i] == y[j] \)

Author(s)

David Schoch

Examples

```r
library(igraph)
tg <- threshold_graph(100, 0.2)
compare_ranks(degree(tg), closeness(tg)) # only concordant pairs
compare_ranks(degree(tg), betweenness(tg)) # no discordant pairs
## Rank Correlation
      cor(degree(tg), closeness(tg), method = "kendall") # 1
      cor(degree(tg), betweenness(tg), method = "kendall") # not 1, although no discordant pairs
```
**dbces11**

**dbces11 graph**

**Description**

Smallest graph (11 nodes and 17 edges) where the centers according to (d)egree, (b)etweenness, (c)loseness, (e)igenvector centrality, and (s)ubgraph centrality are all different.

**Usage**

dbces11

**Format**

igraph object

**dominance_graph**

Partial ranking as directed graph

**Description**

Turns a partial ranking into a directed graph. An edge (u,v) is present if $P[u,v]=1$, meaning that u is dominated by v.

**Usage**

dominance_graph(P)

**Arguments**

P
A partial ranking as matrix object calculated with neighborhood_inclusion or positional_dominance.

**Value**

Directed graph as an igraph object.

**Author(s)**

David Schoch
Examples

```r
library(igraph)
g <- threshold_graph(20, 0.1)
P <- neighborhood_inclusion(g)
d <- dominance_graph(P)
## Not run:
plot(d)
## End(Not run)

# to reduce overplotting use transitive reduction
P <- transitive_reduction(P)
d <- dominance_graph(P)
## Not run:
plot(d)
## End(Not run)
```

exact_rank_prob

**probabilistic centrality rankings**

Description

Performs a complete and exact rank analysis of a given partial ranking. This includes rank probabilities, relative rank probabilities and expected ranks.

Usage

```r
exact_rank_prob(P, only.results = TRUE, verbose = FALSE, force = FALSE)
```

Arguments

- `P` A partial ranking as matrix object calculated with `neighborhood_inclusion` or `positional_dominance`.
- `only.results` Logical. return only results (default) or additionally the ideal tree and lattice if FALSE.
- `verbose` Logical. should diagnostics be printed. Defaults to FALSE.
- `force` Logical. If FALSE (default), stops the analysis if the partial ranking has more than 40 elements and less than 0.4 comparable pairs. Only change if you know what you are doing.

Details

The function derives rank probabilities from a given partial ranking (for instance returned by `neighborhood_inclusion` or `positional_dominance`). This includes the calculation of expected ranks, (relative) rank probabilities and the number of possible rankings. Note that the set of rankings grows exponentially in the number of elements and the exact calculation becomes infeasible quite quickly and approximations need to be used. See vignette("benchmarks") for guidelines and `approx_rank_relative`, `approx_rank_expected`, and `mcmc_rank_prob` for approximative methods.
Value

- **lin.ext**: Number of possible rankings that extend \( P \).
- **mse**: Array giving the equivalence classes of \( P \).
- **rank.prob**: Matrix containing rank probabilities: \( \text{rank.prob}[u,k] \) is the probability that \( u \) has rank \( k \).
- **relative.rank**: Matrix containing relative rank probabilities: \( \text{relative.rank}[u,v] \) is the probability that \( u \) is ranked lower than \( v \).
- **expected.rank**: Expected ranks of nodes in any centrality ranking.
- **rank.spread**: Standard deviation of the ranking probabilities.
- **topo.order**: Random ranking used to build the lattice of ideals (if \( \text{only.results} = \text{FALSE} \)).
- **tree**: Adjacency list (incoming) of the tree of ideals (if \( \text{only.results} = \text{FALSE} \)).
- **lattice**: Adjacency list (incoming) of the lattice of ideals (if \( \text{only.results} = \text{FALSE} \)).
- **ideals**: List of order ideals (if \( \text{only.results} = \text{FALSE} \)).

In all cases, higher numerical ranks imply a higher position in the ranking. That is, the lowest ranked node has rank 1.

