Package 'multinet'

December 14, 2018

Type Package

Title Analysis and Mining of Multilayer Social Networks

Version 2.0.1

Date 2018-12-07

Author Matteo Magnani, Davide Vega, Mikael Dubik (glouvain). The package uses functions from eclat (www.borgelt.net/eclat.html), for association rule mining, Eigen (eigen.tuxfamily.org) and spectra (https://spectralib.org), for matrix manipulation, Infomap (www.mapequation.org), for the Infomap community detection method, and Howard Hinnant's date and time library (https://github.com/HowardHinnant/date). The code from this libraries has been included in our source package.

Maintainer Matteo Magnani <matteo.magnani@it.uu.se>

Description Functions for the creation/generation and analysis of multilayer social networks.

License GPL

Depends igraph (>= 1.0.1), Rcpp (>= 0.12.11), methods

LinkingTo Rcpp

RcppModules multinet

SystemRequirements A C++14 compiler

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-12-14 12:20:03 UTC

R topics documented:

`multinet-package` *Multilayer social network analysis and mining*

Description

This package defines a class to store multilayer networks and functions to pre-process, analyze and mine them.

With *multilayer social network* we indicate a network where nodes (V) are organized into multiple layers (L) and each node corresponds to an actor (A), where the same actor can be mapped to nodes in different layers. Formally, a multilayer social network as implemented in this package is a graph G = (V, E) where V is a subset of A x L.

In this manual, *Getting networks: IO* describes functions to read and write multilayer networks from/to file and the file format. To quickly test some features of the library, some existing multilayer networks are also included (*Getting networks: predefined*). A synthetic multilayer network can be generated using the growing models described in *Getting networks: generation*.

Updating and getting information about the basic components of a multilayer network (layers, actors, nodes and edges) can be done using the methods described in *Network management: properties*, *Network management: updates* and *Network management: edge directionality*. *Navigation* shows how to retrieve the neighbors of a node. Attribute values can also be attached to the basic components of a multilayer network (actors, layers, nodes and edges). Attribute management is described in *Network management: attributes*.

Each individual layer as well as combination of layers obtained using the data pre-processing (flattening) functions described in *Transformation* can be analyzed as a single-layer network using the iGraph package, by converting them as shown in *Conversion*. We can also visualize small networks using the method described in *Plotting*.

Multilayer network analysis measures are described in *Measures: basic* (for single-actor, degree-based measures), *Measures: distance-based* (for measures based on geodesic distances) and *Measures: layer comparison* (to compare different layers).

Communities can be extracted using various clustering algorithms, described in *Mining: communities*.

Most of the methods provided by this package are described in the book "Multilayer Social Networks". These methods have been proposed by many different authors: extensive references are

available in the book, and in the documentation of each function we indicate the main reference we have followed for the implementation. For a few methods developed after the book was published we give specific references to the corresponding literature.

Author(s)

Matteo Magnani <matteo.magnani@it.uu.se>

References

Dickison, Magnani, and Rossi, 2016. Multilayer Social Networks. Cambridge University Press. ISBN: 978-1107438750

Classes *Classes defined by the package*

Description

The multinet package defines two classes to represent multilayer networks (RMLNetwork) and evolutionary models for the generation of networks (REvolutionModel). Objects of these types are used as input or returned as output of the functions provided by the package, as detailed in the description of each function.

Conversion *Conversion to a simple or multi graph*

Description

Constructs a single graph resulting from merging one or more layers of the network and converts it into an iGraph object.

Usage

```
## S3 method for class 'Rcpp_RMLNetwork'
as.igraph(x, layers = NULL, merge.actors=TRUE, all.actors=FALSE, ...)
```

Arguments

x	A multilayer network.
layers	A vector of names of layers. If NULL, all layers are included in the result.
merge.actors	Whether the nodes corresponding to each actor should be merged into a single node (true) or kept separated (false).
all.actors	Whether all actors in the multilayer network should be included in the result (true) or only those present in at least one of the input layers (false).
...	Additional arguments. None currently.

Value

An object of class iGraph.

Examples

```
net <- ml.aucs()
# using the default merge.actors=TRUE we create a multigraph,
# where each actor corresponds to a node in the result
multigraph <- as.igraph(net)
# this is a simple graph corresponding to the facebook layer
facebook1 <- as.igraph(net, "facebook")
# this includes also the actors without a facebook account
facebook2 <- as.igraph(net, "facebook", all.actors=TRUE)
# two layers are converted to an igraph object, where two
# nodes are used for each actor: one corresponding to the
# node on facebook, one to the node on lunch
f_l_net <- as.igraph(net, c("facebook","lunch"),
    merge.actors=FALSE)
```

Getting networks: generation

Generation of multilayer networks

Description

The grow.ml function generates a multilayer network by letting it grow for a number of steps, where for each step three events can happen: (1) evolution according to internal dynamics (in which case a specific internal evolution model is used), (2) evolution importing edges from another layer, and (3) no action.

The functions evolution.pa.ml and evolution.er.ml define, respectively, an evolutionary model based on preferential attachment and an evolutionary model where edges are created by chosing random end points, as in the ER random graph model.

Usage

```
grow.ml(num.actors, num.steps, models, pr.internal, pr.external, dependency)
evolution.pa.ml(m0,m)
evolution.er.ml(n)
```

Arguments

num.actors	The number of actors from which new nodes are selected during the generation process.
num.steps	Number of timestamps.
models	A vector containing one evolutionary model for each layer to be generated. Evolutionary models are defined using the evolution.*.ml functions.

