Package 'modMax'

October 13, 2022

Type Package

Title Community Structure Detection via Modularity Maximization

Version 1.1

Date 2015-07-24

Author Maria Schelling, Cang Hui

Maintainer Maria Schelling <schelling.rmaintainer@vodafone.de>

Depends gtools, igraph

Description

The algorithms implemented here are used to detect the community structure of a network. These algorithms follow different approaches, but are all based on the concept of modularity maximization.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2015-07-24 18:21:32

R topics documented:

modMax-package *Calculate network modularity via maximization algorithms*

Description

Calcuation of modularity and detection of the community strcture of a given network depicted by an (nonnegative symmetric) adjacency matrix using different modularity maximization algorithms

Details

Package:	modMax
Type:	Package
Version:	1.0
Date:	2015-02-09
License:	GPL-2

The `modMax` package implements 38 algorithms of 6 major categories maximizing modularity, including the greedy approach, simulated annealing, extremal optimization, genetic algorithm, mathematical programming and the usage of local modularity.

All algorithms work on connected (consisting of only one connected component), undirected graphs given by their adjacency matrix.

Most algorithms also provide the possibility to compare the estimated modularity of the identified community structure with the modularity for random networks generated by null models with the number of vertices and edges conserved.

Author(s)

Maria Schelling, Cang Hui

Maintainer: Maria Schelling <schelling.rmaintainer@vodafone.de>

extremalOptimization *Extremal optimization (EO) algorithms*

Description

`extremalOptimization` is a function executing the extremal optimization approach and its modifications for calculating modularity and detecting communities (modules of nodes) of a network via modularity maximization

`pcseoss` is a function which uses extremal optimization, but also considers pairwise constraints when calculating the fitness function and the modularity. The violation of constraints is punished, leading to smaller fitness and modularity values for community structures that violate many pairwise constraints. The constraints are predefined as two matrices separately for must-links and cannot-links with punishment for violation.

Usage

```
extremalOptimization(adjacency, numRandom = 0,
                      refine = c("none", "agents"),
                     tau = FALSE, alpha_max = length(adjacency[1,]), steps = 3)
pcseoss(adjacency,constraints_ml,constraints_cl)
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model.
refine	Specify whether or not a refinement step is needed, the default option is none. See details below.
tau	If TRUE, τ-EO is executed where the vertices are ranked according to their fitness values and chosen by a probability depending on this ranking.
alpha_max	It gives the maximum number of iteration steps. If the community structure could not be improved for this number of steps, the algorithm terminates. It is 1 for the normal EO-algorithm and n for the τ-EO where n is the number of vertices in the network
steps	The number of iteration steps for the random local search agent algorithm. The algorithm terminates, if the clusters have not changed for this number of steps. Ignored if refine is none.
constraints_ml	The matrix where each column is a must-link constraint given by two vertices in the first two rows which have to be in the same community and a punishment for the violation of the constraint in the third row
constraints_cl	The matrix where each column is a cannot-link constraint given by two vertices in the first two rows which cannot be in the same community and a punishment for the violation of the constraint in the third row

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

The EO algorithm can be run with a certain refinement step, the local random search agent algorithm, applied at the end of one round of extremal where all communities have been split once.

This refinement algorithm is executed if refine equals agent, otherwise the generic EO algorithm is executed.

Value

The result of the extremal optimization algorithms is a list with the following components

number of communities	The number of communities detected by the algorithm

`modularity` The modularity of the detected community structure

`mean` The mean of the modularity values for random networks, only computed if `numRandom>0`

`standard deviation`
The standard deviation of the modularity values for random networks, only computed if `numRandom>0`

`community structure`
The community structure of the examined network given by a vector assigning each vertex its community number

`random modularity values`
The list of the modularity values for random networks, only computed if `numRandom>0`

Author(s)

Maria Schelling, Cang Hui

References

Duch, J. and Arenas, A. Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 72:027104, Aug 2005.

Azizifard, N., Mahdavi, M. and Nasersharif, B. Modularity optimization for clustering in social networks. 2011.

Li, L., Du, M., Liu, G., Hu, X. and Wu, G. Extremal optimization-based semi-supervised algorithm with conflict pairwise constraints for community detection. In *Advances in Social Network Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, 2014.

Examples