Author(s)

David Schoch, Julian Müller

References


See Also

- `approx_rank_relative`, `approx_rank_expected`, `mcmc_rank_prob`

Examples

```r
P <- matrix(c(0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, rep(0, 10)), 5, 5, byrow = TRUE)
res <- exact_rank_prob(P)

# a warning is displayed if only one ranking is possible
tg <- threshold_graph(20, 0.2)
P <- neighborhood_inclusion(tg)
res <- exact_rank_prob(P)
```
### florentine_m  
*Florentine family marriage network*

#### Description
Florentine family marriage network

#### Usage
```r
florentine_m
```

#### Format
An igraph object containing marriage links of florentine families

#### References

---

### get_rankings  
*Rankings that extend a partial ranking*

#### Description
Returns all possible rankings that extend a partial ranking.

#### Usage
```r
get_rankings(data, force = FALSE)
```

#### Arguments
- **data**: List as returned by `exact_rank_prob` when run with `only.results = FALSE`
- **force**: Logical scalar. Stops function if the number of rankings is too large. Only change to TRUE if you know what you are doing

#### Details
The $i$th row of the matrix contains the rank of node $i$ in all possible rankings that are in accordance with the partial ranking $P$. The lowest rank possible is associated with 1.

#### Value
A matrix containing ranks of nodes in all possible rankings.
**hyperbolic_index**

**Author(s)**
David Schoch

**Examples**

```r
P <- matrix(c(0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, rep(0, 10)), 5, 5, byrow = TRUE)
res <- exact_rank_prob(P, only.results = FALSE)
get_rankings(res)
```

---

**Description**

The hyperbolic index is an index that considers all closed walks of even or odd length on induced neighborhoods of a vertex.

**Usage**

```r
hyperbolic_index(g, type = "odd")
```

**Arguments**

- `g` igraph object.
- `type` string. 'even' if only even length walks should be considered. 'odd' (Default) if only odd length walks should be used.

**Details**

The hyperbolic index is an illustrative index that should not be used for any serious analysis. Its purpose is to show that with enough mathematical trickery, any desired result can be obtained when centrality indices are used.

**Value**

A vector containing centrality scores.

**Author(s)**
David Schoch

**Examples**

```r
library(igraph)
data("dbces11")
hyperbolic_index(dbces11, type = "odd")
hyperbolic_index(dbces11, type = "even")
```
incomparable_pairs (Incomparable pairs in a partial order)

Description

Calculates the fraction of incomparable pairs in a partial order.

Usage

incomparable_pairs(P)

Arguments

P A partial order as matrix object, e.g. calculated with neighborhood_inclusion or positional_dominance.

Value

Fraction of incomparable pairs in P.

Author(s)

David Schoch

See Also

comparable_pairs

Examples

library(igraph)
g <- sample_gnp(100, 0.1)
P <- neighborhood_inclusion(g)
comparable_pairs(P)
# All pairs of vertices are comparable in a threshold graph
tg <- threshold_graph(100, 0.3)
P <- neighborhood_inclusion(g)
comparable_pairs(P)
index_builder  

**Centrality Index Builder**

**Description**

This shiny gadget can be used to build centrality indices based on specific indirect relations, transformations and aggregation functions. Use the dropdown menus to select components that make up the index. Depending on your choices, some options are not available at later stages. At the end, code is being inserted into the current script to use the index.

**Usage**

```
index_builder()
```

**Value**

code to calculate the specified index.

indirect_relations  

**Indirect relations in a network**

**Description**

Derive indirect relations for a given network. Observed relations, like presents or absence of a relation, are commonly not the center of analysis, but are transformed in a new set of indirect relation like shortest path distances among nodes. These transformations are usually an implicit step when centrality indices are used. Making this step explicit gives more possibilities, for example calculating partial centrality rankings with `positional_dominance`.

