MAMMULT: Metrics And Models for MULTilayer networks

19th October 2015

Contents

Chapter 1

Structural descriptors

1.1 Basic node, edge, and layer properties

1.1.1 Node and layer activity

This section includes programs related to the computation of node and layer activity, activity vectors, pairwise multiplexity, pairwise normalised Hamming distance, node degree vectors.

`node_activity.py`

NAME

node_activity.py - compute the activity of the nodes of a multiplex, i.e. the number of layers where each node is not isolated.

SYNOPSYS

node_activity.py *<layer1> [<layer2> ...]*

DESCRIPTION

Compute and print on output the activity of the nodes of a multiplex network, whose layers are given as input in the files *layer1*, *layer2*, etc.

Each file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

A list of lines, where the n-th line is the value of activity of the n-th node, starting from **0**.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`layer_activity.py`

NAME

layer_activity.py - compute the activity of the layers of a multiplex, i.e. the number of active nodes on each layer.

SYNOPSYS

layer_activity.py <*layer1*> [<*layer2*> ...]

DESCRIPTION

Compute and print on output the activity of the layers of a multiplex network, where the layers are given as input in the files *layer1*, *layer2*, etc.

Each file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

A listof lines, where the n-th line is the value of activity of the n-th layer, starting from **0**.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`node_activity_vectors.py`

NAME

node_activity_vectors.py - compute the activity vectors of all the nodes of a multiplex.

SYNOPSYS

node_activity_vectors.py <*layer1*> [<*layer2*> ...]

DESCRIPTION

Compute and print on output the activity vectors of the nodes of a multiplex network, whose layers are given as input in the files *layer1*, *layer2*, etc.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines, where the n-th line contains the activity vector of the n-th node, i.e. a bit-string where each bit is set to "1" if the node is active on the corresponding layer, and to "0" otherwise. As usual, node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers, and will print on output a bit-string of zeros.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`layer_activity_vectors.py`

NAME

layer_activity_vectors.py - compute the activity vectors of all the layers of a multiplex.

SYNOPSYS

layer_activity_vectors.py *<layer1> [<layer2> ...]*

DESCRIPTION

Compute and print on output the activity vectors of the layers of a multiplex network, where the layers are given as input in the files *layer1*, *layer2*, etc.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines, where the n-th line contains the activity vector of the n-th layer, i.e. a bit-string where each bit is set

to "1" if the corresponding node is active on the n-th layer, and to "0" otherwise.
As usual, node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`multiplexity.py`

NAME

multiplexity.py - compute the pairwise multiplexity between all the pairs of layers of a multiplex.

SYNOPSYS

multiplexity.py *<`layer1`> <`layer2`> [<`layer3`>...]*

DESCRIPTION

Compute and print on output the pairwise multiplexity $Q_{\alpha,\beta}$ (i.e., the fraction of nodes active on both layers) between all pairs of layers. The layers are given as input in the files *layer1*, *layer2*, etc.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines, in the format:

layer1 layer2 mult

where *layer1* and *layer2* are the IDs of the layers, and *mult* is the value of the multiplexity $Q_{layer1,layer2}$. Layers IDs start from zero, are are associated to the layers in the same order in which the layer files are provided on the command line.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`hamming_dist.py`

NAME

hamming_dist.py - compute the normalised Hamming distance between all the pairs of layers of a multiplex.

SYNOPSYS

hamming_dist.py *<layer1> <layer2> [<layer3>...]*

DESCRIPTION

Compute and print on output the normalised Hamming distance $H_{\alpha,\beta}$ (i.e., the fraction of nodes which are active on either of the layers, but not on both) between all pairs of layers. The layers are given as input in the files *layer1*, *layer2*, etc.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines, in the format:

layer1 layer2 hamm

where *layer1* and *layer2* are the IDs of the layers, and *hamm* is the value of the normalised Haming distance $H_{layer1,layer2}$. Layers IDs start from zero, are are associated to the layers in the same order in which the layer files are provided on the command line.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`node_degree_vectors.py`

NAME

node_degree_vectors.py - compute the degree vectors of all the nodes of a multiplex network

SYNOPSYS

node_degree_vectors.py *<layer1> [<layer2> ...]*

DESCRIPTION

Compute and print on output the degree vectors of all the nodes of a multiplex network, whose layers are given as input in the files *layer1*, *layer2*, etc.

Each file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

A list of lines, where the n-th line is the vector of degrees of the n-th node, in the format:

noden_deg_lay1 noden_deg_lay2 ... noden_deg_layM

As usual, node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

`degs_to_binary.py`

NAME

degs_to_binary.py - compute the activity vectors of all the nodes of a multiplex.

SYNOPSYS

degs_to_binary.py <*degree_vectors*>

DESCRIPTION

Take a file which contains, on the n-th line, the degrees at each layer of the n-th node, (e.g., the result of the script `node_degree_vectors.py`), in the format:

noden_deg_lay1 noden_deg_lay2 ... noden_deg_layM

and compute the corresponding node activity bit-strings, where a "1" signals the presence of the node on that layer, while a zero indicates its absence.

OUTPUT

The program returns on `stdout` a list of lines, where the n-th line is the activity bit-string of the n-th node. Additionally, the program prints on `stderr` the distribution of all activity bit-strings, in the format:

Bn Bit-string count

Where B is the number of ones in the activity bit-string (i.e., the node-activity associated to that activity bit-string), *Bit-string* is the activity bit-string and *count* is the number of times that particular activity bit-string appears in the multiplex.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`degs_to_activity_overlap.py`

NAME

degs_to_activity_overlap.py - compute the activity and the total (overlapping) degree of all the nodes of a multiplex.

SYNOPSYS

degs_to_activity_overlap.py <*`degree_vectors`*>

DESCRIPTION

Take a file which contains, on the n-th line, the degrees at each layer of the n-th node, (e.g., the result of the script `node_degree_vectors.py`), in the format:

noden_deg_lay1 noden_deg_lay2 ... noden_deg_layM

and compute the activity (i.e., the number of layers in which a node is not isolated) and the total (overlapping) degree of each node.

