Analysis of Multiplex Social Networks with R

Matteo Magnani, InfoLab, Uppsala University
Luca Rossi, Data Science & Society Lab, IT University of Copenhagen

In this workshop we introduce **multinet**: an R package to analyze multiplex social networks represented within the more general framework of multilayer networks.

Multiplex networks are characterized by a common set of *actors* connected through multiple types of relations. Each type of relation defines a network between these actors, and each of these networks is represented as a *layer* in the library. For each of these layers we use the standard terminology from graph theory: *vertices* represent actors who are present in the layer and *edges* connect adjacent vertices. Notice that not all actors are forced to be present in all layers; for example, only some actors having a Facebook account will be present as vertices in a layer representing Facebook friendship relations.

In addition to multinet, this document also uses the R libraries **knitr**, , **gplots**, **ggplot2**, **corrplot**, **pander**, **tibble**, **formatR** and **rmarkdown**. Please install them if you want to run the code contained in this document. The libraries **igraph** and **RColorBrewer** are automatically installed (if needed) and loaded by **multinet**. Using RStudio you can directly modify the source code of the document, execute it and also compile your updated document into a pdf file or other formats.

Part 0: getting network data

First we should load the library:

```
library(multinet)
```

Networks can be created in the following ways:

1. we can use `read_ml` to read a network from file,
2. we can create an empty network using `ml_empty` and add objects and attributes to it using `add_XXX_ml` and `add_attributes_ml` (where XXX should be one of `layers`, `vertices` or `edges` – from version 3.1 actors are automatically added when you add a vertex),
3. we can create an empty network using `ml_empty` and add layers to it in the form of igraph objects using `add_igraph_layer_ml`,
4. we can generate a synthetic network using a growing network model with `grow_ml`,
5. or we can load one of the networks already available in the multinet package.

As reading networks form file is the most typical way to load data, in this section we provide some additional details on the input file format. At the end of this document we also demonstrate how to create synthetic networks. Otherwise, from the next section we will use one of the datasets already available in the library to show how to analyze a real multiplex network. For the other ways to create networks please consult the documentation either in RStudio or by writing `?` followed by the name of the function on the command line. Writing ?multinet-package shows a general description of the functionality offered by the library, with links to pages about sub-topics.

When no special information is needed, that is, when the network has no attributes, it has no isolated nodes and all edges are undirected, the input file is as simple as a list of layer-annotated edges:

```
Luca,Matteo,research
Davide,Matteo,research
Luca,Matteo,friendship
```

If needed the library allows us to specify additional details. In particular: we can specify the directionality of intra-layer edges in the #LAYERS section; we can define attributes for actors, vertices and edges; we can

specify attribute values in the #ACTORS, #VERTICES and #EDGES sections; and we can indicate the presence of isolated vertices in the #VERTICES section, as in the following example:

```
#TYPE multiplex

#LAYERS
research, UNDIRECTED
twitter, DIRECTED

#ACTOR ATTRIBUTES
affiliation,STRING

#VERTEX ATTRIBUTES
twitter, num_tweets, NUMERIC

#EDGE ATTRIBUTES
research, num_publications, NUMERIC

#ACTORS
Luca,ITU
Matteo,UU
Davide,UU

#VERTICES
Luca,twitter,53
Matteo,twitter,13

#EDGES
Luca,Matteo,research,9
Luca,Matteo,twitter
```

When we read a multiplex network from file we can also specify that we want all the actors to be present in all the layers, using the *align* parameter. The difference between the two obtained networks can be seen by checking basic network statistics.

```
net <- read_ml("example_io.mpx")
net
```

```
## Multilayer Network [3 actors, 2 layers, 4 vertices, 2 edges (2,0)]
```

```
aligned_net <- read_ml("example_io.mpx", align=TRUE)
aligned_net
```

```
## Multilayer Network [3 actors, 2 layers, 6 vertices, 2 edges (2,0)]
```

Notice that in the second case the network has six vertices, that is, all actors are present in all the layers.

From now on we will use one of the networks already available in the package.

Part I: network exploration

We can start our analysis by loading the library and retrieving the AUCS dataset, here stored in a variable we call *net*. The AUCS network has been often used in the literature to test new methods thanks to its small size, the presence of attributes, and its easy semantics. The data, described by Rossi and Magnani (2015), were collected at a university research department and include five types of online and offline relations. The population consists of 61 employees (called *actors* in the multinet library), incuding professors, postdocs, PhD students and administrative staff.

```
library(multinet)
net <- ml_aucs()
```

Typing the variable name we get a short description of the network.

```
net
```

```
## Multilayer Network [61 actors, 5 layers, 224 vertices, 620 edges (620,0)]
```

We can also list its attributes

```
attributes_ml(net, target = "actor")
```

```
##    name   type
## 1 group string
## 2  role string
```

and individual objects, for example layers and actors.

```
num_layers_ml(net)
```

```
## [1] 5
```

```
layers_ml(net)
```

```
## [1] "facebook" "leisure"  "work"     "coauthor" "lunch"
```

```
num_actors_ml(net)
```

