Visualising the *Marvel Universe*

Ryan G

April 8, 2020

Contents

Introduction

The Marvel Universe

Marvel comics began it's life as *Timely Comics* in 1939 and eventually became *Atlas* Comics in 1951, this period was known as the *Golden Age* of comics [8] and it was during this time that core characters such as *Captain America* [20] were introduced.

During The Second World War the audience for comic book characters expanded, but following the war superheroes fell from vogue [3] and were often criticised by familial and political groups [12] alike.

In 1961 the publisher took on the name *Marvel* and introduced new types of characters through the work of Stan Lee [12], these characters included *The Fantastic Four* which were characters intended to be very human and flawed [16] , *Spiderman*, authored to appeal to issues faced by teenagers such as

self-doubt [9] and the *X-men* which came broadly to embody an opposition against oppressive forces [18].

These more human characters lead to a resurgence of comic book fiction generally [17] and provides a broad network of interconnected characters in a set commonly referred to as *The Marvel Universe*.

The Data Set

In 2002 R. Alberich, J. Miro-Julia and F. Rossello investigated the relationship between the characters in the Marvel Universe and compared this relational data to real-world social networks [2], the data set used in this research has since been made available through *Kaggle*. [7].

The data set used in the study is derived from the *Marvel Chronology Project* [6] which aims to "*catalog every **actual** appearance by every significant character in the Marvel Universe*". Although this database has had 10 years to develop since the original work, the original database will be used for want of comparison to the original work. It should also be noted that the *Marvel Chronology Project* has no direct affiliation with *Marvel Entertainment Group*.

Collaboration networks are networks in which vertices represent people and the edges that link these individuals represent interaction [4].

The study of social networks can be very useful in that the information can be used to formulate models to describe the network and predict how it might evolve through time [1]. These models can be very important to may fields, one such field is epidemiology wherein the behaviour of a social netwok can be used to forecast the behaviour of disease spread over regions and through time [15].

Literature Review

It can be challenging to develop understandable and concise ways to present data from network analysis [5]. Many tools exist to plot and model network graphs in general scripting languages (e.g. *Python*, ***R***, Julia etc.) such as *NetworkX*, *igraph*, *graph-tool* and python-graph [26].

The chosen library for producing the data for graphs in this investigation is igraph because of it's close integration with ***R*** [14] and a combination of ggplot and plotly because the are *Free Open-Source* tools that have been widely adopted in the data science and machine learning fields.

Compression Techniques

Although networks may generally be shown as mathematical graphs there is a challenge in displaying so much data simultaneously, research has been undertaken in order to simplify visualisation of large network graphs [13] such as exploiting the redundancy in graph topology (i.e. squishing nodes together while preserving the general shape) [19] as shown in figure 1 published by Yifan Hu and Lei Shi. [13]

Other such compression techniques include linking ideas by words, known as *PhraseNet* [22] or replacing common patterns of nodes with a symbol, known as *motif simplification* [11].

The principle idea of these methods is to try and convey as much information as possible without that information being lost to noise, an example of information being lost in noise can be shown by figure 2 which is a visualization produced through research into reciprocal altruism by Gerrit Sander van Doorn and Michael Taborsky [10].

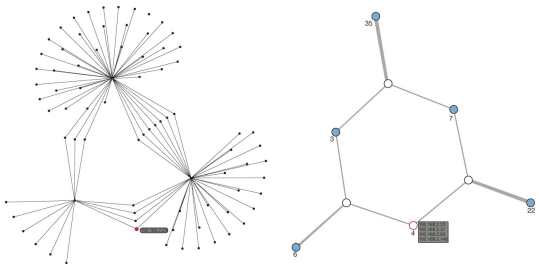

Figure 1: Lossless Compression of a Network graph

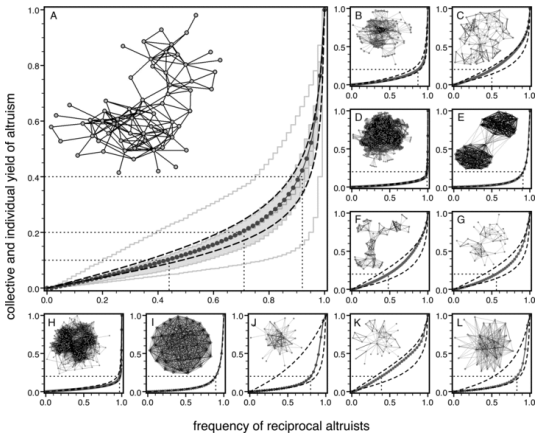

Figure 2: A graph illustrating the frequency of reciprocal altruists

1. Disadvantages to Compression Techniques

   These techniques unfortunately result in a simplification of information that is generally unacceptable for this type of data, the only practical way to compress character interaction without loss of information is to aggregate characters into logical clusters, for example instead of *Professor X*, *Shadowcat* and *Wolverine*, aggregating the characters into a single vertex as *The X-Men*.

   The disadvantages with this type of abstraction is that the process requires a significant amount of human intervention and that the fundamental shape of the graph may change in response.

2. Chosen Method of Compression

   Rather than compress the data, the method chosen to visualise the data will be to:

   (a) subset the data into a more reasonable size based on how interactive that character is
   - Although information is lost in doing this, characters that don't interact are not relevant to the visualisation and so any such removal may not be altogether unduly significant.

   (b) Make the visualisations entirely interactive
   - This will mean that relevant areas of the network can be examined in great detail despite many connected edges.

Interactive Visualisation Techniques

In there textbook, *Interactive Data Visualization*, Ward, Grinstein and Keim provide the following techniques for interactively visualising hierarchical data: [23, p. 15]

- Tree Graphs
  - Nodes and edges connected in a descending fashion to represent hierarchical interconnections.
    - The Advantage to a treemap is that they are simple but they don't show proportional weights between edges very effectively
- Treemaps
  - A subdivided region further subdivided to represent nodes and edges wherein the area of any subdivision is made proportional to the weight of an edge (in this case the weight will be the number of character cross overs and the nodes will be the characters)
    - Treemaps are advantagous to a generic tree because the structure is very consistent, they can however be difficult to interpret and become very cluttered towards the centre of the graphic.
- Sunburst Charts
  - Much like a treemap but the subdivided regions are mapped around the circumference of a circle in a manner that moves away from the centre, unlike the treemap which moves toward the centre.
    - The advantage to this over a treemap is that data moves away and can become less cluttered, however, it can also become very large which can be a disadvantage.
- Cone Tree
  - a hierarchical tree graph in visualsed in 3-dimensions where the 3rd dimsion is conformed to an equal distance from the centre and hence traces the surface of a cone.
    - $(\frac{x^2+y^2}{\frac{r^2}{h^2}} = (z - z_0)^2)$

For non-hierarchical but still relational data, their recommendation is to use force-directed graphing methods which means that a node-vertex graph is drawn and the distance between points is determined via an algorithm relating the weights of the edges to the force applied to a spring.