OUTPUT

The program prints on `stdout` a list of lines, where the n-th line contains the activity and the total degree of the n-th nodem in the format:

noden_activity noden_tot_deg

As usual, the program assumes that node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

1.1.2 Layer aggregation

This section includes programs to obtain various single-layer aggregated graphs associated to a multiplex network.

`aggregate_layers_w.py`

NAME
aggregate_layers_w.py - compute the (weighted) aggregated graph associated to a multiplex.

SYNOPSYS
aggregate_layers_w.py `<layer1> <layer2> [<layer3>...]`

DESCRIPTION
Compute and print on output the edge list of the weighted aggregated graph associated to the multiplex network given on input. An edge is present in the aggregated graph if it exists in at least one of the M layers of the multiplex.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT
The program prints on `stdout` the edge list of the aggregated graph associated to the multiplex network. The edge list is a list of lines in the format:

ID1 ID2 weight

where *ID1* and *ID2* are the IDs of the two nodes and *weight* is the number of layers in which an edge between *ID1* and *ID2* exists.

REFERENCE
V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).
Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`intersect_layers.py`

NAME
intersect_layers.py - compute the intersection graph associated to a multiplex.

SYNOPSYS

intersect_layers.py *<layer1> <layer2> [<layer3>...]*

DESCRIPTION

Compute and print on output the edge list of the intersection graph associated to the multiplex network given on input, where an edge exists only if it is present on **all** the layers of the multiplex.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` the edge list of the intersection graph associated to the multiplex network. The edge list is a list of lines in the format:

ID1 ID2

where *ID1* and *ID2* are the IDs of the two nodes.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

1.1.3 Node degree, participation coefficient, cartography

This section includes programs to compute the total degree and participation coefficient of each node, and to draw the cartography diagram of a multiplex.

`overlap_degree.py`

NAME

overlap_degree.py - compute the total (overlapping) degree of all the nodes of a multiplex and the corresponding Z-score.

SYNOPSYS

overlap_degree.py *<layer1> <layer2> [<layer3>...]*

DESCRIPTION

Compute and print on output the total degree o_i of each node i of a multiplex, defined as:

$$o_i = \sum_{\alpha} \sum_{j} a_{ij}^{[\alpha]}$$

and the corresponding Z-score:

$$z(o_i) = \frac{o_i - \langle o \rangle}{\sigma_o}$$

where $\langle o \rangle$ and σ_o are, respectively, the mean and the standard deviation of the total degree computed over all the active nodes of the multiplex.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines in the format:

ID_n deg_n z_n

where *ID_n* is the ID of the node, *deg_n* is its total degree, and *z_n* is the corresponding Z-score.
As usual, node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers, and the node is omitted from the output.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

`cartography_from_layers.py`

NAME

cartography_from_layers.py - compute the total degree and the multiplex participation coefficient of all the nodes of a multiplex.

SYNOPSYS

cartography_from_layers.py *<layer1> <layer2> [<layer3>...]*

DESCRIPTION

Compute and print on output the total degree and the multiplex participation coefficient P_i for each node i of a multiplex. The participation coefficient is defined as:

$$P_i = \frac{M}{M-1}\left[1 - \sum_{\alpha=1}^{M}\left(\frac{k_i^{[\alpha]}}{o_i}\right)^2\right]$$

Note that P_i takes values in $[0,1]$, where $P_i = 0$ if and only if node i is active on exactly one of the layers, while $P_i = 1$ if node i has equal degree on all the M layers.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines in the format:

deg_n P_n col_n

where *deg_n* is the total degree of node n, *P_n* is the participation coefficient of node n and *col* is the integer representation of the activity bitstring of node n, which is a number between 0 and $2^M - 1$. The field *col* might be useful for the visualisation of the multiplex cartography diagram, where it would be possible to associate different colors to nodes having different node activity patterns.
As usual, node IDs start from zero and proceed sequentially, without gaps, i.e., if a node ID is not present in any of the layer files given as input, the program considers it as being isolated on all the layers, and is set to zero.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

`cartography_from_deg_vectors.py`

NAME

cartography_from_deg_vectors.py - create a multiplex cartography diagram.

SYNOPSYS

cartography_from_deg_vectors.py <*node_deg_vectors*>

DESCRIPTION

Compute and print on output the total degree and the multiplex participation coefficient of all the nodes of a multiplex network whose list of node degree vectors is provided as input. The input file is in the format:

IDn_deg1 IDn_deg_2 ... IDn_degM

where *IDn_degX* is the degree of node n at layer X. The input file can be generated using the script `node_degree_vectors.py`.

OUTPUT

The program prints on `stdout` a list of lines in the format:

tot_deg part_coeff

where *tot_deg* is the total degree of the node and *part_coeff* is the corresponding participation coefficient.

As usual, node IDs start from zero and proceed sequentially, without gaps, so if one of the lines in the input files contains just zeros, the program considers the corresponding node as being isolated on all the layers, and both its total degree and multiplex participation coefficient are set equal to zero.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

`cartography_from_columns.py`

NAME

cartography_from_columns.py - compute total and participation coefficient of generic structural descriptors of the nodes of a multiplex.

SYNOPSYS

cartography_from_columns.py <*filein*> <*col1*> <*col2*> [<*col3*>...]

DESCRIPTION

Compute and print on output the sum and the corresponding participation coefficient of a generic structural descriptor of the nodes of a multiplex. The input file is a generic collection of single-space-separated columns, where each line corresponds to a node. The user must specify the IDs of the columns which contain the node structural descriptors to be used in the cartography diagram. Columns IDs start from ZERO. For example:

python cartography_from_layers.py filein.txt 0 2 4 6 8

will create a cartography diagram assuming that the multiplex network has five layers, and that the node structural descriptors at each layers are contained in the first (0), third (2), fifth (4), seventh (6) and nineth (8) columns of each row.

OUTPUT

The program prints on `stdout` a list of lines in the format:

tot_n P_n

where *tot_n* is the sum over the layers of the considered structural descriptor for node *n*, and *P_n* is the associated participation coefficient

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

1.1.4 Edge overlap, reinforcement

This section includes programs to compute the egde overlap and to evaulate the edge reinforcement effect.

`edge_overlap.py`

NAME

edge_overlap.py - compute the edge overlap of all the edges of the multiplex.