`pr.internal`	A vector with (for each layer) the probability that at each step the layer evolves according to the internal evolutionary model.
`pr.external`	A vector with (for each layer) the probability that at each step the layer evolves importing edges from another layer.
`dependency`	A matrix LxL where element (i,j) indicates the probability that layer i will import an edge from layer j in case an external event is triggered.
`m0`	Initial number of nodes.
`m`	Number of edges created for each new node joining the network.
`n`	Number of nodes (created at the beginning, before starting adding edges).

Value

`grow.ml` returns a multilayer network. `evolution.*.ml` return evolutionary models that are used by `grow.ml` to decide how each layer should grow.

References

Magnani, Matteo, and Luca Rossi. 2013. Formation of Multiple Networks. In Social Computing, Behavioral-Cultural Modeling and Prediction, 257-264. Springer Berlin Heidelberg.

See Also

Getting networks: predefined, Getting networks: IO

Examples

```
# we generate a network with two layers, one growing according
# to the Preferential Attachment model and one growing by selecting
# new edges uniformly at random.
models <- c(evolution.pa.ml(3,1), evolution.er.ml(50))
# all the probability vectors must have the same number of
# fields, one for each layer: two in this example
# by defining pr.internal and pr.external, we are also implicitely defining
# pr.no.action (1 minus the other probabilities, for each field/layer).
pr.external <- c(.5,0)
pr.internal <- c(.5,.5)
# each layer will import edges from the other if needed
# (not the second layer in this example: it has 0 probability of external events)
dependency <- matrix(c(0,1,1,0),2,2)
# 100 steps of network growing, adding actors from a pool of 100
grow.ml(100, 100, models, pr.internal, pr.external, dependency)
```

`Getting networks: IO` *Reading and writing multilayer networks from/to file*

Description

These functions are used to store a multilayer network to a file or load it from a file.

The format used by the package is the following:
-- comment lines start with two dashes (--)
#TYPE multiplex
-- or: "#TYPE multilayer", if there are inter-layer edges
#ACTOR ATTRIBUTES
AttributeName1,STRING
AttributeName2,NUMERIC
-- etc.
#NODE ATTRIBUTES
LayerName1,AttributeName1,STRING
LayerName1,AttributeName2,NUMERIC
LayerName2,AttributeName3,STRING
-- etc.
#EDGE ATTRIBUTES
-- if type is multiplex (default), edge attributes are indicated as follows:
LayerName1,AttributeName,STRING
LayerName2,AttributeName,NUMERIC
-- etc.
-- if type is multilayer, edge attributes are indicated as follows:
-- LayerName1,LayerName1,AttributeName,STRING
-- LayerName1,LayerName2,AttributeName,NUMERIC
-- etc.
#LAYERS
-- if type is multiplex (default), layers are indicated as follows:
LayerName1,UNDIRECTED,AttributeValueList...
LayerName2,DIRECTED,AttributeValueList...
-- etc.
-- if type is multilayer, layers are indicated as follows:
-- LayerName1,LayerName1,UNDIRECTED,AttributeValueList...
-- LayerName2,LayerName2,DIRECTED,AttributeValueList...
-- LayerName1,LayerName2,DIRECTED
-- etc. and all intra-layer directionalities should be listed before inter-layer directionalities
#ACTORS
ActorName1,AttributeValueList...
ActorName2,AttributeValueList...
-- etc.
#NODES
ActorName1,LayerName1,AttributeValueList...
ActorName1,LayerName2,AttributeValueList...
-- etc.

#EDGES
-- if TYPE is multiplex (default), edges are indicated as follows:
ActorName1,ActorName2,LayerName1,AttributeValueList...
-- etc.
-- if TYPE is multilayer, edges are instead indicated as follows:
-- ActorName1,LayerName1,ActorName2,LayerName2,AttributeValueList...
-- etc.

If the #LAYERS section is empty, undirected layers are created as mentioned in the #EDGES section.

If the #LAYER ATTRIBUTES, #ACTOR ATTRIBUTES, #NODE ATTRIBUTES or #EDGE ATTRIBUTES sections are empty, no attributes are created.

The #LAYERS #ACTORS and #NODES sections are useful only if attributes are present, or if there are actors not present in any layer, or if there are isolated nodes, otherwise they can be omitted.

If no section is specified, #EDGES is the default.

Therefore, a minimalistic undirected multiplex network file would look like this:

```
---
Matteo,Luca,Facebook
Matteo,Mark,Facebook
Luca,Mark,Twitter
---
```

Usage

```
read.ml(file, name="unnamed", sep=',', aligned=FALSE)
write.ml(mlnetwork, file, format="multilayer", layers=character(0),
  sep=',', merge.actors=TRUE, all.actors=FALSE)
```

Arguments

`file`	The path of the file storing the multilayer network.
`name`	The name of the multilayer network.
`mlnetwork`	A multilayer network.
`layers`	If specific layers are passed to the function, only those layers are saved to file.
`format`	Either "multilayer", to use the package's internal format, or "graphml".
`sep`	The character used in the file to separate text fields.
`aligned`	If true, all actors are added to all layers.
`merge.actors`	Whether the nodes corresponding to each single actor should be merged into a single node (true) or kept separated (false), when format="graphml" is used.
`all.actors`	Whether all actors in the multilayer network should be included in the output file (true) or only those present in at least one of the input layers (false), when format="graphml" and merge.actors=TRUE are used.

Value

`read.ml` returns a multilayer network. `write.ml` does not return any value.

See Also

Getting networks: predefined, Getting networks: generation

Examples

```
# writing a network to file...
file <- tempfile("aucs.mpx")
net <- ml.aucs()
write.ml(net,file)
# ...and reading it back into a variable
net <- read.ml(file,"AUCS")
net
# the following network has more nodes, because all
# actors are replicated to all graphs
net.aligned <- read.ml(file,"AUCS",aligned=TRUE)
net.aligned
```

Getting networks: predefined

Loading predefined multilayer networks

Description

Creates predefined multilayer networks from the literature.

