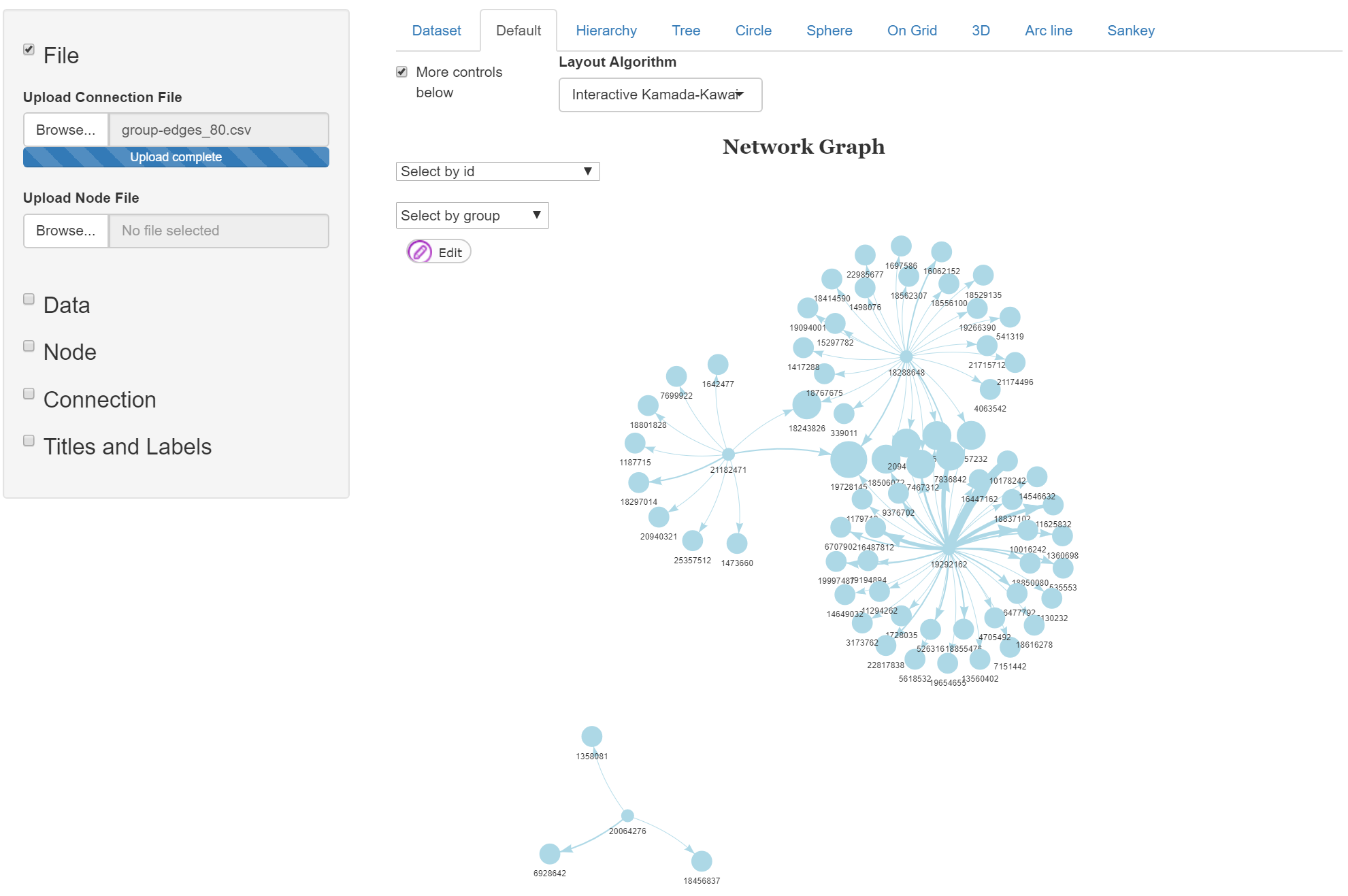
**What do we mean by “Network Data”?**

Network data is a representation of the relationships between entities or individuals, which can be symmetric relations or asymmetric relations.

**What do we want to accomplish?**

The subject of this project is the visualization of network data. Data visualization is used to encode the data as a different graph, which can be excellent tool to discover and communicate with big data. An appropriate graph can often help the users understand the data easily. We want an interface that enable visualization of network data for users who are new to such data as quickly and easily as possible. Some key ideas in network will be introduced in this report.