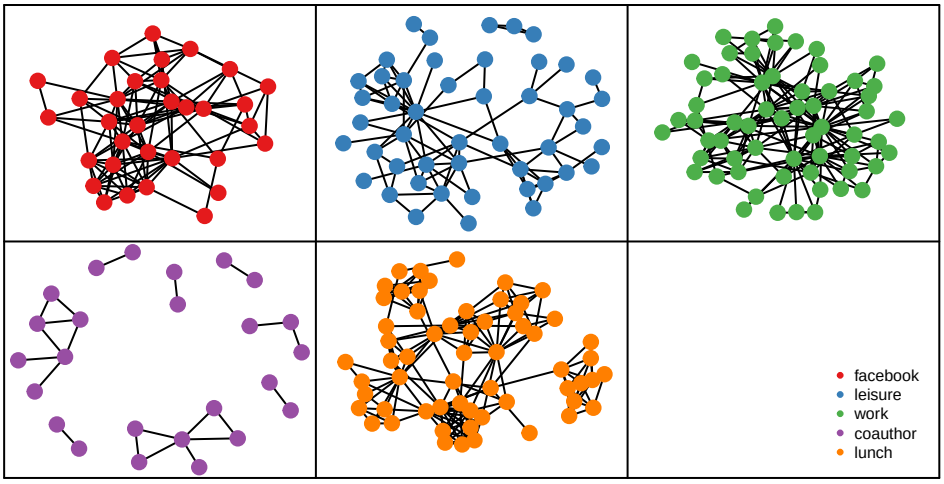
```
## [1] 61
```

```
actors_ml(net)
```

```
##  [1] "U79"  "U65"  "U23"  "U102" "U106" "U33"  "U37"  "U68"  "U69"  "U17"
## [11] "U3"   "U29"  "U42"  "U41"  "U19"  "U97"  "U99"  "U67"  "U22"  "U59"
## [21] "U71"  "U4"   "U126" "U130" "U140" "U123" "U6"   "U134" "U138" "U86"
## [31] "U73"  "U76"  "U14"  "U47"  "U18"  "U48"  "U62"  "U63"  "U107" "U10"
## [41] "U92"  "U72"  "U90"  "U91"  "U118" "U109" "U110" "U112" "U26"  "U13"
## [51] "U21"  "U142" "U113" "U54"  "U124" "U139" "U32"  "U49"  "U53"  "U1"
## [61] "U141"
```

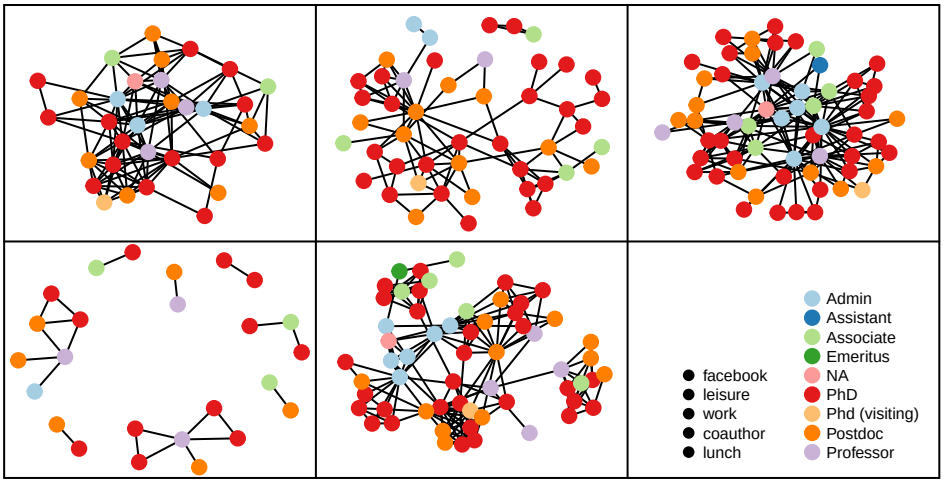
To get a visual overview of the network and get a first idea of its structure we can plot it. We can produce a default visualization just by executing `plot(net)`, but to make the plot more readable we will add a few details. In particular: (1) we explicitely compute a layout that draws each layer independently of the others, as declared by setting interlayer weights (w_inter) to 0, (2) we plot the layers on two rows, (3) we remove the labels of the vertices, to increase readability, and (4) we add a legend with the names of the layers.

```
l <- layout_multiforce_ml(net, w_inter = 0, gravity = 1)

plot(net,
     vertex.labels = "",
     grid = c(2,3),
     layout = l,
     legend.x="bottomright", legend.inset = c(.05,.05)
     )
```