**Usage**

```
indirect_relations(
    g,
    type = "dist_sp",
    ltparam = NULL,
    dwparam = NULL,
    netflowmode = "",
    rpxparam = NULL,
    FUN = identity,
    ...)
```
Arguments

- **g**: igraph object. The network for which relations should be derived.
- **type**: String giving the relation to be calculated. See Details for options.
- **lfparam**: Numeric parameter. Only used if type = "dist_lf".
- **dwparam**: Numeric parameter. Only used if type = "dist_walk".
- **netflowmode**: String, one of raw, frac, or norm. Only used if type = "depend_netflow".
- **rspxparam**: Numeric parameter. Only used if type = "depend_rsp" or type = "depend_rspn".
- **FUN**: A function that allows the transformation of relations. See Details.
- **...**: Additional arguments passed to FUN.

Details

The **type** parameter has the following options.

- **'adjacency'** returns the adjacency matrix of the network.
- **'weights'** returns the weighted adjacency matrix of the network if an edge attribute 'weight' is present.
- **'dist_sp'** returns shortest path distances between all pairs of nodes.
- **'depend_sp'** returns dyadic dependencies
  \[
  \delta(u, s) = \sum_{t \in V} \frac{\sigma(s, t|u)}{\sigma(s, t)}
  \]
  where \( \sigma(s, t|u) \) is the number of shortest paths from \( s \) to \( t \) that include \( u \) and \( \sigma(s, t) \) is the total number of shortest \((s,t)\)-paths. This relation is used for betweenness-like centrality indices.
- **'walks'** returns walk counts between pairs of nodes, usually they are weighted decreasingly in their lengths or other properties which can be done by adding a function in **FUN**. See **transform_relations** for options.
- **'dist_resist'** returns the resistance distance between all pairs of nodes.
- **'dist_lf'** returns a logarithmic forest distance \( d_\alpha(s, t) \). The logarithmic forest distances form a one-parametric family of distances, converging to shortest path distances as \( \alpha \to 0 \) and to the resistance distance as \( \alpha \to \infty \). See (Chebotarev, 2011) for more details. The parameter \( \text{lfparam} \) can be used to tune \( \alpha \).
- **'dist_walk'** returns the walk distance \( d_\alpha^W(s, t) \) between nodes. The walk distances form a one-parametric family of distances, converging to shortest path distances as \( \alpha \to 0 \) and to longest walk distances for \( \alpha \to \infty \). Walk distances contain the logarithmic forest distances as a special case. See (Chebotarev, 2012) for more details.
- **'dist_rwalk'** returns the expected length of a random walk between two nodes. For more details see (Noh and Rieger, 2004)
- **'depend_netflow'** returns dependencies based on network flow (See Freeman et al.,1991). If **netflowmode="raw"**, the function returns
  \[
  \delta(u, s) = \sum_{t \in V} f(s, t, G) - f(s, t, G - v)
  \]
where $f(s,t,G)$ is the maximum flow from $s$ to $t$ in $G$ and $f(s,t,G-v)$ in $G$ without the node $v$. For `netflowmode="frac"` it returns dependencies in the form, similar to shortest path dependencies:

$$
\delta(u,s) = \sum_{t \in V} \frac{f(s,t,G) - f(s,t,G-v)}{f(s,t,G)}
$$

'depend_curflow' returns pairwise dependencies based on current flow. The relation is based on the same idea as 'depend_sp' and 'depend_netflow'. However, instead of considering shortest paths or network flow, the current flow (or equivalent: random walks) between nodes are of interest. See (Newman, 2005) for details.

'depend_exp' returns pairwise dependencies based on 'communicability':

$$
\delta(u,s) = \sum_{t \in V} \frac{\exp(A)_{st} - \exp(A+E(u))_{st}}{\exp(A)_{st}},
$$

where $E(u)$ has nonzeros only in row and column $u$, and in this row and column has -1 if $A$ has +1. See (Estrada et al., 2009) for additional details.

'depend_rsps'. Simple randomized shortest path dependencies. The simple RSP dependency of a node $u$ with respect to absorbing paths from $s$ to $t$, is defined as the expected number of visits through $u$ over all s-t-walks. The parameter $rspxparam$ is the "inverse temperature parameter". If it converges to infinity, only shortest paths are considered and the expected number of visits to a node on a shortest path approaches the probability of following that particular path. When the parameter converges to zero, then the dependencies converge to the expected number of visits to a node over all absorbing walks with respect to the unbiased random walk probabilities. This means for undirected networks, that the relations converge to adjacency. See (Kivimäki et al., 2016) for details.