SYNOPSYS

edge_overlap.py `<layer1>` `[<layer2>...]`

DESCRIPTION

Compute and print on output the edge overlap o_{ij} of each edge of the multiplex. Given a pair of nodes (i, j) that are directly connected on at least one of the M layers, the edge overlap o_{ij} is defined as:

$$o_{ij} = \sum_{\alpha} a_{ij}^{[\alpha]}$$

i.e., the number of layers on which the edge (i, j) exists.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a list of lines in the format:

ID_1 ID_2 overlap

where *ID_1* and *ID_2* are the IDs of the end-points of the edge, and *overlap* is the number of layers in which the edge exists.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

`avg_edge_overlap.py`

NAME

avg_edge_overlap.py - compute the average edge overlap of a multiplex.

SYNOPSYS

avg_edge_overlap.py <*layer1*> [<*layer2*>...]

DESCRIPTION

Compute and print on output the average edge overlap

$$\omega^* = \frac{\sum_i \sum_{j>i} \sum_\alpha a_{ij}^{[\alpha]}}{\sum_i \sum_{j>i}(1 - \delta_{0,\sum_\alpha a_{ij}^{[\alpha]}})}$$

i.e., the expected *number* of layers on which an edge of the multiplex exists, and the corresponding normalised quantity:

$$\omega = \frac{\sum_i \sum_{j>i} \sum_\alpha a_{ij}^{[\alpha]}}{M \sum_i \sum_{j>i}(1 - \delta_{0,\sum_\alpha a_{ij}^{[\alpha]}})}$$

that is the expected *fraction* of layers on which an edge of the multiplex is present.

Each input file contains the (undirected) edge list of a layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

OUTPUT

The program prints on `stdout` a single line, in the format:

omega_star omega

where *omega_star* and *omega* are, respectively, the expected number and fraction of layers in which an edge is present.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

L. Lacasa, V. Nicosia, V. Latora, *"Network structure of multivariate time series"*, accepted for publication in Scientific Reports, arxiv:1408.0925 (2015).

Link to paper: `http://arxiv.org/abs/1408.0925`

`reinforcement.py`

NAME

reinforcement.py - compute the probability to have a link between two nodes in layer 1 given their weight in layer 2.

SYNOPSYS

reinforcement.py <*layer1*> <*layer2*> $< N_{bins} >$ $< min_{value} >$ $< max_{value} >$

DESCRIPTION

Compute and print on output the probability to have a link between two nodes in layer 1 given their weight in layer 2. As input are given the files *layer1*, *layer2*, the number of bins for the link weights of the second layer, the minimum and the maximum values of the binning.

The first file contains the binary edge list of layer 1, the second file contains the weighted edge list of layer 2. each line is in the format:

bin_min bin_max freq

where *bin_min* and *bin_max* are the minimum and maximum values of the link weights of layer 2 in that binning, and *freq* is the probability to have a link on layer 1 given such weight in layer 2.

OUTPUT

A list of lines, where the n-th line is the minimum and maximum values of the weight of the links in layer 2 in the n-th bin, and the frequency to have a link on layer 2 given that weight.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, "Structural measures for multiplex networks", *Phys. Rev. E* **89**, 032804 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.89.032804`

1.2 Inter-layer degree correlations

1.2.1 Node ranking

This section includes various utilities to compute and compare node rankings induced by any generic structural node property, including degree at different layers.

`rank_nodes.py`

NAME

rank_nodes.py - rank the nodes of a layer according to a given structural descriptor.

SYNOPSYS

rank_nodes.py <*prop_file*>

DESCRIPTION

Get a file as input, whose n-th line corresponds to the value of a certain property of the n-th node, and rank the nodes according to that property, taking into account ranking ties properly.

For example, if *propfile* contains the degrees of the nodes at a certain layer of the multiplex, the computes the ranking induced by degrees, where the node with the highest degree will be assigned a rank equal to **1** (one).

OUTPUT

The program prints on `stdout` a list of lines, where the n-th line contains the rank of the n-th node corresponding to the values of the structural descriptor provided in the input file.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`rank_nodes_thresh.py`

NAME

rank_nodes_thresh.py - rank the nodes of a layer whose value of a given structural descriptor is above a threshold.

SYNOPSYS

rank_nodes_thresh.py <*prop_file*> <*thresh*>

DESCRIPTION

Get a file as input, whose n-th line corresponds to the value of a certain property of the n-th node, and rank the nodes according to that property, taking into account ranking ties properly. The rank of all the nodes whose value of the structural descriptor is smaller than the threshold *thresh* specified as second parameter is set to **0** (ZERO).

OUTPUT

The program prints on `stdout` a list of lines, where the n-th line contains the rank of the n-th node corresponding to the values of the structural descriptor provided in the input file, or zero if such desxriptor is below the specified threshold *thresh*.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`rank_occurrence.py`

NAME

rank_occurrence.py - compute the intersection of two rankings.

SYNOPSYS

rank_occurrence.py <*rank1*> <*rank2*> <*increment*>

DESCRIPTION

Get two rankings *rank1* and *rank2* and compute the size of the *k*-intersection, i.e. the number of elements which are present in the first k positions of both rankings, as a function of *k*. The parameter *increment* determines the distance between two subsequent values of *k*.

Each input file is a list of node IDs, one per line, where the first line contains the ID of the highest ranked node.

OUTPUT

The program prints on `stdout` a list of lines in the format:

k num_k

where *num_k* is the number of nodes which are present in the first *k* positions of both rankings.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

1.2.2 Interlayer degree correlation coefficients

This section includes programs for the computation of various inter-layer degree correlation coefficients.

`compute_pearson.py`

NAME

compute_pearson.py - compute the Pearson's linear correlation coefficient between two node properties.

SYNOPSYS

compute_pearson.py <*file1*> <*file2*>

DESCRIPTION

Compute the Pearson's linear correlation coefficient between two sets of (either integer- or real-valued) node properties provided in the input files *file1* and *file2*. Each input file contains a list of lines, where the n-th line contains the value of a node property for the n-th node. For instance, *file1* and *file2* might contain the degrees of nodes at two distinct layers of a multiplex. However, the program is pretty general and can be used to compute the Pearson's correlation coeffcient between any pairs of node properties.