- `ml.empty` returns an empty multilayer network, not containing any actor, layer, node or edge.
- `ml.aucs` returns the AUCS multiplex network described in *Rossi and Magnani, 2015. "Towards effective visual analytics on multiplex networks". Chaos, Solitons and Fractals. Elsevier.*
- `ml.bankwiring` returns Padgett's Florentine Families multiplex network.
- `ml.florentine` returns Padgett's Florentine Families multiplex network.
- `ml.monastery` returns Sampson's monastery multiplex network.
- `ml.tailorshop` returns Kapferer's' tailorshop multiplex network.
- `ml.toy` returns the toy network used as a running example in *Dickison, Magnani and Rossi. "Multilayer Social Networks". Cambridge University Press.*

Usage

```
ml.empty(name="unnamed")
ml.aucs()
ml.bankwiring()
ml.florentine()
ml.monastery()
ml.tailorshop()
ml.toy()
```

Arguments

name	The name of the new multilayer network.

Value

All these functions return a multilayer network.

References

Rossi, Luca, and Magnani, Matteo (2015). Towards effective visual analytics on multiplex and multilayer networks. Chaos, Solitons and Fractals, 72, 68-76. (for ml.aucs()). Padgett, John F., and McLean, Paul D. (2006). Organizational Invention and Elite Transformation: The Birth of Partnership Systems in Renaissance Florence. American Journal of Sociology, 111(5), 1463-1568. (for ml.florentine()). Breiger, R. and Boorman, S. and Arabic, P. (1975). An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. Journal of Mathematical Psychology, 12 (for ml.monastery() and ml.bankwiring() - these authors prepared the datasets, see multilayer.it.uu.se/datasets.html for references to the data collectors). Kapferer, Bruce (1972). Strategy and Transaction in an African Factory: African Workers and Indian Management in a Zambian Town. Manchester University Press (fot ml.tailorshop()).

See Also

Getting networks: IO, Getting networks: generation

Examples

```
empty <- ml.empty("new network")
aucs <- ml.aucs()
bankwiring <- ml.bankwiring()
florentine <- ml.florentine()
monastery <- ml.monastery()
tailorshop <- ml.tailorshop()
```

Layout *Layouts*

Description

These functions compute xyz coordinates for each node in the network.

Usage

```
layout.multiforce.ml(mlnetwork, w_in=1, w_inter=1, gravity=0, iterations=100)
layout.circular.ml(mlnetwork)
```

Arguments

mlnetwork	A multilayer network.
w_in	An array with weights for intralayer forces, or a single number if weights are the same for all layers.
w_inter	An array with weights for interlayer forces, or a single number if weights are the same for all layers.
gravity	An array with weights for gravity forces, or a single number if weights are the same for all layers.
iterations	Number of iterations.

References

Fatemi, Zahra, Salehi, Mostafa, & Magnani, Matteo (2018). A generalised force-based layout for multiplex sociograms. Social Informatics

See Also

Plotting

Examples

```
net <- ml.florentine()
layout.multiforce.ml(net)
l <- layout.circular.ml(net)
## Not run:
plot(net,layout=l)
## End(Not run)
```

`Measures: basic` *Network analysis measures*

Description

These functions compute network analysis measures providing a basic description of the actors in the network.

Usage

```
degree.ml(mlnetwork,actors=character(0),layers=character(0),mode="all")
degree.deviation.ml(mlnetwork,actors=character(0),
  layers=character(0),mode="all")
neighborhood.ml(mlnetwork,actors=character(0),layers=character(0),mode="all")
xneighborhood.ml(mlnetwork,actors=character(0),layers=character(0),mode="all")
connective.redundancy.ml(mlnetwork,actors=character(0),
  layers=character(0),mode="all")
relevance.ml(mlnetwork,actors=character(0),layers=character(0),mode="all")
xrelevance.ml(mlnetwork,actors=character(0),layers=character(0),mode="all")
```

Arguments

`mlnetwork`	A multilayer network.
`actors`	An array of names of actors.
`layers`	An array of names of layers.
`mode`	This argument can take values "in", "out" or "all" to count respectively incoming edges, outgoing edges or both.

Value

`degree.ml` returns the number of edges adjacent to the input actor restricted to the specified layers. `degree.deviation.ml` returns the standard deviation of the degree of an actor on the input layers. An actor with the same degree on all layers will have deviation 0, while an actor with a lot of neighbors on one layer and only a few on another will have a high degree deviation, showing an uneven usage of the layers (or layers with different densities).

`neighborhood.ml` returns the number of actors adjacent to the input actor restricted to the specified layers. `xneighborhood.ml` returns the number of actors adjacent to the input actor restricted to the specified layers and not present in the other layers.

`connective.redundancy.ml` returns 1 minus neighborhood divided by degree.

`relevance.ml` returns the percentage of neighbors present on the specified layers. `xrelevance.ml` returns the percentage of neighbors present on the specified layers and not on others.

References

Berlingerio, Michele, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. 2011. "Foundations of Multidimensional Network Analysis." In International Conference on Social Network Analysis and Mining (ASONAM), 485-89. IEEE Computer Society.

Magnani, Matteo, and Luca Rossi. 2011. "The ML-Model for Multi-Layer Social Networks." In International conference on Social Network Analysis and Mining (ASONAM), 5-12. IEEE Computer Society.

