R documentation

of 'D:/Program' etc.

November 15, 2016

R topics documented:

metric.cluster.global *Global Clustering Coefficient*

Description

Calculate the global clustering coefficient of a graph.

Usage

```
metric.cluster.global(Network, node.sample, triplet.sample)
```

Arguments

`Network`	The input network.
`node.sample`	The percentage of nodes to be selected - random sample (0,1).
`triplet.sample`	The number of triplets to explore for each node - random triplets.

Details

The global clustering coefficient measures the ratio of triples versus the total number of all possible triples in graph g. `metric.cluster.global()` calculates the (estimated) global clustering coefficient of graph g with a justified error.

Value

A real constant.

Author(s)

Luis Castro, Nazrul Shaikh.

References

Wasserman, Stanley, and Katherine Faust. Social network analysis: Methods and applications. Vol. 8. Cambridge university press, 1994.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.cluster.global(x, 0.001,2)
## End(Not run)
```

`metric.cluster.mean` *Mean Local Clustering Coefficient*

Description

Calculate the average local clustering coefficient of a graph.

Usage

```
metric.cluster.mean(g)
```

Arguments

`g`	The input network.

Details

The local clustering coefficient of a node is the ratio of the triangles connected to the node and the triples centered on the `node.metric.cluster.mean()` calculates the (estimated) average clustering coefficient for all nodes in graph *g* with a justified error.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Wasserman, Stanley, and Katherine Faust. Social network analysis: Methods and applications. Vol. 8. Cambridge university press, 1994.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.cluster.mean(x)
## End(Not run)
```

`metric.cluster.median` *Median Local Clustering Coefficient*

Description

Calculate the median local clustering coefficient of a graph.

Usage

```
metric.cluster.median(g)
```

Arguments

`g`	The input network.

Details

The local clustering coefficient of a node is the ratio of the triangles connected to the node and the triples centered on the `node.metric.cluster.median()` calculates the (estimated) median clustering coefficient for all nodes in graph *g* with a justified error.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Wasserman, Stanley, and Katherine Faust. Social network analysis: Methods and applications. Vol. 8. Cambridge university press, 1994.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.cluster.median(x)
## End(Not run)
```

`metric.degree.entropy` *Degree Entropy*

Description

Calculate the degree entropy of a graph.

Usage

```
metric.degree.entropy(g)
```

Arguments

g	The input network.

Details

Calculates the degree entropy of graph g, i.e.

$$ntropy(g) = -\sum_{i=1}^{n} i * \log_2(i)$$

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Anand, Kartik, and Ginestra Bianconi. "Entropy measures for networks: Toward an information theory of complex topologies." Physical Review E 80, no. 4 (2009): 045102.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.degree.entropy(x)
## End(Not run)
```

metric.degree.max *Maximal Degree*

Description

Calculate the maximal degree of a graph.

Usage

```
metric.degree.max(g)
```

Arguments

g	The input network.

Details

The maximal degree.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

Examples

```
## Not run:
metric.degree.max(x)
## End(Not run)
```

metric.degree.max.efficient *Efficient Maximal Degree*

Description

Calculate the efficient maximal degree of a graph.

Usage

```
metric.degree.max.efficient(g)
```

Arguments

g	The input network.

Details

The efficient maximal degree is the 90

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

Examples

```
## Not run: x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.degree.max.efficient(x)
## End(Not run)
```

---

metric.degree.mean   *Mean Degree*

Description

Calculate the mean degree of a graph.

Usage

```
metric.degree.mean(g)
```

Arguments

g	The input network.

Details

The mean degree is the average value of the degrees of all nodes in graph g.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.degree.mean(x)

## End(Not run)
```

metric.degree.median *Median Degree*

Description

Calculate the median degree of a graph.

Usage

```
metric.degree.median(g)
```

Arguments

g	The input network.

Details

The median degree is the median value of the degrees of all nodes in graph g.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.degree.median(x)
## End(Not run)
```

metric.distance.apl *Average Path Length*

Description

Calculate the average path length of a graph.