```
#weighted network
randomgraph <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices <- which(clusters(randomgraph)$membership==1)
graph <- induced.subgraph(randomgraph,vertices)
graph <- set.edge.attribute(graph, "weight", value=runif(ecount(graph),0,1))

adj <- get.adjacency(graph, attr="weight")
result <- extremalOptimization(adj)
```

`geneticAlgorithm` *Genetic algorithm*

Description

`geneticAlgorithm` is a function executing the genetic algorithm and its modifications for identifying the community structure of a network via modularity maximization

Usage

```
geneticAlgorithm(adjacency, numRandom = 0,
                 initial = c("general", "cluster", "own"), p, g,
                 mutRat = 0.5, crossOver = 0.2, beta = 0.1, alpha = 0.4,
                 n_l = 4, local = FALSE)
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model
initial	Specify the community structure to use as initial partition in the algorithm. See details below.
p	Population size
g	Number of generations
mutRat	Mutation rate. Default is 0.5
crossOver	Crossing over rate. Default is 0.2
beta	The fraction of chromosomes to save. The top βp chromosomes are saved in each generation to ensure that the fitness scores of the top βp chromosomes of the child generation are at least as good as the parent population. Default is 0.1
alpha	The fraction of repetitions for the identification of an initial partition according to cluster. Default is 0.4. Ignored if initial is not cluster.
n_l	The number of copies of a chromosome made by the local search operator. Default is 4. Ignored if local is FALSE
local	If TRUE, local search operator is applied at the end of each iteration in the genetic algorithm.

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

The initial partition used in the genetic algorithm can either be the generic one where all vertices are put in their own community (initial=general) or the initial partition can be identified by randomly picking a vertex αn times and assigning its cluster to all its neighbours (initial=cluster) or the initial partition can be given by the user (initial=own). In this case, the user needs to add a last column to the adjacency matrix indicating the initial partition. Hence, the adjacency matrix has to have one column more than the network has vertices.

Value

The result of the genetic algorithm is a list with the following components

number of communities
: The number of communities detected by the algorithm

modularity
: The modularity of the detected community structure

mean
: The mean of the modularity values for random networks, only computed if numRandom>0

standard deviation
: The standard deviation of the modularity values for random networks, only computed if numRandom>0

community structure
: The community structure of the examined network given by a vector assigning each vertex its community number

random modularity values
: The list of the modularity values for random networks, only computed if numRandom>0

Author(s)

Maria Schelling, Cang Hui

References

Tasgin, M., Herdagdelen, A., and Bingol, H. Community detection in complex networks using genetic algorithms. *arXiv preprint arXiv:0711.0491*, 2007.

Li, S., Chen, Y., Du, H., and Feldman, M. W. A genetic algorithm with local search strategy for improved detection of community structure. *Complexity*, 15(4):53-60, 2010.

Examples

```
#unweighted network
randomgraph <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices <- which(clusters(randomgraph)$membership==1)
graph <- induced.subgraph(randomgraph,vertices)

adj <- get.adjacency(graph)
result <- geneticAlgorithm(adj, p=4, g=6)
```

---

greedy *Greedy algorithms*

Description

greedy executes the general CNM algorithm and its modifications for modularity maximization.

rgplus uses the randomized greedy approach to identify core groups (vertices which are always placed into the same community) and uses these core groups as initial partition for the randomized greedy approach to identify the community structure and maximize the modularity.

msgvm is a greedy algorithm which performs more than one merge at one step and applies fast greedy refinement at the end of the algorithm to improve the modularity value.

cd iteratively performs complete greedy refinement on a certain partition and then, moves vertices with a probability p to another community to avoid the greedy algorithm getting trapped in a local optimum.