Examples

```
net <- ml.aucs()
# degrees of all actors, considering edges on all layers
degree.ml(net)
# degree of actors U54 and U3, only considering layers work and coauthor
degree.ml(net,c("U54","U3"),c("work","coauthor"),"in")
# an indication of whether U54 and U3 are selectively active only on some layers
degree.deviation.ml(net,c("U54","U3"))
# co-workers of U54
neighborhood.ml(net,"U54","work")
# co-workers of U54 who are not connected to U54 on other layers
xneighborhood.ml(net,"U54","work")
# percentage of neighbors of U54 who are also co-workers
relevance.ml(net,"U54","work")
# redundancy between work and lunch
connective.redundancy.ml(net,"U54",c("work","lunch"))
# percentage of neighbors of U54 who would no longer
# be neighbors by removing this layer
xrelevance.ml(net,"U54","work")
```

Measures: distance-based

Network analysis measures: distance based

Description

This function is based on the concept of multilayer distance. This concept generalizes single-layer distance to a vector with the distance traveled on each layer (in the "multiplex" case). Therefore, non-dominated path lengths are returned instead of shortest path length, where one path length dominates another if it is not longer on all layers, and shorter on at least one. A non-dominated path length is also known as a Pareto distance. Finding all multilayer distances can be very time-consuming for large networks.

Usage

```
distance.ml(mlnetwork,from,to=character(0),method="multiplex")
```

Arguments

`mlnetwork`	A multilayer network.
`from`	The actor from which the distance is computed.
`to`	The actor(s) to which the distance is computed. If not specified, all actors are considered.
`method`	This argument can take values "simple", "multiplex", "full". Only "multiplex" is currently implemented.

Value

A data frame with one row for each non-dominated distance, specifying the number of steps in each layer.

References

Magnani, Matteo, and Rossi, Luca (2013). Pareto Distance for Multi-layer Network Analysis. In Social Computing, Behavioral-Cultural Modeling and Prediction (Vol. 7812, pp. 249-256). Springer Berlin Heidelberg.

Examples

```
net <- ml.aucs()
distance.ml(net,"U54","U3")
```

`Measures: layer comparison`

Network analysis measures

Description

These functions can be used to compare different layers.

Usage

```
layer.summary.ml(mlnetwork,layer,method="entropy.degree",mode="all")
layer.comparison.ml(mlnetwork,layers=character(0),method="jaccard.edges",mode="all",K=0)
```

Arguments

`mlnetwork`	A multilayer network.
`layer`	The name of a layer.
`layers`	Names of the layers to be compared. If not specified, all layers are used.
`method`	This argument can take several values. For layer summary: "min.degree", "max.degree", "sum.degree", "mean.degree", "sd.degree", "skewness.degree", "kurtosis.degree", "entropy.degree", "CV.degree", "jarque.bera.degree". For layer comparison:

- Overlapping:"jaccard.actors", "jaccard.edges", "jaccard.triangles", "coverage.actors", "coverage.edges", "coverage.triangle","sm.actors", "sm.edges", "sm.triangles", "rr.actors", "rr.edges", "rr.triangles", "kulczynski2.actors", "kulczynski2.edges", "kulczynski2.triangles", "hamann.actors", "hamann.edges", "hamann.triangles". The first part of the value indicates the type of comparison function (Jaccard, Coverage, Simple Matching, Russell Rao, Kulczynski, Hamann), the second part indicates the configurations to which the comparison function is applied.
- Distribution dissimilarity:"dissimilarity.degree", "KL.degree", "jeffrey.degree". Notice that these are dissimilarity functions: 0 means highest similarity
- Correlation:"pearson.degree" and "rho.degree"

`mode` This argument is used for distribution dissimilarities and correlations (that is, those methods based on node degree) and can take values "in", "out" or "all" to consider respectively incoming edges, outgoing edges or both.

`K` This argument is used for distribution dissimilarity measures and indicates the number of histogram bars used to compute the divergence. If 0 is specified, then a "typical" value is used, close to the logarithm of the number of actors.

Value

A data frame with layer-by-layer coparisons. For each pair of layers, the data frame contains a value between 0 and 1 (for overlapping and distribution dissimilarity) or -1 and 1 (for correlation).

References

Brodka, P., Chmiel, A., Magnani, M., and Ragozini, G. (2018). Qunatifying layer similarity in multiplex networks: a systematic study. Royal Sociwty Open Science 5(8)

Examples

```
net <- ml.aucs()

# computing similarity between layer summaries
s1 = layer.summary.ml(net,"facebook",method="entropy.degree")
s2 = layer.summary.ml(net,"lunch",method="entropy.degree")
relative.difference=abs(s1-s2)*2/(abs(s1)+abs(s2))
# other layer summaries
layer.summary.ml(net,"facebook",method="min.degree")
layer.summary.ml(net,"facebook",method="max.degree")
layer.summary.ml(net,"facebook",method="sum.degree")
layer.summary.ml(net,"facebook",method="mean.degree")
layer.summary.ml(net,"facebook",method="sd.degree")
layer.summary.ml(net,"facebook",method="skewness.degree")
layer.summary.ml(net,"facebook",method="kurtosis.degree")
layer.summary.ml(net,"facebook",method="entropy.degree")
layer.summary.ml(net,"facebook",method="CV.degree")
layer.summary.ml(net,"facebook",method="jarque.bera.degree")

# returning the number of common edges divided by the union of all
# edges for all pairs of layers (jaccard.edges)
```

```
layer.comparison.ml(net)
# returning the number of common edges divided by the union of all
# edges only for "lunch" and "facebook" (jaccard.edges)
layer.comparison.ml(net,layers=c("lunch","facebook"))
# returning the percentage of actors in the lunch layer that are
# also present in the facebook layer
layer.comparison.ml(net,method="coverage.actors")
# all overlapping-based measures:
layer.comparison.ml(net,method="jaccard.actors")
layer.comparison.ml(net,method="jaccard.edges")
layer.comparison.ml(net,method="jaccard.triangles")
layer.comparison.ml(net,method="coverage.actors")
layer.comparison.ml(net,method="coverage.edges")
layer.comparison.ml(net,method="coverage.triangles")
layer.comparison.ml(net,method="sm.actors")
layer.comparison.ml(net,method="sm.edges")
layer.comparison.ml(net,method="sm.triangles")
layer.comparison.ml(net,method="rr.actors")
layer.comparison.ml(net,method="rr.edges")
layer.comparison.ml(net,method="rr.triangles")
layer.comparison.ml(net,method="kulczynski2.actors")
layer.comparison.ml(net,method="kulczynski2.edges")
layer.comparison.ml(net,method="kulczynski2.triangles")
layer.comparison.ml(net,method="hamann.actors")
layer.comparison.ml(net,method="hamann.edges")
layer.comparison.ml(net,method="hamann.triangles")

# comparison of degree distributions (divergences)
layer.comparison.ml(net,method="dissimilarity.degree")
layer.comparison.ml(net,method="KL.degree")
layer.comparison.ml(net,method="jeffrey.degree")

# statistical degree correlation
layer.comparison.ml(net,method="pearson.degree")
layer.comparison.ml(net,method="rho.degree")
```