Robert Spence, in his textbook, *Information Visualization*, recommends aggregating data using *Cluster Maps* and mapping the interaction of people around a circular layout. [21]

1. Benefits and Disadvantages of Visualisation Techniques

   The advantages to a cone tree is that it will allow a for a 3D visualisation of the hierarchical relationship between the characters.

   Unfourtunately tree graphs are not built in to *Plotly* or *ggplot* and so the overheads of mapping nodes around the surface of a cone are simply outside the scope of this investigation and will not be implemented.

   Circular layout graphs (also k can be very good for spotting overwhelming large interactions between characters, the disadvantage however is that for very large character counts the circle can become very large. Due to the large number of characters in this data set the circular layout for a graph was not chosen, instead a combination of an automatic layout detection and the *Kamada-Kawai* algorithm were utilised.

   Interactive treemaps, graphs, sunburst diagrams and hierarchical trees will be useful techniques to visualise these relations and so they will be implemented as the chosen methods for investigation.

Visualisation

Visualisation Method

The chosen visualisation method's have previously been discussed but it is necessary to clarify that these visualisations are interactive and that the accompanying images are merely illustrative and should be viewed in that light. The true visualisations (submitted as ***R*** source code and a *HTML* file) should be referred to in order to actually understand and interpret the data.

Inspect the data

The data pertaining to the relationships is provided in two formats, a list of characters and comic books in which they appear as well as a list of 574, 467 edges, this can be viewed by taking a random snapshot of the data as shown in listing 1 and tables 1 2.

```
h_net <- read.csv(file = "./hero-network.csv")
h_net[sample(1:5000, 4, 0),]; print(paste(nrow(h_net), "Observations"))
(h_com <- read.csv(file = "./edges.csv")) %>% head()
h_com[sample(1:5000, 4, 0),]; print(paste(nrow(h_com), "Observations"))
```

Listing 1: Code to Output samples rows of data

Table 1: Snapshot of Character Appearances

Hero	**Comic**
BANNER, BETTY ROSS T	H2 467
ATALANTA	H2 425
BAAL	ROA 1
BARON STRUCKER/WOLFG	HMAG 23/2

Table 2: List of Character Interactions

Hero	**Comic**
ANGEL/WARREN KENNETH	UX 146
ANT-MAN II/SCOTT HAR	A 275
BANSHEE/SEAN CASSIDY	GENX '97
BANNER, BETTY ROSS T	H2 251

Data Cleaning

Before anything can be done with the data, it is necessary to clean it, the weight ascribed to an edge will become the number of interactions between any two characters, this is because it represents a measure of the strength of the relationship between two characters.

The easiest way to acheive this is to create an adjacency matrix where the value of each cell is the weight of the edge and then melt that adjacency matrix into a long-format table as shown in listing 2, the data will then be of the form as shown in table 3

```r
#+begin_src r
# Create Adjacency Matrix ----------------------------------------------
make_adj <- function(mat){
ig_adj <- get.adjacency(graph.edgelist(as.matrix(mat), directed=FALSE))
as.matrix(ig_adj)
}

# Plot Matrix ----------------------------------------------------------

# Rotate matrix 90-degrees clockwise.
rotate <- function(x) {
    t(apply(x, 2, rev))
}

# create the image
plot_mat <- function(mat){
image(rotate(mat), col = c(3, 6), axes = FALSE, frame.plot = TRUE, main = "Values > 10")
}

data_adj <- make_adj(h_net)
if(!((colnames(data_adj)==rownames(data_adj)) %>% mean())) print("The Adjacency Matrix is not symmetrical about i=j")
mean_social <- apply(data_adj, 1, mean)
data <- melt(data_adj)
data <- cbind(data, mean_social)
data <- data[data$value > 0,]
data <- data[order(-data$mean_social),] # This is the wrong order, I should use mean level
names(data) <- c("Source", "Target", "Value", "Mean_Social")
data_all <- data
head(data_all)
```

Listing 2: Coercing the Data in order to get a long-table with weights.

Table 3: Output Data set after y

Row	Source	Target	Value	Mean Social
12913	CAPTAIN AMERICA	BLACK PANTHER/T'CHAL	131	2.548864
25765	CAPTAIN AMERICA	FORTUNE, DOMINIC	1	2.548864
38617	CAPTAIN AMERICA	IRON MAN/TONY STARK	446	2.548864
45043	CAPTAIN AMERICA	IRON MAN IV/JAMES R.	37	2.548864
57895	CAPTAIN AMERICA	CARNIVORE/COUNT ANDR	3	2.548864
64321	CAPTAIN AMERICA	GHOST	1	2.548864

1. Sample the Data

   One of the issues with this data set however is that there is simply too much data to visualise effecively (or frankly to compute in any reasonable time frame in a scripting language) and so instead of using aggregation or abstraction the data will be subset, in order to keep the interactive element of the mouse-hover name display an element of the visualisation, in conjunction with re-sampling this could be an effective strategy to visualise the data-set while still preserving the ability to preview character names. The method used to subset the data is shown in listing 3.

```
dsamp_social <- function(n, seed) {
index <- sample(1:nrow(data), size = n, replace = FALSE)
data_all[index,]
}

data <- dsamp_social(1500, 309320932932)
```

Listing 3: Subset the data by sampling random observations

2. Relational Investigation

   Primarily this dataset is relational without direction or heirarchy and so the most appropriate method to visualise it would be with a graph depicting the inter-relation of characters via edges and vertices.

3. 2D Network Graph

   A 2D Network Graph describing the inter-relations of vertices and edges can be built using the `igraph` library as shown in listing 4, producing the data output beneath the listing.

```
Edges <- data
G <- igraph::graph_from_data_frame(d = Edges, directed = FALSE)
   laytg <- igraph::layout.auto(G)
   colnames(laytg) <- c("xval", "yval")
   laytg <- as_tibble(laytg)
   laytg$node <- vertex_attr(G)[[1]]
   laytg %>% head()
```

Listing 4: Use `igraph` to produce Cartesian Co-Ordinates for a graph

```
## # A tibble: 6 x 3
##     xval   yval node
##    <dbl>  <dbl> <chr>
## 1   14.2 -150.  ZABU
## 2  -65.0   21.5 HULK/DR. ROBERT BRUC
## 3   35.7  103.  WHITE QUEEN/EMMA FRO
## 4   42.0 -309.  MR. TERMINEUS/AMODEU
```

```
## 5 -122.   101.  PEREZ, GEORGE
## 6  -39.5 -305.  BROTHERS GRIMM II/BA
```

With the vertices described it is necessary to build a data frame describing the lines by creating a data frame of the starting and Ending points of edges as shown in listing 5.

```
ne <- nrow(Edges)
   ys <- xe <- ye <- xs <- vector(length = ne)
   for (i in seq_len(ne)) {
     xs[i] <- laytg$xval[laytg$node==Edges$Source[i]]
     ys[i] <- laytg$yval[laytg$node==Edges$Source[i]]
     xe[i] <- laytg$xval[laytg$node==Edges$Target[i]]
     ye[i] <- laytg$yval[laytg$node==Edges$Target[i]]
   }

   EdgeNames <- paste0("From:", Edges$Source, "To:", Edges$Target)
   starts    <- data.frame("xval" = xs, "yval" = ys, edgeid =
    ↪  EdgeNames) # TODO Make a factor
   ends      <- data.frame("xval" = xe, "yval" = ye, edgeid = EdgeNames)
   Edges_val <- as_tibble(rbind(starts,ends))
   Edges_val %>% tail()
```

Listing 5: Use a `for` loop to build a data frame of edges.

