Data Visualization – Assignment 5

– Visual analysis of graphs –

P. Rüdiger, J. Lukasczyk · H. Leitte – winter term 2018/19

In the fifth assignment we will work on graph data and combine multiple libraries to obtain best results. We will look into graph handling in python, graph layout, and graph rendering/interaction.

In exercise 1, we will introduce the necessary libraries and procedures to work with graphs. In exercise 2, we will extend the existing rendering code to obtain custimzed designs for hierarchies. In exercise 3, we will use graph visualization techniques to analyze data.

Please hand in your solutions in the OLAT system. Submit the jupyter notebook (including the analysis task as text cells with markup) and a hmtl copy of the notebook.

Due date: Wednesday, 06. Feb. 2018, 22:00

Remark: The text on the assignment sheet and in the notebook are almost identical (some links are only provided in the notebook). We provide the text here for people who prefer a compact printout. If you want to work with the notebook only, you have the same amount of information (except for the credit points per exercise).

Exercise 1 – Getting started with graphs in bokeh — 6 points

Bokeh recently started to provide graph rendering support: `https://bokeh.pydata.org/en/latest/docs/user_guide/graph.html#`

This first exercise shall help you work with the required packages.

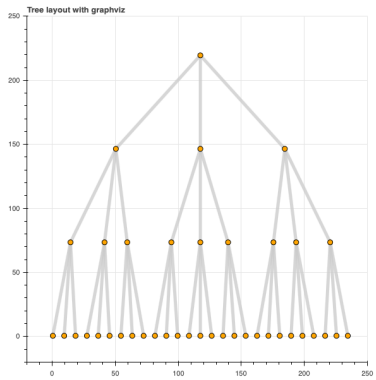
To support graph analysis, Bokeh has a networkx integration. Networkx `https://networkx.github.io` is a python package to work with complex networks (creation, manipulation, IO, analysis, layout). Networkx is very strong with algorithmic analysis of graphs, but provides only a few basic layouts that we will test first. In the next part we will extend the code and switch to the more powerful layouts from the GraphViz library `https://www.graphviz.org`, provided as python package pygraphviz `https://pygraphviz.github.io`.

Hints: see notebook

Task 1a) (2P): You are given a triary tree and bokeh code for rendering below. Render the tree using the networkx native layout algorithms. Check all algorithms in the list below (using ☒) that are helpful in analyzing the given tree. Give a short reason for your choice.

Task 1b) (3P): As you may have noticed, none of the algorithms provided a classical hierarchical tree layout. We will work towards this now and style the graph. We will now use layout algorithms from the graphviz library - a widely used graph drawing library with very good layout algorithms - probided through pygraphviz `https://pygraphviz.github.io`.

Your goal is to obtain the following layout for the triary tree:



The code below demonstrates how to use a graphviz layout in bokeh [1] (using networkx [2] and its access to pygraphviz [3]). To obtain the desired layout, update the following three parameters:

- **Choose the correct layout**: The prog parameter specifies the layout routine. Select a fitting layout.
- **Distances settings**: The current layout has a poor aspect ratio. Adjust the distance settings in the args to improve aspect ratio and make the tree fit in the given x-/y-range. Documentation: `https://graphviz.gitlab.io/_pages/pdf/dot.1.pdf`
- **Change style**: The glyphs for the nodes and edges are defined in the bokeh part by means of renderers. See the examples from bokeh [1] and change the shape of the glyph (draw larger circles in a new color, e.g., orange) as well as the color and line style of the edges.

Exercise 2 – Visualizing hierarchies

8 points

In this exercise, we will extend the bokeh code to render hierarchical graphs with directed, curved edges and labels as depicted in Figure 1.

