Package ‘multinet’

January 5, 2022

Type     Package
Title    Analysis and Mining of Multilayer Social Networks
Version  4.0.1
Date     2022-01-04
Maintainer Matteo Magnani <matteo.magnani@it.uu.se>
Description Functions for the creation/generation and analysis of multilayer social networks <doi:10.18637/jss.v098.i08>.
License  GPL
Depends  igraph (>= 1.2), Rcpp (>= 1.0), methods, RColorBrewer
LinkingTo Rcpp
RcppModules multinet
SystemRequirements A C++14 compiler
NeedsCompilation yes
Repository CRAN
Note     a previous version of the library (main version number: 3) was developed as part of the European Union's Horizon 2020 research and innovation programme under grant agreement No. 727040 (Virt-EU). The package uses functions from eclat <https://borgelt.net/eclat.html>, for association rule mining, Infomap <https://www.mapequation.org>, for the Infomap community detection method (please refer to their web site for up-to-date code), and Howard Hinnant's date and time library <https://github.com/HowardHinnant/date>. The code from these libraries has been included in our source package.

Author   Matteo Magnani [aut, cre],
          Luca Rossi [aut] (API design),
          Davide Vega [aut] (API and code design),
          Obaida Hanteer [ctb] (mdlpa, flat_ec, flat_nw, some community evaluation functions)

Date/Publication 2022-01-05 00:20:02 UTC
**R topics documented:**

- multinet-package .................................................. 2
- multinet.actor_measures ........................................... 3
- multinet.attributes .............................................. 5
- multinet.classes .................................................. 7
- multinet.communities ............................................ 7
- multinet.community.generation .................. 9
- multinet.conversion .......................................... 11
- multinet.distance ............................................ 12
- multinet.edge_directionality .................. 13
- multinet.generation ........................................ 14
- multinet.IO ......................................................... 15
- multinet.layer_comparison ..................................... 18
- multinet.layout ............................................. 20
- multinet.navigation ........................................ 22
- multinet.plotting ........................................ 23
- multinet.predefined .......................................... 26
- multinet.properties ........................................ 27
- multinet.transformation ..................................... 29
- multinet.update ............................................. 30

**Index**

---

**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 vertices (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 \( A \times L \).

In this manual, **multinet.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 (**multinet.predefined**). A synthetic multilayer network can be generated using the growing models described in **multinet.generation**.

Updating and getting information about the basic components of a multilayer network (layers, actors, vertices and edges) can be done using the methods described in **multinet.properties**, **multinet.update** and **multinet.edge_directionality**. **multinet.navigation** shows how to retrieve the neighbors of a node. Attribute values can also be attached to the basic objects in a multilayer network (actors, layers, vertices and edges). Attribute management is described in **multinet.attributes**.
Each individual layer as well as combination of layers obtained using the data pre-processing (flattening) functions described in `multinet.transformation` can be analyzed as a single-layer network using the igraph package, by converting them as shown in `multinet.conversion`. We can also visualize small networks using the method described in `multinet.plotting` and the layouts in `multinet.layout`.

Multilayer network analysis measures are described in `multinet.actor_measures` (for single-actor, degree-based measures), `multinet.distance` (for measures based on geodesic distances) and `multinet.layer_comparison` (to compare different layers).

Communities can be extracted using various clustering algorithms, described in `multinet.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**


---

**multinet.actor_measures**

**Network analysis measures**

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

**Usage**
```r
degree_ml(n, actors = character(0), layers = character(0), mode = "all")
degree_deviation_ml(n, actors = character(0), layers = character(0), mode = "all")
neighborhood_ml(n, actors = character(0), layers = character(0), mode = "all")
neighborhood_ml(n, actors = character(0), layers = character(0), mode = "all")
connective_redundancy_ml(n, actors = character(0), layers = character(0), mode = "all")
relevance_ml(n, actors = character(0), layers = character(0), mode = "all")
xrelevance_ml(n, actors = character(0), layers = character(0), mode = "all")
```
Arguments

n
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


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 U3
neighborhood_ml(net,"U3","work")
# co-workers of U3 who are not connected to U3 on other layers
xneighborhood_ml(net,"U3","work")
# percentage of neighbors of U3 who are also co-workers
relevance_ml(net,"U3","work")
# redundancy between work and lunch
connective_redundancy_ml(net,"U3",c("work","lunch"))
# percentage of neighbors of U3 who would no longer
multinet.attributes

Managing attributes

Description

These functions are used to assign and retrieve values to/from actors, vertices.