Using `ggplot2`, as shown in listing 6, the vertices can be plotted as points and the edges as lines, it is important to specify that the `group` the lines belong to are the names of the edges (`edgeid` values) because otherwise all the lines will be connected, this may be acceptable for a graph where all vertices are connected by edges, but for this data it is quite possible that two characters will never interact (for example the *Avengers* might not interact with the *X-men* or the Gods of the Aesir and Vanir might not interact with the *Fantastic Four* etc.), such a graph is known as a ***disconnected*** *graph*. [24]

Wrapping the call to `ggplot` in `plot_ly::ggplotly()` will pass instructions to the plotly graphing library to produce an interactive *Plotly* graph based on the `ggplot` object, an illustration of the output is shown in figures 3 and 4.

```
p <-  ggplot(rbind(starts,ends), aes(x = xval, y = yval)) +
      geom_line(aes(group = edgeid)) +
      geom_point(data = laytg, aes(x = xval, y = yval, col = node),
       ↪  size = 2) +
      theme_classic()
 ggplotly(p)
```

Listing 6: Use `ggplot` to build a network graph

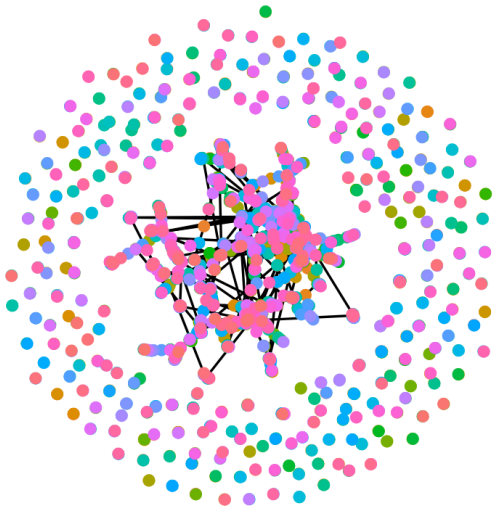

Figure 3: 2D Network Graph of Marvel Interactions

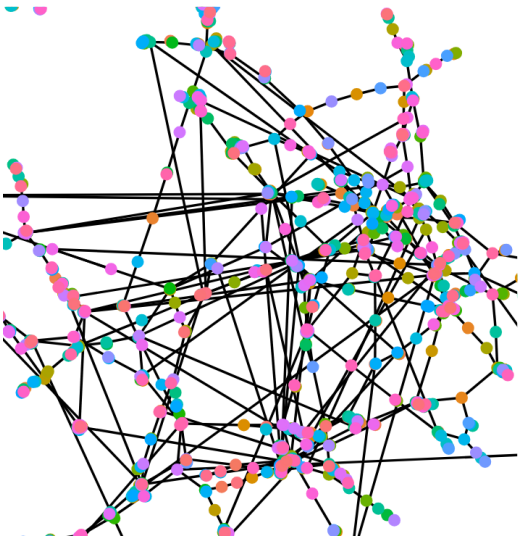

Figure 4: 2D Network Graph of Marvel Interactions, with a closer perspective.

Unfourtunately this method produces a lot of islands and the interactions are difficult to understand because of the edge overlaps, so instead of taking a sample, it would be better instead to choose from the characters that have the highest absolute cross overs (i.e. the weight of the edge) as shown in listing 7.

This visualisation though does suggest that either observations are connected or they are only connected to a very small number of individuals, so perhaps if more data was visualised most characters would connect.

```
data_all <- data_all[order(-data_all$Value),] # This is the wrong
  ↪  order, I should use mean level
data <- data_all[1:300,]
```

Listing 7: Sample the Data with the largest weights

The graph can then be re-created as shown in Listing 8, the ouput of which is illustrated in 5.

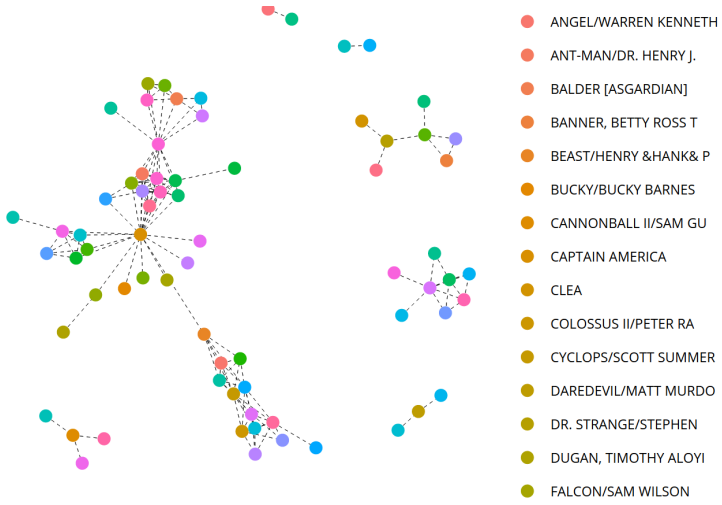

Figure 5: 2D Network Graph of Marvel Interactions, top 300 most interactive characters

Although this method of sampling the data is less of a fair sample (biasing selection for characters that often interact such as the *X-Men*, *Fantastic Four* and *Avengers*) it does provide for an effective way to observe the most frequent form of interaction between any two characters as opposed to trying to see every interaction between all characters through a random sample; for this reason this form of sampling is more appropriate for making a visualisation of the relationships between characters and will be used henceforth.