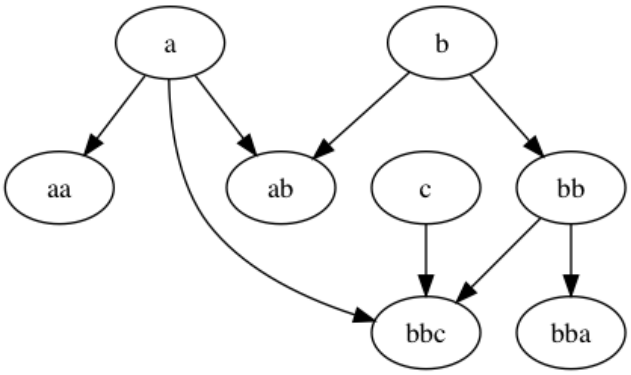


Figure 1: Hierarchy G rendered using dot command line tool. Command line code to recreate the image (you have to have graphviz installed): `dot -Tpdf tree.gv -o tree.pdf`

Below you are given starter code which does the following things: - Compute the dot layout for a given graph. - Extract layout information for nodes from the dot algorithm and provide it to bokeh (see more details below). - Render edges as B-splines.

The function `directedGraphLayout( G )` returns a `GraphRenderer` object for which you have access to the following information:

- Nodes:
  - width and height (`graph_renderer.node_renderer.data_source`)
  - label and position. Iterate over nodes and their positions using the following loop
    `for label,pos in graph.layout_provider.graph_layout.items():`
- Edges:
  - Sampled B-spline positions (xs and ys in `graph_renderer.edge_renderer.data_source`) Iterate over the edges polylines using:
    `for x,y in zip(graph.edge_renderer.data_source.data['xs'], graph.edge_renderer.data_source.data['ys']):`
  - The last two vertices are the start and end of the arrow tip still to be drawn.

Task: Edit the rendering code to make G look as similar as possible to the output from the command line dot algorithm in figure 1 (see notebook).

ToDo:

- Adjust the size and shape of the glyphs (use Ellipse glyph to have correct width/height values).
- Render the text using Bokeh's labels (change font, font size, position).

- Add tips to the edges (you'll have to use annotations, lines with arrows do not work yet).

Hints: You can print the graphs and their information using the following code

- `print(G)` for networkx graph objects
- `graph.pprint()` for GraphRenderer

Exercise 3 – Analyzing graphs

18 points

In the material folder you'll find three graphs that you shall render using an appropriate technique and answer questions relating to the data.

Task 3a) (9P): The file `lesmis.gml` contains a coappearance network of characters in the novel Les Miserables. Display the graph using a node-link-diagram of your choice.

Answer the following questions:

- (3P) Graph type (+implementation): Which type of graph do you have and what is your chosen layout?
- (3P) A community in a graph is a set of nodes that is densely connected. See the example below where gray circles enclose communities: Use the routine greedy_modularity_communities(G) from networkx to extract communities and color the nodes of each community in the same color. Now answer the following questions:
  - Name the color of a community that matches your expected structure of a community.
  - Name the color of a community that you consider not optimal. Explain problems of this community.
- (3P) Unusual people:
  - Find three people that should be members of multiple communities. State their names and list the communities they should belong to.
  - Find a person that cooccurs with many people that do not occur with anyone else.
  - Can you find a person that is always alone on stage? What would their nodes look like?

Data courtesy: D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, Reading, MA (1993).
image: By j_ham3 - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=17125894`

Task 3b) (9P): The file `programmingLanguages.gml` contains information on the genealogy of some of the more influential or widely used programming languages as discussed by (Scott, 2000). Edges indicate principal influences on design. Many influences, of course, cannot be shown in a single figure.

The graph also contains the date for each language which indicates the approximate time at which its features became widely known. This can be used for rendering but cannot be realized with the algorithm we have used so far. It is enough to show dependencies here.

Render the graph using an appropriate layout and provide a method to color a list or set of nodes.

Now answer the following questions:

- (2P) Initial languages: Which languages were not influenced by others? How can they be identified?
- (2P) Long history: Which language has the longest history, i.e., the longest path to a upper leaf? What is the path length? Solve this problem algorithmically using an appropriate routine from networkx and color the respective nodes.
- (1P) Most influenced: Which language has the largest number of direct influences? How many?
- (2P) Java influences: List all languages that influenced Java (directly and indirectly). Solve this question algorithmically using networkx and color the nodes.

- (2P) Showing time: Each language also contains a date. How could this be incorporated into the visualization? Can you base the new design on the given one? If no, say why. If yes, reason how difficult the adaption would be and where you would have to be careful.

Scott, Michael Lee. Programming language pragmatics. Morgan Kaufmann, 2000.