louvain performs fast greedy refinement and uses the resulting community structure to build a new network where vertices in the new network are the communities in the original network. For this new network, all vertices are assigned to their own community, and the fast greedy refinement is applied again.

vertexSim uses a vertex similarity measure to identify the initial partition and further improves this community structure by merging neighbouring communities.

mome consists of the two phases of coarsening and uncoarsening with refinement. In the coarsening phase, two vertices are collapsed into one vertex for which the increase in modularity is maximal. In the uncoarsening phase, each intermediate graph of the coarsening phase is revisited and its community structure is refined by applying fast greedy refinement. After revisiting the different steps, the community structure for the original graph can be reconstructed from different coarsening levels.

Usage

```
greedy(adjacency, numRandom = 0,
        q = c("general", "danon", "wakita1", "wakita2", "wakita3"),
       initial = c("general", "prior", "walkers", "subgraph", "adclust", "own"),
        randomized = 0, refine = c("none", "complete", "fast", "kernighan"),
        coarse = 0)
rgplus(adjacency,numRandom=0,z,randomized)
msgvm(adjacency,numRandom=0,initial=c("general","own"), parL)
cd(adjacency, numRandom=0,initial=c("general","own"),maxC=length(adjacency[,1]),
  iter,p)
louvain(adjacency, numRandom=0, initial=c("general","own"))
vertexSim(adjacency, numRandom=0, frac=0.5)
mome(adjacency, numRandom=0)
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model.

q	Specify whether the general ΔQ value or a modification should be used. See details below.
initial	Specify the community structure to be used as initial partition in the algorithm. See details below.
z	The number of executions of the randomized greedy approach to identify the core groups.
randomized	The number of rows to use for the randomized greedy approach. Ignored when set to 0 (default)
refine	specifies which refinement algorithm should be used. See details below.
coarse	Define the percentage by which the number of communities has to be decreased since the last coarsening level to consider the current clustering as a new coarsening level and apply refinement on this clustering
parL	The number of merges at one step in the msgvm algorithm
maxC	The maximum number of communities for the initial partition used in the cd algorithm
iter	The number of iterations in the cd algorithm
p	The probability with which a vertex is moved into another community in the dilation step of the cd algorithm
frac	The fraction of iteration steps for which "pairwise" merging is performed in the vertexSim algorithm. Remaining iteration steps are "single neighbour" merges.

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

For the identification of the best merging event leading to a maximum increase in modularity, different values of the modularity were proposed. Which modularity value to use is specified by the parameter q. The options are general where the normal value for ΔQ is used, danon where ΔQ is normalized by the number of overall edges of vertices in a community and wakita1, wakita2 and wakita3 where ΔQ is multiplied by the consolidation ratio.

The greedy algorithms can be run on different initial partitions. The used initial partition is specified by parameter initial. The options are general where all vertices are assigned to their own community, prior where the initial community structure is identified by using prior knowledge, walkers where the initial community structure is identified by using random walkers, subgraph where the initial community structure is identified by using subgraph similarity, adclust where the general initial partition is refined using fast greedy refinement and own where the user can specify an initial partition to use with the greedy approach. In this case, the user needs to add a last column to the adjacency matrix indicating the initial partition. Hence, the adjacency matrix has to have one column more than the network has vertices.

