Mediaviz Documentation

Release 0.1

Tahsin Mayeesha

Aug 13, 2018

Contents:

CHAPTER 1

mediaviz package

1.1 Submodules

1.2 mediaviz.community_utils module

mediaviz.community_utils.**get_community_colormap**(*partition*)

Takes dict of partitions after community detection and returns dict with colormap for partitions.

The function is currently used with python-louvain package, but it's a general purpose function for randomly generating colorlaps from node attributes.Expected to change into get_random_colormap(node_attributes) in the next version.

Parameters **partition** (*dict*) – Dictionary of form {node1:partition,node2:partition....} generated after community detection.

Examples

```
>>> import community #python-louvain package
>>> import networkx as nx
>>> from mediaviz.community_utils import get_community_colormap
>>> G = nx.florentine_families_graph()
>>> partitions = community.best_partition(G)
>>> get_community_colormap(partitions)
{0: '#A0FD29', 1: '#C3E994', 2: '#61EDD2', 3: '#DCB0DF'}
```

Returns Dictionary containing partitions and colormaps.

Return type dict

mediaviz.community_utils.**get_community_graph**(*G*)

Takes a Graph, generates partitions from python-louvain package, sets the partitions as node attributes and returns the Graph and partitions.

Helper function to integrate community detection with the draw function. In the draw function color_by parameter takes a node attribute along with a colormap for the attributes. So here python-louvain is used for getting the community partitions first and then we set those partitions as node attributes.

Examples

```
>>> import networkx as nx
>>> G = nx.florentine_families_graph()
>>> from mediaviz.community_utils import get_community_graph
>>> G, partitions = get_community_graph(G)
```

Parameters **G** (*nx.Graph*) – A networkx graph.

Notes

If G is a digraph , as python-louvain package does not currently support digraphs, it's changed to the undirected graph of the largest weakly connected component.See source if necessary.

Returns Returns a tuple of form (G, partitions) where G has partitions set as node attributes and partitions are partitions generated from the python-louvain package.

Return type tuple

1.3 mediaviz.draw module

mediaviz.draw.**draw_forceatlas2_network**(*G*, *pos=None*, *fa2l_iterations=50*, *fa2l_scaling_ratio=38*, *scale='auto'*, *num_labels=None*, *node_list=None*, *node_color='red'*, *color_by=None*, *colormap=None*, *node_size=10*, *size_field=None*, *min_size=0.1*, *max_size=100*, *with_labels=False*, *label_field=None*, *filter_by=None*, *top=None*, *adjust_labels=True*, *node_opacity=1*, *edge_opacity=0.05*, *font_size=8*, *font_color='k'*, *font_family='sans-serif'*, *filename='untitled.png'*, *title=None*, *edge_color='lightgray'*, *edge_color_by_source=False*, *figsize=(10, 10)*, *fig_dpi=100*, ***kwargs*)

Main function for drawing graph.

The function has sensible defaults. If no scale is provided scale for the graph is set automatically. If no pos is given then force atlas 2 layout is used. Other parameters can be described below can be used to customize different aspects of the visualization.

See Notes section for dependencies.

Check README for examples.

Parameters

- **G** (*nx.Graph*) – A networkx graph.
- **pos** (*dict or None. optional. default None.*) – pos containing positions for the graph generated by some network layout algorithm.

If None, force_atlas2_layout function from fa2l package is used with default parameters to generate the layout for the graph. To use with user given parameters for the force_atlas2_layout, calculate pos outside draw function and then pass to the draw function.

Pos can also be calculated with any other network layout algorithms like nx.spring_layout(G) and passed to the draw function.

- **fa2l_iterations** (*int, optional. Default 50.*) – number of times to run the default force atlas 2 layout algorithm from fa2l package.
- **fa2l_scaling_ratio** (*float, optional. Default 38.*) – How much repulsion you want. More makes a more sparse graph.
- **scale** (*"auto" or float or int. optional. default "auto".*) – Used for scaling the graph automatically.Also used for expanding and contracting the graph.

  If "auto", then scale is set automatically. see : mediaviz.scaling.get_auto_scale(...) for how scale is set. If given a float or int, positions are scaled according to the number.