OUTPUT

The program prints on `stdout` the value of the Pearson's linear correlation coefficient between the two sets of node properties.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

`compute_rho.py`

NAME

compute_rho.py - compute the Spearman's rank correlation coefficient ρ between two rankings.

SYNOPSYS

compute_rho.py <*file1*> <*file2*>

DESCRIPTION

Compute the Spearman's rank correlation coefficient ρ between two rankings provided in the input files *file1* and *file2*. Each input file contains a list of lines, where the n-th line contains the value of rank of the n-th node. For instance, *file1* and *file2* might contain the ranks of nodes induced by the degree sequences of two distinct layers of a multiplex.

However, the program is pretty general and can be used to compute the Spearman's rank correlation coefficient between any generic pair of rankings.

N.B.: A C implementation of this program, with the same interface is also available in the executable file `compute_rho`.

OUTPUT

The program prints on `stdout` the value of the Spearman's rank correlation coefficient ρ between the two rankings provided as input.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

`compute_tau.py`

NAME

compute_tau.py - compute the Kendall's rank correlation coefficient τ_b between two rankings.

SYNOPSYS

compute_tau.py <*file1*> <*file2*>

DESCRIPTION

Compute the Kendall's rank correlation coefficient τ_b between two rankings provided in the input files *file1* and *file2*. Each input file contains a list of lines, where the n-th line contains the value of rank of the n-th node. For instance, *file1* and *file2* might contain the ranks of nodes induced by the degree sequences of two distinct layers of a multiplex.

However, the program is pretty general and can be used to compute the Kendall's rank correlation coefficient between any generic pair of rankings.

N.B.: This implementation takes properly into account rank ties.

OUTPUT

The program prints on `stdout` the value of the Kendall's rank correlation coefficient τ_b between the two rankings provided as input.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

1.2.3 Interlayer degree correlation functions

This section includes programs to compute intra-layer and inter-layer degree correlation functions, and to fit those functions with a power-law.

M

`dump_k_q`

NAME

dump_k_q - compute the degree sequences of two layers of a multiplex.

SYNOPSYS

dump_k_q <*layer1*> <*layer2*> <*pairing*>

DESCRIPTION

Compute and dump on `stdout` the degree sequences of two layers of a multiplex. The input files *layer1* and *layer2* contain the (undirected) edge lists of the two layers, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The third file *pairing* is a list of lines in the format:

IDL1 IDL2

where *IDL1* is the ID of a node on layer 1 and *IDL2* is the ID of the same node on layer 2. For instance, the line:

5 27

indicates that node 5 on layer 1 has ID 27 on layer 2.

OUTPUT

The program prints on `stdout` the degree of each node on the two layers, in the format:

ki qi

where *ki* is the degree of node *i* on layer 1 and *qi* is the degree of node *i* on layer 2.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

`knn_q_from_layers.py`

NAME

knn_q_from_layers.py - compute intra-layer and inter-layer degree-degree correlation coefficients.

SYNOPSYS

knn_q_from_layers.py <*layer1*> <*layer2*>

DESCRIPTION

Compute the intra-layer and the inter-layer degree correlation functions for two layers given as input. The intra-layer degree correlation function quantifies the presence of degree-degree correlations in a single layer network, and is defined as:

$$\langle k_{nn}(k)\rangle = \frac{1}{kN_k}\sum_{k'} k'P(k'|k)$$

where $P(k'|k)$ is the probability that a neighbour of a node with degree k has degree k', and N_k is the number of nodes with degree k. The quantity $\langle k_{nn}(k)\rangle$ is the average degree of the neighbours of nodes having degree equal to k.

If we consider two layers of a multiplex, and we denote by k the degree of a node on the first layer and by q the degree of the same node on the second layers, the inter-layer degree correlation function is defined as

$$\overline{k}(q) = \sum_{k'} k'P(k'|q)$$

where $P(k'|q)$ is the probability that a node with degree q on the second layer has degree equal to k' on the first layer, and N_q is the number of nodes with degree q on the second layer. The quantity $\overline{k}(q)$ is the expected degree at layer 1 of node that have degree equal to q on layer 2. The dual quantity:

$$\overline{q}(k) = \sum_{q'} q'P(q'|k)$$

is the average degree on layer 2 of nodes having degree k on layer 1.

OUTPUT

The program creates two output files, respectively called

file1_file2_k1

and

file1_file2_k2

The first file contains a list of lines in the format:

k $\langle k_{nn}(k)\rangle$ σ_k $\overline{q}(k)$ $\sigma_{\overline{q}}$

where k is the degree at first layer, $\langle k_{nn}(k)\rangle$ is the average degree of the neighbours at layer 1 of nodes having degree k at layer 1, σ_k is the standard deviation associated to $\langle k_{nn}(k)\rangle$, $\overline{q}(k)$ is the average degree at layer 2 of nodes having degree equal to k at layer 1, and $\sigma_{\overline{q}}$ is the standard deviation associated to $\overline{q}(k)$.

The second file contains a similar list of lines, in the format:

q $\langle q_{nn}(q)\rangle$ σ_q $\overline{k}(q)$ $\sigma_{\overline{k}}$

with obvious meaning.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

`knn_q_from_degrees.py`

NAME

knn_q_from_degrees.py - compute the inter-layer degree-degree correlation function.

SYNOPSYS

knn_q_from_degrees.py <*filein*>

DESCRIPTION

Compute the inter-layer degree correlation functions for two layers of a multiplex, using the degrees of the nodes specified in the input file. The format of the input file is as follows

ki qi

where *ki* and *qi* are, respectively, the degree at layer 1 and the degree at layer 2 of node *i*.

If we consider two layers of a multiplex, and we denote by k the degree of a node on the first layer and by q the degree of the same node on the second layers, the inter-layer degree correlation function is defined as

$$\overline{k}(q) = \frac{1}{N_k}\sum_{k'} k' P(k'|q)$$

where $P(k'|q)$ is the probability that a node with degree q on the second layer has degree equal to k' on the first layer, and N_k is the number of nodes with degree k on the first layer. The quantity $\overline{k}(q)$ is the expected degree at layer 1 of node that have degree equal to q on layer 2. The dual quantity:

$$\overline{q}(k) = \frac{1}{N_q}\sum_{q'} q' P(q'|k)$$

is the average degree on layer 2 of nodes having degree k on layer 1.