'depend_rspn' Net randomized shortest path dependencies. The parameter $rspxparam$ is the "inverse temperature parameter". The asymptotic for the infinity case are the same as for 'depend_rsps'. If the parameter approaches zero, then it converges to 'depend_curflow'. The net randomized shortest path dependencies are closely related to the random walk interpretation of current flows. See (Kivimäki et al., 2016) for technical details.

The function `FUN` is used to transform the indirect relation. See `transform_relations` for predefined functions and additional help.

**Value**

A matrix containing indirect relations in a network.

**Author(s)**

David Schoch

**References**


See Also

aggregate_positions to build centrality indices, positional_dominance to derive dominance relations

Examples

```r
library(igraph)

data("dbces11")

# shortest path distances
D <- indirect_relations(dbces11, type = "dist_sp")

# inverted shortest path distances
D <- indirect_relations(dbces11, type = "dist_sp", FUN = dist_inv)

# shortest path dependencies (used for betweenness)
D <- indirect_relations(dbces11, type = "depend_sp")

# walks attenuated exponentially by their length
W <- indirect_relations(dbces11, type = "walks", FUN = walks_exp)
```

---

**is_preserved**

*Check preservation*

**Description**

Checks if a partial ranking is preserved in the ranking induced by scores.

**Usage**

```r
is_preserved(P, scores)
```

**Arguments**

- **P**
  - A partial ranking as matrix object calculated with neighborhood_inclusion or positional_dominance.
- **scores**
  - Numeric vector containing the scores of a centrality index.
**majorization_gap**

**Details**

In order for a score vector to preserve a partial ranking, the following condition must be fulfilled: \( P[u,v] = 1 \) & \( \text{scores}[i] \leq \text{scores}[j] \).

**Value**

Logical scaler whether \( \text{scores} \) preserves the relations in \( P \).

**Author(s)**

David Schoch

**Examples**

```r
library(igraph)
# standard measures of centrality preserve the neighborhood inclusion preorder
data("dbces11")
P <- neighborhood_inclusion(dbces11)

is_preserved(P, degree(dbces11))
is_preserved(P, betweenness(dbces11))
is_preserved(P, closeness(dbces11))
```

**Description**

Calculates the (normalized) majorization gap of an undirected graph. The majorization gap indicates how far the degree sequence of a graph is from a degree sequence of a threshold_graph.

**Usage**

```r
majorization_gap(g, norm = TRUE)
```

**Arguments**

- `g`: An igraph object
- `norm`: True (Default) if the normalized majorization gap should be returned.
Details

The distance is measured by the number of reverse unit transformations necessary to turn the degree sequence into a threshold sequence. First, the corrected conjugated degree sequence \(d'\) is calculated from the degree sequence \(d\) as follows:

\[
d'_k = |\{i : i < k \land d_i \geq k - 1\}| + |\{i : i > k \land d_i \geq k\}|.
\]

the majorization gap is then defined as

\[
\frac{1}{2} \sum_{k=1}^{n} \max\{d'_k - d_k, 0\}
\]

The higher the value, the further away is a graph to be a threshold graph.

Value

Majorization gap of an undirected graph.

Author(s)

David Schoch

References


Examples

library(igraph)
g <- graph.star(5, "undirected")majorization_gap(g) # 0 since star graphs are threshold graphs

g <- sample_gnp(100, 0.15)
majorization_gap(g, norm = TRUE) # fraction of reverse unit transformation
majorization_gap(g, norm = FALSE) # number of reverse unit transformation

mcmc_rank_prob

Estimate rank probabilities with Markov Chains

Description

Performs a probabilistic rank analysis based on an almost uniform sample of possible rankings that preserve a partial ranking.
mcmc_rank_prob

Usage

```r
mcmc_rank_prob(P, rp = nrow(P)^3)
```

Arguments

- `P`: A partial ranking as matrix object calculated with `neighborhood_inclusion` or `positional_dominance`.
- `rp`: Integer indicating the number of samples to be drawn.