This visualisation clearly show of the most interactive characters, that there are four distinct clusters, the smallest containing *Hulk* and *Doctor Strange* (characters I wouldn't have thought to interact), *Spiderman* and the largest centre island being a cluster of the Avengers on the left connected by *Beast* to the *X-Men*. This Shows that those characters are not just interconnected in there own wolds, but that it doesn't take many characters before those worlds become connected and interconnected.

```r
Edges <- data
gg_network_graph <- function() {
G <- igraph::graph_from_data_frame(d = Edges, directed = FALSE)
   laytg <- igraph::layout.auto(G)
   colnames(laytg) <- c("xval", "yval")
   laytg <- as_tibble(laytg)
   laytg$node <- vertex_attr(G)[[1]]
   laytg %>% head()


ne <- nrow(Edges)
   ys <- xe <- ye <- xs <- vector(length = ne)
   for (i in seq_len(ne)) {
     xs[i] <- laytg$xval[laytg$node==Edges$Source[i]]
     ys[i] <- laytg$yval[laytg$node==Edges$Source[i]]
     xe[i] <- laytg$xval[laytg$node==Edges$Target[i]]
     ye[i] <- laytg$yval[laytg$node==Edges$Target[i]]
   }

   EdgeNames <- paste0("From:", Edges$Source, "To:", Edges$Target)
   starts    <- data.frame("xval" = xs, "yval" = ys, edgeid =
    ↪  EdgeNames) # TODO Make a factor
   ends      <- data.frame("xval" = xe, "yval" = ye, edgeid = EdgeNames)
   Edges_val <- as_tibble(rbind(starts,ends))
   # Edges_val %>% tail()


  p <-  ggplot(rbind(starts,ends), aes(x = xval, y = yval)) +
        geom_line(aes(group = edgeid), size = 0.1, lty = 3) +
        geom_point(data = laytg, aes(x = xval, y = yval, col = node),
        ↪  size = 2.5) +
        labs(x = "", y = "") +
        theme_classic() +
        theme(axis.line = element_blank(),  #
        ↪  https://stackoverflow.com/a/6542792/12843551
              axis.text.y=element_blank(),axis.ticks=element_blank(),
              axis.text.x=element_blank())
  return(p)
}

ggplotly(gg_network_graph())
```

Listing 8: Code to produce 2D Network Graph and plot with ggplot2

(a) Visualisation Techniques

For this plot 300 edges were chosen because the interactive nature of the plot is such that the overlap of edges can be made arbitrarily small by zooming sufficiently far in (bearing in mind that this is an interactive visualisation), given that there are distinct islands this is necessary in order to visualise the data and hence more data points may as well be visualised. It was necessary to balance the need to zoom in with the size of the vertices/points because the points will respond to a mouse hover by returning the name of the node.

Genereally only a small amount of colours can be used to distinguish variables in a plot, however in this case the variables are going to be considered relative to other local vertices and so multiple colours can be an effective way to distinguish nodes in relative proximity to one another.

Although the legend is quite long and not useful on it's own, in conjunction with the capacity to adjust the range of the plot by zooming and the fact that the legend can be scrolled it becomes an effective way to compare multiple plots that are in close proximity to each other as a reference second to the mouse-hover dialog.

(b) Very Large Sample

A much larger sample can be used, as shown in listing 9, in order to get an understanding of how often the separate disconnected 'islands' of characters might become connected by another character taken into the snapshot as, this visualisation is illustrated by figures 6, and .

```
1 Edges <- data_all[1:999,]
2 ggplotly(gg_network_graph())
```

Listing 9: Visualise a large number of characters

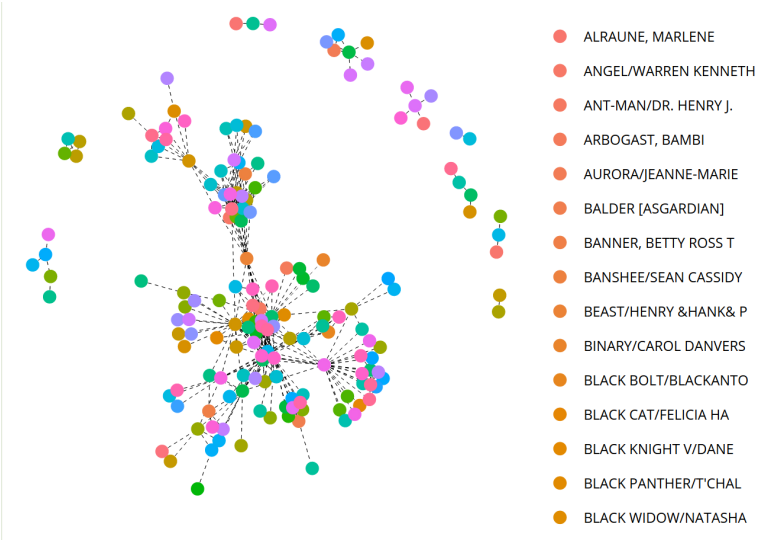

Figure 6: Overview of Large 2D Network Graph

This larger graph shows that as more characters have been introduced *Spiderman* has become cluster connected to the *Avengers* by *Captain America* and *Iron Man* while *Hulk* has been

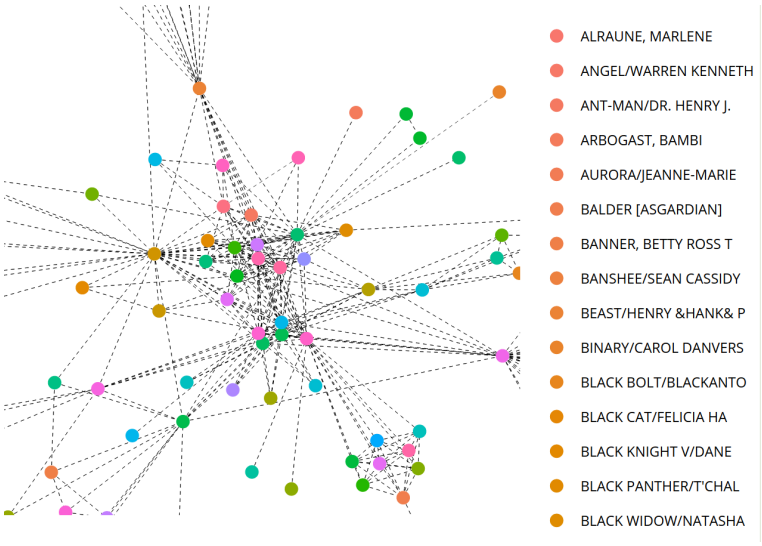

Figure 7: Closer view of Large 2D Network Graph

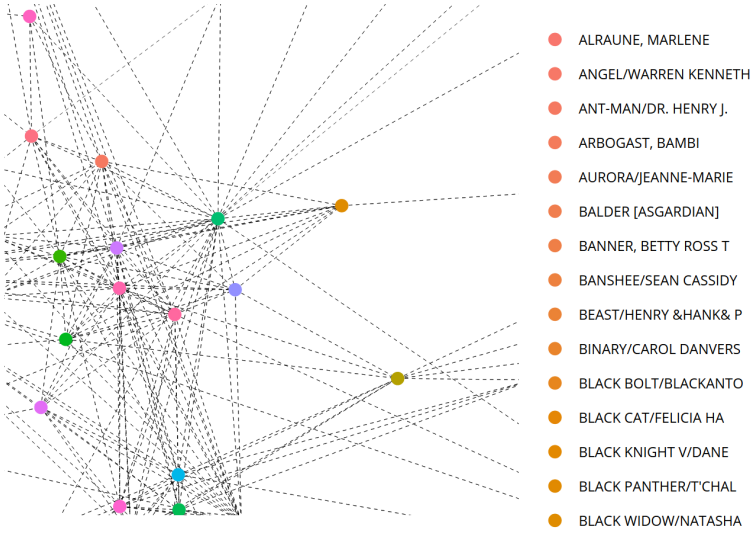

Figure 8: Closest view of Large 2D Network Graph

connected by *Spiderman* and the *Fantastic Four*; the *X-men* remain a connected albeit distinct cluster, indicating that their world is rather separate from the other characters, this stands to reason given that most characters in the *X-Men* stories have unique abilities minimising the need for cross overs.