Usage

```
metric.distance.apl(Network, probability = 0.95, error = 0.03,
  Cores = detectCores(), full.apl = FALSE)
```

Arguments

`Network`	The input network.
`probability`	The confidence level probability.
`error`	The sampling error.
`Cores`	Number of cores to use in the computations. By default uses *parallel* function `detecCores()`.
`full.apl`	It will calculate the sampling version by default. If it is set to true, the population APL will be calculated and the rest of the parameters will be ignored.

Details

The average path length (APL) is the average shortest path lengths of all pairs of nodes in graph *Network*. `metric.distance.apl` calculates the population APL and estimated APL of graph g with a sampling error set by the user.

The calculation uses a parallel load balancing approach, distributing jobs equally among the cores defined by the user.

Value

A real value.

Author(s)

Luis Castro, Nazrul Shaikh.

References

Dijkstra EW. A note on two problems in connexion with graphs:(numerische mathematik, _1 (1959), p 269-271). 1959.

Castro L, Shaikh N. Estimation of Average Path Lengths of Social Networks via Random Node Pair Sampling. Department of Industrial Engineering, University of Miami. 2016.

Examples

```
## Not run:
##Default function
x <-  net.erdos.renyi.gnp(1000,0.01)
metric.distance.apl(x)
##Population APL
metric.distance.apl(x, full.apl=TRUE)
##Sampling at 99% level with an error of 10% using 5 cores
metric.distance.apl(Network = x, probability=0.99, error=0.1, Cores=5)

## End(Not run)
```

metric.distance.diameter
Diameter

Description

Calculate the diameter of a graph.

Usage

```
metric.distance.diameter(Network, probability = 0.95, error = 0.03,
  Cores = detectCores(), full = TRUE)
```

Arguments

`Network`	The input network.
`probability`	The confidence level probability
`error`	The sampling error
`Cores`	Number of cores to use in the computations. By default uses *parallel* function `detecCores()`.
`full`	It will calculate the popular full version by default. If it is set to FALSE, the estimated diameter will be calculated.

Details

The diameter is the largest shortest path lengths of all pairs of nodes in graph *Network*. `metric.distance.diameter` calculates the (estimated) diameter of graph *Network* with a justified error.

Value

A real value.

Author(s)

Luis Castro, Nazrul Shaikh.

References

Dijkstra EW. A note on two problems in connexion with graphs:(numerische mathematik, _1 (1959), p 269-271). 1959.

Castro L, Shaikh N. Estimation of Average Path Lengths of Social Networks via Random Node Pair Sampling. Department of Industrial Engineering, University of Miami. 2016.

Examples

```
## Not run:
##Default function
x <-  net.erdos.renyi.gnp(1000,0.01)
metric.distance.diameter(x)
##Population APL
metric.distance.diameter(x, full=TRUE)
```

```
##Sampling at 99% level with an error of 10% using 5 cores
metric.distance.diameter(Network = x, probability=0.99, error=0.1, Cores=5)

## End(Not run)
```

---

metric.distance.meanecc
Mean Eccentricity

Description

Calculate the mean eccentricity of a graph.

Usage

```
metric.distance.meanecc(g, p)
```

Arguments

g	The input network.
p	The sampling probability.

Details

The mean eccentricities of all nodes in graph g. Calculates the (estimated) mean eccentricity of graph g with a justified error.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

West, Douglas Brent. Introduction to graph theory. Vol. 2. Upper Saddle River: Prentice Hall, 2001.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.distance.meanecc(x, 0.01)
## End(Not run)
```

`metric.distance.medianecc` *Median Eccentricity*

Description

Calculate the (estimated) median eccentricity of a graph.

Usage

```
metric.distance.medianecc(g, p)
```

Arguments

g	The input network.
p	The sampling probability.

Details

The median eccentricity is the median eccentricities of all nodes in graph g. metric.distance.medianecc calculates the (estimated) median eccentricity of graph g with a justified error.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

West, Douglas Brent. Introduction to graph theory. Vol. 2. Upper Saddle River: Prentice hall, 2001.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.distance.medianecc(x, 0.01)
## End(Not run)
```

`metric.distance.mpl` *Median Path Length*

Description

Calculate the median path length (MPL) of a network.