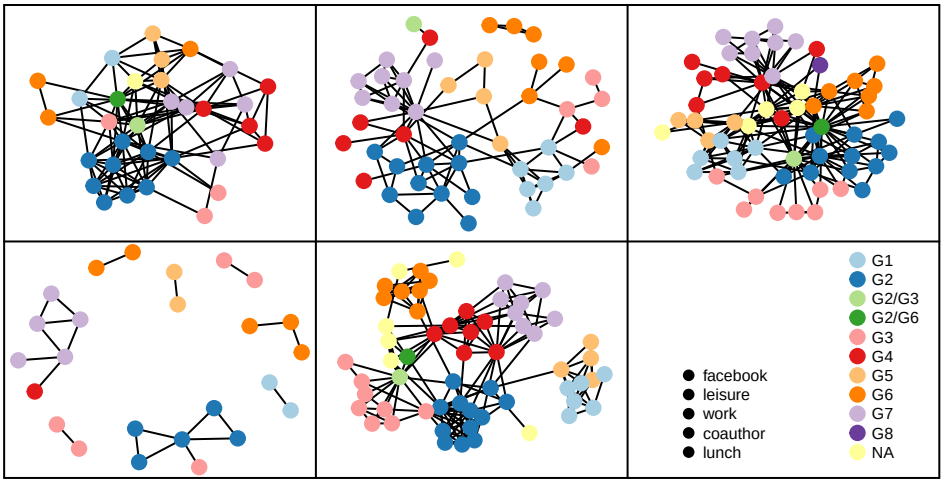


We can also use the attributes to inspect the relationship between the role and group of the actors and the topology of the network.

```
role_attributes <- get_values_ml(net,  actors = vertices_ml(net)[[1]],  attribute = "role")
gr <- values2graphics(role_attributes[[1]])

plot(net,
    layout = l,
    grid = c(2,3),
    vertex.labels = "",
    vertex.color = gr$color
)

legend("bottomright",
  legend = gr$legend.text,
  col = gr$legend.col,
  pt.bg = gr$legend.col,
  pch = gr$legend.pch,
  bty = "n", pt.cex = 1, cex = .5,
  inset = c(0.05, 0.05)
)

legend("bottomright",
  legend = layers_ml(net),
   bty = "n", pch=20, pt.cex = 1, cex = .5,
   inset = c(0.2, 0.05)
)
```




```
group_attributes <- get_values_ml(net,  actors = vertices_ml(net)[[1]],  attribute = "group")
gr <- values2graphics(group_attributes[[1]])

plot(net,
    layout = l,
    grid = c(2,3),
    vertex.labels = "",
    vertex.color = gr$color
)

legend("bottomright",
       legend = gr$legend.text,
       col = gr$legend.col,
       pt.bg = gr$legend.col,
       pch = gr$legend.pch,
       bty = "n", pt.cex = 1, cex = .5,
       inset = c(0.05, 0.05)
       )

legend("bottomright",
       legend = layers_ml(net),
       bty = "n", pch=20, pt.cex = 1, cex = .5,
       inset = c(0.2, 0.05)
       )
```



Part II: layer comparison

After getting an idea of the general structure of the network, we can start computing some quantitative summaries, starting from the macro level: the structure of the layers and the relationships between layers.

Layer-by-layer statistics

A first quantitative comparison of the layers can be done computing basic network measures for each layer. The `summary` function computes a selection of measures on all the layers, and also on the flattened network.

Table 1: Basic layer statistics. n: order (number of vertices), m: size (number of edges), dir: edge directionality, nc: number of connected components, dens: density, cc: clustering coefficient, apl: average path length, dia: diameter. *flat* is the combination of all the layers

	n	m	dir	nc	slc	dens	cc	apl	dia
flat	61	620	0	1	61	0.34	0.48	2.06	4
coauthor	25	21	0	8	6	0.07	0.43	1.50	3
facebook	32	124	0	1	32	0.25	0.48	1.96	4
leisure	47	88	0	2	44	0.08	0.34	3.12	8
lunch	60	193	0	1	60	0.11	0.57	3.19	7
work	60	194	0	1	60	0.11	0.34	2.39	4

To compute other functions or perform another type of layer-by-layer analysis we can convert the layers into igraph objects, using the `as.igraph` function, for a single (group of) layer(s), or the `as.list` function to obtain a list with all the layers and the flattened network as igraph objects. Once the igraph objects have been generated, all the network measures available in igraph can be computed.

```
as.igraph(net, layers = c("facebook", "leisure"))
```

```
## IGRAPH 30cf9d5 UN-- 52 212 --
## + attr: leisure (v/c), facebook (v/c), group (v/c), role (v/c),
## | name (v/c), id (v/c), e_type (e/c), id (e/c)
## + edges from 30cf9d5 (vertex names):
##  [1] U6  --U90  U79 --U99  U90 --U91  U106--U118 U10 --U1   U106--U41
##  [7] U17 --U107 U126--U109 U107--U32  U107--U91  U109--U54  U29 --U126
## [13] U126--U124 U69 --U126 U126--U90  U76 --U54  U68 --U142 U19 --U14
```

```
## [19] U42 --U142 U138--U72  U18 --U124 U18 --U62  U126--U54  U138--U91
## [25] U68 --U141 U37 --U142 U17 --U14  U14 --U1   U79 --U73  U79 --U76
## [31] U76 --U90  U79 --U90  U126--U21  U23 --U14  U99 --U124 U91 --U124
## [37] U73 --U14  U23 --U17  U17 --U91  U17 --U73  U126--U91  U76 --U18
## + ... omitted several edges
```

```
layers <- as.list(net)
names(layers)
```