Although such a large graph is inherently difficult to interpret for clusters close to one another it is useful for observing broader relations in conjuction with the zooming, panning and using the mouse-hover feature.

(c) Smaller Sample

A graph destined for physical print will be more appropriately constructed if the scope is restricted to a smaller number of characters and annotations are added to the vertices such that the audience can come to understand the potential connections between the characters (as opposed to a broad overview that can be adjusted in digital media), such a graph can be produced as before by gaving `ggplot` include a `geom_label` layer for the plot as shown in listing 10. This produced output is shown in figure 9.

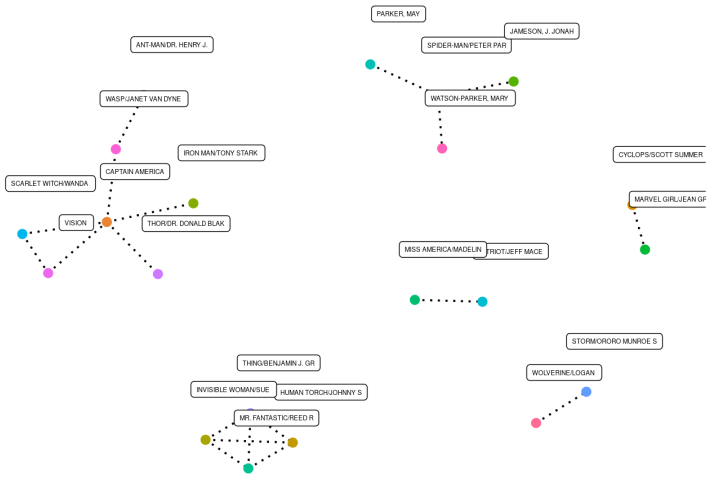

Figure 9: Smaller Network for Print Media

4. 3D Network Graph

Although a 2D graph can provide a better seperation of nodes the capacity to visualise more elements by way of a third dimension can provide a deeper understanding of the connections between the data.

In order to create a 3d graph the *Plotly* graphing library may be used, this is built on the D3JS library and geared towards data science and AI.

(a) Smaller Sample

i. Build the Graph Building Network Graphs in plotly requires some assembly out of the box, first Subset the Data and create an `igraph` object as shown in listing 11

ii. Layout the Nodes From the constructed `igraph` object a layout of nodes in 3 dimensions may be produced as shown in 12

```r
gg_network_graph_print <- function() {
G <- igraph::graph_from_data_frame(d = Edges, directed = FALSE)
   laytg <- igraph::layout.auto(G)
   colnames(laytg) <- c("xval", "yval")
   laytg <- as_tibble(laytg)
   laytg$node <- vertex_attr(G)[[1]]
   laytg %>% head()


ne <- nrow(Edges)
   ys <- xe <- ye <- xs <- vector(length = ne)
   for (i in seq_len(ne)) {
     xs[i] <- laytg$xval[laytg$node==Edges$Source[i]]
     ys[i] <- laytg$yval[laytg$node==Edges$Source[i]]
     xe[i] <- laytg$xval[laytg$node==Edges$Target[i]]
     ye[i] <- laytg$yval[laytg$node==Edges$Target[i]]
   }

   EdgeNames <- paste0("From:", Edges$Source, "To:", Edges$Target)
   starts    <- data.frame("xval" = xs, "yval" = ys, edgeid =
    ↪  EdgeNames) # TODO Make a factor
   ends      <- data.frame("xval" = xe, "yval" = ye, edgeid = EdgeNames)
   Edges_val <- as_tibble(rbind(starts,ends))
   # Edges_val %>% tail()


  p <-  ggplot(rbind(starts,ends), aes(x = xval, y = yval)) +
        geom_line(aes(group = edgeid), size = 0.6, lty = 3) +
        geom_point(data = laytg, aes(x = xval, y = yval, col = node),
        ↪  size = 2.5) +
        geom_label(data = laytg, aes(x = xval, y = yval, label = node),
        ↪  size = 1.5, nudge_x = 0.3, nudge_y = 0.8, ) +
        labs(x = "", y = "") +
        guides(col = FALSE) +
        theme_classic() +
        theme(axis.line = element_blank(),  #
        ↪  https://stackoverflow.com/a/6542792/12843551
              axis.text.y=element_blank(),axis.ticks=element_blank(),
              axis.text.x=element_blank())
  return(p)
}


Edges <- data_all[1:40,]
gg_network_graph_print()
```

Listing 10: Less Nodes and Labels will be necessary for print media.

```r
Edges <- data_all[1:300,]

# Get the number of nodes and edges
nodes <- factor(c(as.character(data$Source), as.character(data$Target)))
nodes <- nodes[!duplicated(nodes)]
nn <- length(nodes)
ne <- nrow(data)

# Build the Graph
G <- igraph::graph_from_data_frame(Edges, directed = FALSE)
```

Listing 11: Building a Network graph for *Plotly*

```r
# Get the node positions set by the layout for 3D graphs (I like Kamada
↪  Kawai)
layt <- as.data.frame(igraph::layout.kamada.kawai(G, dim = 3))
names(layt) <- c("xval", "yval", "zval")
layt$label <- vertex_attr(G)[[1]]

# Get the X Co-ordinates of the Nodes
   xn <- layt[,1] # X co-ordinate of nodes
   yn <- layt[,2]
   zn <- layt[,3]
```

Listing 12: Create the Cartesian layout of nodes in 3-Dimensions

```r
Zs <- Ys <- Xs <- Ze <- Ye <- Xe <- vector(length = ne)
for (i in seq_len(ne)) {
  Xs[i] <- layt$xval[layt$label==Edges$Source[i]]
  Ys[i] <- layt$yval[layt$label==Edges$Source[i]]
  Zs[i] <- layt$zval[layt$label==Edges$Source[i]]
  Xe[i] <- layt$xval[layt$label==Edges$Target[i]]
  Ye[i] <- layt$yval[layt$label==Edges$Target[i]]
  Ze[i] <- layt$zval[layt$label==Edges$Target[i]]
}

starts    <- data.frame("xval" = Xs, "yval" = Ys, "zval" = Zs, edgenum
↪  = 1:ne) # TODO Make a factor
ends      <- data.frame("xval" = Xe, "yval" = Ye, "zval" = Ze, edgenum
↪  = 1:ne)
Edges_val <- as_tibble(rbind(starts,ends))
# Edges_val
```

Listing 13: Use a `for` loop to get the Cartesian Points for the Edges

iii. Replace the Edges with the Node Values The corresponding endpoints for the edges can be taken from the listed nodes and built into a data-set as shown in listing 13

iv. Create the Plotly Elements Now that the ground work in creating the graph data is complete it is necessary to pass that data to *Plotly*.