Usage

```
metric.distance.mpl(Network, probability = 0.95, error = 0.03,
  Cores = detectCores(), full = FALSE)
```

Arguments

`Network`	The input network.
`probability`	The confidence level probability
`error`	The sampling error
`Cores`	Number of cores to use in the computations. By default *parallel* function `detecCores()`.
`full`	It calculates the sampling version by default. If it is set to true, the population MPL will be calculated and the rest of the parameters will be ignored.

Details

The median path length (MPL) is the median shortest path lengths of all pairs of nodes in *Network*. *metric.distance.mpl(g)* calculates the population MPL OR estimated MPL of network g with a sampling error set by the user. The calculation uses a parallel load balancing approach, distributing jobs equally among the cores defined by the user.

Value

A real integer

Author(s)

Luis Castro, Nazrul Shaikh.

References

Dijkstra EW. A note on two problems in connexion with graphs:(numerische mathematik, _1 (1959), p 269-271). 1959.

Castro L, Shaikh N. Estimation of Average Path Lengths of Social Networks via Random Node Pair Sampling. Department of Industrial Engineering, University of Miami. 2016.

Examples

```
## Not run:
##Default function
x <-  net.erdos.renyi.gnp(1000,0.01)
metric.distance.mpl(x)
##Population MPL
metric.distance.mpl(x, full=TRUE)
```

```
##Sampling at 99% level with an error of 10% using 5 cores
metric.distance.mpl(Network = x, probability=0.99, error=0.1, Cores=5)

## End(Not run)
```

`metric.eigen.mean` *Mean Eigenvalue Centrality*

Description

Calculate the mean eigenvalue centrality of a graph.

Usage

```
metric.eigen.mean(g)
```

Arguments

`g`	The input network.

Details

`metric.eigen.mean` calculates the mean eigenvalue centrality score of graph *g*.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Bonacich, Phillip, and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations." Social networks 23, no. 3 (2001): 191-201.

Borgatti, Stephen P. "Centrality and network flow." Social networks 27, no. 1 (2005): 55-71.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.eigen.mean(x)
## End(Not run)
```

`metric.eigen.median` *Median Eigenvalue Centrality*

Description

Calculate the median eigenvalue centrality of a graph.

Usage

```
metric.eigen.median(g)
```

Arguments

g	The input network.

Details

`metric.eigen.median` calculates the median eigenvalue centrality score of graph *g*.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Bonacich, Phillip, and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations." Social networks 23, no. 3 (2001): 191-201.

Borgatti, Stephen P. "Centrality and network flow." Social networks 27, no. 1 (2005): 55-71.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.eigen.median(x)
## End(Not run)
```

`metric.eigen.value` *Eigenvalue Score*

Description

Calculate the eigenvalue centrality score of a graph.

Usage

```
metric.eigen.value(g)
```

Arguments

g	The input network.

Details

`metric.eigen.value` calculates the eigenvalue centrality score of graph *g*.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Bonacich, Phillip, and Paulette Lloyd. "Eigenvector-like measures of centrality for asymmetric relations." Social networks 23, no. 3 (2001): 191-201.

Borgatti, Stephen P. "Centrality and network flow." Social networks 27, no. 1 (2005): 55-71.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.eigen.value(x)
## End(Not run)
```

`metric.graph.density` *Graph Density*

Description

Calculate the density of a graph.

Usage

```
metric.graph.density(g)
```

Arguments

g	The input network.

Details

Computes the ratio of the number of edges and the number of possible edges.

Value

A real constant.

Author(s)

Xu Dong, Nazrul Shaikh.

Examples

```
## Not run:
x <-  net.erdos.renyi.gnp(1000, 0.01)
metric.graph.density(x)
## End(Not run)
```

`net.barabasi.albert` *Barabasi-Albert Scale-Free Network*

Description

Simulate a scale-free network using a preferential attachment mechanism (Barabasi and Albert, 1999)

Usage