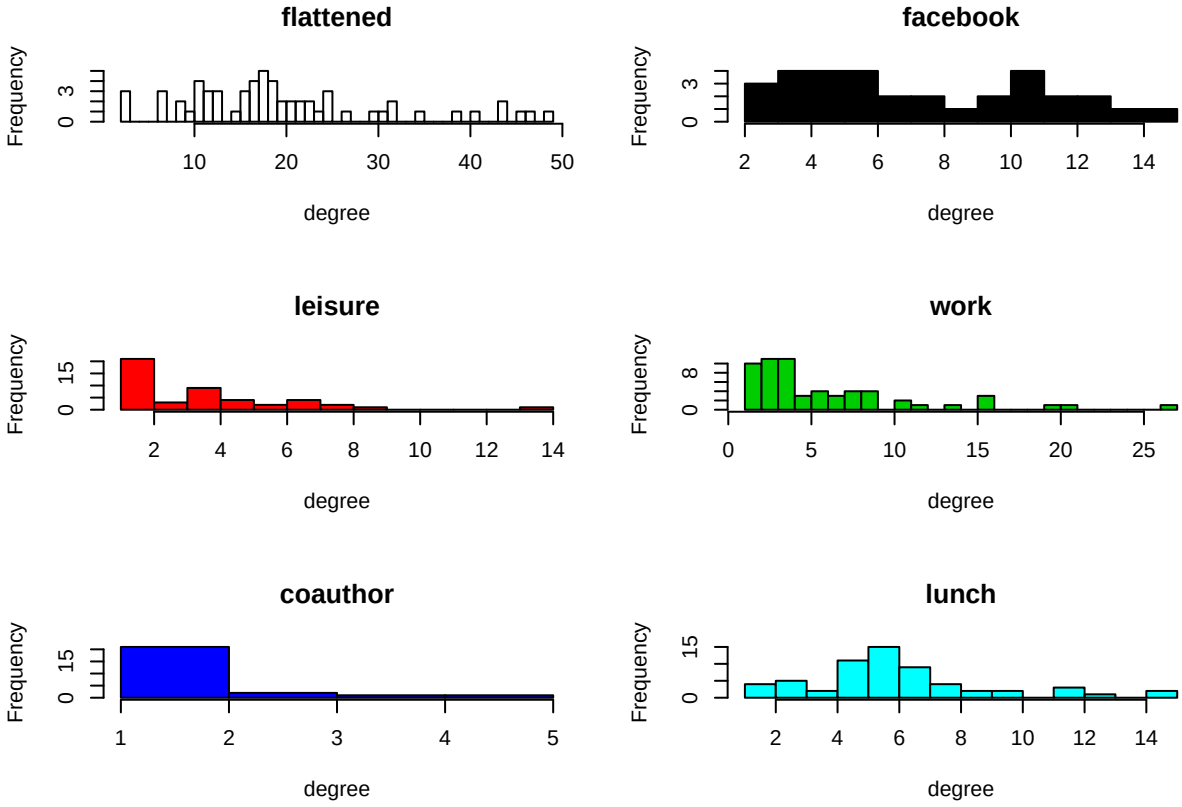
```
## [1] "_flat_"   "coauthor" "facebook" "leisure"  "lunch"    "work"
```

```
transitivity(as.list(net)[[1]])
```

```
## [1] 0.4761508
```

Degree distributions

The degree distribution often reveals interesting dynamics, similarities and dissimilarities between the layers and thus between the relations that are represented.



To quantify the difference between these distributions we can use the `layer_comparison_ml` function that returns a table with pair-wise comparisons:

```
layer_comparison_ml(net, method = "jeffrey.degree")
```

Table 2: Dissimilarity between degree distributions, computed using the Jeffrey dissimilarity function

	facebook	leisure	work	coauthor	lunch
facebook	0.00	1.02	0.71	2.02	0.42

	facebook	leisure	work	coauthor	lunch
leisure	1.02	0.00	0.21	0.45	1.33
work	0.71	0.21	0.00	0.59	0.84
coauthor	2.02	0.45	0.59	0.00	2.90
lunch	0.42	1.33	0.84	2.90	0.00

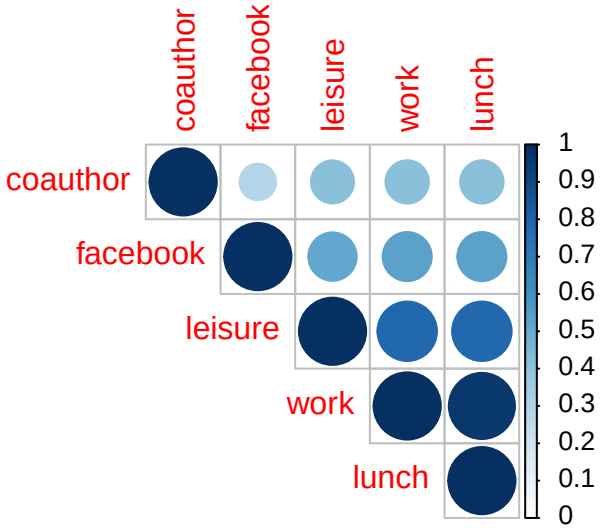
Actor-based layer comparison

The `layer_comparison_ml` function can also be used to compute multiplex-specific comparisons considering the fact that the same actors may be present on the different layers. In fact, a first important comparison can be used to check to what extent this is true:

```
comp <- layer_comparison_ml(net, method = "jaccard.actors")
```

Table 3: Overlapping between actors in the two layers. 0: no common actors. 1: the same actors are present in both layers

	facebook	leisure	work	coauthor	lunch
facebook	1.00	0.52	0.53	0.30	0.53
leisure	0.52	1.00	0.78	0.41	0.78
work	0.53	0.78	1.00	0.42	0.97
coauthor	0.30	0.41	0.42	1.00	0.42
lunch	0.53	0.78	0.97	0.42	1.00

