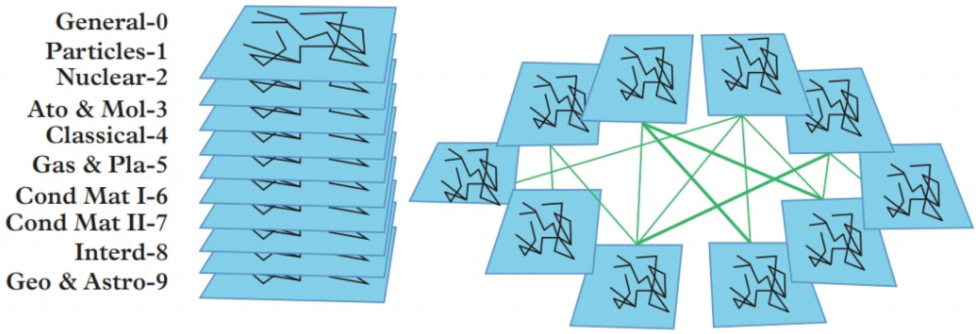
Multilayer Networks

Introduction

First introduced under social science domain, multilayer networks represented different type of relationships of the same nodes. The representation is now generalized to allow different nodes at each layer, different relationships and connections between layers. Transportation networks, social networks, biological networks, and financial networks are examples of multilayer networks.



reference 1.

Basic Concepts

1. Multilayer Network Structure

Definition: A multilayer network consists of multiple layers, where each layer represents a different type of interaction or relationship between the same set of nodes.

Components:

- **Nodes**: Entities that exist across multiple layers
- **Intra-layer edges**: Connections within the same layer
- **Inter-layer edges**: Connections between different layers

- **Layers**: Different types of relationships or time periods

2. Types of Multilayer Networks

Temporal Networks

- **Definition**: Networks that evolve over time
- **Layers**: Different time snapshots
- **Applications**: Social network evolution, transportation networks

Multiplex Networks

- **Definition**: Networks with different types of relationships
- **Layers**: Different relationship types
- **Applications**: Social networks (friendship, work, family)

Interdependent Networks

- **Definition**: Networks where functionality depends on other networks
- **Layers**: Different infrastructure systems
- **Applications**: Power grid and communication networks

Mathematical Representation

1. Supra-Adjacency Matrix

Definition: A block matrix that represents the entire multilayer network.

Structure:

$$A = \begin{bmatrix} A_1 & C_{12} & \cdots & C_{1L} \\ C_{21} & A_2 & \cdots & C_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ C_{L1} & C_{L2} & \cdots & A_L \end{bmatrix}$$

where:

- A_i is the adjacency matrix of layer i
- C_{ij} represents inter-layer connections between layers i and j

2. Tensor Representation

Definition: Using tensors to represent multilayer networks.

Advantages:

- Natural representation of multi-dimensional data
- Efficient mathematical operations
- Clear separation of intra and inter-layer connections

Centrality in Multilayer Networks

1. Eigenvector Centrality

Mathematical Definition:

$$x_v = \frac{1}{\lambda} \sum_{u \in \mathcal{N}(v)} A_{uv} x_u$$

where $\mathcal{N}(v)$ includes neighbors from all layers.

2. PageRank Centrality

Extension to Multilayer:

$$x_v = (1-d) + d \sum_{u \in \mathcal{N}(v)} \frac{A_{uv}}{k_u} x_u$$

where k_u is the total degree across all layers.

3. Versatility

Definition: A centrality measure that considers both intra and inter-layer connections.

Mathematical Definition:

$$V_i = \frac{1}{L} \sum_{\alpha=1}^{L} \frac{k_i^{[\alpha]}}{k_{max}^{[\alpha]}}$$

where:

- $k_i^{[\alpha]}$ is the degree of node i in layer α
- $k_{max}^{[\alpha]}$ is the maximum degree in layer α

Community Detection

1. Multilayer Modularity

Definition: Extension of modularity to multilayer networks.

Mathematical Definition:

$$Q = \frac{1}{2\mu} \sum_{ij\alpha} \left[A_{ij}^{[\alpha]} - \gamma_\alpha \frac{k_i^{[\alpha]} k_j^{[\alpha]}}{2m^{[\alpha]}} \right] \delta(c_i, c_j)$$

where:

- μ is the total number of edges across all layers
- γ_α is the resolution parameter for layer α
- $\delta(c_i, c_j)$ indicates if nodes i and j are in the same community

2. Multilayer Infomap

Algorithm: Extends the Infomap algorithm to multilayer networks.

Steps:

1. Create a supra-graph with inter-layer connections
2. Apply Infomap algorithm
3. Project communities back to individual layers

3. Tensor Decomposition

Method: Use tensor decomposition techniques to find communities.

Approaches:

- **CP Decomposition**: Canonical Polyadic decomposition
- **Tucker Decomposition**: Higher-order SVD
- **Non-negative Tensor Factorization**: For overlapping communities

Random Walk Models

1. Multilayer Random Walk

Definition: Random walk that can move both within and between layers.

