pymnet.draw

pymnet.**draw**(*net*, *layout='spring'*, *layershape='rectangle'*, *azim=- 51*, *elev=22*, *show=False*, *layergap=1.0*, *camera_dist=None*, *autoscale=True*, *backend='mpl'*, *figsize=None*, *nodeCoords={}*, *nodelayerCoords={}*, *layerPadding=0.05*, *alignedNodes=True*, *ax=None*, *layerColorDict={}*, *layerColorRule={}*, *defaultLayerColor='#29b7c1'*, *layerAlphaDict={}*, *layerAlphaRule={}*, *defaultLayerAlpha=0.75*, *layerLabelDict={}*, *layerLabelRule={'rule': 'name'}*, *defaultLayerLabel=None*, *layerLabelLocDict={}*, *layerLabelLocRule={}*, *defaultLayerLabelLoc=(1, 1)*, *layerLabelSizeDict={}*, *layerLabelSizeRule={}*, *defaultLayerLabelSize=None*, *layerLabelColorDict={}*, *layerLabelColorRule={}*, *defaultLayerLabelColor='k'*, *layerLabelStyleDict={}*, *layerLabelStyleRule={}*, *defaultLayerLabelStyle='normal'*, *layerLabelAlphaDict={}*, *layerLabelAlphaRule={}*, *defaultLayerLabelAlpha=1.0*, *layerOrderDict={}*, *layerOrderRule={'rule': 'name'}*, *defaultLayerOrder=0*, *nodeLabelDict={}*, *nodeLabelRule={'rule': 'nodename'}*, *defaultNodeLabel=None*, *nodeLabelSizeDict={}*, *nodeLabelSizeRule={}*, *defaultNodeLabelSize=None*, *nodeLabelColorDict={}*, *nodeLabelColorRule={}*, *defaultNodeLabelColor='k'*, *nodeLabelStyleDict={}*, *nodeLabelStyleRule={}*, *defaultNodeLabelStyle='normal'*, *nodeLabelAlphaDict={}*, *nodeLabelAlphaRule={}*, *defaultNodeLabelAlpha=1.0*, *nodeSizeDict={}*, *nodeSizeRule={'rule': 'scaled', 'scalecoeff': 0.2}*, *defaultNodeSize=None*, *nodeColorDict={}*, *nodeColorRule={}*, *defaultNodeColor='black'*, *edgeColorDict={}*, *edgeColorRule={}*, *defaultEdgeColor='gray'*, *edgeWidthDict={}*, *edgeWidthRule={}*, *defaultEdgeWidth=1.5*, *edgeAlphaDict={}*, *edgeAlphaRule={}*, *defaultEdgeAlpha=1*, *edgeZDict={}*, *edgeZRule={}*, *defaultEdgeZ=0*, *edgeStyleDict={}*, *edgeStyleRule={'inter': ':', 'intra': '-', 'rule': 'edgetype'}*, *defaultEdgeStyle='-'*)

Visualize a multilayer network.

Creates a 3D pictures of multilayer networks are drawn using Matplotlib. The network can be any type of multilayer network with a single aspect.

Parameters:

net : *MultilayerNetwork*

Network that is to be drawn

layout : *string*

Layout algorithm. Use “fr” for Fruchterman-Reingold layout. Options using networkx are “circular”,”shell”,”spring”, or “spectral”.

layershape : *string*

Shape of the layers. Options are “rectangle” or “circular”.

azim : *float*

Azimuth of the layers given in degrees.

elev : *float*

Elevation of the layers given in degrees.

show : *bool*

If true, the picture of the network is displayed using the default Matplotlib backend.

layergap : *float*

The gap between the layers. See also autoscale.

camera_dist : *float, None*

The distance of the camera to the layers. See also autoscale.

autoscale : *bool*

If true, the layergap and camera distance is scaled automatically such that the whole drawing fits the figure. This is done if the layergap times 3 is larger than 3.

backend : *string*

The backend for drawing the network. ‘mpl’ = Matplotlib

figsize : *tuple of integers, None*

The figsize argument is forwarded to pyplot.figure when a new figure is created.

alignedNodes : *bool, None*

Should each node have the same coordinate in each layer. If None, then True for multiplex networks and False for multilayer networks.

layerPadding : *float*

Space between nodes and the edge of each layer.

ax : *[axes object]*

The axes where the figure is drawn. (Only when Matplotlib is used for drawing. Axes need to be of Axes3D type.)

[property]Dict : *dict*

Dictionary giving each element a property value. Keys are

Notes

Setting properties

Various visible elements can be set values using a property setting scheme which is similar for all of the following properties: layer color, layer label, node labe, node size, node color, edge color, edge width, and edge style.

Each of each property has three parameters that can be used to set the values of the elements: [property]Dict, [property]Rule, and default[property]. (Here the word [property] is replaced by the property name.) Each of these parameters can give a way to set a value for property of an element, and the parameters are gone through in the order [property]Dict, [property]Rule, and default[property] until a property value is found.

The format for defining edges in property dictionaries is tuples with two node-layer names. For example, and edges between node 1 in layer ‘a’ and node 2 in layer ‘b’ is specified with tuple ((1,’a’),(2,’b’)).

All z-coordinate modifiers (e.g., edgeZ) must lie between 0 and 1.

Property rules

The [property]Rule parameter can be used to set property values by giving a rule for determining the property value. The rules can be generic or specific to the property type. For example, using node degree as a property value is specific to node properites such as node size or node color. Empty property dictionary means that there is no rule for setting the property, and a rule can be set by adding an item to the property rule dictionary with “rule” as a key and value correspoding to the specific rule.

Generic properties:

- “order” : Order at which the iterator for the object gives a value of the property. First object gets value 0.
- “name” : Object name is used as a value for the property

Node properties (node color, node label, node size):

- “degree” : Degree of the node.

Node label property:

- “nodename” : Name of the node (note that “name” returns the node-layer tuple).

Edge properties (edge color,edge width, edge style):

- “edgetype” : *Properties are given by edge type. You can include keys “intra” and/or “inter” in the property rule dictionary*

  to give values for intra-layer edges and inter-layer edges.

- “edgeweight” : Weight of the edge.

Property modifiers

Properties generated by rules can be modified before they are assigined as final properties of the elements. This is done by property modifiers and it is useful for example when converting numeric values to colors. Property modifiers can be stacked and they are evaluated in an order that is reverse to the order in which they are introduced next. Each property modifier is an item in the property rule dictionary.

Generic modifiers:

- “colormap” : Use a Matplotlib color map to map a number to a color. Value is the colormap name, e.g. “jet”.
- “scaleby” : Multiply the property values by a constant given by the value
- “f” : Any function take takes the value as an argument an returns the modified value

Node size modifiers:

- “propscale” : Multiply everytnig by a constant given as value and divide by the sqrt of the number of nodes in the net.