It is about art and statistics. Colours, transparency, size helps the trends, patterns, or outliers to be detected intuitively. This project aims to develop an interface displaying a set of network data by three different diagrams: node-connection graph, arc diagram, and Sankey diagram. A control panel is provided for users to change the appearances of the diagram. Both diagrams are interactive so that users can play around the diagram and know more about the data. Hopefully, those diagrams can inspire users some ideas which will help them to conduct a network analysis.



The control on the left in this interface is limited because our major consideration is to get users to learn network fast. We do not want to offer a lot of user control so that the interface becomes overwhelmingly complicated. The control there are the most common features that users are interested in. For example, the appearance of the nodes and connections and the controls for clusters. We can see there is a control in the graph panel called “More control below”, which displays whole set of controls available in visNetwork package. It is helpful when users want more advanced options for the graph. Different layouts are available and they will be displayed in different tabs, which give users a instant access to different visualization types.

**Some key ideas in networks**

**Type of network graph layout**

**Useful R packages**

**NetworkD3**

NetworkD3 is a R package that helps to visualise network data, using javascript library.

NetworkD3 is 0-based. Node data should contain at least one column called ID, including the ID of node, and ID should be integer starting from 0. To be consistent, the source and destination node will be represented by ID numbers in the connection data. Unlike, only column name “I”""“s” required for node-set, no name requirement for other columns and connection set. On the other hand, all attributes have to be predefined in the function, and no small widgets are showing up with the graph, so users cannot change any details on the graph on the interface.

Four layouts are available in networkD3. There are force-directed networks with forceNetwork, Sankey graph with sankeyNetwork, radial graph with radialNetwork and dendro graph with dendroNetwork.

One special feature for is the Sankey graph layout. provides the easiest way to produce a Sankey graph. Sankey graph is a special type of flow graph, representing flow direction from left to right and the flow quantity. It is a weighted graph. Each flow quantity is represented by the width of the connection, which is directly proportional to the total flow quantity. Sankey graph puts an emphasis on the major flows, and it is easier to detect the primary source, which contributes most to the destination by using the Sankey graph. Also, the breakdown of contribution from different sources for one destination is easy to be compared.

The node is displayed by a rectangle with the node label. The size of the rectangles located on the left column is represented how much the flow coming out from the node and the size of the rectangles located on the right column is represented how much the flow coming into the node. Since nodes are located on columns, the Sankey graph might not be suitable for large datasets. If a large dataset is plotted, then we might not only see the node label, but not the rectangle and the width of connection will all be the same, and it will be skinny. The suggested threshold will be 200 or 300 connections.

There are two structures for the Sankey graph: one is called evolution, and the other is called source to end. Evolution means only two columns will be displayed. The left column represents the source, and the right column represents the destination. Unlike other layouts, the node can be duplicated here. A node can be on both sides. To create this structure, more work has to be done on node data. A full list of the nodes will be copied so that one node will have two entries with the same label but different id so that the Sankey graph algorithm will treat these two entries to be two nodes. For the connection data, the IDs for destination have to change to the ID of the second entries of that node in the node-set. The source will stay the same. It is used when users want to explore the direct flow of each node. We can see which flow is heaviest. The major contributor of each node and the major receiver are easy to identity.

Source to end means there are not only two columns, but more columns are also possible. It shows a series of flow, from sources to one or multiple intermediate and finally to the destinations. The node should be unique in the dataset. We do not do any data manipulation. We need to convert data ID to be 0-based and then apply the Sankey graph algorithm. The algorithm will optimize the number of columns to the output and optimize the position of each node. In this way, it is clear to see the contributors and the receivers in one node. Also, we can explore which nodes are involved in a flow chain, and it is possible to track back to all sources and intermediates from a destination. Source to end structure conveys more information of a dataset than the evolution structure.

When hovering the nodes, all connections to that node will be highlighted. When hovering the connections, the information of the source and the destination and the flow unit will be displayed. Users cannot interact too much with the Sankey graph. They can only move the position of a node vertically. The default data is ungrouped. Therefore each rectangle is filled with one colour.