Transition Matrix:

$$P_{ij}^{[\alpha\beta]} = \frac{A_{ij}^{[\alpha\beta]}}{\sum_k A_{ik}^{[\alpha\beta]}}$$

where α and β represent layers.

2. Supra-Laplacian

Definition: Laplacian matrix of the supra-adjacency matrix.

Properties:

- Eigenvalues provide information about network structure
- Second smallest eigenvalue indicates algebraic connectivity
- Used for spectral clustering and community detection

Applications

1. Social Networks

Multiplex Social Networks

- **Layers**: Friendship, work, family, online interactions
- **Analysis**: Influence spread, community detection
- **Applications**: Marketing, social influence analysis

Temporal Social Networks

- **Layers**: Time periods (days, weeks, months)
- **Analysis**: Network evolution, temporal patterns
- **Applications**: Social dynamics, trend prediction

2. Transportation Networks

Multimodal Transportation

- **Layers**: Bus, train, subway, walking
- **Analysis**: Route optimization, accessibility
- **Applications**: Urban planning, transportation optimization

Temporal Transportation

- **Layers**: Different time periods (peak hours, off-peak)
- **Analysis**: Temporal patterns, capacity planning
- **Applications**: Traffic management, infrastructure planning

3. Biological Networks

Protein Interaction Networks

- **Layers**: Different experimental conditions
- **Analysis**: Functional modules, disease mechanisms
- **Applications**: Drug discovery, disease understanding

Brain Networks

- **Layers**: Different frequency bands, brain regions
- **Analysis**: Functional connectivity, cognitive processes
- **Applications**: Neuroscience, brain-computer interfaces

4. Infrastructure Networks

Interdependent Infrastructure

- **Layers**: Power grid, communication, water, transportation
- **Analysis**: Cascading failures, system resilience
- **Applications**: Infrastructure planning, disaster management

Analysis Methods

1. Aggregation Methods

Edge Aggregation

```
def aggregate_edges(multilayer_network):
    aggregated = nx.Graph()
    for layer in multilayer_network.layers:
        for edge in layer.edges():
            if aggregated.has_edge(edge[0], edge[1]):
                aggregated[edge[0]][edge[1]]['weight'] += 1
            else:
                aggregated.add_edge(edge[0], edge[1], weight=1)
    return aggregated
```

Node Aggregation

```
def aggregate_nodes(multilayer_network):
    node_attributes = {}
    for node in multilayer_network.nodes:
        node_attributes[node] = {
            'total_degree': sum(layer.degree(node) for layer in
multilayer_network.layers),
            'layer_degrees': [layer.degree(node) for layer in
multilayer_network.layers]
        }
    return node_attributes
```

2. Layer Similarity Analysis

Jaccard Similarity

```
def layer_similarity(layer1, layer2):
    edges1 = set(layer1.edges())
    edges2 = set(layer2.edges())
    intersection = edges1.intersection(edges2)
    union = edges1.union(edges2)
    return len(intersection) / len(union)
```

Correlation Analysis

```
def layer_correlation(multilayer_network):
    layers = list(multilayer_network.layers)
    n_layers = len(layers)
    correlation_matrix = np.zeros((n_layers, n_layers))

    for i in range(n_layers):
        for j in range(n_layers):
            if i != j:
                correlation_matrix[i][j] =
layer_similarity(layers[i], layers[j])

    return correlation_matrix
```

Actor Analysis

Degree standard deviation of an actor through layers is defined as

$$\sigma_k = \sqrt{\frac{1}{L-1}\sum_{\alpha=1}^{L}(k_\alpha - \bar{k})^2}$$

where k_α is the degree of the actor in layer α, $\bar{k}$ is the average degree of the actor, and L is the number of layers.

uunet Python package can be used to analyze multiplex networks. Detailed documentation and a tutorial can be found at uunet

```
#install uunet
import uunet.multinet as ml
n = ml.read('tutorial/example1.txt')
ml.plot(n, vertex_labels_bbox = {"boxstyle":'round4',
"fc":'white'})
```

Helper function defined to convert the output of the library to pandas dataframes.

```
import pandas as pd
# transforms the typical output of the library (dictionaries) into
pandas dataframes
def df(d):
  return pd.DataFrame.from_dict(d)
```

Use networkx objects as layers of the multiplex network

```
import networkx as nx
l1 = nx.read_edgelist("tutorial/example_igraph1.dat")
l2 = nx.read_edgelist("tutorial/example_igraph2.dat")
n = ml.empty()
ml.add_nx_layer(n, l1, "layer1")
ml.add_nx_layer(n, l2, "layer2")
df( ml.edges(n) )
```