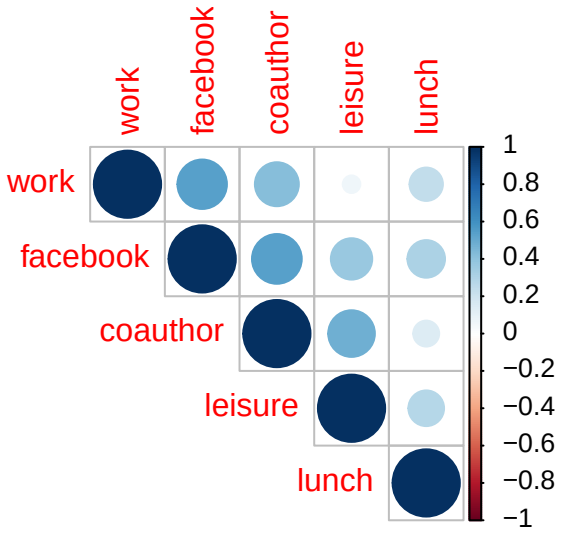


If there is a strong overlapping between the actors, then we can ask whether actors having a high (or low) degree on one layer behave similarly in other layers. To do this we can compute the correlation between the degrees:

```
comp <- layer_comparison_ml(net, method = "pearson.degree")
```

Table 4: Linear correlation between the degree of actors in the two layers, from -1 (top actors in one layer are not active in the other and vice versa) to 1 (top actors in one layer are top actors in the other and vice versa)

	facebook	leisure	work	coauthor	lunch
facebook	1.00	0.38	0.54	0.55	0.31
leisure	0.38	1.00	0.07	0.48	0.28
work	0.54	0.07	1.00	0.43	0.25
coauthor	0.55	0.48	0.43	1.00	0.15
lunch	0.31	0.28	0.25	0.15	1.00

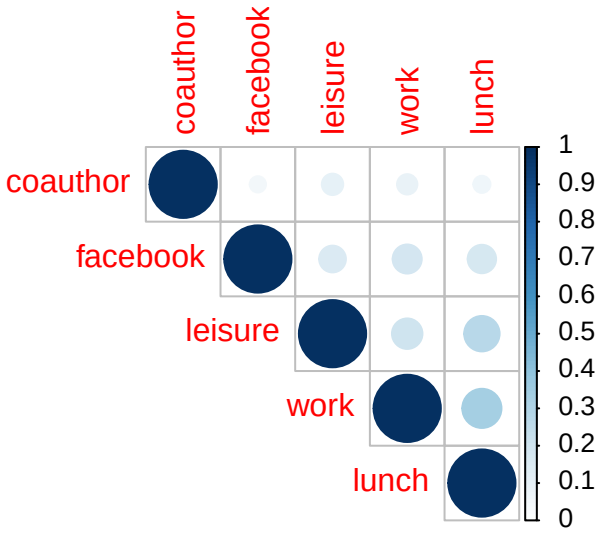


Correlation only depends on the number of incident edges for each pair (actor, layer). We can also check to what extent actors are adjacent to the same other actors in different layers:

```
comp <- layer_comparison_ml(net, method = "jaccard.edges")
```

Table 5: Overlapping between edges in the two layers. 0: no actors adjacent in one layer are also adjacent in the other. 1: all pairs of actors are either adjacent in both layers or in none

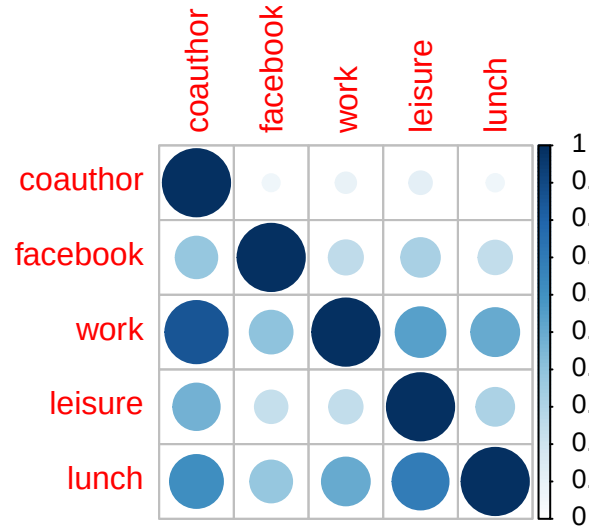
	facebook	leisure	work	coauthor	lunch
facebook	1.00	0.16	0.19	0.06	0.18
leisure	0.16	1.00	0.21	0.10	0.28
work	0.19	0.21	1.00	0.09	0.34
coauthor	0.06	0.10	0.09	1.00	0.06
lunch	0.18	0.28	0.34	0.06	1.00




```
comp <- layer_comparison_ml(net, method = "coverage.edges")
```

Table 6: Directional overlapping (coverage) between edges in the two layers.

	facebook	leisure	work	coauthor	lunch
facebook	1.00	0.33	0.26	0.38	0.25
leisure	0.23	1.00	0.25	0.48	0.32
work	0.40	0.55	1.00	0.86	0.51
coauthor	0.06	0.11	0.09	1.00	0.07
lunch	0.39	0.69	0.51	0.62	1.00



The package provides additional similarity functions, listed in the following table.