Usage

add_attributes_ml(n, attributes, type="string", target="actor", layer="", layer1="", layer2="")

attributes_ml(n, target="actor")

get_values_ml(n, attribute, actors=character(0), vertices=character(0), edges=character(0))

set_values_ml(n, attribute, actors=character(0), vertices=character(0), edges=character(0), values)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>A multilayer network.</td>
</tr>
<tr>
<td>attributes</td>
<td>Name(s) of the attributes to be created.</td>
</tr>
<tr>
<td>target</td>
<td>Can be &quot;actor&quot; (attributes attached to actors), &quot;vertex&quot; (attributes attached to vertices) or &quot;edge&quot; (attributes attached to edges). Layer attributes are not available in this version.</td>
</tr>
<tr>
<td>type</td>
<td>Can be &quot;string&quot; or &quot;numeric&quot;.</td>
</tr>
<tr>
<td>layer</td>
<td>This can be specified only for targets &quot;vertex&quot; (so that the attribute exists only for the vertices in that layer) or &quot;edge&quot; (in which case the attribute applies to intra-layer edges in that layer).</td>
</tr>
<tr>
<td>layer1</td>
<td>This can be specified only for target &quot;edge&quot;, together with layer2, so that the attribute applies to inter-layer edges from layer1 to layer2. If layer1 and layer2 are specified, the parameter layer should not be used.</td>
</tr>
<tr>
<td>layer2</td>
<td>See layer1.</td>
</tr>
<tr>
<td>attribute</td>
<td>The name of the attribute to be updated.</td>
</tr>
<tr>
<td>actors</td>
<td>A vector of actor names. If this is specified, layers, vertices and edges should not.</td>
</tr>
<tr>
<td>vertices</td>
<td>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.</td>
</tr>
<tr>
<td>edges</td>
<td>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.</td>
</tr>
<tr>
<td>values</td>
<td>A vector of values to be set for the corresponding actors, vertices or edges.</td>
</tr>
</tbody>
</table>
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.

See Also

multinet.properties, multinet.edge_directionality

Examples

```r
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)$actor -> 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)
```
multinet.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.

multinet.communities

Community detection algorithms and evaluation functions

Description

Various algorithms to compute communities in multiplex networks, based on flattening (flat_ec, weighted, and flat_wc, unweighted), frequent itemset mining (abacus), adjacent cliques (clique percolation), modularity optimization (generalized louvain), random walks (infomap) and label propagation (mdlp). glouvain2_ml is a more efficient implementation of the original glouvain_ml, no longer based on matrices: it is equivalent to glouvain_ml with gamma set by default to 1.0 (apart from undeterministic behaviour: individual executions are not guaranteed to return the same result). get_community_list_ml is a commodity function translating the result of these algorithms into a list of vertex identifiers, and is internally used by the plotting function.

There are also algorithms to evaluate the resulting communities: generalized modularity (as optimized by glouvain) and normalized mutual information (nmi_ml) and omega index (omega_index_ml) to compare respectively partitioning and general communities. Please consider that both comparison functions use the number of vertices in the network to make a computation, so the absence of actors from some layers would change their result.

Usage

abacus_ml(n, min.actors=3, min.layers=1)
flat_ec_ml(n)
flat_nw_ml(n)
clique_percolation_ml(n, k=3, m=1)
glovain_ml(n, gamma=1, omega=1)
inomap_ml(n, overlapping=FALSE, directed=FALSE, self.links=TRUE)
mdlp_ml(n)

modularity_ml(n, comm.struct, gamma=1, omega=1)
nmi_ml(n, com1, com2)
omega_index_ml(n, com1, com2)
get_community_list_ml(comm.struct, n)
**Arguments**

- **n**: 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 method.
- **omega**: Inter-layer weight parameter in the generalized louvain method.
- **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.
- **com1**: The result of a community detection method.
- **com2**: 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.