Details

This function can be used instead of `exact_rank_prob` if the number of elements in `P` is too large for an exact computation. As a rule of thumb, the number of samples should be at least cubic in the number of elements in `P`. See vignette("benchmarks",package="netrankr") for guidelines and benchmark results.

Value

- `expected.rank`: Estimated expected ranks of nodes
- `relative.rank`: Matrix containing estimated relative rank probabilities: `relative.rank[u,v]` is the probability that `u` is ranked lower than `v`.

Author(s)

David Schoch

References


See Also

`exact_rank_prob, approx_rank_relative, approx_rank_expected`

Examples

```r
## Not run:
data("florentine_m")
P <- neighborhood_inclusion(florentine_m)
res <- exact_rank_prob(P)
mcmc <- mcmc_rank_prob(P, rp = vcount(g)^3)

# mean absolute error (expected ranks)
mean(abs(res$expected.rank - mcmc$expected.rank))

## End(Not run)
```
neighborhood_inclusion

*Neighborhood-inclusion preorder*

**Description**
Calculates the neighborhood-inclusion preorder of an undirected graph.

**Usage**
```r
eighborhood_inclusion(g, sparse = FALSE)
```

**Arguments**
- `g` An igraph object
- `sparse` Logical scalar, whether to create a sparse matrix

**Details**
Neighborhood-inclusion is defined as

\[ N(u) \subseteq N[v] \]

where \( N(u) \) is the neighborhood of \( u \) and \( N[v] = N(v) \cup \{v\} \) is the closed neighborhood of \( v \). \( N(u) \subseteq N[v] \) implies that \( c(u) \leq c(v) \), where \( c \) is a centrality index based on a specific path algebra. Indices falling into this category are closeness (and variants), betweenness (and variants) as well as many walk-based indices (eigenvector and subgraph centrality, total communicability,...).

**Value**
The neighborhood-inclusion preorder of \( g \) as matrix object. \( P[u,v]=1 \) if \( N(u) \subseteq N[v] \)

**Author(s)**
David Schoch

**References**


**See Also**
- positional_dominance, exact_rank_prob
Examples

```r
library(igraph)
# the neighborhood inclusion preorder of a star graph is complete
g <- graph.star(5, "undirected")
P <- neighborhood_inclusion(g)
comparable_pairs(P)

# the same holds for threshold graphs
tg <- threshold_graph(50, 0.1)
P <- neighborhood_inclusion(tg)
comparable_pairs(P)

# standard centrality indices preserve neighborhood-inclusion
data("dbces11")
P <- neighborhood_inclusion(dbces11)

is_preserved(P, degree(dbces11))
is_preserved(P, closeness(dbces11))
is_preserved(P, betweenness(dbces11))
```

---

**plot.netrankr_full**  
*Plot netrankr_full object*

**Description**

Plots the result of an object obtained from `exact_rank_prob`

**Usage**

```r
## S3 method for class 'netrankr_full'
plot(x, icols = NULL, bcol = "grey66", ecol = "black", ...)
```

**Arguments**

- `x` A netrankr_full object
- `icols` a list of colors (an internal palette is used if missing)
- `bcol` color used for the barcharts
- `ecol` color used for errorbars
- `...` additional plot parameters

**Author(s)**

David Schoch
plot.netrankr_interval

*plot netrankr_interval objects*

**Description**

Plots results from `rank_intervals`

**Usage**

```r
## S3 method for class 'netrankr_interval'
plot(x, cent_scores = NULL, cent_cols = NULL, ties.method = "min", ...)
```

**Arguments**

- `x`: A netrank object
- `cent_scores`: A data frame containing centrality scores of indices (optional)
- `cent_cols`: colors for centrality indices. If NULL a default palette is used. Length must be equal to columns in `cent_scores`.
- `ties.method`: how to treat ties in the rankings. see `rank` for details
- `...`: additional arguments to `plot`

**Author(s)**

David Schoch

---

plot.netrankr_mcmc

*Plot netrankr_mcmc object*

**Description**

Plots the result of an object obtained from `mcmc_rank_prob`

**Usage**

```r
## S3 method for class 'netrankr_mcmc'
plot(x, icols = NULL, bcol = "grey66", ...)
```

**Arguments**

- `x`: A netrank_mcmc object
- `icols`: a list of colors (an internal)
- `bcol`: color used for the barcharts
- `...`: additional plot parameters
plot_rank_intervals

Author(s)
David Schoch

Description
This function is deprecated. Use `plot(rank_intervals(P))` instead.