Overlapping	Distribution dissimilarity	Correlation
jaccard.actors	dissimilarity.degree	pearson.degree
jaccard.edges	KL.degree	rho.degree
jaccard.triangles	jeffrey.degree	
coverage.actors		

Overlapping	Distribution dissimilarity	Correlation
coverage.edges		
coverage.triangles		
sm.actors		
sm.edges		
sm.triangles		
rr.actors		
rr.edges		
rr.triangles		
kulczynski2.actors		
kulczynski2.edges		
kulczynski2.triangles		
hamann.actors		
hamann.edges		
hamann.triangles		

Part III: actor-level analysis

Degree and degree deviation

The following is the list of highest-degree actors on the whole multiplex network:

```
deg <- degree_ml(net)
top_degrees <- head(deg[order(-deg)])
top_actors <- head(actors_ml(net)[order(-deg)])
top_actors
```

```
## [1] "U4"   "U67"  "U91"  "U79"  "U123" "U110"
```

However, in a multiplex context degree becomes a layer-specific measure. We can no longer just ask "who is the most central actor" but we should ask "who is the most central actor on this layer?" Let us see how the most central actors look like when we "unpack" their centrality on the different layers:

Table 8: Degree for the top actors on each layer

actors	facebook	leisure	lunch	coauthor	work	flat
U4	12	1	15	NA	21	49
U67	13	2	12	NA	20	47
U91	14	14	7	3	8	46
U79	15	7	13	NA	9	44
U123	11	NA	6	NA	27	44
U110	9	7	7	4	14	41

If we want to quantify to what extent actors have similar or different degrees on the different (combinations of) layers, we can compute the standard deviation of the degree:

```
degree_deviation_ml(net, actors = top_actors)
```

```
## [1] 8.133880 7.418895 4.261455 5.230679 9.987993 3.310589
```

Neighborhood and exclusive neighborhood

The layer structure, the concept of actor and the concept of node allow us to define *neighborhood* and *exclusive neighborhood*. The neighbors of an actor a are those distinct actors that are connected to a on a specific layer or on a set of layers. While on a single layer degree and neighborhood size have the same value, they can be different when layers and nodes are taken into account.

Starting from the idea of *neighborhood*, *exclusive neighborhood* counts the neighbours that are connected to a specific actor only on that layer. That layer is thus important to preserve the full connectivity of the actor. In the following example U4 has 5 exclusive neighbors on the Facebook layer.

It is now possible to visualize the top actors according to their neighborhood size (= degree) for each layer, for example work:

```
m = neighborhood_ml(net, layers = "work")
```

Table 9: Top-neighborhood actors, work layer

actors	neighborhood
U123	27
U4	21
U67	20
U71	16
U130	16
U26	16

and leisure:

```
m = neighborhood_ml(net, layers = "leisure")
```

Table 10: Top-neighborhood actors, leisure layer

actors	neighborhood
U91	14
U126	9
U90	8
U54	8
U79	7
U73	7

It is also possible to visualize the exclusive neighborhood for every single actor for a layer:

```
m = xneighborhood_ml(net, layers = "work")
```

Table 11: Top-xneighborhood actors, work layer

actors	neighborhood
U123	15
U139	11
U26	10
U71	7
U97	6
U130	6

or a combination of layers:

```
m = xneighborhood_ml(net, layers = c("work","facebook"))
```

Table 12: Top-xneighborhood actors, work & facebook layer

actors	neighborhood
U123	23
U71	12
U124	12
U4	11
U130	11
U139	11

Relevance

Once we have introduced the concept of *neighborhood*, we can easily introduce the idea of *relevance*. *Relevance* computes the ratio between the neighbors of an actor connected by edges belonging to a specific layer (or set of) and the total number of her neighbors. Every actor could be described as having a specific "signature" represented by her presence on the different layers.

Table 13: Relevance of each layer for U4

facebook	leisure	lunch	coauthor	work
0.3793103	NA	0.2068966	NA	0.9310345

Similarly to *neighborhood* also *relevance* can be extended into an exclusive version where the ratio is defined using the *exclusive neighbors*. While relevance is an effective way to observe how much of an actor's neighborhood exists on each layer it does not allow to observe all the complexity that can be represented with a multilayer structure such as knowing how much the general connectivity of an actor would be affected by removing a specific layer. This can be answered looking at the *exclusive relevance*:

Table 14: Exclusive Relevance of each layer for U4

facebook	leisure	lunch	coauthor	work
0.0689655	NA	0	NA	0.5172414

Distances

In addition to single-actor measures, the package can also be used to compute multilayer distances between pairs of actors. Distances are defined as sets of lengths of Pareto-optimal multidimensional paths. As an example, if two actors are adjacent on two layers, both edges would qualify as Pareto-optimal paths from one actor to the other, as edges on different layers are considered incomparable (that is, it is assumed that it makes no sense in general to claim that two adjacent vertices on Facebook are closer or further than two adjacent vertices on the co-author layer). Pareto-optimal paths can also span multiple layers.