The evaluation functions return a number between -1 and 1. For the comparison functions, 1 indicates that the two community structures are equivalent. The maximum possible value of modularity is <= 1 and depends on the network, so modularity results should not be compared across different networks. Also, notice that modularity is only defined for partitioning community structures.

`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())

Oualid Boutemine and Mohamed Bouguessa. Mining Community Structures in Multidimensional Networks. ACM Transactions on Knowledge Discovery from Data, 11(4):1-36, 2017 (for mdp_ml())

See Also

multinet.plotting

Examples

net <- ml_florentine()
abacus_ml(net)
flat_ec_ml(net)
flat_nw_ml(net)
clique_percolation_ml(net)
glovain_ml(net)
infomap_ml(net)
mdlp_ml(net)

# evaluation

c1 <- glouvain_ml(net)
modularity_ml(net, c1)

c2 <- flat_ec_ml(net)
nmi_ml(net, c1, c2)

c3 <- abacus_ml(net)
omega_index_ml(net, c1, c2)
Usage

```
generate_communities_ml(type, num.actors, num.layers, num.communities, overlap=0, 
pr.internal=.4, pr.external=.01)
```

Arguments

- **type**: Type of community structure: pep, peo, sep or seo.
- **num.actors**: The number of actors in the generated network.
- **num.layers**: The number of layers in the generated network.
- **num.communities**: The number of communities in the generated network.
- **overlap**: Number of actors at the end of one community to be also included in the following community.
- **pr.internal**: A vector with the probability of adjacency for two vertices on the same layer and community (either a single value, or one value for each layer).
- **pr.external**: A vector with the probability of adjacency for two vertices on the same layer but different communities (either a single value, or one value for each layer).

Value

`generate_communities_ml` returns a list with two elements: a multilayer network and the community structure used to generate it.

References


See Also

`multinet.generation`, `multinet.IO`

Examples

```
# we generate a network with three layers and 10 communities.
generate_communities_ml("pep", 50, 3, 10)
# the following command also adds some overlapping (1 actor shared between consecutive communities).
generate_communities_ml("pep", 50, 3, 10, 1)
# the following command adds 10 different communities on the last layer.
generate_communities_ml("sep", 50, 3, 20)
# here we add some noise and make communities less dense than the defaults.
generate_communities_ml("pep", 50, 3, 10, pr.internal=.3, pr.external=.05)
```
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

```r
## 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 vertices corresponding to each actor should be merged into a single vertex (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). This option does not currently make any difference, as only actors who are present in at least one layer can exist since version 4.0.
- **...**: Additional arguments. None currently.

Value

An object of class iGraph.

See Also

- `multinet.transformation`

Examples

```r
net <- ml_aucs()
# using the default merge.actors=TRUE we create a multigraph, # where each actor corresponds to a vertex 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 # vertices are used for each actor: one corresponding to the # vertex on facebook, one to the vertex on lunch
f_l_net <- as.igraph(net, c("facebook","lunch"),
                     merge.actors=FALSE)
```
multinet.distance

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(n, from, to=character(0), method="multiplex")

Arguments

- **n**: 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


See Also

- `multinet.actor_measures`
- `multinet.layer_comparison`

Examples

```r
net <- ml_aucs()
distance_ml(net,"U54","U3")
```
multinet.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

```r
set_directed_ml(n, directionalities)
```

```r
is_directed_ml(n, layers1 = character(0), layers2 = character(0))
```

Arguments

- `n`: 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).

See Also

`multinet.properties`, `multinet.attributes`

Examples