A. Create the Points Vertices The points can then be passed to plotly to build a scatter-plot object as shown in figure 14.

```r
ponts <- plot_ly(x = layt[,1],
                 y = layt[,2],
                 z = layt[,3],
                # marker = list(size = 9),
                 text = layt$label,
                 color = layt$label,
                # text = 0:(nn-1),
                type = "scatter3d",
                mode = "markers")
```

Listing 14: Build a scatter plot object in plotly from the Vertices

B. Create the Lines (Edges)
Building the edges is a little tricker, the *Plotly* libray doesn't support grouping line elements and so each edge must be produced as a seperate plot and then combined as submodules (if the graph was assured to be completely connected then the endpoints could simply be provided and all connected). This is shown in listing .

```
 line_plots <- list()   # Does not support groups so you must create a seperate line plot so it doesn't just join everything

for (i in 1:ne) {                # the idea is to create a different sub plot for every line.
line_plots[i] <- plot_ly(
    x = c(starts$xval[i], ends$xval[i]),
    y = c(starts$yval[i], ends$yval[i]),
    z = c(starts$zval[i], ends$zval[i]),
    mode = 'lines',
    type = "scatter3d",
    showlegend = FALSE) # You don't want each edge to be labelled, only the nodes
}
lines_grouped <- subplot(line_plots)
```

C. Remove Axis Elements
The axis Elements are not desirable in a network graph and may be removed as a by using the `layout` function as shown in listing 15

```
# Pull out background.
noback <- list(showticks = FALSE,
               showgrid = FALSE,
               showaxis = FALSE,
               gridcolor = "white",
               showline=FALSE,
               zeroline=FALSE,
               showticklabels=FALSE, title='')


# lines <- lines %>% layout(scene = list(xaxis=noback,yaxis=noback,zaxis=noback))
lines_grouped <- lines_grouped %>% layout(scene = list(xaxis=noback,yaxis=noback,zaxis=noback))
ponts <- ponts %>% layout(scene = list(xaxis=noback,yaxis=noback,zaxis=noback)) # you need to do this for points to get rid of x,y,z labels
```

Listing 15: Remove the Axis Elements

D. Put it all Together
All the points can now be combined to create the network graph as shown in listing 16 and illustrated in figures 10, 11, 12, 13 .

```
subplot(lines_grouped, ponts)
```

Listing 16: use `subplot` to combine the lines and nodes.