OUTPUT

The program prints on `stdout` a list of lines in the format:

$k\ \overline{q}(k)$

where k is the degree on layer 1 and $\overline{q}(k)$ is the average degree on layer 2 of nodes having degree equal to k on layer 1.

The program also prints on `stderr` a list of lines in the format:

$q\ \overline{k}(q)$

where q is the degree on layer 2 and $\overline{k}(q)$ is the average degree on layer 1 of nodes having degree equal to q on layer 2.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

`fit_knn`

NAME

fit_knn - power-law fit of the inter-layer degree correlation function.

SYNOPSYS

fit_knn <*filein*> <*alpha*>

DESCRIPTION

Perform a power-law fit of the inter-layer degree correlation function:

$$\overline{q}(k) = \frac{1}{N_q}\sum_{q'} q' P(q'|k)$$

where k is the degree of a node on layer 1, q is the degree on layer 2 and $P(q|k)$ is the probability that a node with degree k on layer 1 has degree q on layer 2. The program assumes that $\overline{q}(k)$ can be written in the form ak^b, and computes the two parameters a and b through a linear fit of the log-log plot of $\overline{q}(k)$.

The input file *filein* contains a list of lines in the format:

ki qi

where *ki* is the degree of node i at layer 1 and *qi* is the degree of node i at layer 2.

The second parameter *alpha* is the ratio of the progression used to generate the exponentially-distributed bins for the log-log plot. Typical values of *alpha* are between 1.1 and 2.0.

N.B.: The exponent b computed with this method is known to be inaccurate.

OUTPUT

The program prints on `stdout` the values of the parameters a and b of the power-law fit $\overline{q}(k) = ak^b$.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Non-linear growth and condensation in multiplex networks", *Phys. Rev. E* **90**, 042807 (2014).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.90.042807`

Chapter 2

Models of multi-layer networks

2.1 Null models

2.1.1 Null-models of node and layer activity

`model_hypergeometric.py`

NAME

model_hypergeometric.py - Hypergeometric node activity null model.

SYNOPSYS

model_hypergeometric.py <*layer_N_file*> <*N*>

DESCRIPTION

This is the hypergeometric model of node activation. In this model each layer has exactly the same number of active node of a reference multiplex network, but nodes on each layer are activated uniformly at random, thus destroying all inter-layer activity correlation patterns.

The file *layer_N_file* reports on the n-th line the number of active nodes on the n-th layer (starting from zero). The second parameter N is the total number of active nodes in the multiplex.

OUTPUT

The program prints on `stdout` a node-layer list of lines in the format:

node_i layer_i

where *node_i* is the ID of a node and *layre_i* is the ID of a layer. This list indicates which nodes are active in which layer. For instance, the line:

24 3

indicates that the node with ID *24* is active on layer *3*.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`model_MDM.py`

NAME

model_MDM.py - Multi-activity Deterministic Model.

SYNOPSYS

model_MDM.py <*Bi_file*> <*M*>

DESCRIPTION

This is the Multi-activity Deterministic Model (MDM). In this model each node i is considered active if it was active in the reference multiplex, maintains the same value of node activity B_i (i.e., the number of layers in which it was active) and is associated an activity vector sampled uniformly at random from the $\binom{M}{B_i}$ possible activity vectors with B_i non-null entries.

The file *Bi_file* is in the format:

Bi N(Bi)

where *Bi* is a value of node activity and *N(Bi)* is the number of nodes which had node activity equaly to *Bi* in the reference multiplex.

The parameter M is the number of layers in the multiplex.

OUTPUT

The program prints on `stdout` a distribution of bit-strings, in the format:

Bi bitstring count

where *bitstring* is the activity bitstring, *Bi* is the number of non-zero entries of *bitstring* and *count* is the number of times that *bitstrings* appear in the null model.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`model_MSM.py`

NAME

model_MSM.py - Multi-activity Stochastic Model.

SYNOPSYS

model_MSM.py <*node_Bi_file*> <*M*>

DESCRIPTION

This is the Multi-activity Stochastic Model (MSM). In this model each node i is considered active if it was active in the reference multiplex, and is activated on each layer with a probability equal to B_i/M where B_i was the activity of node i in the reference multiplex.

The file *node_Bi_file* is in the format:

node_i Bi)

where *Bi* is the value of node activity of *node_i* in the reference multiplex.

The parameter M is the number of layers in the multiplex.

OUTPUT

The program prints on `stdout` a node-layer list of lines in the format:

node_i layer_i

where *node_i* is the ID of a node and *layre_i* is the ID of a layer. This list indicates which nodes are active in which layer. For instance, the line:

24 3

indicates that the node with ID *24* is active on layer *3*.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`model_layer_growth.py`

NAME

model_layer_growth.py - Layer growth with preferential activation model.

SYNOPSYS

model_layer_growth.py <*layer_N_file*> <*N*> <*M0*> <*A*> *[RND]*

DESCRIPTION

This is the model of layer growth with preferential node activation. In this model an entire new layer arrives at time t and a number of nodes N_t is activated (N_t is equal to the number of nodes active on that layer in the reference multiplex). Then, each node i of the new layer is activated with a probability:

$$P_i(t) \propto A + B_i(t)$$

where $B_i(t)$ is the activity of node i at time t (i.e., the number of layers in which node i is active at time t) while $A > 0$ is an intrinsic attractiveness.

The file *layer_N_file* reports on the n-th line the number of active nodes on the n-th layer.

The parameter N is the number of nodes in the multiplex, *M0* is the number of layers in the initial network, A is the value of node attractiveness.

If the user specifies `RND` as the last parameter, the sequence of layers is

OUTPUT

The program prints on `stdout` a node-layer list of lines in the format:

node_i layer_i

where *node_i* is the ID of a node and *layre_i* is the ID of a layer. This list indicates which nodes are active in which layer. For instance, the line:

24 3

indicates that the node with ID *24* is active on layer *3*.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

2.2 Growing multiplex networks

2.2.1 Linear preferential attachment

`nibilab_linear_delta`

NAME

nibilab_linear_delta - Multiplex linear preferential attachment model – Synchronous arrival.

