COMP5048 Week 3 Extra Tutorial

1. Size scales

1. Examine the source code for *noscale.html*. Note the fruits variable - we will create bars for each entry with the bar length varying according to the counts.

2. The bars are given the class "bar" to differentiate from the rectangle that makes the SVG area border (classed "border"), and to select them we use the dot prefix (*svg.selectAll(".bar")*). Once the data is assigned to the selection and rect objects appended, the "bar" class is assigned (*.attr("class", "bar")*).

3. The width assigned to the bars are simply the value of the "count" property of each entry (*.attr("width", function (d) {return d.count;})*).

4. Open the page in a browser. You will see that the bars go over the borders of the SVG area.

5. To make sure the bars fit completely inside the SVG area, we will scale the widths to occupy only the available area. Examine the source code of scale.html, which is similar to noscale.html up to the declaration of the border.

6. To scale the widths of the bars, we first need to know the maximum of the counts of the objects in the fruits array. This can be done using d3.max. The first argument is an array, while the (optional) second argument is an accessor, denoting how the members of the array should be compared - in this case, we compare the items using their "count" values (*function(d) {return d.count;}*).

7. The scaling is done using a scale (d3.scale). There are various types of scales provided by D3, including linear, square root, exponential, and power scales - in this case, we will use the linear scale (*var width scale = d3.scale.linear()*).

8. To use a scale, the domain (range of input values) and range (range of output values) must be defined. In our case, the domain is between 0 and the maximum count, while the range is between 0 and the width of the area. These are set using the domain() and range() values, which both take as an argument an array of length 2, containing the minimum and maximum values of the domain/range respectively.

9. Scales are functions, and they are called by passing in a value within the specified domain and returns that value mapped to a value in the specified range. In this case, we pass the count property of the entries of fruits and it returns a value shorter than the width of the SVG area, which we then use to specify the bar lengths (*.attr("width", function (d) {return width scale(d.count);})*).

10. Open scale.html in a browser. You will see now that the bars are contained fully inside the SVG area, while preserving the proper ratio of lengths depending on each entries' count property.

For the homework, you can use a scale to limit the possible sizes of the nodes or links (e.g. restrict node diameter between 0 and 15).

2. Colour scales

This example takes the same data array as the previous example, but in addition to the bar width, the colour is varied according to the count value as well.

1. Examine the source code for colourscale.html. Notice the additional variable colour scale - this is a scale similar to width scale, but the range is set between 0 and 255. This is the range of values each component of the RGB (red green blue) colour model can take.

2. In this example, we will make higher colour value correspond to higher blue values. RGB colour values are expressed using strings in the format RGB(r,g,b), where r, g, and b are

integer values between 0 and 255 denoting the value of each colour component. We use colour scale to calculate the value of the blue component depending on the count value (*return "rgb(0,0," + Math.round(colour scale(d.count)) + ")"*. Math.round is used as the RGB component values have to be integers).

3. colourscale hsl.html is a variation which uses the HSL (hue saturation lightness) colour model instead of RGB. Here, we vary the hue according to the count value, and hue scale takes as a range a subset of the full hue range (0 to 360), in this case from green to blue. Meanwhile, saturation and lightness takes values between 0-100%.

For the homework, try playing around with the range of colour values and/or changing different components (e.g. modify red instead of blue) if using colour to encode node degree or link weight/value.

3. Opacity

1. Open scale.html again. Here, all the bars have an opacity of 1, i.e. fully solid for both its fill and stroke (border). Opacity can take values between 0 and 1, where lower opacity values make the objects more "transparent".

2. To change the overall opacity of the item, change the opacity attribute: *bars.attr("opacity", 0.3)*. Restore it back to 1.

3. To change only the opacity of the fill, use fill-opacity: *bars.attr("fill-opacity", 0.3)*.

4. To change only the opacity of the stroke, use stroke-opacity: *bars.attr("stroke-opacity", 0.3)*. This is also how you change the opacity of lines.