**Layout Algorithms**

**Force-directed algorithm**

The force-directed layout algorithm places nodes based on the force between any two nodes. It draws graphs in an aesthetically pleasing way by following two principals: connections are tried to have equal length and reducing crossing edges as much as possible. Also, there are two principles when developing a graph layout algorithm: nodes connected by a connection should be close together, and nodes should not be drawn too close to each other.

Force-directed layout simulates a graph as a physical system which assigns forces into edges and nodes and try to find an equilibrium so that every node is settled, and a graph is generated. There are two electrical concepts to be used in the algorithm, attractive force, and repulsive forces. Attractive forces measure how much a pair of endpoints of a connection attract each other, which is used to decide the length of a connection. Repulsive forces are used to disperse all pairs of nodes. The direction of the attractive force and repulsive force are inversed. If the distance between two nodes decreases towards zero, then the repulsive force increases infinitely. If the distance between two connected nodes increases, then the attractive force between two connected nodes increases.

The algorithm first calculates the attractive force and repulsive force of every node and then sum the forces of each node, respectively. The final force is now calculated. The algorithm moves the nodes by a constant step size to minimize the energy system and finally reach an equilibrium.

This algorithm results in a high-quality layout when the data has around 50 to 500 vertices. The position of nodes is best so that they are not overlapped with each other. The distribution of nodes tends to symmetric. The graph gets smaller, then the degree of connection gets higher because a higher degree causes shorter distance when equilibrium is reached. It is flexible to use in a 2-dimensional graph or 3-dimensional graph. The algorithm is easy to understand since not much knowledge about graph theory is required when users go through the algorithm, but it is suggested. Therefore, the graph is easy to understand, and the outliers, the clusters are easy to interpret. It becomes to be the default layout in most of the graphics tools because it is the right choice when users have no idea of the underlying structure of the data before they plot the data. Then they can expect to see some information from the general layout generated by the force-directed algorithm.

However, it might take a long time to generate the graph. In each iteration, it is expensive to calculate the repulsive forces because all nodes have to be in consideration. Therefore, when a graph is relatively large, a force-directed algorithm results in high running time to reach the equilibrium. If a constant step width is defined when the nodes are moving to try to minimize the energy system, equilibrium might not be reached. The Fruchterman-Reingold Algorithm is a force-directed layout algorithm. In the Fruchterman Reingold model, a concept of temperature is introduced, which is a dynamic step width. The temperature is high at the very first iteration, which means the nodes will move a greater distance in a step, and then the temperature cools down at each iteration, which means the distance for a step decrease. The process stops until the nodes are static. The algorithm assumes the shape of nodes is a dot, a one-dimension shape. When the algorithm is applied to two-dimensional shaped nodes, it cannot make sure that the nodes will not overlap, and it just prevents the centre of each node will not overlap.

**Multidimensional Scaling**

Multidimensional scaling layout is a way to visualize the dataset with a large dimension by re-projecting the nodes to a lower dimension, usually two or three. In multivariate statistics, if we have a dataset with many variables and we detect some variables are related to each other, we might want to explore the data to see if there are some principal components can explain most of the variance in the data. Then we can reduce the large dimension to a small one, which only includes the principal components and do the data analysis easily. An analogy to this, multidimensional scaling does similarly in data visualization. When there are lots of nodes and connections in the network data, then we might want to reduce the dimension of distances between nodes to make some clusters. These clusters are similar to the principal components. We try to reduce the dimensions while still preserving as much information as possible. If we look at the data by table or distance matrix we get from the data, and we cannot detect the underlying structure of the data. Therefore, we can try this layout to pull out a set of relevant information. The distances here represent the length of the connections.

The original coordinates of nodes are thrown away, and the distance relationship is used to try to preserve the distance. Then to find the new coordinates of the nodes, multidimensional scaling tries to minimize the squared error between Euclidean distances of the connections. Then the nodes are re-projected into the two-dimensional plot. In the graph, we might not see each connection clearly because the connections in a cluster will have a relatively short length, and some nodes or labels will be overlapped with each other if the lengths of connections are extremely short. Nodes in one cluster indicate those nodes have some common features. Different groups and the distance between them can be used to interpret the dissimilarity.