SYNOPSYS

nibilab_linear_delta <*N*> <*m*> <*m0*> <*outfile*> <*a*> <*b*> <*c*> <*d*>

DESCRIPTION

Grow a two-layer multiplex network using the multiplex linear preferential attachment model by Nicosia, Bianconi, Latora, Barthelemy (NiBiLaB).

The probability for a newly arrived node i to create a link to node j on layer 1 is:

$$\Pi^1_{i \to j} \propto a k_j^{[1]} + b k_j^{[2]}$$

and the dual probability for i to create a link to j on layer 2 is:

$$\Pi^2_{i \to j} \propto c k_j^{[1]} + d k_j^{[2]}$$

Each new node arrives at the same time on both layers.

The (mandatory) parameters are as follows:

- **N** number of nodes in the final graph
- **m** number of new edges brought by each new node
- **m0** number of nodes in the initial seed graph. *m0* must be larger than of equal to *m*.
- **outfile** the name of the file which will contain the
- **a,b,c,d** the coefficients of the attaching probability function

OUTPUT

The program dumps on the file `outfile` the (undirected) edge list of the resulting network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

`nibilab_linear_delay`

NAME

nibilab_linear_delay - Multiplex linear preferential attachment model – Asynchronous arrival.

SYNOPSYS

nibilab_linear_delay <*N*> <*m*> <*m0*> <*outfile*> <*a*> <*b*> <*c*> <*d*> <*beta*>

DESCRIPTION

Grow a two-layer multiplex network using the multiplex linear preferential attachment model by Nicosia, Bianconi, Latora, Barthelemy (NiBiLaB).

The probability for a newly arrived node i to create a link to node j on layer 1 is:

$$\Pi^1_{i\to j} \propto ak_j^{[1]} + bk_j^{[2]}$$

and the dual probability for i to create a link to j on layer 2 is:

$$\Pi^2_{i\to j} \propto ck_j^{[1]} + dk_j^{[2]}$$

Each new node arrives first on layer 1, and its replica on the layer 2 appears after a time delay τ sampled from the power-law function:

$$P(\tau) \sim \tau^{-\beta}$$

The (mandatory) parameters are as follows:

- **N** number of nodes in the final graph
- **m** number of new edges brought by each new node
- **m0** number of nodes in the initial seed graph. *m0* must be larger than of equal to *m*.
- **outfile** the name of the file which will contain the
- **a,b,c,d** the coefficients of the attaching probability function
- **beta** the exponent of the power-law delay function which determines the arrival of replicas on layer 2

OUTPUT

The program dumps on the file `outfile` the (undirected) edge list of the resulting network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

`nibilab_linear_delay_mix`

NAME

nibilab_linear_delay_mix - Multiplex linear preferential attachment model – Asynchronous arrival and randomly selected first layer.

SYNOPSYS

nibilab_linear_delay_mix <*N*> <*m*> <*m0*> <*outfile*> <*a*> <*b*> <*c*> <*d*> <*beta*>

DESCRIPTION

Grow a two-layer multiplex network using the multiplex linear preferential attachment model by Nicosia, Bianconi, Latora, Barthelemy (NiBiLaB).

The probability for a newly arrived node i to create a link to node j on layer 1 is:

$$\Pi^1_{i \to j} \propto a k_j^{[1]} + b k_j^{[2]}$$

and the dual probability for i to create a link to j on layer 2 is:

$$\Pi^2_{i \to j} \propto c k_j^{[1]} + d k_j^{[2]}$$

Each new node arrives on one of the two layers, chosen uniformly at random, and its replica on the other layer appears after a time delay τ sampled from the power-law function:

$$P(\tau) \sim \tau^{-\beta}$$

The (mandatory) parameters are as follows:

- **N** number of nodes in the final graph
- **m** number of new edges brought by each new node

- **m0** number of nodes in the initial seed graph. *m0* must be larger than of equal to *m*.
- **outfile** the name of the file which will contain the
- **a,b,c,d** the coefficients of the attaching probability function
- **beta** the exponent of the power-law delay function which determines the arrival of replicas on layer 2

OUTPUT

The program dumps on the file `outfile` the (undirected) edge list of the resulting network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

`nibilab_linear_random_times`

NAME

nibilab_linear_random_times - Multiplex linear preferential attachment model – Asynchronous arrival with randomly sampled arrival times on layer 2.

SYNOPSYS

nibilab_linear_random_times <*N*> <*m*> <*m0*> <*outfile*> <*a*> <*b*> <*c*> <*d*>

DESCRIPTION

Grow a two-layer multiplex network using the multiplex linear preferential attachment model by Nicosia, Bianconi, Latora, Barthelemy (NiBiLaB).

The probability for a newly arrived node i to create a link to node j on layer 1 is:

$$\Pi_{i\to j}^{1} \propto ak_j^{[1]} + bk_j^{[2]}$$

and the dual probability for i to create a link to j on layer 2 is:

$$\Pi_{i\to j}^{2} \propto ck_j^{[1]} + dk_j^{[2]}$$

Each new node arrives on layer 1, but its replica on the other layer appears at a uniformly chosen random time in $[m0 + 1; N]$.

The (mandatory) parameters are as follows:

- **N** number of nodes in the final graph
- **m** number of new edges brought by each new node
- **m0** number of nodes in the initial seed graph. *m0* must be larger than of equal to *m*.
- **outfile** the name of the file which will contain the
- **a,b,c,d** the coefficients of the attaching probability function

OUTPUT

The program dumps on the file `outfile` the (undirected) edge list of the resulting network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

2.2.2 Non-linear preferential attachment

`nibilab_nonlinear`

NAME

nibilab_nonlinear - Multiplex non-linear preferential attachment model – Synchronous arrival.

SYNOPSYS

nibilab_nonlinear <*N*> <*m*> <*m0*> <*outfile*> <*alpha*> <*beta*>

DESCRIPTION

Grow a two-layer multiplex network using the multiplex non-linear preferential attachment model by Nicosia, Bianconi, Latora, Barthelemy (NiBiLaB).