Usage

```
plot_rank_intervals(P, cent.df = NULL, ties.method = "min")
```

Arguments

- `P`: A partial ranking as matrix object calculated with `neighborhood_inclusion` or `positional_dominance`.
- `cent.df`: A data frame containing centrality scores of indices (optional). See Details.
- `ties.method`: String specifying how ties are treated in the base `rank` function.

Author(s)
David Schoch

See Also
`rank_intervals`

Examples

```
library(igraph)
data("dbces11")
P <- neighborhood_inclusion(dbces11)
## Not run:
plot_rank_intervals(P)
## End(Not run)

# adding index based rankings
cent_scores <- data.frame(  
  degree = degree(dbces11),  
  betweenness = round(betweenness(dbces11), 4),  
  closeness = round(closeness(dbces11), 4),  
  eigenvector = round(eigen_centrality(dbces11)$vector, 4)
)
## Not run:
plot_rank_intervals(P, cent.df = cent_scores)
## End(Not run)
```
Generalized Dominance Relations

Description

Generalized dominance relations that can be computed on one and two mode networks.

Usage

`positional_dominance(A, type = "one-mode", map = FALSE, benefit = TRUE)`

Arguments

- **A**: Matrix containing attributes or relations, for instance calculated by `indirect_relations`.
- **type**: A string which is either 'one-mode' (Default) if `A` is a regular one-mode network or 'two-mode' if `A` is a general data matrix.
- **map**: Logical scalar, whether rows can be sorted or not (Default). See Details.
- **benefit**: Logical scalar, whether the attributes or relations are benefit or cost variables.

Details

Positional dominance is a generalization of neighborhood-inclusion for arbitrary network data. In the default case, it checks for all pairs `u,v` if `A_{ut} \geq A_{vt}` holds for all `t` if `benefit = TRUE` or `A_{ut} \leq A_{vt}` holds for all `t` if `benefit = FALSE`. This form of dominance is referred to as dominance under total heterogeneity. If `map=TRUE`, the rows of `A` are sorted decreasingly (`benefit = TRUE`) or increasing (`benefit = FALSE`) and then the dominance condition is checked. This second form of dominance is referred to as dominance under total homogeneity, while the first is called dominance under total heterogeneity.

Value

Dominance relations as matrix object. An entry `[u,v]` is 1 if `u` is dominated by `v`.

Author(s)

David Schoch

References


See Also

`neighborhood_inclusion`, `indirect_relations`, `exact_rank_prob`
Examples

library(igraph)

data("dbces11")

P <- neighborhood_inclusion(dbces11)
comparable_pairs(P)

# positional dominance under total heterogeneity
dist <- indirect_relations(dbces11, type = "dist_sp")
D <- positional_dominance(dist, map = FALSE, benefit = FALSE)
comparable_pairs(D)

# positional dominance under total homogeneity
D_map <- positional_dominance(dist, map = TRUE, benefit = FALSE)
comparable_pairs(D_map)

print.netrankr_full  
Print netrankr_full object to terminal

Description

Prints the result of an object obtained from exact_rank_prob to terminal

Usage

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

Arguments

x  
A netrankr_full object

...  
additional arguments to print

Author(s)

David Schoch
print.netrankr_interval

Print netrankr_interval object to terminal

Description
Prints the result of an object obtained from rank_intervals to terminal

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

Arguments
x A netrankr_interval object
...

Author(s)
David Schoch

print.netrankr_mcmc

Print netrankr_mcmc object to terminal

Description
Prints the result of an object obtained from mcmc_rank_prob to terminal

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

Arguments
x A netrank object
...