  For example, scale = 2 will expand the graph to twice of its current size while scale = 0.5 will contract the graph to half of its original size.

  When scaling graph node sizes are not changed, only the relative positions are changed.

  For example, after doubling a position (2,4) will become (4,8).
- **num_labels** (*int, optional. default None.*) – Number of nodes to label sorted by the node size.

  Recommended to use only with varying node sizes.

  If all node sizes are equal it will just return the nodes from G.nodes() in the same order.
- **node_list** (*list. optional, default None.*) – list of nodes to draw. if None and there's no filtering done all nodes are drawn.
- **node_color** (*str or list. default 'red'.*) – Multiple variations are allowed.

  node_color can be color name like "r","b" or hex code in string format.

  node_color can also be a list of colors like ["r","b"...] or ["#FFFFFF","#..."...].

  If node_colors are being passed then color_by and colormap should be None(their defaults)
- **color_by** (*node attribute. optional. Default None.*) – categorical node attribute to color the nodes by.

  If color by is given, a colormap dictionary must also be provided.

  Example : if "gender" in graph node attribute, then color_by = "gender" and colormap = {"M":'b',"F":'r'}
- **colormap** (*dict, optional. Default None.*) – dictionaries containing color assignment for each value of the categorical node attribute in color_by.

  Assumes color_by is a categorical node_attribute like "gender" or "partition" or "country".

  colormap is then a dictionary that assigns color for each unique value of that categorical node attribute.

  Example : If there's two unique values "M" and "F" in "gender" node attribute, then color_by = "gender" and colormap = {"M":'b',"F":'r'}.

- **node_size** (*int or float or list. Default 10.*) – Multiple values are supported.

  int or float value can be provided to make all node sizes same.

  list of node sizes can also be passed.

  if node_size is given, size_field should not be used. Default of size_field is None.

- **size_field** (*str, optional. Default None.*) – Node attribute to resize the nodes by. Must be numeric node attribute.

  If given node sizes are resized to [min_size, max_size] using set_node_size from utils module.

- **min_size** (*float, optional. Default 0.1*) – Minimum size for the nodes.

- **max_size** (*float , optional. Default : 100.*) – maximum size for the nodes

- **with_labels** (*bool, optional. Default False.*) – Whether to show the labels or not. If label_field is not provided node names are used.

- **filter_by** (*str, optional. Default None.*) – Numeric Node attribute to filter the graph by. If filter_by is given, top must be provided.

  Filter_by filters the graph to draw only the top k nodes.

  For example, if filter_by = "inlink_count" and top = 100, the graph is filtered to the subgraph containing the top 100 nodes sorted by inlink count and only the subgraph is drawn.

- **top** (*int, optional. Default None.*) – Number of largest nodes to keep after filtering using filter_by.

- **label_field** (*str, optional. Default None.*) – Node attribute to label the nodes by.

- **adjust_labels** (*bool, optional. Default True.*) – If True, Adjust Text package is used for adjusting the labels to prevent label overlap.

  Label texts are iteratively adjusted until there's no overlap.

  See : https://github.com/Phlya/adjustText for details.

- **node_opacity** (*float. Default 1*) – Node alpha.

- **edge_opacity** (*float.Default 0.05.*) – Edge alpha.

- **font_size** (*int, optional. Default 8.*) – Label font size.

- **font_color** (*str, optional. Default 'k'*) – Label font color.

- **font_family** (*str. optional. Default 'sans-serif'*) – Label font family.

- **filename** (*str, optional. Default "untitled.png"*) – File name to save the figure by. See fname from https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html

- **title** (*str, optional. Default None.*) – Plt title.

- **edge_color_by_source** (*bool, optional. Default False.*) – If true, edge colors are same as the source node for the edge. If edge_color is given, then edge_color_by_source should be set to default False.

- **figsize** (*tuple, optional. Default : (10,10)*) – figure size.

- **fig_dpi** (*float, optional. Default : 100.*) – See https://matplotlib.org/api/figure_api.html for details.

1.4 mediaviz.rotation module

mediaviz.rotation.**rotation_layout**(*pos*, *angle*, *origin=(0, 0)*, *unit='degree'*)

Rotates the pos to the given angle with respect to origin.