```
net.barabasi.albert(n, m, ncores = detectCores(), d = FALSE)
```

Arguments

`n`	Number of nodes of the network.
`m`	Number of nodes to which a new node connects at each iteration.
`ncores`	Number of cores, by default `detectCores()` from `parallel`.
`d`	A logical value determining whether the generated network is a directed or undirected (default) network.

Details

Starting with *m* nodes, the preferential attachment mechaism adds one node and *m* edges in each step. The edges will be placed with one end on the newly-added node and the other end on the existing nodes, according to probabilities that associate with their current degrees.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Luis Castro, Xu Dong, Nazrul Shaikh.

References

Barabasi, A.- L. and Albert R. 1999. Emergence of scaling in random networks. Science, 286:509-512.

Examples

```
## Not run:
x <- net.barabasi.albert(1000, 20) # using default ncores
## End(Not run)
```

`net.caveman` *Caveman Network*

Description

Simulate a (connected) caveman network of *m* cliques of size *k*.

Usage

```
net.caveman(m, k, ncores = detectCores())
```

Arguments

`m`	Number of cliques (or caves) in the network.
`k`	Number of nodes per clique.
`ncores`	Number of cores, by default `detectCores()` from `parallel`.

Details

The (connected) caveman network is formed by connecting a set of isolated *k* - cliques (or "caves"), neighbor by neighbor and head to toe, using one edge that removed from each clique such that all *m* cliques form a single circle (Watts 1999). The total number of nodes, i.e. *n*, in this network is given by $k * m$.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh.

References

Watts, D. J. Networks, Dynamics, and the Small-World Phenomenon. Amer. J. Soc. 105, 493-527, 1999.

Examples

```
## Not run:
x <- net.caveman(50, 20) #using ncores by default
## End(Not run)
```

`net.complete` *Complete Network*

Description

Simulate a complete (or full) network.

Usage

```
net.complete(n, ncores = detectCores())
```

Arguments

`n`	Number of nodes of the network.
`ncores`	Number of cores, by default `detectCores()` from `parallel`.

Details

The *n* nodes in the network are fully connected.

Note that the input *n* should not excess 10000, for the sake of memory overflow.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh.

References

...

Examples

```
## Not run:
x <- net.complete(1000) #using ncores by default
## End(Not run)
```

---

net.erdos.renyi.gnm *Directed / Undirected Erdos-Renyi $G(n, m)$ network using a fix edge size.*

Description

Simulate a random network with *n* nodes and *m* edges, according to Erdos and Renyi (1959).

Usage

```
net.erdos.renyi.gnm(n, m, ncores = detectCores(), d = TRUE)
```

Arguments

n	Number of nodes of the network.
m	Number of edges of the network.
ncores	Number of cores, by default `detectCores()` from `parallel`.
d	A logical value determining whether is a network directed (default) or indirected.

Details

In this (simplest) random network, *m* edges are formed at random among *n* nodes. When `d = TRUE` is a directed network.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaihk.

References

Erdos, P. and Renyi, A., On random graphs, Publicationes Mathematicae 6, 290-297 (1959).

Examples

```
## Not run:
x <- net.erdos.renyi.gnm(1000, 100)
## End(Not run)
```

`net.erdos.renyi.gnp` *Directed / Undirected Erdos-Renyi $G(n, p)$ network*

Description

Simulate a random network with *n* nodes and a link connecting probability of *p*, according to Erdos and Renyi (1959).

Usage

```
net.erdos.renyi.gnp(n, p, ncores = detectCores(), d = TRUE)
```

Arguments

n	Number of nodes of the network.
p	Connecting probability.
ncores	Number of cores, by default `detectCores()` from `parallel`.
d	A logical value determining whether is a network directed (default) or indirected.

Details

In this (simplest) random network, each edge is formed at random with a constant probability. When `d = TRUE` is a directed network.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Luis Castro, Xu Dong, Nazrul Shaihk.

References

Erdos, P. and Renyi, A., On random graphs, Publicationes Mathematicae 6, 290-297 (1959).

Examples

