COMP5048 Week 6 Tutorial

**1. D3 examples**

Some of the examples need to be run on a server due to the need to load other files from the filesystem. The instructions below has been copied from the Week 3 tutorial:

1. Extract comp5048-wk6.zip to a folder.
2. Open the command line and navigate to the destination of extraction.
3. Run *C:\Python27\python.exe -m SimpleHTTPServer 8888* from the command line.
4. Run python -V from and note the first digit of the version number returned.
5. If the first digit is 2, run python -m SimpleHTTPServer 8888 from the command line.
6. If the first digit is 3, run python -m http.server 8888 from the command line.
7. The example web pages can now be accessed through localhost; that is, localhost:8888
8. To stop the server, press ctrl+c on the command line.

**1.2 Force Layout example**

The following examples has been modified from *http://bl.ocks.org/mbostock/4062045*.

1. Examine the source for force linkscale.html. Open in a browser to check the visualisation.
2. var force is the variable storing the force layout function d3.layout.force. The following functions allows modification of the parameters of the layout:
   * *size*: Layout size, default [1,1] ([width, height]).
   * *charge*: Charge value, default -30. Negative values denote repulsion (graph layouts mostlyuse this), while positive values denote attraction.
   * *linkStrength*: Link strength in the range [0, 1], default 1. Can be a constant or a function.
   * *linkDistance*: Target distance between nodes, default 20. Can be a constant or a function.
   * *gravity*: Gravitational strength of the layout’s centre, default 0.1. This prevents disconnectednodes from escaping the layout, which may happen if gravity is set to 0.
   * *friction*: Friction coefficient which decays node velocities, default 0.9. 0 freezes all items inplace, while 1 means no friction. Any of these values can be edited after the force variable is declared, however for the changes to take effect the start function must be called afterwards, e.g. *force.charge(-20).start()*.
3. var *widthscale* is an example of a scale, which is a function that maps values from a domain to values from a range. This particular scale is a linear one. Here, we only set the range first, which is the desired range of link widths of the graph.
4. var *color* is another type of scale. This particular scale, *d3.scale.category20()* maps values to a range of 20 colour values.
5. Examine the contents of miserables.json. The object contains two properties: "nodes", a list of vertices in the graph, and "links", a list of edges.
6. The *force.nodes()* and *force.links()* function assigns the graph to the force layout, however at this point no layout is calculated yet. The layout calculation is started by calling *force.start()*.
7. The domain of widthscale is defined by taking the extent of values (weights) associated with the links. *d3.extent* takes as an argument a list of values and returns an array of size 2 containing the smallest and largest items in the list respectively. Here, an optional accessor function is specified - this can be used when the array contains objects and we want to find the extent of a certain property of the objects (in this case value).
8. The links and nodes are drawn in a similar way to the tree examples, with small differences. Here, the widths of the links depend on the value associated with the edge, and node colours depend on the group recorded (*color(d.group)*). (*node.call(force.drag)*) initialises the dragging function, which allows the nodes to be dragged by holding down the mouse.
9. *force.on("tick", ...)* defines the action taken for every tick (iteration) of the force layoutcalculation. Here, the co-ordinates of the links and nodes are adjusted after every iteration, which will eventually settle as the calculation converges.
10. Try changing the parameters of the function described in step 2 and reload to see the effects on the layout.
11. *force\_linkcolours.html* is a variation where the colour of the links varies by the edge weight.

**1.3 Additional Example: Reading Multiple Files**

The resulting layout from *force\_csv.html* is the same as *force\_linkscale.html*, but it reads from two CSV files instead of one JSON file.

* 1. This example uses the *queue.js* module, which is the second included script under the *D3.js* import.
  2. *queue()* enables asynchronous functions to be queued up using the defer() function, and acallback defined using await() will only be executed once all deferred functions have finished. Each *defer()* call takes as the first argument the function to be called, with subsequent arguments being the arguments of said function. Here, (*d3.csv*) is called to read CSV files, with the first argument being the filename and the second being an accessor that reformats each row of the input, in this case to enforce numeric type on certain fields using the + operator.
  3. The argument passed to the callback function defined using *await()* consists of an error, if it occurred, followed by one argument each for the return values of the deferred functions, in the order they are deferred. Alternatively, *awaitAll()* may be used instead, which always gives the callback two arguments: the error, and an array storing the results of the deferred functions.

1. **iGraph and NetworkD3**

The iGraph is a collection network analysis tools with the emphasis on efficiency, portability and ease of use. Igraph is an open source and free. Igraph can be programmed in R, Python and C/C++.

In this tutorial, we will use R.

The NetworkD3 creates D3 JavaScript Network graphs from R.

To download and install the software:

1. Go to the R then RStudio websites at <https://www.r-project.org/> and <https://www.rstudio.com/> and go to the download page.
2. Select the appropriate package for your system from both and download. Then run the installers or uncompress the packages.

This section of the tutorial will give a few examples to show how to use the software.

Runthrough iGraph examples

1. Open RStudio
2. Make sure to install igraph and networkD3 packages before using it, go to console and type:

install.packages(“igraph”)

1. To use the packages, go to console and type:

library(igraph)

1. To create a graph with Zachary karate club dataset:

g = graph(“Zachary”)

1. To display the edges, vertices and the number of vertices and edges of the graph:

E(g)

V(g)

summary(g)

1. To generate coreness clustering of the graph (this prints out the centrality value of each vertex in the same order as the vertices are printed out with V(g)):

coreness(g, mode=”all”)

1. To generate degree centrality of the graph (similar output format as #6):

degree(g, mode=”all”)

1. To generate Fruchterman-Reingold layout of the graph (this prints out the 2D coordinates of each vertex in the same order as the vertices are printed out with V(g)):

layout\_with\_fr(g)

1. To plot a graph (the drawing will be displayed in a sidebar):

plot(g, layout= layout\_with\_fr(g))

1. Save the image using the Export button

Runthrough network3D examples

1. Make sure to install networkD3 packages before using it, go to console and type:

install.packages(“networkD3”)

1. To use the packages, go to console and type:

library(networkD3)

1. To create a force network graph with Les Mis dataset (uses same force layout function as in D3 example):

data(MisLinks)

data(MisNodes)

forceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",

Target = "target", Value = "value", NodeID = "name",

Nodesize="size", Group = "group", opacity = 0.8, zoom = TRUE, legend = TRUE)

For more examples, you can open week6\_tutorial.Rmd in Rstudio, which also contains all the above igraph & NetworkD3 examples.

References:

http://igraph.org/r/

https://cran.r-project.org/web/packages/networkD3/networkD3.pdf