An interesting application for multidimensional scaling is to organize the buttons in an interface. In human computation interaction, for user interface design, if there are lots of buttons in a dashboard, the designers can layout the button base on how similarity of the buttons to do a task and then apply multidimensional scaling to compute the coordinate of buttons. In this way, buttons that usually work together can locate close to each other.

**Shiny App layout and controls**

The app is designed for visualizing network data based on the shiny app. There are two sections in this app. A set of appearance control locates on the left and the graph will be displayed in the main panel. There are nine layouts provided and most of the dataset can be represented well by one of these layouts. Two sections are separated clearly so that the user can modify the options on the left and see how they affect the graph on the right. Nine layouts are designed to be in one row. Therefore, users can try different layouts quickly. By using this interface, users can custom the appearance of the graph and find a suitable layout to explore the information containing in the data from the graph structure.

There are five submenus in the side panel: File, Data, Node, Connection and Titles and Labels. When ticking those buttons, a few controls corresponding to the submenus will uncollapse. On the other hand, unticking the buttons will collapse the controls. User can import the edge dataset and node dataset from their local drive under File. Text and CSV file is acceptable. Data can be removed. The controls under Data help us to select fewer rows from the original data and display it. If “value” column is in the edge data, users can select the rows with how large the value they want to up to. The unit is a percentage and the default is 5%, which means the default setting will sort the data by value first and then select the top 5% value in the data. On the other hand, if the “value” column is missing, then the selection is based on the degree of each node. Here, the concept of degree is worth to introduce. Degree is frequently used in graph theory. Degree is the number of edges having a node as the endpoint. For example, if we have a connecting to b and b connecting to c, then the degrees of a, b, c are 2, 1, 0 respectively. Using degree as criterion allows us to build the graph with the most important nodes so that we might catch importance message from the data.

The controls under Node and Connection are for manipulating the appearance of the graphs. All control work on default, hierarchy, tree, circle, sphere, on-grid graph and some of them work on 3D, arc line. Node colour works across the layouts except Sankey graph and it provides a colour palette for picking a colour. If users want to classify groups by different colours, then users can choose a variable containing group information. Then the option of colour palette will hide and if colour by variable is none, colour palette turns up. Same as shape and shape by variable. There are 12 shapes: dot, ellipse, box, circle, database, diamond, square, star, text, triangle, triangle Down, icon. Cluster is an important feature in the graph, and we do not want to miss out. Cluster shows the internal structure of data. The default colour is blue, and the default shape is a dot. Node transparency ranges from 0 to 1 with 0 indicating non-transparent and 1 indicating transparent. The shadow of a node is on by default and the transparency of shadow can be adjusted. When hovering the nodes, all edges connecting to that node will be highlighted because the default setting is 1. User can custom the number of degrees to be displayed. Furthermore, the transparency of the non-highlighted node can be adjusted. In terms of connection, the arrow points to the destination by default and the options that point to source node or both directions are available. The colour palette for connection colour works across the layouts except for Sankey graph. The connections can be styled to dash line and they can be set to fully invisible.

Titles, subtitles and footers are available except arc graph and Sankey graph. Nodes are named by labels and colour and size can be modified.

Default Layout

In this panel, the graph is generated using visNetwork package. The checkbox called more controls below provides all available options that manipulate the graph in visNetwork. The options are located under the graph. They are advanced options that help to polish lots of details in the graph. There are four main parts in the advanced options: nodes, edges, layout, interaction, manipulation and physics. At the end of those options, users can click on the button called “generate options”, which generates a short JAVASCRIPT code that put the new settings into a list. This code can be used in JAVASCRIPT and reproduce the same effect on the graph. The options inside a menu like a shortcut. All of them are the most common setting that users want to see.

However, since it contains a full list of controls and it locates under the graph, when we make some change in the controls, we cannot see them as the controls are too far from the graph. This is also why we give some common settings beside the graph panel. We can see the change in graph directly and we do not need to scroll back to the graph. That full list of controls includes many details that might not be easily understood by users who are new to the graph theory but it is a friendly tool for people who have advanced knowledge.

There are 10 layout algorithms available for the graph: Interactive Kamada-Kawai used in visNetwork, Fruchterman-Reingold layout, Davidson-Harel layout, DrL graph, GEM layout, stable Kamada-Kawai algorithm used in igraph, Multidimensional scaling, Sugiyama graph, graphopt, Large graph. Different layout algorithms may convey different pieces of information so it is worth to try different algorithms. The default layout algorithm in is the Kamada-Kawai layout. It tries to optimise the speed of displaying the graph and make the graph symmetric.

