# Package 'diffusr'

April 21, 2018

**Type** Package

**Title** Network Diffusion Algorithms

**Version** 0.1.3

**Date** 2018-04-20

**Maintainer** Simon Dirmeier <simon.dirmeier@gmx.de>

**Description** Implementation of network diffusion algorithms such as heat diffusion or Markov random walks. Network diffusion algorithms generally spread information in the form of node weights along the edges of a graph to other nodes. These weights can for example be interpreted as temperature, an initial amount of water, the activation of neurons in the brain, or the location of a random surfer in the internet. The information (node weights) is iteratively propagated to other nodes until a equilibrium state or stop criterion occurs.

**URL** https://github.com/dirmeier/diffusr

**BugReports** https://github.com/dirmeier/diffusr/issues

**License** GPL (>= 3)

**Depends** R (>= 3.4)

**LazyData** TRUE

**LinkingTo** Rcpp, RcppEigen

**Imports** Rcpp, igraph, methods

**Suggests** knitr, rmarkdown, testthat, lintr

**VignetteBuilder** knitr

**RoxygenNote** 6.0.1

**SystemRequirements** C++11

**NeedsCompilation** yes

**Author** Simon Dirmeier [aut, cre]

**Repository** CRAN

**Date/Publication** 2018-04-21 21:55:29 UTC

# R topics documented:



---

`diffusr-package` *diffusr*

---

## Description

Network diffusion algorithms in R.

## Author(s)

Simon Dirmeier <simon.dirmeier@gmx.de>

## References

Tong, H., Faloutsos, C., & Pan, J. Y. (2006), Fast random walk with restart and its applications.

Koehler, S., Bauer, S., Horn, D., & Robinson, P. N. (2008), Walking the interactome for prioritization of candidate disease genes. *The American Journal of Human Genetics*

Bonacich, P. (1987), Power and centrality: A family of measures. *American Journal of Sociology*

Leiserson, M. D., Vandin, F., Wu, H. T., Dobson, J. R., Eldridge, J. V., Thomas, J. L., ... & Lawrence, M. S. (2015), Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. *Nature genetics*

https://en.wikipedia.org/wiki/Laplacian_matrix
https://en.wikipedia.org/wiki/Heat_equation

heat.diffusion *Graph diffusion using a heat diffusion process on a Laplacian matrix.*

## Description

An amount of starting heat gets distribution using the Laplacian matrix of a graph. Every iteration (or time interval) `t` heat streams from the starting nodes into surrounding nodes.

## Usage

```
heat.diffusion(h0, graph, t = 0.5, ...)

## S4 method for signature 'numeric,matrix'
heat.diffusion(h0, graph, t = 0.5, ...)

## S4 method for signature 'matrix,matrix'
heat.diffusion(h0, graph, t = 0.5, ...)
```

## Arguments

| | |
|---|---|
| `h0` | an `n x p`-dimensional numeric non-negative vector/matrix of starting temperatures |
| `graph` | an (`n x n`)-dimensional numeric non-negative adjacence matrix representing the graph |
| `t` | time point when heat is measured |
| `...` | additional parameters |

## Value

returns the heat on every node as numeric vector

## References

https://en.wikipedia.org/wiki/Laplacian_matrix
https://en.wikipedia.org/wiki/Heat_equation

## Examples

```
# count of nodes
n <- 5
# starting distribution (has to sum to one)
h0 <- as.vector(rmultinom(1, 1, prob=rep(.2, n)))
# adjacency matrix (either normalized or not)
graph <- matrix(abs(rnorm(n*n)), n, n)
# computation of stationary distribution
ht <- heat.diffusion(h0, graph)
```

## hub.correction *Correct for hubs in an adjacency matrix*

### Description

Correct for hubs in an adjacency matrix

### Usage

```
hub.correction(obj)
```

### Arguments

| | |
|---|---|
| obj | matrix for which hubs are corrected |

### Value

returns the matrix with hub correction

### Examples

```
W <- matrix(abs(rnorm(10000)), 100, 100)
cor.hub <- hub.correction(W)
```

## nearest.neighbors *Graph diffusion using nearest neighbors*

### Description

For every node in a set of nodes the graph gets traversed along the node's shortest paths to its neighbors. Nearest neighbors are added until a maximum depth of k is reached. For settings where there are more than k neighbors having the same distance, all neighbors are returned.