The library provides a multiplex mixture growth model based on some evolution functions. It's limited to ER and PA models for now. A layer can grow internally or externally based on probabilities. If externally effected from other layers, dependency matrix can be used to define the dependency between layers.

```
models_mix = [ ml.evolution_pa(3, 1), ml.evolution_er(100),
ml.evolution_er(100) ]
pr_internal = [1, .2, 1]
pr_external = [0, .8, 0]
dependency = [ [1, 0, 0], [0.5, 0, 0.5], [0, 0, 1] ]
generated_mix = ml.grow(100, 150, models_mix, pr_internal,
pr_external, dependency)
l = ml.layout_multiforce(generated_mix, gravity = [.5])
ver = ml.vertices(generated_mix)
deg = [ml.degree(generated_mix, [a], [l])[0] for a,l in
zip(ver['actor'], ver['layer'])]
ml.plot(generated_mix, layout = l, vertex_labels=[],
vertex_size=deg)
```

It is also possible to read existing datasets in the library.

```
net = ml.data("aucs")
ml.layers(net)
```

You can merge some layers into a single layer.

```
ml.flatten(net, "offline", ['work', 'leisure', 'lunch'] )
ml.layers(net)
```

```
net = ml.data("aucs")
l2 = ml.layout_multiforce(net, w_inter = [1], w_in = [1, 0, 0, 0,
0], gravity = [1, 0, 0, 0, 0])
ml.plot(net, layout = l2, grid = [2, 3], vertex_labels = [])
```

Some summary network metrics can be obtained from the library. n, m are number of nodes and edges, nc is number of components, slc is the size of larges component, $dens$ is the density, cc is the average clustering coefficient, apl is the average shortest path length, dia is the diameter of the layer.

```
df( ml.summary(net) )
```

Other network statistics can be obtained converting multiplex object to networkx object.

```
import networkx as nx
layers = ml.to_nx_dict(net)
nx.degree_assortativity_coefficient(layers["facebook"])
```

Layers can be compared in pairs based on difference/similarity of node/edge properties. The formulations are explained in the reference 3.

```
comp = df( ml.layer_comparison(net, method = "jeffrey.degree") )
comp.columns = ml.layers(net)
```

```
comp.index = ml.layers(net)
comp
```

```
df( ml.layer_comparison(net, method = "jaccard.actors") )
```

```
df( ml.layer_comparison(net, method = "pearson.degree") )
```

```
df( ml.layer_comparison(net, method = "jaccard.edges") )
```

Degrees and degree deviations of actors can be obtained.

```
ml.degree(net, actors=['U4'])
```

```
deg = ml.degree(net)
act = ml.actors(net)
list(act.values())[0]

degrees = [ [deg[i], list(act.values())[0][i]] for i in
range(len(deg)) ]
degrees.sort(reverse = True)

degrees[0:5]
```

```
top_actors = []
for el in degrees[0:5]:
      top_actors.append( el[1] )

layer_deg = dict()
layer_deg["actor"] = top_actors
for layer in ml.layers(net):
    layer_deg[layer] = ml.degree(net, actors = top_actors, layers
= [layer] )
```

```
df( layer_deg )
```

```
deg_dev = dict()
deg_dev["actor"] = top_actors
deg_dev["dd"] = ml.degree_deviation(net, actors = top_actors)

df( deg_dev )
```

Neigborhood centralities can be calculated. See the difference from degrees.

```
ml.degree(net, actors = ["U4"], layers = ["work", "lunch"])
ml.neighborhood(net, actors = ["U91"], layers = ["facebook",
"leisure"])
```

Neigborhood centrality exclusive to the queried layers can be calculated.

```
ml.xneighborhood(net, actors = ["U91"], layers = ["facebook",
"leisure"])
```

Relevance is the ratio between neighborhood centrality of the queried layers and all layers.

```
layer_rel = dict()
layer_rel["actor"] = top_actors
for layer in ml.layers(net):
    layer_rel[layer] = ml.relevance(net, actors = top_actors,
layers = [layer] )

df( layer_rel )
```

Multiplex networks are special cases of multilayer networks. Multilayer networks can be defined with different nodes at each layer. Pymnet libary can be used to visulaize and analyze multilayer networks.

```
#!git clone https://github.com/bolozna/Multilayer-networks-
library.git
#!pip install ./Multilayer-networks-library
import pymnet as pn
fig=pn.draw(pn.er(10,3*[0.4]),layout="spring", show=True)
fig.savefig("net.pdf")
```

```
gml = pn.er(10,3*[0.4])
#list(gml)
#list(gml.edges)
#list(gml.iter_layers())
#list(gml.iter_node_layers())
#list(gml[0,0])
#gml[0,0].deg()
#gml[0,0].str()
#pn.degs(gml)
#pn.density(gml)
#pn.cc_barrat(gml,(0,0))
#pn.multiplex_density(gml)
#pn.supra_adjacency_matrix(gml)
#help(pn.models)
#help(pn.netio)
```

References

1: Bianconi, Ginestra. Multilayer Networks. Available from: VitalSource Bookshelf, Oxford University Press Academic UK, 2018.

2: uunet Python library

3: Bródka, Piotr, et al. "Quantifying layer similarity in multiplex networks: a systematic study." Royal Society open science 5.8 (2018): 171747.