The probability for a newly arrived node i to create a link to node j on layer 1 is:

$$\Pi^1_{i\to j} \propto \frac{\left(k_j^{[1]}\right)^\alpha}{\left(k_j^{[2]}\right)^\beta}$$

and the dual probability for i to create a link to j on layer 2 is:

$$\Pi^2_{i\to j} \propto \frac{\left(k_j^{[2]}\right)^\alpha}{\left(k_j^{[1]}\right)^\beta}$$

Each node arrives simultaneously on both layers.

The (mandatory) parameters are as follows:

- **N** number of nodes in the final graph
- **m** number of new edges brought by each new node
- **m0** number of nodes in the initial seed graph. *m0* must be larger than of equal to *m*.
- **outfile** the name of the file which will contain the
- **alpha, beta** exponents of of the attaching probability function

OUTPUT

The program dumps on the file `outfile` the (undirected) edge list of the resulting network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

2.2.3 Utilities

`node_deg_over_time.py`

NAME

node_deg_over_time.py - Time evolution of the degree of a node in a growing graph.

SYNOPSYS

node_deg_over_time.py <*layer*> <*arrival_times*> <*node_id*> [<*node_id*> ...]

DESCRIPTION

Compute the degree $k_i(t)$ of node i in a growing network as a function of time. The file *layer* contains the edge list of the final network. Each line of the file is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The file *arrival_times* is a list of node arrival times, in the format:

time_i node_i

where *time_i* is the time at which *node_i* arrived in the graph. Notice that *time_i* must be an integer in the range [0, N-1], where N is the total number of nodes in the final graph.

The third parameter *node_id* is the ID of the node whose degree over time will be printed on output. If more than one *node_id* is provided, the degrees over time of all the corresponding nodes are printed on output.

OUTPUT

The program prints on `stdout` a list of lines in the format:

t kit

where *kit* is the degree of node i at time t. The first line of output is in the format:

#### node_id

where *node_id* is the ID of node i.

If more than one *node_id*s is provided as input, the program prints the degree over time of all of them, sequentially.

REFERENCE

V. Nicosia, G. Bianconi, V. Latora, M. Barthelemy, "Growing multiplex networks", *Phys. Rev. Lett.* **111**, 058701 (2013).

Link to paper: `http://prl.aps.org/abstract/PRL/v111/i5/e058701`

2.3 Multiplex networks with inter-layer degree correlations

2.3.1 Models based on simulated annealing

`tune_rho`

NAME

tune_rho - Construct a multiplex with prescribed inter-layer correlations.

SYNOPSYS

tune_rho <*rank1*> <*rank2*> <*rho*> <*eps*> <*beta*> *[RND|NAT|INV]*

DESCRIPTION

This programs tunes the inter-layer degree correlation coefficient ρ (Spearman's rank correlation) of two layers, by adjusting the inter-layer pairing of nodes. The files *rank1* and *rank2* are the rankings of nodes in the first and second layer, where the n-th line of the file contains the rank of the n-th node (the highest ranked node has rank equal to 1).

The parameter *rho* is the desired value of the Spearman's rank correlation coefficient, while *eps* is the accuracy of *rho*. For instance, if *rho* is set equal to -0.25 and *eps* is equal to 0.0001, the program stops when the configuration of node pairing corresponds to a value of ρ which differs from -0.25 by less than 0.0001.

The parameter *beta* is the typical inverse temperature of simulated annealing.

If no other parameter is specified, or if the last parameter is `RND`, the program starts from a random pairing of nodes. If the last parameter is `NAT` then the program assumes that the initial pairing is the natural one, where the nodes have the same ID on both layers. Finally, if `INV` is specified, the initial pairing is the inverse pairing, i.e. the one where node 0 on layer 1 is paired with node N-1 on layer 2, and so on.

OUTPUT

The program prints on `stdout` a pairing, i.e. a list of lines in the format:

IDL1 IDL2

where *IDL1* is the ID of the node on layer 1 and *IDL2* is the corresponding ID of the same node on layer 2.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

`tune_qnn_adaptive`

NAME

tune_qnn_adaptive - Construct a multiplex with prescribed inter-layer correlations.

SYNOPSYS

tune_qnn_adaptive <*degs1*> <*degs2*> <*mu*> <*eps*> <*beta*> *[RND|NAT|INV]*

DESCRIPTION

This programs tunes the inter-layer degree correlation exponent μ. If we consider two layers of a multiplex, and we denote by k the degree of a node on the first layer and by q the degree of the same node on the second layers, the inter-layer degree correlation function is defined as:

$$\overline{q}(k) = \sum_{q'} q' P(q'|k)$$

where $\overline{q}(k)$ is the average degree on layer 2 of nodes having degree k on layer 1.

The program assumes that we want to set the degree correlation function such that:

$$\overline{q}(k) = ak^{\mu}$$

where the exponent of the power-law function is given by the user (it is indeed the parameter *mu*), and successively adjusts the pairing between nodes at the two layers in order to obtain a correlation function as close as possible to the desired one. The files *degs1* and *degs2* contain, respectively, the degrees of the nodes on the first layer and on the second layer.

The parameter *eps* is the accuracy of *mu*. For instance, if *mu* is set equal to -0.25 and *eps* is equal to 0.0001, the program stops when the configuration of node pairing corresponds to a value of the exponent μ which differs from -0.25 by less than 0.0001.

The parameter *beta* is the typical inverse temperature of simulated annealing.

If no other parameter is specified, or if the last parameter is `RND`, the program starts from a random pairing of nodes. If the last parameter is `NAT` then the program assumes that the initial pairing is the natural one, where the nodes have the same ID on both layers. Finally, if `INV` is specified, the initial pairing is the inverse pairing, i.e. the one where node 0 on layer 1 is paired with node N-1 on layer 2, and so on.

OUTPUT

The program prints on `stdout` a pairing, i.e. a list of lines in the format:

IDL1 IDL2

where *IDL1* is the ID of the node on layer 1 and *IDL2* is the corresponding ID of the same node on layer 2.

REFERENCE

V. Nicosia, V. Latora, "Measuring and modeling correlations in multiplex networks", *Phys. Rev. E* **92**, 032805 (2015).