A network graph will be displayed with two drop-down menus, one for highlighting the node with all other nodes and connections connected to it and the other for highlighting the group if groups are specified. When users click the node, its label turns up in the box of the first drop-down menu. On the other hand, uses can select which node they want to highlight by choosing the label of a node. If groups are classified, then uses can highlight all the nodes belonging to that group using the second drop-down menu. A small widget below the drop-down menus can edit nodes and connections. There is a default title called network graph.

The graph can be zoomed in or zoomed out. Users can drag the graph everywhere inside the graph area. The effect of clicking or hovering the node are the same. They both make the label of highlighted node bold and highlight the neighbours in term of how many degree users want. Nodes can change their positions but they have to correspond to the endpoints of their connection. This layout is the most interactive layout among the app.

Hierarchy Layout

Advanced options are available for hierarchy layout. The default direction is up-town where the roots are located on top and children are below the roots. Zooming and highlighting work the same as the default layout. The whole graph can be dragged everywhere. Each layer can only be dragged horizontally. The direction can be changed from the advanced options. The root can be put on top, down, left or right. If there are too many layers, the space of layers can be customed. The space between children can be customed as well. Hierarchy layout does not suitable for every dataset.

Tree Layout

Advanced options are available for tree graph. Users can explore the graph by hovering the nodes or clicking the nodes. Dragging one node cannot move the whole graph as what uses can do in the first two layouts. It is worth to mention that the tree graph here is not designed for the data containing a cycle. If the graph contains a cycle, then users will get an error message. Tree layout does not fit in every dataset.

Circle Layout

Circle layout provides a tidy graph. The default graph is a big circle. All nodes form a circle and all connections are displayed within the circle. Nodes are draggable but then the circle will be destroyed. When dragging one node, the other nodes will keep stable and the connections are flexible to move. The order of nodes is based on the order in the dataset. Advanced options and two drops down menus for highlighting are available. Nodes and connections can be added or deleted by the small widget under the drop-down menus.

Sphere Layout

Sphere layout is similar to circle layout but the nodes do not form a circle. Nodes spread out to be a cloud shape.

On-Grid Layout

Nodes are located on the grid and the order of nodes is the same as the dataset.

3D

The 3D graph is created using package. The manipulation of the 3D graph is limited in this shiny app. Five layout algorithms are available in three dimensions: random, On-Grid, Multidimensional Scaling, Fruchterman-Reingold, Kamada-Kawai, Drl Graph. The default algorithm is random, which uses layout\_randomly function. It may not be a popular algorithm because it randomly places a node in a square shape when the graph is designed on two dimensions. If the graph is designed in three dimensions, then the graph will be like a cube shape.

The graph can be zoomed in and zoomed out and the colour of nodes and connections can be changed. The whole graph cannot be moved but then it can be rotated. The node cannot be highlighted. When hovering a node, its label shows in the title. The default title is 3D Network.

Arc Graph

The arc graph is generated by JavaScript because JavaScript is more flexible to generate arc graph and the arc graph are more interactive comparing to the one generated by some R packages. The controls are limited. Users can custom the colour of nodes and connections, also the transparency of labels can be modified. When there are many nodes, and the labels are not separated clearly, users can make the labels fully transparent. When they hover a node, the node label will turn up. When a user hovers a connection, the connection and its corresponding endpoints will be highlighted. The labels of two points will show up as well.

All nodes are located in the middle of the graph area along a line horizontally. The connections are arc instead of a straight line as the previous graph. Each arc is a semicircle. The arc graph can show the direction of a connection. If the arc is above the line, then the source is the left endpoint and the destination is the right endpoints. If the arc is under the line, then the source is the right endpoint and the destination id the left endpoint. The graph can be weighted or unweighted.

Sankey Graph

Sankey graph displays the flow from the left column to the right column with the width proportional to the weight of the edge. “sankeyNetwork” function in package is used to create the graph. Interaction in this graph is limited, we can just move the nodes up and down and highlight the connections When a connection is highlighted, a piece of information tells users who are the source and the destination and the connection weight.