The community structure identified by the CNM algorithm can be refined by applying a refinement step at the end of the algorithm. The used refinement algorithm is specified by the parameter refine. The options are none where no refinement algorithm is applied, complete where the complete greedy refinement is applied, fast where the fast greedy refinement is applied, kernighan where the adapted Kernighan-Lin refinement is applied. Besides, if initial is set to adclust, fast greedy refinement is applied to the community structure after each merging event. If coarse != 0,

the refinement algorithm specified by `refine` is not only applied at the end of the algorithm, but at each coarsening level where coarsening levels are defined according to `coarse`.

Value

The result of the greedy algorithms is a list with the following components

`number of communities`
: The number of communities detected by the algorithm

`modularity`
: The modularity of the detected community structure

`mean`
: The mean of the modularity values for random networks, only computed if `numRandom>0`

`standard deviation`
: The standard deviation of the modularity values for random networks, only computed if `numRandom>0`

`community structure`
: The community structure of the examined network given by a vector assigning each vertex its community number

`random modularity values`
: The list of the modularity values for random networks, only computed if `numRandom>0`

Author(s)

Maria Schelling, Cang Hui

References

Clauset, A., Newman, M. and Moore, C. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004.

Danon, L., Daz-Guilera, A. and Arenas, A. The effect of size heterogeneity on community identifcation in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(11):P11010, 2006.

Wakita, K. and Tsurumi, T. Finding community structure in mega-scale social networks: [extended abstract]. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1275- 1276, New York, NY, USA, 2007. ACM.

Ovelgonne, M. and Geyer-Schulz, A. Cluster cores and modularity maximization. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1204-1213, Dec 2010.

Du, H., Feldman, M. W., Li, S. and Jin, X. An algorithm for detecting community structure of social networks based on prior knowledge and modularity. *Complexity*, 12(3):53-60, 2007.

Pujol, J., Bejar, J. and Delgado, J. Clustering algorithm for determining community structure in large networks. *Phys. Rev. E*, 74:016107, Jul 2006.

Xiang, B., Chen, E.-H. and Zhou, T. Finding community structure based on subgraph similarity. In Santo Fortunato, Giuseppe Mangioni, Ronaldo Menezes, and Vincenzo Nicosia, editors, *Complex Networks*, volume 207 of *Studies in Computational Intelligence*, pages 73-81. Springer Berlin Heidelberg, 2009.

Noack, A. and Rotta, R. Multi-level algorithms for modularity clustering. Technical report, 2008.

Ye, Z., Hu, S. and Yu, J. Adaptive clustering algorithm for community detection in complex networks. *Phys. Rev. E*, 78:046115, Oct 2008.

Schuetz, P. and Caflisch, A. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Phys. Rev. E*, 77:046112, Apr 2008.

Mei, J., He, S., Shi, G., Wang, Z., and Li, W. Revealing network communities through modularity maximization by a contractiondilation method. *New Journal of Physics*, 11(4):043025, 2009.

Blondel, V. D., Guillaume. J.-L., Lambiotte, R. and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

Arab, M. and Afsharchi, M. A modularity maximization algorithm for community detection in social networks with low time complexity. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, volume 1, pages 480-487, Dec 2012.

Zhu, Z., Wang, C., Ma, L., Pan, Y. and Ding, Z. Scalable community discovery of large networks. In *Web-Age Information Management, 2008. WAIM '08. The Ninth International Conference on*, pages 381-388, July 2008.

Examples