Parameters

- **pos** (*dict*) – A dictionary with nodes as keys and positions as values.
- **angle** (*float*) – angle in degree or radian
- **origin** (*tuple*) – the point will rotate with respect to the origin.
- **unit** (*str*) – 'degree'/'radian' to indicate if the angle is in degrees or radians.If given in degrees angle is converted to radians.

Returns Returns the pos dict with positions rotated by the given angle with respect to the given origin.

Return type dict

1.5 mediaviz.scaling module

mediaviz.scaling.**get_auto_scale**(*G*, *pos*, *node_sizes*, *k=20*)

Returns the ratio of the ideal vs current top node distance.

The scale is being set with a heuristic that we consider k largest nodes to prevent overlap, take the distances of all these nodes, find the current minimum top node distance. We also find the ideal minimum top node distance that will prevent overlap and return their ratio to scale the layout by that scale.

Parameters

- **G** (*nx.Graph*) – A networkx Graph
- **pos** (*dict*) – A dictionary containing the node positions
- **node_sizes** (*dict*) – A dictionary containing the node sizes.
- **k** (*int, default 20.*) – Number of nodes to consider when setting the scale automatically.

mediaviz.scaling.**scale_layout**(*pos*, *scale*)

Scales layout

Parameters

- **pos** (*dict*) – A dictionary containing the positions of the nodes.
- **scale** (*int or float.*) – Number to scale the positions by.

Returns Returns Dictionary containing Positions for the nodes scaled by the scale.

Return type dict

1.6 mediaviz.utils module

mediaviz.utils.**draw_networkx_nodes_custom**(*G*, *pos*, *node_size*, *node_color='r'*, *alpha=1*, *ax=None*, ***kwargs*)

Draws networkx graph nodes with circles instead of using plt.scatter function like draw_networkx_nodes

Parameters

- **G** (*nx.Graph*) – A networkx graph object.
- **pos** (*dict*) – A dictionary with nodes as keys and positions as values.
- **node_size** (*list*) – A list containing node sizes for each node.
- **node_color** (*str or list*) – A single color or list containing color values. Behavior same as draw_networkx_nodes
- **alpha** (*float*) – opacity (between 0-1) for setting transparency of the nodes
- **ax** (*axis*) –

mediaviz.utils.**edgecolor_by_source**(*G*, *node_colors*)

Returns a list of colors to set as edge colors based on the source node for each edge.

Parameters

- **G** (*graph.*) – A networkx graph.
- **node_colors** (*list*) – List of node colors.

Example

```
>>> colormap = {'male':'b','female':'r'}
>>> node_colors = set_node_color(G,"gender",colormap)
>>> edge_colors = edgecolor_by_source(G,node_colors)
```

Returns list of colors for each edge of the graph color set by the source node.

Return type list

mediaviz.utils.**filter_graph**(*G*, *filter_by=None*, *top=None*)

Filters the graph by returning the subgraph for the top x nodes for the given filter_by field.

To learn about subgraphs see : https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.Graph.subgraph.html

Parameters

- **G** (*graph.*) – A networkx graph.
- **filter_by** (*graph node attribute*) – Numerical node attribute to filter the graph.
- **top** (*int*) – Number of top nodes to return. E.g if the node attribute is node_size then the function returns top = 100 largest node subgraph.

Example

```
>>> filter_field = "inlink_count"
>>> filter_graph(G,filter_by=filter_field,top=k)
```

Returns Subgraph after filtering by the node attribute.

Return type nx.Graph

mediaviz.utils.**get_subgraph_pos**(*G*, *pos*)

Returns the filtered positions for subgraph G. If subgraph = original graph then pos will be returned.

Parameters

- **G** (*nx.Graph*) – A graph object.
- **Pos** (*dict*) – A dictionary with nodes as keys and positions as values.

Example

```
>>> pos  = nx.spring_layout(G)
>>> subgraph_nodes = ['1','2','3']
>>> subgraph = G.subgraph(subgraph_nodes)
>>> subgraph_positions = get_subgraph_pos(subgraph,pos)
```

Returns Assuming positions were generated earlier for a larger graph with some layout algorithm this functions returns the filtered positions by the subgraph.