`Mining: communities` *Community detection algorithms*

Description

Various algorithms to compute communities in multiplex networks, based on frequent itemset mining (abacus), adjacent cliques (clique percolation), modularity optimization (generalized louvain) and random walks (lart). get.community.list.ml is a commodity function translating the result of these algorithms into a list of node identifiers, and is internally used by the plotting function.

Usage

```
abacus.ml(mlnetwork, min.actors=3, min.layers=1)
clique.percolation.ml(mlnetwork, k=3, m=1)
```

```
glouvain.ml(mlnetwork, gamma=1, omega=1, limit=0)
infomap.ml(mlnetwork, overlapping=FALSE, directed=FALSE, self.links=TRUE)

get.community.list.ml(comm.struct, mlnetwork)
```

Arguments

mlnetwork	A multilayer network.
min.actors	Minimum number of actors to form a community.
min.layers	Minimum number of times two actors must be in the same single-layer community to be considered in the same multi-layer community.
k	Minimum number of actors in a clique. Must be at least 3.
m	Minimum number of common layers in a clique.
gamma	Resolution parameter for modularity in the generalized louvain and lart methods.
omega	Inter-layer weight parameter in the generalized louvain method.
limit	Limit parameter in the generalized louvain method: if the number of nodes exceeds this limit, then the modularity is computed on the fly without keeping the full data in memory.
overlapping	Specifies if overlapping clusters can be returned.
directed	Specifies whether the edges should be considered as directed.
self.links	Specifies whether self links should be considered or not.
comm.struct	The result of a community detection method.

Value

All community detection algorithms return a data frame where each row contains actor name, layer name and community identifier.

`get.community.list.ml` transforms the output of a community detection function into a list by grouping all the nodes having the same community identifier and the same layer.

References

Berlingerio, Michele, Pinelli, Fabio, and Calabrese, Francesco (2013). ABACUS: frequent pAttern mining-BAsed Community discovery in mUltidimensional networkS. Data Mining and Knowledge Discovery, 27(3), 294-320. (for abacus.ml()) Afsarmanesh, Nazanin, and Magnani, Matteo (2018). Partial and overlapping community detection in multiplex social networks. Social informatics (for clique.percolation.ml()) Mucha, Peter J., Richardson, Thomas, Macon, Kevin, Porter, Mason A., and Onnela, Jukka-Pekka (2010). Community structure in time-dependent, multiscale, and multiplex networks. Science (New York, N.Y.), 328(5980), 876-8. Data Analysis, Statistics and Probability; Physics and Society. (for glouvain.ml()) De Domenico, M., Lancichinetti, A., Arenas, A., and Rosvall, M. (2015) Identifying Modular Flows on Multilayer Networks Reveals Highly Overlapping Organization in Interconnected Systems. PHYSICAL REVIEW X 5, 011027 (for infomap.ml())

See Also

Plotting

Examples

```
net <- ml.florentine()
abacus.ml(net)
clique.percolation.ml(net)
glouvain.ml(net)
infomap.ml(net)
```

Navigation *Functions to extract neighbors of nodes, to navigate the network*

Description

These functions return actors who are connected to the input actor through an edge. They can be used to navigate the graph, following paths inside it.

Usage

```
neighbors.ml(mlnetwork, actor, layers=character(0), mode="all")
xneighbors.ml(mlnetwork, actor, layers=character(0), mode="all")
```

Arguments

mlnetwork	A multilayer network.
actor	An actor name present in the network, whose neighbors are extracted.
layers	An array of layers belonging to the network. Only the nodes in these layers are returned. If the array is empty, all the nodes in the network are returned.
mode	This argument can take values "in", "out" or "all" to indicate respectively neighbors reachable via incoming edges, via outgoing edges or both.

Value

`neighbors.ml` returns the actors who are connected to the input actor on at least one of the specified layers. `xneighbors.ml` (eXclusive neighbors) returns the actors who are connected to the input actor on at least one of the specified layers, and on none of the other layers. Exclusive neighbors are those neighbors that would be lost by removing the input layers.

References

Berlingerio, Michele, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. 2011. "Foundations of Multidimensional Network Analysis." In International Conference on Social Network Analysis and Mining (ASONAM), 485-89. IEEE Computer Society.