Link to paper: `http://journals.aps.org/pre/abstract/10.1103/PhysRevE.92.032805`

Chapter 3

Dynamics on multi-layer networks

3.1 Interacting opinions - Multilayer ising model

`multiplex_ising`

NAME

multiplex_ising - compute the coupled ising model in a multiplex with 2 layers.

SYNOPSYS

multiplex_ising <*layer1*> <*layer2*> <*T*> <*J*> $< \gamma >$ $< h^{[1]} >$ $< h^{[2]} >$ $< p_1 >$ $< p_2 >$ $< numepochs >$

DESCRIPTION

Compute and print the output of the ising dynamics on two coupled layers of a multiplex network. Files *layer1*, *layer2*, contain the (undirected) edge list of the two layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

T is the value of thermal noise in the system, J the value of peer pressure, γ the relative ratio between internal coupling and peer pressure, $h^{[1]}$ and $h^{[2]}$ the external fields acting on the two layers, p_1 the probability for a spin on layer 1 at $t = 0$ to be up, p_2 the same probability for spins on layer 2, *numepochs* the number of epochs for the simulation.

OUTPUT

One line, reporting all controlling parameter, the value of consensus in layer 1 $m^{[1]}$, the value of consensus in layer 2 $m^{[2]}$ and the coherence C.

REFERENCE

F. Battiston, A. Cairoli, V. Nicosia, A. Baule, V. Latora, *"Interplay between consensus and coherence in a model of interacting opinions"*, accepted for publication in Physica D, arxiv:1506.04544 (2015).

Link to paper: `http://arxiv.org/abs/1506.04544`

3.2 Biased random walks

3.2.1 Stationary distribution

`statdistr2`

NAME

statdistr2 - compute the stationary distribution of additive, multiplicative and intensive biased walks in a multiplex with 2 layers.

SYNOPSYS

statdistr2 <*`layer1`*> <*`layer2`*> < *overlappingnetwork* > <*`N`*> b_1 b_2

DESCRIPTION

Compute and print the stationary distribution of additive, multiplicative and intensive biased walks in a multiplex with 2 layers. Files *layer1*, *layer2*, contain the (undirected) edge list of the two layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The file *overlapping network* has also a third column indicating the number of times two nodes are connected across all layers.

N is the number of nodes, b_1 is the first bias exponent (the bias exponent for layer 1 for additive and multiplicative walks, the bias exponent on the participation coefficient for intensive walks), b_2 is the second bias exponent (the bias exponent for layer 1 for additive and multiplicative walks, the bias exponent on the participation coefficient for intensive walks).

OUTPUT

N lines. In the n-th line we report the node ID, the stationary distribution of that node for additive walks with exponents b_1 and b_2, the stationary distribution for multiplicative walks with exponents b_1 and b_2, the stationary distribution for multiplicative walks with exponents b_1 and b_2, the values of the bias exponents b_1 and b_2.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, *"Biased random walks on multiplex networks"*, arxiv:1505.01378 (2015).

Link to paper: `http://arxiv.org/abs/1505.01378`

3.2.2 Entropy rate

`entropyrate2add`

NAME

entropyrate2add - compute the entropy rate of additive biased walks in a multiplex with 2 layers.

SYNOPSYS

entropyrate2add <*layer1*> <*layer2*> < *overlappingnetwork* > <*N*> b_1 b_2

DESCRIPTION

Compute and print the entropy rate of an additive biased walk in a multiplex with 2 layers and bias parameters b_1 and b_2. Files *layer1*, *layer2*, contain the (undirected) edge list of the two layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The file *overlapping network* has also a third column indicating the number of times two nodes are connected across all layers.

N is the number of nodes, b_1 is the degree-biased exponent for layer 1, b_2 is the degree-biased exponent for layer 2.

OUTPUT

One line, reporting the value of the entropy rate h of an additive biased random walks with b_1 and b_2 as bias exponents, b_1 and b_2.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, *"Biased random walks on multiplex networks"*, arxiv:1505.01378 (2015).

Link to paper: `http://arxiv.org/abs/1505.01378`

`entropyrate2mult`

NAME

entropyrate2mult - compute the entropy rate of multiplicative biased walks in a multiplex with 2 layers.

SYNOPSYS

entropyrate2mult <*layer1*> <*layer2*> < *overlappingnetwork* > <*N*> b_1 b_2

DESCRIPTION

Compute and print the entropy rate of a multiplicative biased walk in a multiplex with 2 layers and bias parameters b_1 and b_2. Files *layer1*, *layer2*, contain the (undirected) edge list of the two layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The file *overlapping network* has also a third column indicating the number of times two nodes are connected across all layers.

N is the number of nodes, b_1 is the degree-biased exponent for layer 1, b_2 is the degree-biased exponent for layer 2.

OUTPUT

One line, reporting the value of the entropy rate h of an multiplicative biased random walks with b_1 and b_2 as bias exponents, b_1 and b_2.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, *"Biased random walks on multiplex networks"*, arxiv:1505.01378 (2015).

Link to paper: `http://arxiv.org/abs/1505.01378`

`entropyrate2int`

NAME

entropyrate2int - compute the entropy rate of intensive biased walks in a multiplex with 2 layers.

SYNOPSYS

entropyrate2int <*layer1*> <*layer2*> < *overlappingnetwork* > <*N*> b_1 b_2

DESCRIPTION

Compute and print the entropy rate of an intensive biased walks in a multiplex with 2 layers and bias parameters b_p and b_o. Files *layer1*, *layer2*, contain the (undirected) edge list of the two layer, and each line is in the format:

src_ID dest_ID

where *src_ID* and *dest_ID* are the IDs of the two endpoints of an edge.

The file *overlapping network* has also a third column indicating the number of times two nodes are connected across all layers.

N is the number of nodes, b_p is the biased exponent on the participation coefficient, b_o is the biased exponent on the overlapping degree.

OUTPUT

One line, reporting the value of the entropy rate h of an intensive biased random walks with b_p and b_o as bias exponents, b_p and b_o.

REFERENCE

F. Battiston, V. Nicosia, V. Latora, *"Biased random walks on multiplex networks"*, arxiv:1505.01378 (2015).

Link to paper: `http://arxiv.org/abs/1505.01378`