Return type dict

mediaviz.utils.**set_node_color**(*G*, *color_by*, *colormap=None*)

Returns a list of colors for each node based on the provided colormap.

Parameters

- **G** (*graph.*) – A networkx graph.
- **color_by** (*str*) – Graph node attribute with finite discrete categorical values. See : https://networkx.github.io/documentation/networkx-1.10/tutorial/tutorial.html#adding-attributes-to-graphs-nodes-and-edges for learning about node attributes
- **colormap** – Dict mapping values of color_by attribute to different colors. colors can be a single color like 'r' or hex code of form '#FFFFFF'

Returns A list containing the colors for each node.

Return type list

Example

```
>>> colormap = {'male':'b','female':'r'}
>>> node_colors = set_node_color(G,"gender",colormap)
['b','b','r','b']
```

mediaviz.utils.**set_node_label**(*G*, *label_field*)

Returns a dict mapping nodes to the labels.

Helper function for use in visualization functions like nx.draw_networkx_labels.

Parameters

- **G** (*graph.*) – A networkx graph.
- **label_field** (*node attribute*) – Node attribute to be set as a the label for the node.

Example

```
>>> label_field = "label"
>>> labels = set_node_label(G,label_field)
```

Returns List of node labels

Return type list

mediaviz.utils.**set_node_size**(*G*, *size_field*, *min_size*, *max_size*)

Returns a list containing values of the size_field attribute from Graph range normalized between [min_size,max_size] interval.

Parameters

- **G** (*graph.*) – A networkx graph.
- **size_field** (*graph attribute*) – graph node attribute containing numbers(integers or floats). See : https://networkx.github.io/documentation/networkx-1.10/tutorial/tutorial.html#adding-attributes-to-graphs-nodes-and-edges for learning about graph attributes
- **min_size** (*float*) – minimum value for the new range
- **max_size** (*float*) – maximum value for the new range.

Notes

Mapping/scaling a number to a certain range like [0,1] or [-1,1]

y = (x - min(d))*(max(n)-min(n))

_____ **+ min(n)** (max(d)-min(d))

min(d) = minimum value in the data

max(d) = maximum value in the data

min(n) = minimum value in the new range

max(n) = maximum value in the new range input number : x , output : y

Example

>>>size_field = 'inlink_count' >>>node_sizes = set_node_size(G,size_field= "inlink_count",min_size = 0.1, max_size=800)

Returns list of node sizes normalized to the range.

Return type list

1.7 mediaviz.viz_parser module

mediaviz.viz_parser.**parse_colors**(*path*, *hex=False*)

Parses the color attribute from .gexf file viz tag and returns the dict with the node colors.

Default nx.read_gexf() does not read the attributes from the visualization namespace of .gexf files so this function is used to read the color codes.

To learn more see : https://gephi.org/gexf/format/viz.html

Parameters

- **path** (*str*) – File or filename of the graph in .gexf format.
- **hex** (*bool*) – whether to return the color codes in hex format. if False rgb color codes are returned.

Returns

Returns dict containing the color codes.

Example : {'1':'#FFFFFF'} or {'1':{'r':0,'g':0,'b':0}}

Return type dict

mediaviz.viz_parser.**parse_position**(*path*)

Parses the position attribute from .gexf file viz tag and returns the dict with the node sizes.

Default nx.read_gexf() does not read attributes from the visualization namespace of .gexf files so we use this function to get the node positions, possibly after using layout algorithms in Gephi.

To learn more see : https://gephi.org/gexf/format/viz.html

Parameters **path** (*str*) – File or filename of the graph in .gexf format.

Returns Returns dict containing the node positions. Example : {'1':(2.0,3.0)}

Return type dict

mediaviz.viz_parser.**parse_size**(*path*)

Parses the size attribute from .gexf file viz tag and returns the dict with the node sizes.

Default nx.read_gexf() does not read attributes from the visualization namespace of .gexf files so we use this function to get the node sizes.

To learn more see : https://gephi.org/gexf/format/viz.html

Parameters **path** (*str*) – File or filename of the graph in .gexf format.

Returns Returns dict containing the node sizes. Example : {'1':52.0}

Return type dict

1.8 Module contents

Python Module Index

m

Index