```
## Not run:
x <- net.erdos.renyi.gnp(1000, 0.01)
## End(Not run)
```

net.holme.kim *Holme-Kim Network*

Description

Simulate a scale-free network with relatively high clustering, comparing to B-A networks (Holme and Kim, 1999).

Usage

```
net.holme.kim(n, m, pt)
```

Arguments

n	Number of nodes of the network.
m	Number of nodes to which a new node connects at each iteration.
pt	Triad formation probability after each preferential attachment mechanism.

Details

The Holme-Kim network model is a simple extension of B-A model. It adds an additional step, called "Triad formation", with the probability *pt* that compensates the low clustering in B-A networks.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh

References

Holme, Petter, and Beom Jun Kim. "Growing scale-free networks with tunable clustering."Physical review E65, no. 2 (2002): 026107.

Examples

```
## Not run:
x <- net.holme.kim (1000, 20, 0.1)
## End(Not run)
```

`net.random.plc` *Random Network with a Power-law Degree Distribution that Has An Exponential Cutoff*

Description

Simulate a random network with a power-law degree distribution that has an exponential cutoff, according to Newman et al. (2001).

Usage

```
net.random.plc(n, cutoff, exponent)
```

Arguments

`n`	The number of the nodes in the network.
`cutoff`	Exponential cutoff of the degree distribution of the network.
`exponent`	Exponent of the degree distribution of the network.

Details

The generated random network has a power-law degree distribution with an exponential degree cutoff.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh

References

Newman, Mark EJ, Steven H. Strogatz, and Duncan J. Watts. "Random graphs with arbitrary degree distributions and their applications." Physical review E 64, no. 2 (2001): 026118.

Examples

```
## Not run:
x <- net.random.plc(1000, 10, 2)
## End(Not run)
```

`net.rewired.caveman` *Rewired (Connected) Caveman Network*

Description

Simulate a rewired caveman network of m cliques of size k, and with a link rewiring probability p.

Usage

```
net.rewired.caveman(nc, m, p)
```

Arguments

`nc`	Number of cliques (or caves) in the network.
`m`	Number of nodes per clique.
`p`	Link rewiring probability.

Details

The rewired caveman network is built on the corresponding regular caveman network with m cliques of size k. Then the links in this caveman network are rewired with probability p.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh

References

Watts, D. J. Networks, Dynamics, and the Small-World Phenomenon. Amer. J. Soc. 105, 493-527, 1999.

Examples

```
## Not run:
x <- net.rewired.caveman(50, 20, 0.0005)
## End(Not run)
```

net.ring.lattice — k - *regular ring lattice*

Description

Simulate a network with a k -regular ring lattice structure.

Usage

```
net.ring.lattice(n, k)
```

Arguments

n	Number of nodes in the network.
k	Number of edges per node.

Details

The n nodes are placed on a circle and each node is connected to the nearest k neighbors.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh

References

Duncan J Watts and Steven H Strogatz: Collective dynamics of 'small world' networks, Nature 393, 440-442, 1998.

Examples

```
## Not run:
x <- net.ring.lattice(1000, 10)
## End(Not run)
```

`net.small.world` *Watts-Strogatz Small-world Network*

Description

Simulate a small-world network according to the model of Watts and Strogatz (1998).

Usage

```
net.small.world(n, k, re)
```

Arguments

`n`	The number of the nodes in the network (or lattice).
`k`	Number of edges per node.
`re`	Rewiring probability.

Details

The formation of Watts-Strogatz network starts with a ring lattice with n nodes and k edges per node, then each edge is rewired at random with probability re.

Value

A list containing the nodes of the network and their respective neighbors.

Author(s)

Xu Dong, Nazrul Shaikh

References

Duncan J. Watts and Steven H. Strogatz: Collective dynamics of 'small world' networks, Nature 393, 440-442, 1998.

Examples

```
## Not run:
x <- net.small.world(1000, 10, 0.05)
## End(Not run)
```

Index