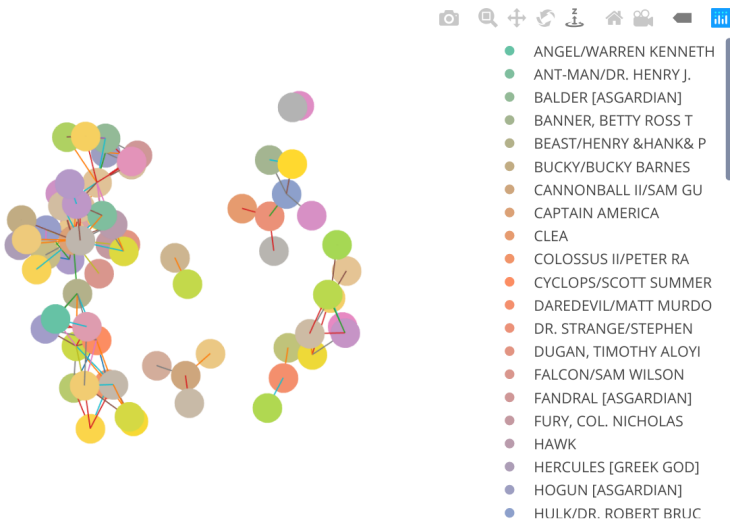

Figure 10: Overview of 3D Network Graph

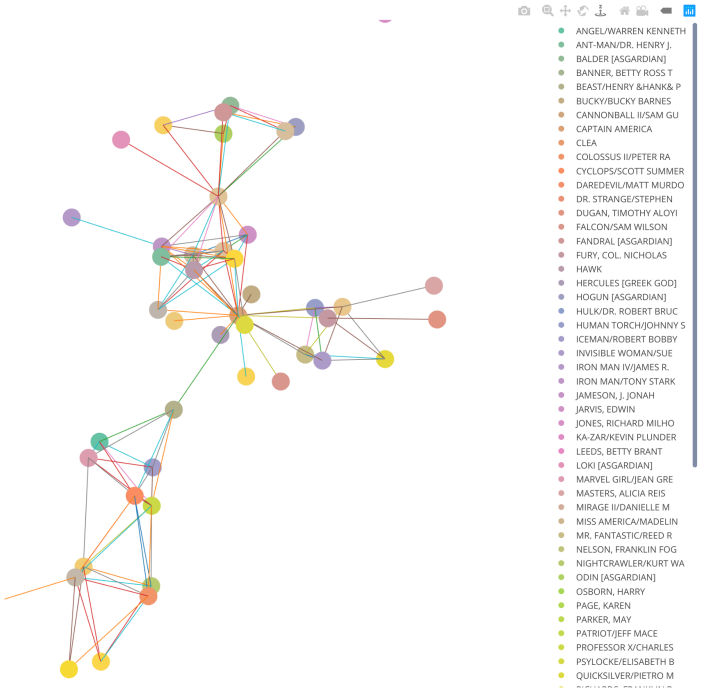

Figure 11: Closer view of 3d Network Graph

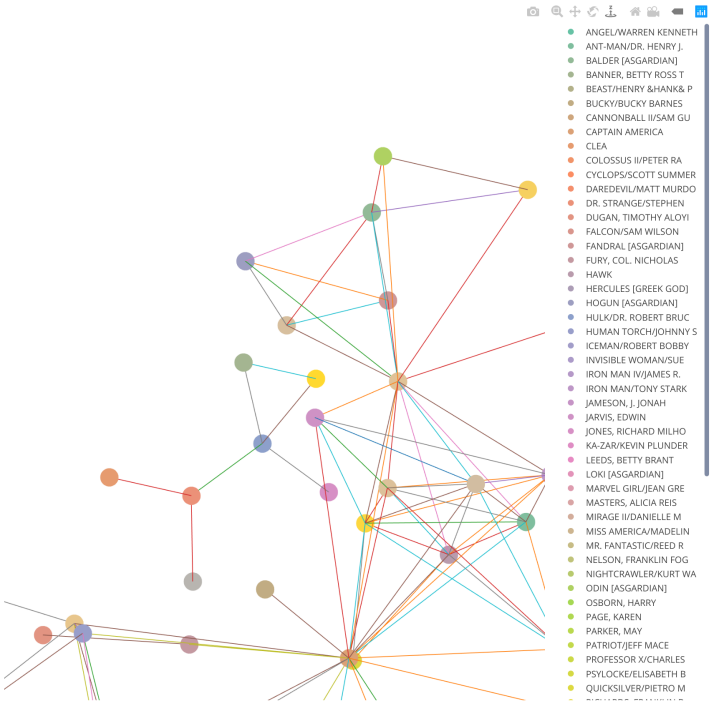

Figure 12: Alternate Perspective of 3D Network Graph

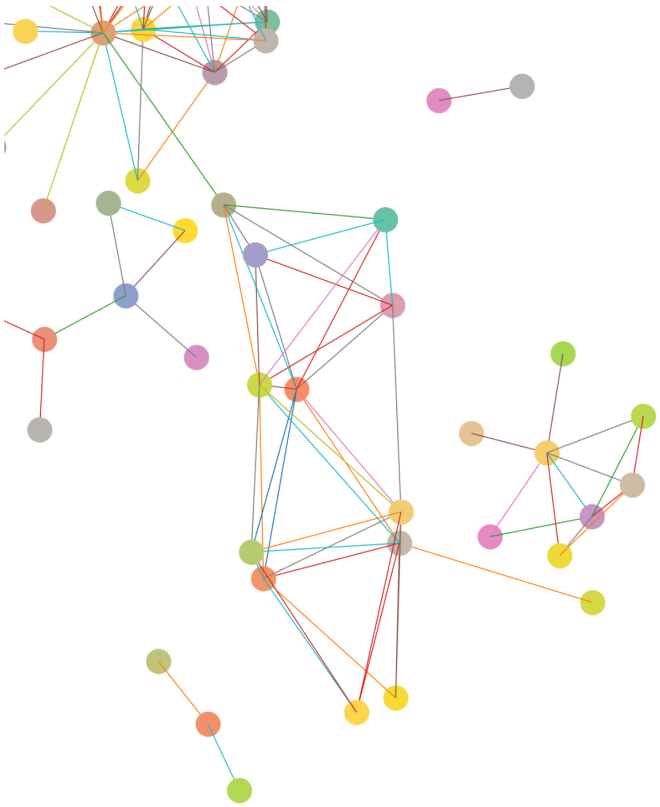

Figure 13: Rotated Perspective of 3D Network Graph

Although the points are very large, after zooming in to a degree that the edges do not overalap too much, the size of the nodes is sufficiently large to respond to a mouse hover yet not too large to obstruct from the sahpe of the graph and so such a size was settled on.

The shown distribution is identical to the 2D graph but the visualisation is significantly more immersive.

(b) Large Sample

This can be repeated for a much larger sample by using the previous code but substituting `data_all[1:1500,]` Having more nodes and edges will provide a more immersive (although less clear) visualisation of the data, this can be useful for observing large patterns but less useful for identifying individual relations, this is illustrated in 14, 15 and .

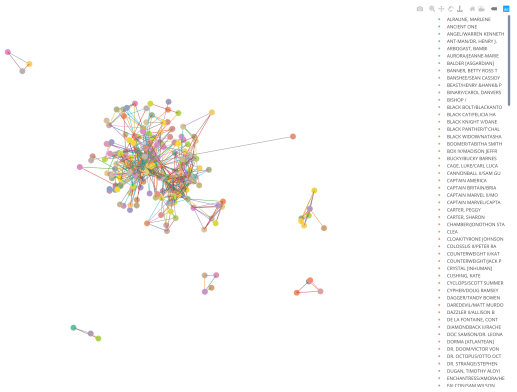

Figure 14: Overview of Large 3D Network Graph

Although the large vertex size does obstruct the fuzzy shape created by the edges, they were elected for as a compromise for the benefit of the mouse-hover annotations that allow a character to be tied to a node.

This graph has too many nodes to show relations between individuals but can be useful to show relations between clusters, for example the Gods of the Aesir/Vanir appear as a distinct cluster linked by Thor and Captain American can be seen to be a very central character that acts as the main tie between much of the universe.

5. Results

The relational network graphs show generally that the Marvel Characters tend to form there own clusters, as more characters are added (in order of the number of cross overs that character has had generally), these clusters become more and more connected through different characters.

The large network graph shows two very interesting features, primarily that certain characters such the Gods are a cluster quite far from most other characters, representing the fact that Gods don't interact with humans often in the comics. Secondly it shows that characters such as *Iron Man*, *Captain America* and *Thor* are very central to all other characters, having many cross-overs generally with a wide-diversity of characters, demonstrating that these core-characters represent a central binding tool to tie the diverse stories together into one universe.