### Usage

```
nearest.neighbors(nodes, graph, k = 1L, ...)

## S4 method for signature 'integer,matrix'
nearest.neighbors(nodes, graph, k = 1L, ...)
```

### Arguments

| | |
|---|---|
| nodes | a n-dimensional integer vector of node indexes (1-based) for which the algorithm is applied iteratively |
| graph | an (n x n)-dimensional numeric non-negative adjacence matrix representing the graph |
| k | the depth of the nearest neighbor search, e.g. the depth of the graph traversal |
| ... | additional parameters |

## Value

returns the kNN nodes as list of integer vectors of node indexes

## Examples

```
# count of nodes
n <- 10
# indexes (integer) of nodes for which neighbors should be searched
node.idxs <- c(1L, 5L)
# the adjaceny matrix (does not need to be symmetric)
graph <- rbind(cbind(0, diag(n-1)), 0)
# compute the neighbors until depth 3
neighs <- nearest.neighbors(node.idxs, graph, 3)
```

---

# normalize.laplacian *Calculate the Laplacian of a matrix*

---

## Description

Calculate the Laplacian of a matrix

## Usage

```
normalize.laplacian(obj, ...)
```

## Arguments

| | |
|---|---|
| obj | matrix for which the Laplacian is calculated |
| ... | additional params |

## Value

returns the Laplacian

## Examples

```
W <- matrix(abs(rnorm(10000)), 100, 100)
lapl.W <- normalize.laplacian(W)
```

## normalize.stochastic *Create a stochastically normalized matrix/vector*

### Description

Create a stochastically normalized matrix/vector

### Usage

```
normalize.stochastic(obj, ...)
```

### Arguments

| | |
|---|---|
| obj | matrix/vector that is stochstically normalized |
| ... | additional params |

### Value

returns the normalized matrix/vector

### Examples

```
W <- matrix(abs(rnorm(10000)), 100, 100)
stoch.W <- normalize.stochastic(W)
```

## random.walk *Graph diffusion using a Markov random walk*

### Description

A Markov Random Walk takes an inital distribution `p0` and calculates the stationary distribution of that. The diffusion process is regulated by a restart probability `r` which controls how often the MRW jumps back to the initial values.

### Usage

```
random.walk(p0, graph, r = 0.5, niter = 10000, thresh = 1e-04,
  do.analytical = FALSE, correct.for.hubs = FALSE)

## S4 method for signature 'numeric,matrix'
random.walk(p0, graph, r = 0.5, niter = 10000,
  thresh = 1e-04, do.analytical = FALSE, correct.for.hubs = FALSE)

## S4 method for signature 'matrix,matrix'
random.walk(p0, graph, r = 0.5, niter = 10000,
  thresh = 1e-04, do.analytical = FALSE, correct.for.hubs = FALSE)
```

## Arguments

| | |
|---|---|
| p0 | an n x p-dimensional numeric non-negative vector/matrix representing the starting distribution of the Markov chain (does not need to sum to one). |
| graph | an (n x n)-dimensional numeric non-negative adjacence matrix representing the graph |
| r | a scalar between (0, 1). restart probability if a Markov random walk with restart is desired |
| niter | maximal number of iterations for computation of the Markov chain. If thresh is not reached, then niter is used as stop criterion. |
| thresh | threshold for breaking the iterative computation of the stationary distribution. If the absolute difference of the distribution at time point $t-1$ and $t$ is less than thresh, then the algorithm stops. If thresh is not reached before niter, then the algorithm stops as well. |
| do.analytical | boolean if the stationary distribution shall be computed solving the analytical solution or rather iteratively |
| correct.for.hubs | if TRUE multiplies a correction factor to the nodes, such that the random walk gets not biased to nodes with high degree. In that case the original input matrix will be normalized as: $$P(j\|i) = 1/degree(i) * min(1, degree(j)/degree(j))$$ *Note that this will not consider edge weights.* |

## Value

returns a list with the following elements

- p.inf the stationary distribution as numeric vector
- transition.matrix the column normalized transition matrix used for the random walk

## References

Tong, H., Faloutsos, C., & Pan, J. Y. (2006), Fast random walk with restart and its applications.

Koehler, S., Bauer, S., Horn, D., & Robinson, P. N. (2008), Walking the interactome for prioritization of candidate disease genes. *The American Journal of Human Genetics*

## Examples

```
# count of nodes
n <- 5
# starting distribution (has to sum to one)
p0    <- as.vector(rmultinom(1, 1, prob=rep(.2, n)))
# adjacency matrix (either normalized or not)
graph <- matrix(abs(rnorm(n*n)), n, n)
```

```
# computation of stationary distribution
pt    <- random.walk(p0, graph)
```

# Index