```
#unweighted network
randomgraph1 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices1 <- which(clusters(randomgraph1)$membership==1)
graph1 <- induced.subgraph(randomgraph1,vertices1)

adj1 <- get.adjacency(graph1)
result1 <- greedy(adj1, refine = "fast")

#weighted network
randomgraph2 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices2 <- which(clusters(randomgraph2)$membership==1)
graph2 <- induced.subgraph(randomgraph2,vertices2)
graph2 <- set.edge.attribute(graph2, "weight", value=runif(ecount(graph2),0,1))

adj2 <- get.adjacency(graph2, attr="weight")
result2 <- louvain(adj2)
```

`localModularity` *Algorithms using local modularity*

Description

`localModularity` uses the local modularity to identify the local community structure around a certain vertex

`localModularityWang` uses the local modularity to identify the community structure of the entire network

Usage

```
localModularity(adjacency, srcV, k)
localModularityWang(adjacency,numRandom=0)
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
srcV	A given vertex whose local community structure should be determined by localModularity
k	The maximum number of vertices to add to the local community of srcV
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model.

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

Value

The result for localModularity is returned as a list with the following components

local community structure
: Vertices assigned to the same community as the source vertex srcV

local modularity
: The local modularity value for the determined local community

The result for localModularityWang is returned as a list with the following components

number of communities
: The number of communities detected by the algorithm

modularity
: The modularity of the detected community structure

mean
: The mean of the modularity values for random networks, only computed if numRandom>0

standard deviation
: The standard deviation of the modularity values for random networks, only computed if numRandom>0

community structure
: The community structure of the examined network given by a vector assigning each vertex its community number

random modularity values
: The list of the modularity values for random networks, only computed if numRandom>0

Author(s)

Maria Schelling, Cang Hui

References

Clauset, A. Finding local community structure in networks. *Phys. Rev. E*, 72:026132, Aug 2005.

Wang, X., Chen, G. and Lu, H. A very fast algorithm for detecting community structures in complex networks. *Physica A: Statistical Mechanics and its Applications*, 384(2):667-674, 2007.

Examples

```
#unweighted network
randomgraph1 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices1 <- which(clusters(randomgraph1)$membership==1)
graph1 <- induced.subgraph(randomgraph1,vertices1)

adj1 <- get.adjacency(graph1)
result1 <- localModularity(adj1, srcV=1, k=4)

#weighted network
randomgraph2 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices2 <- which(clusters(randomgraph2)$membership==1)
graph2 <- induced.subgraph(randomgraph2,vertices2)
graph2 <- set.edge.attribute(graph2, "weight", value=runif(ecount(graph2),0,1))

adj2 <- get.adjacency(graph2, attr="weight")
result2 <- localModularityWang(adj2)
```

---

simulatedAnnealing *Simulated annealing algorithms*

Description

The functions presented here are based on simulated annealing and identify the community structure and maximize the modularity. `simulatedAnnealing` is only based on moving a single vertex from one community to another, while `saIndividualCollectiveMoves` considers movements of vertices, merging of communities and splitting of communities as alternatives to increase the modularity.

Usage

```
simulatedAnnealing(adjacency, numRandom = 0,
                   initial = c("general", "random","greedy", "own"),
                   beta = length(adjacency[1, ])/2, alpha = 1.005, fixed)
```

```
saIndividualCollectiveMoves(adjacency,numRandom=0,initial=c("general","own"),
                            beta=length(adjacency[1,])/2,alpha=1.005,
                            fixed=25,numIter=1.0)
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model.
initial	Specify the community structure to use as the initial partition in the algorithm. See details below.
beta	Define the initial inverse temperature. Default is (network size)/2
alpha	Define the cooling parameter. Default is 1.005
fixed	If the community structure has not changed for this specified number of steps, the algorithm is terminated.
numIter	Define the iteration factor. At each temperature, the algorithm performs fn^2 individual moves (movement of a single vertex) and fn collective moves (merge or split of a community) where n is the number of vertices in the network.

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

The initial partition used in the simulated annealing algorithms can either be the generic one where all vertices are put in their own community (initial=general) or the initial partition can be identified by randomly identifying the initial number of communities and randomly assigning the vertices to one of these communities (initial=random) or the initial partition can be the community structure identified by the greedy algorithm (initial=greedy) or the initial partition can be given by the user (initial=own). In this case, the user needs to add a last column to the adjacency matrix indicating the initial partition. Hence, the adjacency matrix has to have one column more than the network has vertices.

Value

The result of the simulated annealing algorithms is a list with the following components

number of communities	The number of communities detected by the algorithm
modularity	The modularity of the detected community structure
mean	The mean of the modularity values for random networks, only computed if numRandom>0
standard deviation	The standard deviation of the modularity values for random networks, only computed if numRandom>0

`community structure`
: The community structure of the examined network given by a vector assigning each vertex its community number

`random modularity values`
: The list of the modularity values for random networks, only computed if `numRandom>0`

Author(s)

Maria Schelling, Cang Hui

References

Medus, A., Acua, G. and Dorso, C.O. Detection of community structures in networks via global optimization. *Physica A: Statistical Mechanics and its Applications*, 358(24):593-604, 2005.

Massen, C. and Doye, J. Identifying communities within energy landscapes. *Phys. Rev. E*, 71:046101, Apr 2005.

Guimera, R. and Amaral, L. A. N. Nunes amaral. Functional cartography of complex metabolic networks. *Nature*, 2005.

Examples