Author(s)
David Schoch
Description

Calculate the maximal and minimal rank possible for each node in any ranking that is in accordance with the partial ranking \( P \).

Usage

\[
\text{rank_intervals}(P)
\]

Arguments

\( P \)  
A partial ranking as matrix object calculated with \textit{neighborhood\_inclusion} or \textit{positional\_dominance}.

Details

Note that the returned \textit{mid\_point} is not the same as the expected rank, for instance computed with \textit{exact\_rank\_prob}. It is simply the average of \textit{min\_rank} and \textit{max\_rank}. For exact rank probabilities use \textit{exact\_rank\_prob}.

Value

An object of type \textit{netrankr\_interval}

Author(s)

David Schoch

See Also

\textit{exact\_rank\_prob}

Examples

\[
P <- \text{matrix(c(0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, rep(0, 10)), 5, 5, byrow = TRUE)}
\]
\[
\text{rank\_intervals}(P)
\]
**spectral_gap**

*Description*

The spectral (or eigen) gap of a graph is the absolute difference between the biggest and second biggest eigenvalue of the adjacency matrix. To compare spectral gaps across networks, the fraction can be used.

**Usage**

```r
spectral_gap(g, method = "frac")
```

**Arguments**

- `g`: igraph object
- `method`: A string, either "frac" or "abs"

**Details**

The spectral gap is bounded between 0 and 1 if `method="frac"`. The closer the value to one, the bigger the gap.

**Value**

Numeric value

**Author(s)**

David Schoch

**Examples**

```r
# The fractional spectral gap of a threshold graph is usually close to 1
g <- threshold_graph(50, 0.3)
spectral_gap(g, method = "frac")
```
Summary of a netrankr_full object

Description
Summarizes the result of an object obtained from exact_rank_prob to terminal

Usage
```r
## S3 method for class 'netrankr_full'
summary(object, ...)
```

Arguments
- `object`: A netrankr_full object
- `...`: Additional arguments to summary

Author(s)
David Schoch

Random threshold graphs

Description
Constructs a random threshold graph. A threshold graph is a graph where the neighborhood inclusion preorder is complete.

Usage
```r
threshold_graph(n, p, bseq)
```

Arguments
- `n`: The number of vertices in the graph.
- `p`: The probability of inserting dominating vertices. Equates approximately to the density of the graph. See Details.
- `bseq`: (0,1)-vector a binary sequence that produces a threshold graph. See details

Details
Either `n` and `p`, or `bseq` must be specified. Threshold graphs can be constructed with a binary sequence. For each 0, an isolated vertex is inserted and for each 1, a vertex is inserted that connects to all previously inserted vertices. The probability of inserting a dominating vertices is controlled with parameter `p`. If `bseq` is given instead, a threshold graph is constructed from that sequence. An important property of threshold graphs is, that all centrality indices induce the same ranking.
transform_relations

Value

A threshold graph as igraph object

Author(s)

David Schoch

References


See Also

neighborhood_inclusion, positional_dominance

Examples

```r
library(igraph)
g <- threshold_graph(10, 0.3)
## Not run:
plot(g)

# star graphs and complete graphs are threshold graphs
complete <- threshold_graph(10, 1) # complete graph
plot(complete)

star <- threshold_graph(10, 0) # star graph
plot(star)

## End(Not run)

# centrality scores are perfectly rank correlated
cor(degree(g), closeness(g), method = "kendall")
```

---

transform_relations  Transform indirect relations

Description

Mostly wrapper functions that can be used in conjunction with indirect_relations to fine tune indirect relations.
Usage

- dist_2pow(x)
- dist_inv(x)
- dist_dpow(x, alpha = 1)
- dist_powd(x, alpha = 0.5)
- walks_limit_prop(x)
- walks_exp(x, alpha = 1)
- walks_exp_even(x, alpha = 1)
- walks_exp_odd(x, alpha = 1)
- walks_attenuated(x, alpha = 1/\max(x) * 0.99)
- walks_uptok(x, alpha = 1, k = 3)

Arguments

- x: Matrix of relations.
- alpha: Potential weighting factor.
- k: For walk counts up to a certain length.