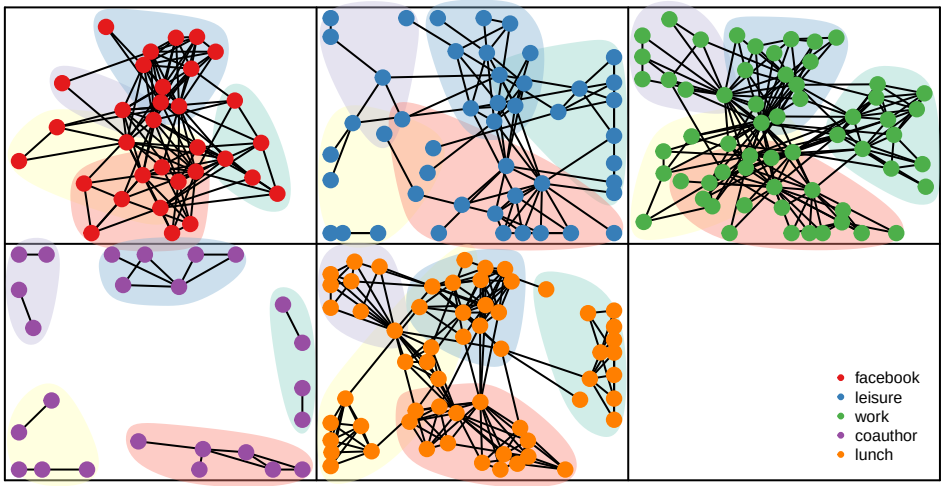
from	to	facebook	leisure	work	coauthor	lunch
U91	U4	1	0	0	0	0
U91	U4	0	1	1	0	0
U91	U4	0	0	0	2	1

from	to	facebook	leisure	work	coauthor	lunch
U91	U4	0	0	1	0	1
U91	U4	0	3	0	0	0
U91	U4	0	1	0	0	1
U91	U4	0	0	1	2	0
U91	U4	0	0	2	0	0
U91	U4	0	0	0	0	2
U91	U4	0	2	0	1	0

Part IV: community detection

Network analysis is commonly used to find communities. A community could be defined as a subgroup of users who are more densely connected among themselves than with the rest of the network. This intuition is formalized through an approach to community detection known as modularity optimization. Adding a multilayer perspective means that interlayer modularity also needs to be taken into account.

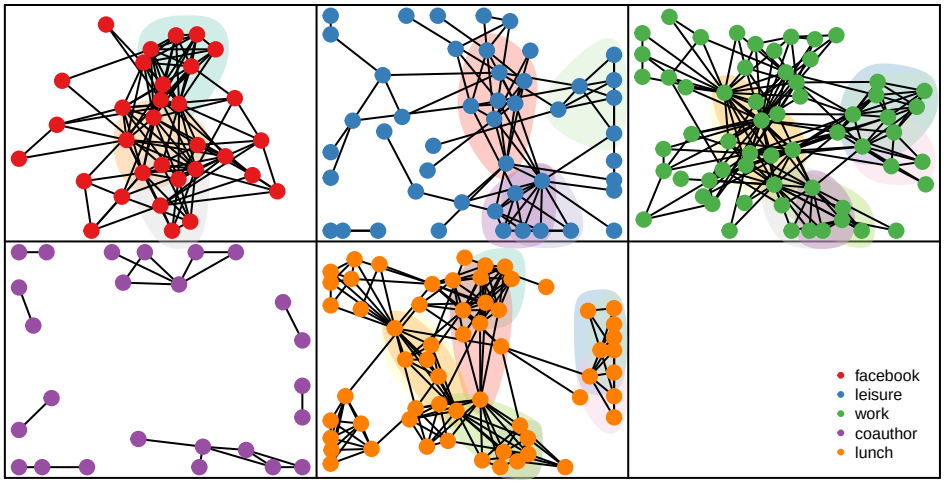
```
ml_clust <- glouvain_ml(net)

l <- layout_multiforce_ml(net)

plot(net,
     com = ml_clust,
     vertex.labels = "",
     layout=l, grid = c(2,3),
     legend.x="bottomright",
     legend.inset = c(.05, .05)
     )
```



The following are communties on the whole multilayer network spanning at least two layers, identified using the clique percolation algorithm. The main practical difference between generalized Louvain and clique percolation is that the latter does not necessarily include all vertices in a community, and the same vertex can be included in more than one community. In addition, using clique percolation we can express the minimum connectivity requirements to identify a community (k = minimum clique size, m = minimum number of layers where the clique is present).

```
ml_clust <- clique_percolation_ml(net, k=4, m=2)
```

```
plot(net,
     com = ml_clust,
     vertex.labels = "",
     layout=l,
     grid = c(2,3),
     legend.x="bottomright",
     legend.inset = c(.05, .05)
     )
```



The following is part of the result of the second algorithm, set to find communities spanning at least two layers:

	actor	layer	cid
9	U109	facebook	0
10	U109	lunch	0
7	U18	facebook	0
8	U18	lunch	0
3	U3	facebook	0
4	U3	lunch	0
11	U54	facebook	0
12	U54	lunch	0
5	U76	facebook	0
6	U76	lunch	0
1	U79	facebook	0
2	U79	lunch	0
22	U123	lunch	1
21	U123	work	1
14	U33	lunch	1
13	U33	work	1
20	U4	lunch	1
19	U4	work	1
24	U63	lunch	1
23	U63	work	1

Part V: Network growing

A growing area of interest is constituted by generative models of multiplex networks. A simple way to approach the problem is to imagine layers that can evolve based on internal or external dynamics. Internal dynamics will be modelled after existing network models (considering a layer as a single layer network) while external dynamics will be represented by importing on layer a an edge already existing on layer b. Within this perspective the intuition is that relations existing on a layer might naturally expand over time into other layers (e.g. co-workers starting to add each other on Facebook because of their offline relationship).

Multinet allows to generate a network with n layers internally growing according to the Preferential Attachment model or growing by selecting new edges uniformly at random. Currently multinet also allows different growing rates for different layers. All the probability vectors must have the same number of fields, one for each layer: two in this example. By defining the parameters *pr.internal* and *pr.external*, we are also implicitely defining *pr.no.action* (1 minus the other probabilities, for each layer).