See Also

Network management: properties

Examples

```
net <- ml.aucs()
# out-neighbors of U54, that is, all A such that there is an edge ("U54",A)
neigh <- neighbors.ml(net, "U54", mode="out")
# all in-neighbors of U54 on the "work" layer who are not in-neighbors
# in any other layer
xneigh <- xneighbors.ml(net, "U54", "work", mode="in")
# all neighbors (in- and out-) of U54 on the "work" and "lunch" layers
# who are not neighbors in any other layer
xneigh <- xneighbors.ml(net, "U54", c("work","lunch"))
```

Network management: attributes

Managing attributes

Description

These functions are used to add values to the network components and retrieve them.

The functions `new.attributes.ml` and `list.attributes.ml` are deprecated in the current version of the library.

Usage

```
add.attributes.ml(mlnetwork, attributes, type="string", target="actor",
layer="", layer1="", layer2="")
attributes.ml(mlnetwork, target="actor")
get.values.ml(mlnetwork, attribute, actors=character(0),
vertices =character(0), edges=character(0))
set.values.ml(mlnetwork, attribute, actors=character(0),
vertices=character(0), edges=character(0), values)
```

Arguments

`mlnetwork`	A multilayer network.
`attributes`	Name(s) of the attributes to be created.
`target`	Can be "actor" (attributes attached to actors), "vertex" (attributes attached to vertices) or "edge" (attributes attached to edges). Layer attributes are not available in this version.
`type`	Can be "string" or "numeric".
`layer`	This can be specified only for targets "vertex" (so that the attribute exists only for the vertices in that layer) or "edge" (in which case the attribute applies to intra-layer edges in that layer).
`layer1`	This can be specified only for target "edge", together with layer2, so that the attribute applies to inter-layer edges from layer1 to layer2. If layer1 and layer2 are specified, layer should not.

`layer2`	See layer1.
`attribute`	The name of the attribute to be updated.
`actors`	A vector of actor names. If this is specified, layers, vertices and edges should not.
`vertices`	A dataframe of vertices to be updated. The first column specifies actor names, the second layer names. If this is specified, actors, layers and edges should not.
`edges`	A dataframe containing the vertices to be connected. The four columns must contain, in this order: actor1 name, layer1 name, actor2 name, layer2 name. If this is specified, actors, layers and vertices should not.
`values`	A vector of values to be set for the corresponding actors, layers, vertices or edges.

Value

`attributes.ml` returns a data frame with columns: "name", and "type". If vertex attributes are listed, an additional "layer" column is used. If edge attributes are listed, two columns "layer1" and "layer2" are included. `get.values.ml` returns a data frame with the values for the requested objects.

Examples

```
net <- ml.aucs()
attributes.ml(net)
# actor attributes, of string type (default)
add.attributes.ml(net,c("name","surname"))
# a numeric attribute associated to the layers (not available in this version)
# add.attributes.ml(net,"num vertices",type="numeric",target="layer")
# attributes for vertices on the facebook layer
add.attributes.ml(net,"username",type="string",target="vertex",layer="facebook")
# attributes for edges on the work layer
add.attributes.ml(net,"strength",type="numeric",target="edge",layer="work")
# listing the attributes
attributes.ml(net)
# attributes.ml(net,"layer") # not available in this version
attributes.ml(net,"vertex")
attributes.ml(net,"edge")
# setting some values for the newly created attributes
set.values.ml(net,"name",actors=c("U54","U139"),values=c("John","Johanna"))
e <- data.frame(
    c("U139","U139"),
    c("work","work"),
    c("U71","U97"),
    c("work","work"))
set.values.ml(net,"strength",edges=e,values=.47)
# getting the values back
get.values.ml(net,"name",actors=c("U139"))
get.values.ml(net,"strength",edges=e)
# setting attributes based on network properties: create a "degree"
# attribute and set its value to the degree of each actor
actors.ml(net) -> a
```

```
layers.ml(net) -> l
degree.ml(net,actors=a,layers=l,mode="all") -> d
add.attributes.ml(net,target="actor",type="numeric",attributes="degree")
set.values.ml(net,attribute="degree",actors=a,values=d)
get.values.ml(net,attribute="degree",actors="U54")
# select actors based on attribute values (e.g., with degree greater than 40)
get.values.ml(net,attribute="degree",actors=a) -> degrees
a[degrees>40]
# list all the attributes again
attributes.ml(net)
```

`Network management: edge directionality`
Controlling edge directionality

Description

Functions to get and set the edge directionality of one or more pairs of layers (that is, the directionality of edges connecting nodes in those layers).

Usage

```
set.directed.ml(mlnetwork, directionalities)
is.directed.ml(mlnetwork, layers1=character(0), layers2=character(0))
```

Arguments

`mlnetwork`	A multilayer network.
`directionalities`	A dataframe with three columns where each row contains a pair of layers (l1,l2) and 0 or 1 (indicating resp. undirected and directed edges). Directionality is automatically set for both (l1,l2) and (l2,l1).
`layers1`	The layer(s) from where the edges start. If layers1 is not provided, all layers are considered.
`layers2`	The layer(s) where the edges end. If an empty list of layers is passed (default), the ending layers are set as equal to those in parameter layers1.

Value

`is.directed.ml` returns a data frame where each row contains the name of two layers and the corresponding type of edges (directed/undirected).

Examples

```
net <- ml.empty()
# Adding some layers, one directed and one undirected
add.layers.ml(net,c("l1","l2"),c(TRUE,FALSE))
# Setting the directionality of inter-layer edges
layers = c("l1","l2")
dir <- data.frame(layers,layers,c(0,1))
set.directed.ml(net,dir)
# retrieving all directionalities
dir <- is.directed.ml(net)
# copying directionalities to a new network
net2 <- ml.empty()
add.layers.ml(net2,c("l1","l2"))
set.directed.ml(net2,dir)
```

Network management: properties

Listing network properties

Description

These functions are used to list basic information about the components of a multilayer network (actors, layers, vertices and edges).

