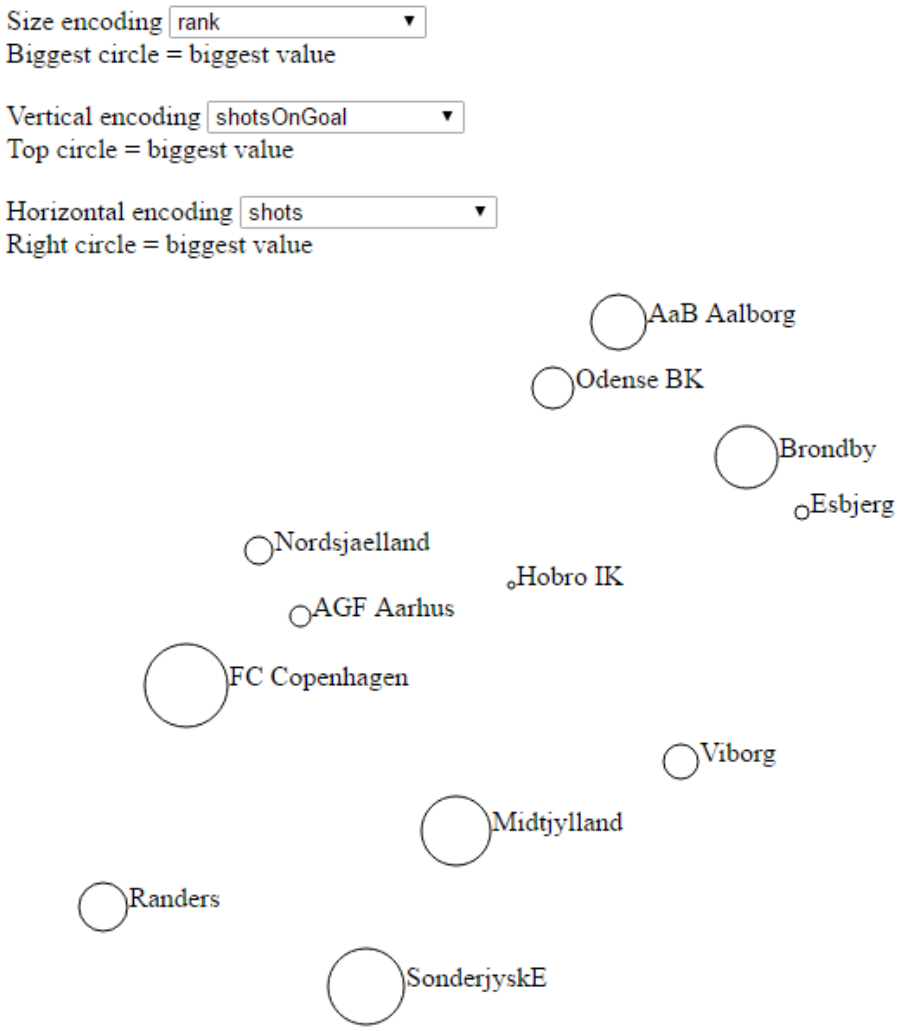

Figure 1: Visualization of statistics of teams from the current season. Encodes rank, shots and shots on goal

0.0.1 What-why-how

What. The data being presented in the visualization is the statistics of the teams from the current season. The includes parameters such as their current ranking in the league and the total shots on goal, number of yellow cards etc.

Why. As the user is able to choose what is encoded in the different dimensions, the user can easily test hypotheses about correlations between different parameters, for example how rank, shots and shots on goal affect eachother. The user can also discover new correlations by simply browsing through the different parameters and trying different combinations. The visualization offers a "fun" platform where the user can play with the different parameters. The transition when the user has changed the options is animated such that the user can easily follow the circles with their eyes, and in that way see how a change in an option changes the order between the circles.

How. In the beginning of the iterative design process, the visualization started out by only having a vertical axis and size encoding. This provided meaningful information to the user,

but based on feedback another axis was added such that three different parameters could be encoded in the graph. However, as the last axis was added, it turns out that the bubble chart had been reinvented. The only minor difference between this visualization idiom and the traditional bubble chart is that this visualization allows the user to interact with the visualization and change the parameters when they wish to. The time complexity of this specific implementation is O(nlgn) where n is the amount of different parameters available to be chosen. The reason being that when a user has selected the options, the coordinates and sizes of the circles are calculated in the following way: First they are sorted based on the first parameter, and then the radius of the circle is assigned in the order that they are now in. Then they are sorted based on the second parameter, and the y coordinate is assigned based on the radius and the order they are in (we have to base it on the radius as well such that there is equal distance between them and no overlaps). Finally they are sorted based on the third parameter, and the x coordinate is assigned. Since we sort 3 times, the complexity is going to be O(nlgn), assuming that we can sort in O(nlgn) time. While there might be smarter approaches than this very straightforward approach, it is not of any concern at the moment, as the different number of parameters a user can choose in football data is very limited. The smart thing about the implementation is that it is very flexible. All of the data that the user is being presented with is based on the data that is in the data file that the algorithm imports. The data file that is supplied should be encoded in JSON as a list of objects, where all of the objects have the same attributes, and all of the values are numerical except for the "name" attribute which is excluded from the drop down list, and is used as a label on the circles. This means that the implementation can be used to visualize many different things. It automatically extracts the attributes that must go in the drop down list, and assigns the names to the circles. The mark chosen in this visualization is a point, and the visual channel chosen to encode data into the mark is size and position. Color could have been chosen aswell, and a fourth parameter could be encoded within that, but for football data purposes, 3 dimensions suffices. The data that the visualization uses is preproccessed in R and exported in the format that was described above.

0.0.2 Code

This is the pseudocode for the way that the x, y and size dimensions are calculated for each circle. Since we receive the data as a list of objects, we can append/change the x, y and radius of each circle directly on the object as each circle represents an object. The updateDimensions function takes as arguments the list of objects, the parameters that it should encode the circle by and the distance between each circle. Note that when calculating the y coordinate of the circles, we sort the objects in reverse order as coordinates in graphical spaces usually have the origin in the upper left corner, meaning that it is counter intuitive for users compared to how they normally view graphs, meaning that we have to inverse the y-coordinates as to not make it confusing. After the dimensions have been updated, the circles would transition into their new dimensions (i.e. grow or shrink and move along the axes).

```
1 updateDimensions(objects, sizePar, xPar, yPar, distance) {
2   sortByParameter(objects, sizePar)
3
4   currentRadius = 1
5
6   for(object in objects)
7     object.radius = currentRadius
```

```
    currentRadius++

  sortByParameterReverse(objects, yPar)

  prevRadius = 0
  prevY = 0

  for(object in objects)
    object.y = prevY + prevRadius + distance + object.radius
    prevY = object.y
    prevRadius = object.radius

  sortByParameter(objects, xPar)

  prevRadius = 0
  prevX = 0

  for(object in objects)
    object.x = prevX + prevRadius + distance + object.radius
    prevX = object.x
    prevRadius = object.radius
}
```