```r
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 directionality to a new network
net2 <- ml_empty()
```
add_layers_ml(net2,c("l1","l2"))
set_directed_ml(net2,dir)

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 choosing 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 vertex joining the network.
- **n**: Number of vertices (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

See Also
multinet.predefined, multinet.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 implicitly 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)

Description
These functions are used to store a multilayer network to a file or load it from a file.
There are two network formats accepted: multiplex (default) or multilayer. A full multiplex network
input file has the following format:
-- comment lines start with two dashes (--)#VERSION
3.0#TYPE
multiplex#ACTOR ATTRIBUTES
AttributeName1,STRING
AttributeName2,NUMERIC
-- etc.#NODE ATTRIBUTES
LayerName1,AttributeName1,STRING
LayerName1,AttributeName2,NUMERIC
LayerName2,AttributeName3,STRING
-- etc.
#EDGE ATTRIBUTES
-- edge attributes can be defined for specific layers (called local attributes):
LayerName1,AttributeName,STRING
LayerName1,AttributeName,NUMERIC
-- or for all layers (called global attributes):
AnotherAttributeName,NUMERIC
-- etc.
#LAYERS
LayerName1,UNDIRECTED
LayerName2,DIRECTED
LayerName3,UNDIRECTED,LOOPS
-- etc.
-- LOOPS indicates that edges from one vertex to itself (called loops) are allowed on that layer
#ACTORS
ActorName1,AttributeValueList...
ActorName2,AttributeValueList...
-- etc.
#VERTICES
ActorName1,LayerName1,AttributeValueList...
ActorName1,LayerName2,AttributeValueList...
-- etc.
#EDGES
ActorName1,ActorName2,LayerName1,LocalAttributeValueList,GlobalAttributeValueList...
-- etc.
-- the attribute values must be specified in the same order in which they are defined above
----------------------------------
If the #LAYERS section is empty, all edges are created as undirected.
If the #ACTOR ATTRIBUTES, #VERTEX ATTRIBUTES or #EDGE ATTRIBUTES sections are empty, no attributes are created.
The #LAYERS, #ACTORS and #VERTICES sections are useful only if attributes are present, or if there are actors that are not present in any layer (#ACTORS), or if there are isolated vertices (#VERTICES), otherwise they can be omitted.
If no section is specified, #EDGES is the default.
Therefore, a non attributed, undirected multiplex network file can be as simple as:
----------------------------------
Actor1,Actor2,Layer1
Actor1,Actor3,Layer1
Actor4,Actor2,Layer2
----------------------------------
If interlayer edges exist, then type "multilayer" must be specified, and layers and edges are formatted in a different way:
#VERSION
3.0
#TYPE
multilayer
# ACTOR ATTRIBUTES
AttributeName1, STRING
AttributeName2, NUMERIC
-- etc.

# NODE ATTRIBUTES
LayerName1, AttributeName1, STRING
LayerName1, AttributeName2, NUMERIC
LayerName2, AttributeName3, STRING
-- etc.

# EDGE ATTRIBUTES
-- edge attributes can be defined for specific layers:
LayerName1, AttributeName, STRING
LayerName1, AttributeName, NUMERIC
-- or for all layers (called global attributes):
AnotherAttributeName, NUMERIC
-- etc.

# LAYERS
-- LayerName1, LayerName1, UNDIRECTED
-- LayerName2, LayerName2, DIRECTED
-- LayerName3, LayerName3, DIRECTED, LOOPS
-- LayerName1, LayerName2, DIRECTED
-- etc.
-- all intra-layer specifications (where the first and second layers are the same)
-- should be listed first.
-- LOOPS is only allowed for intra-layer specifications.

# ACTORS
ActorName1, AttributeValueList...
ActorName2, AttributeValueList...
-- etc.

# VERTICES
ActorName1, LayerName1, AttributeValueList...
ActorName1, LayerName2, AttributeValueList...
-- etc.

# EDGES
-- ActorName1, LayerName1, ActorName2, LayerName2, LocalAttributeValueList, GlobalAttributeValueList...
-- etc.

----------------------------------
Usage

read_ml(file, name = "unnamed", sep = ",", aligned = FALSE)
write_ml(n, 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.
multinet.layer_comparison

Description

These functions can be used to compare different layers.

Usage

layer_summary_ml(n, layer, method = "entropy.degree", mode = "all")
layer_comparison_ml(n, layers = character(0), method = "jaccard.edges", mode = "all", K = 0)

Value

read_ml returns a multilayer network. write_ml does not return any value.

See Also

multinet.predefined, multinet.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

n 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.
Arguments

n
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 comparisons. 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


Examples

```r
net <- ml_auc()  

# 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")

multinet.layout

Description

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

\[
\text{layout\_multiforce\_ml}(n, w\_in = 1, w\_inter = 1, gravity = 0, \text{iterations} = 100) \\
\text{layout\_circular\_ml}(n)
\]