The functions `nodes.ml` and `num.nodes.ml` are deprecated in the current version of the library. Vertex/vertices are now preferentially used over node/nodes.

Usage

```
layers.ml(mlnetwork)
actors.ml(mlnetwork, layers=character(0))
vertices.ml(mlnetwork, layers=character(0))
edges.ml(mlnetwork, layers1=character(0), layers2=character(0))
edges.idx.ml(mlnetwork)

num.layers.ml(mlnetwork)
num.actors.ml(mlnetwork, layers=character(0))
num.vertices.ml(mlnetwork, layers=character(0))
num.edges.ml(mlnetwork, layers1=character(0), layers2=character(0))
```

Arguments

`mlnetwork`	A multilayer network.
`layers`	An array of names of layers belonging to the network. Only the actors/vertices in these layers are returned. If the array is empty, all the vertices in the network are returned. Notice that this may not correspond to the list of actors: there can be actors that are not present in any layer. These would be returned only using the `actors.ml` function.

layers1	The layer(s) from where the edges to be extracted start. If an empty list of layers is passed (default), all the layers are considered.
layers2	The layer(s) where the edges to be extracted end. If an empty list of layers is passed (default), the ending layers are set as equal to those in parameter layer1.

Value

actors.ml and layers.ml return an array of respectively actor and layer names. vertices.ml returns a data frame where each row contains the name of the actor corresponding to that vertex and the layer of the vertex. edges.ml returns a data frame where each row contains two actor names (i.e., an edge), the name of the two layers connected by the edge (which can be the same layer if it is an intra-layer edge) and the type of edge (directed/undirected).

edges.idx.ml returns the index of the vertex as returned by the vertices.ml function instead of its name - this is used internally by the plotting function.

The functions num.* compute the number of components of the requested type. If the number of actors is requested without specifying any layer, the total number of actors is returned, including those not present in any layer.

See Also

Network management: updates, Network management: edge directionality

Examples

```
net <- ml.aucs()
actors.ml(net)
layers.ml(net)
vertices.ml(net)
# only vertices in the "facebook" layer
vertices.ml(net,"facebook")
# all edges
edges.ml(net)
# Only edges inside the "lunch" layer
edges.ml(net,"lunch","lunch")
# Does the same as in the previous line
edges.ml(net,"lunch")
# Returns an empty  data frame, because there are no edges from the
# "lunch" layer to the "facebook" layer
edges.ml(net,"lunch","facebook")

num.actors.ml(net)
num.layers.ml(net)
num.vertices.ml(net)
# Only vertices in the "facebook" layer are counted
num.vertices.ml(net,"facebook")
num.edges.ml(net)
# Only edges inside the "lunch" layer are counted
num.edges.ml(net,"lunch","lunch")
# Does the same as in the previous line
num.edges.ml(net,"lunch")
```

```
# Returns 0, because there are no edges from the "lunch" layer to
# the "facebook" layer
num.edges.ml(net,"lunch","facebook")
```

Network management: updates

Manipulation of multilayer networks

Description

Functions to add or remove components of a multilayer network.

The functions `add.vertices.ml` and `delete.vertices.ml` add/remove the input actors to/from the input layers, but do not add/remove the actors from the multilayer network.

A layer can also be added from an igraph object, where the vertex attribute `name` represents the actor name, using the `add.igraph.layer.ml` function. `add.vertices.ml` and `delete.vertices.ml` add/remove the input actors to/from the input layers, but do not add/remove the actors from the multilayer network.

The functions `add.nodes.ml` and `delete.nodes.ml` are deprecated in the current version of the library. Vertex/vertices are now preferentially used over node/nodes.

Usage

```
add.layers.ml(mlnetwork, layers, directed=FALSE)
add.actors.ml(mlnetwork, actors)
add.vertices.ml(mlnetwork, vertices)
add.edges.ml(mlnetwork, edges)

add.igraph.layer.ml(mlnetwork, g, name)

delete.layers.ml(mlnetwork, layers)
delete.actors.ml(mlnetwork, actors)
delete.vertices.ml(mlnetwork, vertices)
delete.edges.ml(mlnetwork, edges)
```

Arguments

`mlnetwork`	A multilayer network.
`layers`	An array of names of layers.
`actors`	An array of names of actors.
`g`	An igraph object with simple edges and a vertex attribute called name storing the actor name corresponding to the vertex.
`name`	Name of the new layer.
`directed`	Determines if the layer(s) is (are) directed or undirected. If multiple layers are specified, directed should be either a single value or an array with as many values as the number of layers.

vertices — A dataframe of vertices to be updated. The first column specifies actor names, the second layer names.

edges — A dataframe containing the vertices to be connected. The four columns must contain, in this order: actor1 name, layer1 name, actor2 name, layer2 name. The directionality of the edge (directed/undirected) is pre-defined depending on the layer(s).