Details

The predefined functions follow the naming scheme `relation_transformation`. Predefined functions `walks_*` are thus best used with `type="walks"` in `indirect_relations`. Theoretically, however, any transformation can be used with any relation. The results might, however, not be interpretable.

The following functions are implemented so far:
- dist_2pow returns $2^{-x}$
- dist_inv returns $1/x$
- dist_dpow returns $x^{-\alpha}$ where $\alpha$ should be chosen greater than 0.
- dist_powd returns $\alpha x^{\alpha}$ where $\alpha$ should be chosen between 0 and 1.
- walks_limit_prop returns the limit proportion of walks between pairs of nodes. Calculating row-Sums of this relation will result in the principle eigenvector of the network.
- walks_exp returns $\sum_{k=0}^{\infty} \frac{A^k}{k!}$
- walks_exp_even returns $\sum_{k=0}^{\infty} \frac{A^{2k}}{(2k)!}$
- walks_exp_odd returns $\sum_{k=0}^{\infty} \frac{A^{2k+1}}{(2k+1)!}$
- walks_attenuated returns $\sum_{k=0}^{\infty} \alpha^k A^k$
Walks up to \( k \) returns \( \sum_{j=0}^{k} \alpha^j A^j \)

Walk based transformation are defined on the eigen decomposition of the adjacency matrix using the fact that

\[
f(A) = Xf(A)X^T.
\]

Care has to be taken when using user defined functions.

**Value**

Transformed relations as matrix

**Author(s)**

David Schoch

---

**transitive_reduction**  
*Transitive Reduction*

---

**Description**

Calculates the transitive reduction of a partial ranking.

**Usage**

\[
\text{transitive_reduction}(P)
\]

**Arguments**

- \( P \)  
  A partial ranking as matrix object calculated with `neighborhood_inclusion` or `positional_dominance`.

**Value**

transitive reduction of \( P \)

**Author(s)**

David Schoch

**Examples**

```r
library(igraph)

g <- threshold_graph(100, 0.1)
P <- neighborhood_inclusion(g)
sum(P)

R <- transitive_reduction(P)
sum(R)
```
Index

* datasets
  - dbces11, 9
  - florentine_m, 12
aggregate_positions, 2, 18
approx_rank_expected, 4, 6, 10, 11, 21
approx_rank_relative, 5, 5, 10, 11, 21
as.matrix.netrankr_full, 6
comparable_pairs, 7, 14
compare_ranks, 8
dbces11, 9
dist_2pow(transform_relations), 32
dist_dpow(transform_relations), 32
dist_inv(transform_relations), 32
dist_powd(transform_relations), 32
dominance_graph, 9
exact_rank_prob, 5, 6, 10, 12, 21–23, 26, 27, 29, 31
florentine_m, 12
get_rankings, 12
hyperbolic_index, 13
incomparable_pairs, 7, 14
index_builder, 15
indirect_relations, 3, 15, 26, 32, 33
is_preserved, 18
majorization_gap, 19
mcmc_rank_prob, 5, 6, 10, 11, 20, 24, 28
neighborhood_inclusion, 4, 5, 7, 9, 10, 14, 18, 21, 22, 25, 26, 29, 32, 34
plot.netrankr_full, 23
plot.netrankr_interval, 24
plot.netrankr_mcmc, 24
plot_rank_intervals, 25
positional_dominance, 4, 5, 7, 9, 10, 14, 15, 18, 21, 22, 25, 26, 29, 32, 34
print.netrankr_full, 27
print.netrankr_interval, 28
print.netrankr_mcmc, 28
rank, 24, 25
rank_intervals, 24, 25, 28, 29
spectral_gap, 30
summary.netrankr_full, 31
threshold_graph, 19, 31
transform_relations, 3, 16, 17, 32
transitive_reduction, 34
walks_attenuated(transform_relations), 32
walks_exp(transform_relations), 32
walks_exp_even(transform_relations), 32
walks_exp_odd(transform_relations), 32
walks_limit_prop(transform_relations), 32
walks_uptok(transform_relations), 32