Arguments

- **n**: A multilayer network.
- **w\_in**: An array with weights for intralayer forces, or a single number if weights are the same for all layers. When \( w\_in \) is positive, vertices in the corresponding layer will be positioned as if a force was applied to them, repelling vertices that are close to each other and attracting adjacent vertices, all proportional to the specified weight.
- **w\_inter**: An array with weights for interlayer forces, or a single number if weights are the same for all layers. When \( w\_inter \) is positive, vertices in the corresponding layer will be positioned as if a force was applied to them, trying to keep them aligned with the vertices corresponding to the same actors on other layers, proportionally to the specified weight.
- **gravity**: An array with weights for gravity forces, or a single number if weights are the same for all layers. This parameter results in the application of a force to the vertices, directed toward the center of the plot. It can be useful when there are multiple components, so that they do not drift away from each other because of the repulsion force applied to their vertices.
- **iterations**: Number of iterations.

Value

These functions return a data frame with columns: actor, layer, x, y, z. Each value of z corresponds to one layer, and x and y are the coordinates of the actor inside that layer.

References


See Also

multinet.plotting

Examples

```r
net <- ml\_florentine()
layout\_multiforce\_ml(net)
l <- layout\_circular\_ml(net)
## Not run:
plot(net, layout=l)
## End(Not run)
```
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(n, actor, layers = character(0), mode = "all")
xneighbors_ml(n, actor, layers = character(0), mode = "all")

Arguments

n
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


See Also

multinet.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")
multinet.plotting  

```r
xneigh <- xneighbors_ml(net, "U54", c("work","lunch"))
```

### multinet.plotting  

**Drawing a multilayer network**

#### Description

The plot function draws a multilayer network. `values2graphics` is a support function translating discrete attribute values to graphical parameters.

#### Usage

```r
## S3 method for class 'Rcpp_RMLNetwork'
plot(x,
     layout = NULL, grid = NULL, mai = c(.1,.1,.1,.1),
     layers = NULL,
     vertex.shape = 21, vertex.cex = 1, vertex.size = vertex.cex, vertex.color = 1,
     vertex.labels = NULL, vertex.labels.pos = 3,
     vertex.labels.offset = .5, vertex.labels.cex = 1, vertex.labels.col=1,
     edge.type = 1, edge.width = 1, edge.col = 1, edge.alpha=.5,
     edge.arrow.length = 0.1, edge.arrow.angle = 20,
     legend.x = NULL, legend.y = NULL,
     legend.pch = 20, legend.cex = 0.5,
     legend.inset = c(0, 0),
     com = NULL, com.cex = 1,
     show.layer.names=TRUE, layer.names.cex=1, ...)

values2graphics(values, output = "color")
```

#### 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 (left, top, right, bottom). This only concerns vertices: text labels can be printed inside the margin or even outside the frame depending on their offset.

- `layers`  
  A vector of layer names, that determine which layers and in which order are plotted.

- `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.size  synonym of vertex.cex.
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.
vertex.labels.col Color of the labels.
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.col     Color of the edges.
edge.alpha   Transparency 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.
legend.x, legend.y the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by xy.coords.
legend.pch   the plotting symbols appearing in the legend, as numeric vector or a vector of 1-character strings
legend.cex   character expansion factor relative to current par("cex"). Used for text.
legend.inset inset distance(s) from the margins as a fraction of the plot region when legend
is placed by keyword.

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.

show.layer.names
if TRUE adds the name of each layer at the center bottom of it.

layer.names.cex
 Increases (>1) or decreases (<1) the size of the layer names.

... Other graphical parameters.

values A vector with values.

output The type of graphical objects the values should be translated to. It can currently
take values "color" and "shape".

Value

plot returns no value. values2graphics returns an object with fields: legend.text, legend.pch,
legend.col and color or shape, containing respectively the text entries for the legend, their characher
shapes, their colours, and the color or shape of the entities in the values input parameter.

See Also

t multinet.layout, multinet.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)
values2graphics(c("a", "b", "b", "c"))
**Description**

Creates predefined multilayer networks from the literature.