```
#unweighted network
randomgraph <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices <- which(clusters(randomgraph)$membership==1)
graph <- induced.subgraph(randomgraph,vertices)

adj <- get.adjacency(graph)
result <- simulatedAnnealing(adj, fixed=10)
```

`spectralOptimization` *Spectral optimization algorithms*

Description

`spectralOptimization` uses the leading eigenvector to recursively split the communities of a network into two until no further improvement of modularity is possible.

`multiWay`, `spectral1` and `spectral2` use $k-1$ leading eigenvectors to split the network into k communities. The value for k leading to the best community structure is chosen as the final number of communities and the resulting split of the network into k communities as the final community structure. The 3 functions implement slightly different approaches leading to possibly different results.

Usage

```
spectralOptimization(adjacency, numRandom = 0, initial = c("general", "own"),
                     refine = FALSE)
multiWay(adjacency, numRandom=0, maxComm=length(adjacency[1,]))
spectral1(adjacency, numRandom=0, maxComm=(length(adjacency[1,])-1))
spectral2(adjacency, numRandom=0, maxComm=(length(adjacency[1,])-1))
```

Arguments

adjacency	A nonnegative symmetric adjacency matrix of the network whose community structur will be analyzed
numRandom	The number of random networks with which the modularity of the resulting community structure should be compared (default: no comparison). see details below for further explanation of the used null model.
initial	Specify the community structure to use as initial partition in the algorithm. See details below.
refine	If TRUE, Kernighan-Lin refinement is applied after splitting a community into two communities only on this part of the network.
maxComm	THe maximum number of communities that the network allows

Details

The used random networks have the same number of vertices and the same degree distribution as the original network.

The initial partition used in the spectral optimization algorithm can either be the generic one where all vertices are put in their own community (initial=general) or the initial partition can be given by the user (initial=own). In this case, the user needs to add a last column to the adjacency matrix indicating the initial partition. Hence, the adjacency matrix has to have one column more than the network has vertices.

Value

The result of the spectral optimization algorithms is a list with the following components

number of communities	The number of communities detected by the algorithm
modularity	The modularity of the detected community structure
mean	The mean of the modularity values for random networks, only computed if numRandom>0
standard deviation	The standard deviation of the modularity values for random networks, only computed if numRandom>0
community structure	The community structure of the examined network given by a vector assigning each vertex its community number
random modularity values	The list of the modularity values for random networks, only computed if numRandom>0

Author(s)

Maria Schelling, Cang Hui

References

Newman, M. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74:036104, Sep 2006.

Newman, M. E. J. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577-8582, 2006.

Wang, G., Shen, Y., and Ouyang, M. A vector partitioning approach to detecting community structure in complex networks. *Computers and Mathematics with Applications*, 55(12):2746-2752, 2008.

White, S. and Smyth, P. A spectral clustering approach to finding communities in graphs. In *In SIAM International Conference on Data Mining*, 2005.

Examples

```
#unweighted network
randomgraph1 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices1 <- which(clusters(randomgraph1)$membership==1)
graph1 <- induced.subgraph(randomgraph1,vertices1)

adj1 <- get.adjacency(graph1)
result1 <- spectralOptimization(adj1, refine = TRUE)

#weighted network
randomgraph2 <- erdos.renyi.game(10, 0.3, type="gnp",directed = FALSE, loops = FALSE)

#to ensure that the graph is connected
vertices2 <- which(clusters(randomgraph2)$membership==1)
graph2 <- induced.subgraph(randomgraph2,vertices2)
graph2 <- set.edge.attribute(graph2, "weight", value=runif(ecount(graph2),0,1))

adj2 <- get.adjacency(graph2, attr="weight")
result2 <- multiWay(adj2, maxComm=3)
```

Index