**Implementation**

**Data Input**

A dataset with connection information is required. Node data is optional, but it is suggested to have it as extra information, which helps to improve data visualization.

**Connection Data**

Two columns called “from” and “to” are compulsory in the connection data. The first column “from” records the name or id number of the source node, while the second column “to” records name or id of the destination node. With these two columns, this shiny app will generate an unweighted graph. If users want to generate a weighted graph, then a column called “value” should be included. If node data is not provided, then the information recorded in from and to the column will be treated as the labels of nodes. If node data is uploaded, then the source and destination information have to both match id or label column in the node dataset. For example, connection file records source id and destination label are not acceptable. One piece of information will generate one connection in the graph.

Here is the connection set example.

**Node Data**

Two columns called “id” and “label” are compulsory in the node data. The first column “id” is usually a sequence number starting from 1. This column has to be unique. Second column “label” is the name of the node. Each pair of id and label should be unique. Since node data is optional, when it is absent, this shiny app will generate node data internally based on the connection data. Moreover, extra information can be included such as node shape, node group and node size and the column name should be “shape”, “group” and “size” respectively. In the initial graph, nodes are represented by dot shapes and sizes of in-degree.

Here is the node dataset example.

**Dealing with Issues of Speed**

If the network is large, then it takes long time to generate the graph. To improve the performance of the shiny app, a few things have been adjusted.

In terms of visNetwork package, when generating the default, hierarchy, tree, circle, sphere, and on-grid graph, the stabilization control is disabled. Therefore, we can see the graph animations- the process of how vis.js computes coordinates for nodes fast and dynamically. If the stabilization is on, we will not see the graph display until the coordinates for each node are finally computed.

It is suggested that to disable the smooth curve for connections. However, we keep the curve in the default setting because curves are more aesthetically appealing. Moreover, the performance can be significantly improved by using igraph layout. Since igraph layout is less interactive than the default layout in visNetwork, we want to first display our perfect interactive graph.

To improve the performance of our shiny app, we limit the number of connections to be displayed. If the connections of whole network are less or equal to 80, then the whole network will be displayed. If the network is large, only partial network will be displayed by default. If the weight of connections is provided, then the network graph will display the connections with top 50% of weight. When weight is not provided, then only the nodes with higher out-degree will be displayed. Users can control how large the network to be displayed.

**Future Work**

In our shiny app, users can only select the node colours and the connections colours for 3D graph, and not too many interaction controls can be applied to them. The advantage of 3D graph is that they can be rotated. However, the position of the graph is fixed, and the labels are not well displayed beside the corresponding nodes. The layout algorithms for generating a 3D graph is limited. More R packages will be explored in the future to improve the interactivity of the 3D graph and we hope our 3D graphs can be as flexible as other 2D graphs.

Some graph sections will be added to the shiny app, for more different types of the network graph, for example, overlaying the networks on geographic maps and some statistics charts. There are many R packages such as maps and geosphere that provide different kinds of geographic map that can be used as background. Overlaying the network in a map allow us to visualize the flow of the world more straightforward, which will be a great help for social network analysis. The choices of map will be flexible, world map and country map with state border and more maps with different scale are provided. Then nodes with different shapes and colours and connections with different width and colours can be put in the map and the positions of nodes will exactly match the region they represent.

Ggraph might be introduced since this package offers more interesting visualization types to represent network, such as hive plots and heat maps.

Currently the data input is assumed to be a collection of relationships between two entities at a single time point. In the future, we will allow the data input recording the small to medium size networks over time. We will deal with different data structure to decide whether it is at a single time point or over time. We can try to explore package networkDynamic and ndtv that have capabilities to work with the network animations. The format can be discrete or continuous. If it is discrete, that means we will generate multiple static network at different time points and the animations just show different networks at different time. If it is continuous, then timpestamp is set for nodes and connections. When the network animation are done, it will show how nodes and connections move over time.

More R packages are helpful for network analysis, for example: sna for sociometric analysis of networks and network for constructing network objects that can be applied to other network analysis package. We want to add a section to do some network modelling and inferences, which can improve the capabilities of our shiny app. It can explore the data and visualize how data structure is and then the network model is applied to the data so that we can get some evidence to suggest the hypothesis made on the data. Statistics charts allow to visualize the network modelling results and help us to choose a better fit. More mathematics methods for network analysis will be added to this section.