- `ml_empty` returns an empty multilayer network, not containing any actor, layer, node or edge.
- `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**

```r
ml_empty(name="")
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**

multinet.properties

• 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).


See Also

multinet.IO, multinet.generation

Examples

empty <- ml_empty("new network")
aucs <- ml_aucs()
bankwiring <- ml_bankwiring()
florentine <- ml_florentine()
monastery <- ml_monastery()
tailorshop <- ml_tailorshop()

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. The names vertex/vertices are now preferentially used over node/nodes.

Usage

layers_ml(n)
actors_ml(n, layers = character(0), add.attributes = FALSE)
vertices_ml(n, layers = character(0), add.attributes = FALSE)
edges_ml(n, layers1 = character(0), layers2 = character(0), add.attributes = FALSE)
edges_idx_ml(n)

num_layers_ml(n)
num_actors_ml(n, layers = character(0))
num_vertices_ml(n, layers = character(0))
num_edges_ml(n, layers1 = character(0), layers2 = character(0))
Arguments

n
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.

add.attributes
If set to TRUE, one column for each attribute is added to the data frame, with the corresponding attribute value.

Value

layers.ml returns an array of layer names. actors.ml returns a data frame with one column, containing actors 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 objects of the requested type.

See Also

multinet.attributes, multinet.update, multinet.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)
multinet.transformation

Functions to transform existing layers into new ones.

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 explicitly 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.

project_ml adds a new layer with the actors in the first input layer and an edge between A and B if they are connected to the same actor in the second layer.

Usage

flatten_ml(n, new.layer = "flattening", layers = character(0),
          method = "weighted", force.directed = FALSE, all.actors = FALSE)
project_ml(n, new.layer = "projection", layer1, layer2,
          method = "clique")

Arguments

n A multilayer network.
new.layer Name of the new layer.
layers An array of layers belonging to the network.
layer1 Name of a layer belonging to the network.
layer2 Name of a layer belonging to the network.
method This argument can take values "weighted" or "or" for flatten_ml and "clique" for project_ml. "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.
Value

These functions return no value: they modify the input network.

References

ISBN: 978-1107438750

See Also

multinet.conversion

Examples

```r
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))
# Bipartite network
from_actor=c("A","B")
to_actor=c("1","1")
from_layer=c("l1","l1")
to_layer=c("l2","l2")
edges = data.frame(from_actor, from_layer, to_actor, to_layer)
n = ml_empty()
add_edges_ml(n, edges)
project_ml(n, layer1 = "l1", layer2="l2")
```

Description

Functions to add or remove objects in a multilayer network.

The functions add_vertices_ml and delete_vertices_ml add/remove the input actors to/from the input layers. Since version 3.1, the actors in the network correspond to the union of all the actors in the various layers (that is, the vertices).

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.

The functions add_nodes_ml and delete_nodes_ml are deprecated in the current version of the library. The names vertex/vertices are now preferentially used over node/nodes.

Usage