See Also

Network management: properties, Network management: edge directionality

Examples

```
net <- ml.empty()
# Adding some layers
add.layers.ml(net,"l1")
add.layers.ml(net,c("l2","l3"),c(TRUE,FALSE))
layers.ml(net)
# Adding actors A1, A2, A3
add.actors.ml(net,"A1")
add.actors.ml(net,c("A2","A3"))
# Verifying that the actors have been added correctly
num.actors.ml(net)
actors.ml(net)
# Adding some vertices (actor A3 is not present in layer l3: no corresponding vertex there)
vertices <- data.frame(
    c("A1","A2","A3","A1","A2","A3"),
    c("l1","l1","l1","l2","l2","l2"))
add.vertices.ml(net,vertices)
vertices <- data.frame(
    c("A1","A2"),
    c("l3","l3"))
add.vertices.ml(net,vertices)
vertices.ml(net)
# We create a data frame specifying two edges:
# A2,l2 -- A3,l1
# A2,l2 -- A3,l2
edges <- data.frame(
    c("A2","A2"),
    c("l2","l2"),
    c("A3","A3"),
    c("l1","l2"))
add.edges.ml(net,edges)
edges.ml(net)

# The following deletes layer 1, ans also deletes
# all vertices from "l1" and the edge with an end-point in "l1"
delete.layers.ml(net,"l1")
# The following also deletes the vertices associated to
# "A1" in layers "l2" and "l3"
delete.actors.ml(net,"A1")
# deleting vertex A2,l3 and edge A2,l2 -- A3,l2
```

```
delete.vertices.ml(net,data.frame("A2","l3"))
edges <- data.frame("A2","l2","A3","l2")
delete.edges.ml(net,edges)
net
```

`Plotting` *Drawing a multilayer network*

Description

This function draws a multilayer network.

Usage

```
## S3 method for class 'Rcpp_RMLNetwork'
plot(x,
layout=NULL, grid=NULL, mai=.1,
vertex.shape=16, vertex.cex=1, vertex.color=NULL,
vertex.labels=NULL, vertex.labels.pos=3, vertex.labels.offset=.5, vertex.labels.cex=1,
edge.type=1, edge.width=1, edge.color=1,
edge.arrow.length=0.1, edge.arrow.angle=20,
com=NULL, com.cex=1, ...)
```

Arguments

`x`	A multilayer network.
`layout`	A data frame indicating the position of nodes. If NULL, the function layout.multiforce.ml is used to compute it.
`grid`	A vector of size 2 indicating the number of rows and columns where to draw the layers.
`mai`	Percentage of each frame reserved as internal margin. This only concerns nodes: text labels can be printed inside the margin or even outside the frame depending on their offset.
`vertex.shape`	Symbol to use for nodes, corresponding to the parameter pch of the R `points` function. This can either be a single character or an integer code for one of a set of graphics symbols. See ?`points` for more details.
`vertex.cex`	Numeric *c*haracter *ex*pansion factor; multiplied by `par("cex")` yields the final node size.
`vertex.color`	Color of the vertexes. If NULL, all vertexes in the same layer are plotted using the same color.
`vertex.labels`	A character vector or expression specifying the text to be written besides each node. It corresponds to the parameter `labels` of the R `text` function.
`vertex.labels.pos`	A position specifier for the text. Values of '1', '2', '3' and '4', respectively indicate positions below, to the left of, above and to the right of the specified coordinates. It corresponds to the parameter `pos` of the R `text` function.

vertex.labels.offset
: When vertex.labels.pos is specified, this value gives the offset of the label from the specified coordinate in fractions of a character width. It corresponds to the parameter offset of the R text function.

vertex.labels.cex
: Numeric *c*haracter *ex*pansion factor; multiplied by 'par("cex")' yields the final character size. 'NULL' and 'NA' are equivalent to '1.0'. It corresponds to the parameter cex of the R text function.

edge.type
: Edge line type, corresponding to the 'lty' parameter of the R par function. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings '"blank"', '"solid"', '"dashed"', '"dotted"', '"dotdash"', '"longdash"', or '"twodash"', where '"blank"' uses 'invisible lines' (i.e., does not draw them). See ?par for more details. It accepts a vector of values which are recycled.

edge.width
: Edge line width, corresponding to the 'lwd' parameter of the R 'par' function. See ?par for more details. It accepts a vector of values which are recycled.

edge.color
: Color of the edges.

edge.arrow.length
: Length of the edges of the arrow head (in inches) - corresponding to the parameter of the R arrows function with the same name.

edge.arrow.angle
: Angle from the shaft of the arrow to the edge of the arrow head - corresponding to the parameter of the R arrows function with the same name.

com
: The result of a community detection algorithm. When this parameter is set, a colored area is added behind each community.

com.cex
: Increases (>1) or decreases (<1) the margin around the nodes when the colored areas are drawn around the communities.

...
: Other graphical parameters.

See Also

Mining: communities

Examples

```
net <- ml.florentine()
## Not run:
plot(net)
c <- clique.percolation.ml(net)
plot(net,vertex.labels.cex=.5,com=c)

## End(Not run)
net <- ml.aucs()
## Not run:
plot(net,vertex.labels=NA)
title("AUCS network")

## End(Not run)
```

`Transformation` *Functions to merge multiple layers into one: flattening*

Description

These functions merge multiple layers into one. The new layer is added to the network. If the input layers are no longer necessary, they must be explicitely erased.

`flatten.ml` adds a new layer with the actors in the input layers and an edge between A and B if they are connected in any of the merged layers.

Usage

```
flatten.ml(mlnetwork, new.layer="flattening", layers=character(0),
  method="weighted", force.directed=FALSE, all.actors=FALSE)
```

Arguments

`mlnetwork`	A multilayer network.
`new.layer`	Name of the new layer.
`layers`	An array of layers belonging to the network.
`method`	This argument can take values "weighted" or "or". "weighted" adds an attribute to the new edges with the number of layers where the two actors are connected
`force.directed`	The new layer is set as directed. If this is false, the new layer is set as directed if at least one of the merged layers is directed.
`all.actors`	If TRUE, then all the actors are included in the new layer, even if they are not present in any of the merged layers.

Examples

```
net <- ml.aucs()
# A new layer is added to the network, with a flattening of all the other layers
flatten.ml(net, layers=layers.ml(net))
```

Index