In this first example we create a multilayer network with two layers both based on Preferential Attachment. A layer will only evolve according to its internal model, while the other will have a probability of .8 of evolving according to the external dynamic (importing an edge from the other layer).

```
models_mix <- c(evolution_pa_ml(3,1), evolution_er_ml(100))
pr.external <- c(0,.5)
pr.internal <- c(1,.5)
dependency <- matrix(c(1,1,0,0),2,2)
ml_generated_mix <- grow_ml(100, 100, models_mix, pr.internal, pr.external, dependency)
```

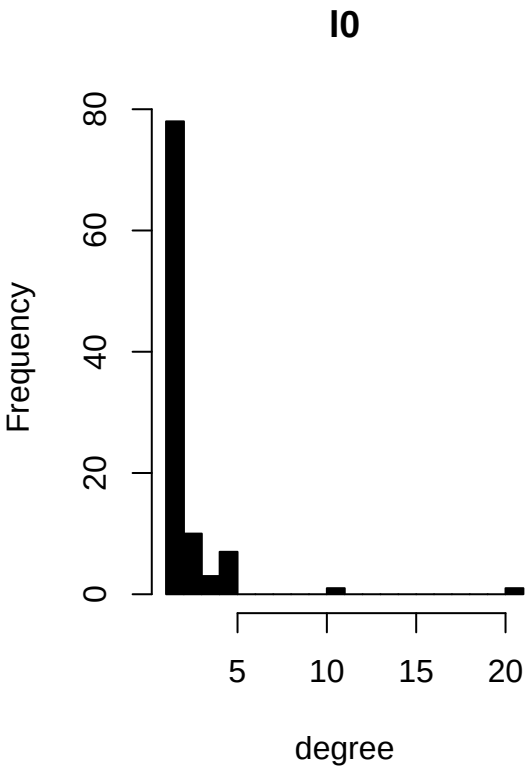
Now we can see the actors with the highest degree on each layer, highlighting how some of the hubs in the Preferential Attachment layer have also high connectivity in the other layer.

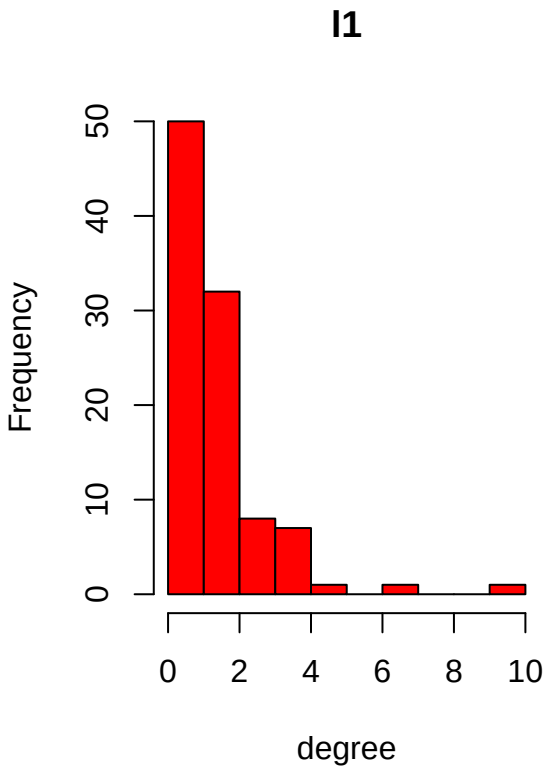
```
deg0 <- degree_ml(ml_generated_mix, layers="l0")
deg1 <- degree_ml(ml_generated_mix, layers="l1")

top_actors_l0 <- head(actors_ml(ml_generated_mix)[order(-deg0)])
top_actors_l1 <- head(actors_ml(ml_generated_mix)[order(-deg1)])

top_actors_l0
```

```
## [1] "a47" "a56" "a7"  "a5"  "a4"  "a26"
```

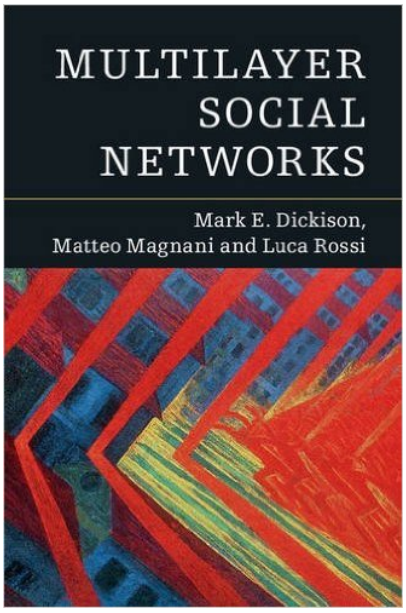
```
top_actors_l1
```

```
## [1] "a47" "a56" "a5"  "a4"  "a26" "a31"
```





References



Dickison M., Magnani M., Rossi L., "Multilayer Social Netwrok Analysis", Cambridge University Press, 2016
http://multilayer.it.uu.se/

Acknowledgments

This material was partially supported by the European Community through the project "Values and ethics in Innovation for Responsible Technology in Europe" (Virt-EU) funded under Horizon 2020 ICT-35-RIA call Enabling Responsible ICT-related Research and Innovation.