```r
add_layers_ml(n, layers, directed=FALSE)
add_vertices_ml(n, vertices)
add_edges_ml(n, edges)
```
add_igraph_layer_ml(n, g, name)
delete_layers_ml(n, layers)
delete_actors_ml(n, actors)
delete_vertices_ml(n, vertices)
delete_edges_ml(n, edges)

Arguments

n  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 or deleted. The first column specifies actor names, the second layer names.
edges  A dataframe containing the edges to be connected or deleted. The four columns must contain, in this order: actor1 name, layer1 name, actor2 name, layer2 name.

Value

These functions return no value: they modify the input network.

See Also

multinet.properties, multinet.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 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(netvertices)
vertices <- data.frame(
            c("A1","A2"),
            c("l3","l3"))
add_vertices_ml(netvertices)
vertices_ml(net)
# Verifying that the actors have been added correctly
num_actors_ml(net)
actors_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, and 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
Index

* multilayer network social analysis mining
  multinet-package, 2

abacus_ml (multinet.communities), 7
actors_ml (multinet.properties), 27
add_attributes_ml
  (multinet.attributes), 5
add_edges_ml (multinet.update), 30
add_igraph_layer_ml (multinet.update), 30
add_layers_ml (multinet.update), 30
add_nodes_ml (multinet.update), 30
add_vertices_ml (multinet.update), 30
as.igraph.multinet
  (multinet.conversion), 11
as.igraph.Rcpp_RMLNetwork
  (multinet.conversion), 11
attributes_ml (multinet.attributes), 5

clique_percolation_ml
  (multinet.communities), 7
connective_redundancy_ml
  (multinet.actor_measures), 3
degree_deviation_ml
  (multinet.actor_measures), 3
degree_ml (multinet.actor_measures), 3
delete_actors_ml (multinet.update), 30
delete_edges_ml (multinet.update), 30
delete_layers_ml (multinet.update), 30
delete_nodes_ml (multinet.update), 30
delete_vertices_ml (multinet.update), 30
distance_ml (multinet.distance), 12
distance_idx_ml (multinet.properties), 27
distances_ml (multinet.properties), 27
edge_partition_ml (multinet.actor_measures), 3
evolution_er_ml (multinet.generation), 14
evolution_pa_ml (multinet.generation), 14
flatten_ml (multinet.transformations), 29

flat_ec_ml (multinet.communities), 7
flat_nw_ml (multinet.communities), 7
get_community_list_ml
  (multinet.communities), 9
get_values_ml (multinet.attributes), 5
glp_ml (multinet.attributes), 5
grow_ml (multinet.generation), 14
infomap_ml (multinet.communities), 7
is_directed_ml
  (multinet.edge_directionality), 13

layer_comparison_ml
  (multinet.layer_comparison), 18
layer_summary_ml
  (multinet.layer_comparison), 20
layers_ml (multinet.properties), 27
layout_circular_ml (multinet.layout), 20
layout_multiforce_ml (multinet.layout), 20
modularity_ml (multinet.communities), 7
ml_aucs (multinet.predefined), 26
ml_bankwiring (multinet.predefined), 26
ml_empty (multinet.predefined), 26
ml_florentine (multinet.predefined), 26
ml_monastery (multinet.predefined), 26
ml_tailorshop (multinet.predefined), 26
ml_toy (multinet.predefined), 26
multinet (multinet-package), 2
multinet-package, 2
multinet.actor_measures, 2, 3, 12
multinet.attributes, 2, 5, 13, 28
multinet.classes, 7
multinet.communities, 3, 7, 25
multinet.community.generation, 9
multinet.conversion, 3, 11, 30
multinet.distance, 3, 12
multinet.edge_directionality, 2, 6, 13, 28, 31
multinet.generation, 2, 10, 14, 18, 27
multinet.IO, 2, 10, 15, 15, 27
multinet.layer.comparison, 3, 12, 18
multinet.layout, 3, 20, 25
multinet.navigation, 2, 22
multinet.plotting, 3, 9, 21, 23
multinet.predefined, 2, 15, 18, 26
multinet.properties, 2, 6, 13, 22, 27, 31
multinet.transformation, 3, 11, 29
multinet.update, 2, 28, 30

neighborhood_ml
  (multinet.actor.measures), 3
neighbors_ml (multinet.navigation), 22
nmi_ml (multinet.communities), 7
nodes_ml (multinet.properties), 27
num_actors_ml (multinet.properties), 27
num_edges_ml (multinet.properties), 27
num_layers_ml (multinet.properties), 27
num_nodes_ml (multinet.properties), 27
num_vertices_ml (multinet.properties), 27

omega_index_ml (multinet.communities), 7

plot.ml (multinet.plotting), 23
plot.multinet (multinet.plotting), 23
plot.Rcpp.RMLNetwork
  (multinet.plotting), 23
project_ml (multinet.transformation), 29

Rcpp.REvolutionModel-class
  (multinet.classes), 7
read_ml (multinet.IO), 15
relevance_ml (multinet.actor.measures), 3
REvolutionModel (multinet.classes), 7
RMLNetwork-class (multinet.classes), 7

set_directed_ml
  (multinet.edge_directionality), 13

set_values_ml (multinet.attributes), 5
values2graphics (multinet.plotting), 23
vertices_ml (multinet.properties), 27
write_ml (multinet.IO), 15

xneighborhood_ml
  (multinet.actor.measures), 3
xneighbors_ml (multinet.navigation), 22
xrelevance_ml
  (multinet.actor.measures), 3
xy.coords, 24