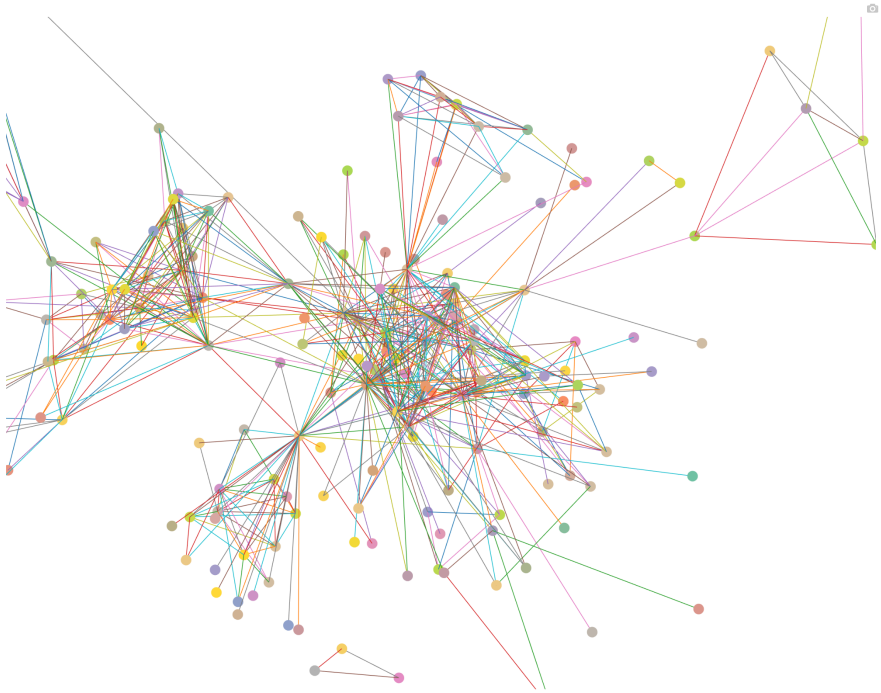

Figure 15: Closer view Large 3D Network Graph

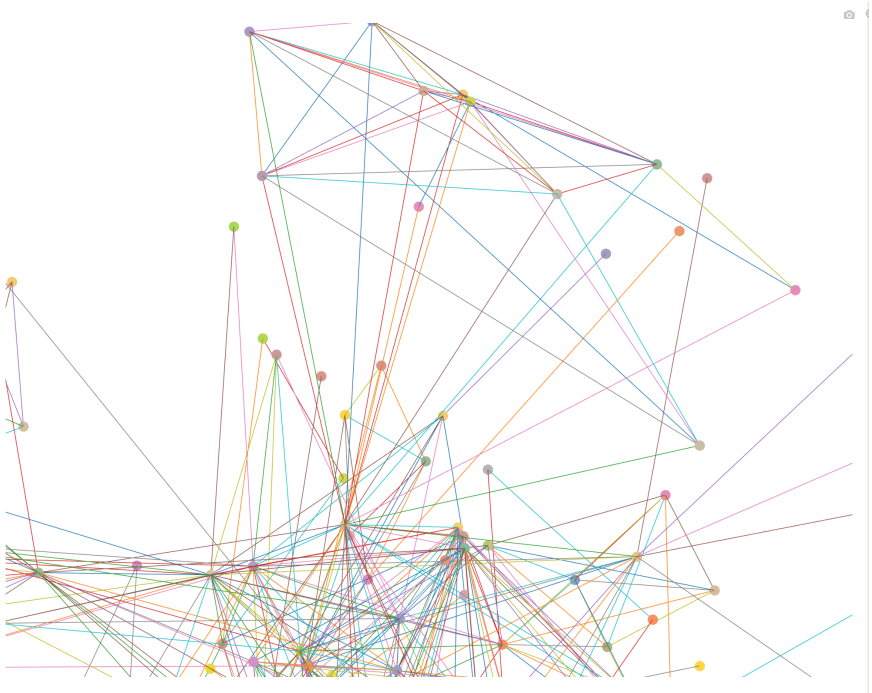

Figure 16: Closest view Large 3D Network Graph

Heirarchical Investigation

Another way to visualise this extremely large data set is to impose upon it, some degree of artificial Heirarchical structure.

If the character with the highest amount of cross overs between the most amount of characters is taken as the root and the characters closest connections mapped and those subsequent connections mapped similarly, a hierarchical data set from the *most socially diverse* character can be built. Although this does seem artificial it allows a view from the centre of the *Marvel Universe* (with respect to cross-over interaction) out to all other characters.

Sunburst

An interactive sunburst visualisation could be used to visualise this heirarchy and can be built by first subsetting the data as shown in listings 17 and 18. It is necessary to be mindful of the structure of the data before passing it to *Plotly* because it will not tolerate a data-frame of edges that breaks the heirarchy. This visualisation is illustrated in figures 17 and 18.

```
c <- 1  # Index of Most Social Character
# Choose the root value
  ## Because this will descend by number of cross overs, I will choose
  ↪  the individual with the most crossovers (including doubles)
socialability <- apply(data_adj, 2, sum)
(socialability <- socialability[order(-socialability)]) %>% head()
```

Listing 17: Determine that character with the most diverse interactions

```
##      CAPTAIN AMERICA SPIDER-MAN/PETER PAR IRON MAN/TONY STARK
##                16379                13717                11817
## THOR/DR. DONALD BLAK THING/BENJAMIN J. GR     WOLVERINE/LOGAN
##                11427                10681                10353
```

Unfourtunately the characters are very difficult to discern around the circumference of the circle, using a treemap instead it may be possible to visualise more connections, repeating the same structure for the subsequent most social character provides:

TreeMap

1. Spider-Man Repeating the Previous Heirarchical loop for the character with the 2nd highest average connections (which so happens to be *Spiderman*) as shown in produces the treemap visualisations shown in figures 19 and 20.

```r
top <- names(socialability)[which(socialability ==
  ↪  max(socialability))]; i <- 1
top <- names(socialability)[c]

sd <- data_adj[1:500,1:500]
# Build a Data Frame and set up the root value
bfdf   <- data.frame(parents = "", labels = top, values = max(sd)+1) #
  ↪  reset to 1 later, it's easier than reordering


##### Loop

while (i <= nrow(bfdf)) {
  ## Identify matches
  matches <- sd[,top] # Extract column of interest
  matches <- matches[matches>0] #Only consider characters that have met
  matches <- matches[!(names(matches) %in% bfdf$labels)] #Remove
    ↪  Characters already used

  (matches <- matches[order(-matches)]) %>% head() # Arrange the matches

  ## Apend the value to the data frame
  bfrow  <- data.frame(parents = rep(top, length(matches)), labels =
    ↪  names(matches), values = matches)
  (bfdf <- rbind(bfdf, bfrow)) %>% head()

  ## Remove Duplicates in df (failsafe)
  # n <- which(duplicated(bfdf$labels))
  # bfdf <- bfdf[-n,]

  ## Set the next top value
  i <- i+1
  top <- bfdf$labels[i]
  while (is.na(top) && i <= nrow(bfdf)) {
    i <- i+1
    top <- bfdf$labels[i]
  }
 # print(i/nrow(bfdf))

  ## Inspect Parents
  # unique(bfdf$parents) %>% print()
}

bfdf$values[1] <- 1
## Test plotting
#(fig <- plot_ly( labels  = bfdf$labels, parents = bfdf$parents, values
  ↪   = bfdf$values, type = 'sunburst' ))
```

Listing 18: Use a loop to build a heirarchical data set

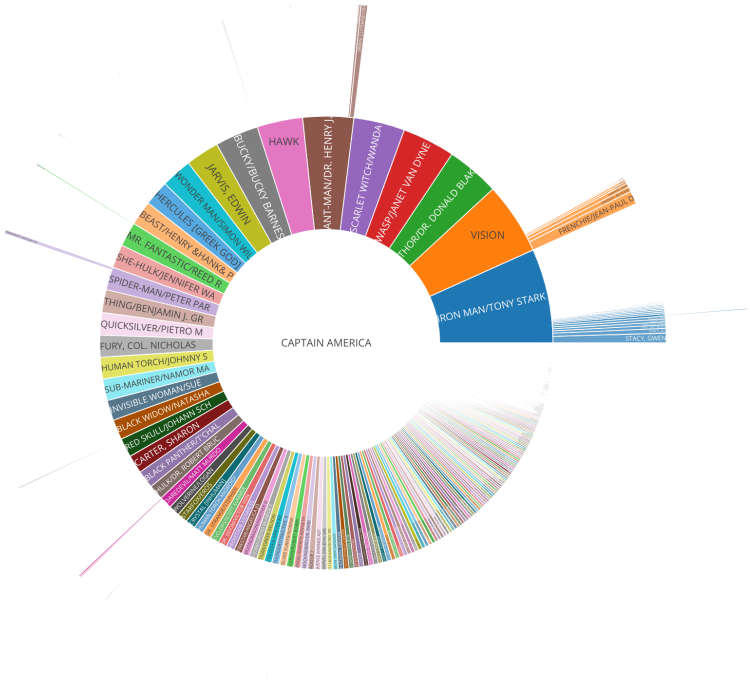

Figure 17: Sunburst Diagram of Relationships from *Captain America*'s perspective

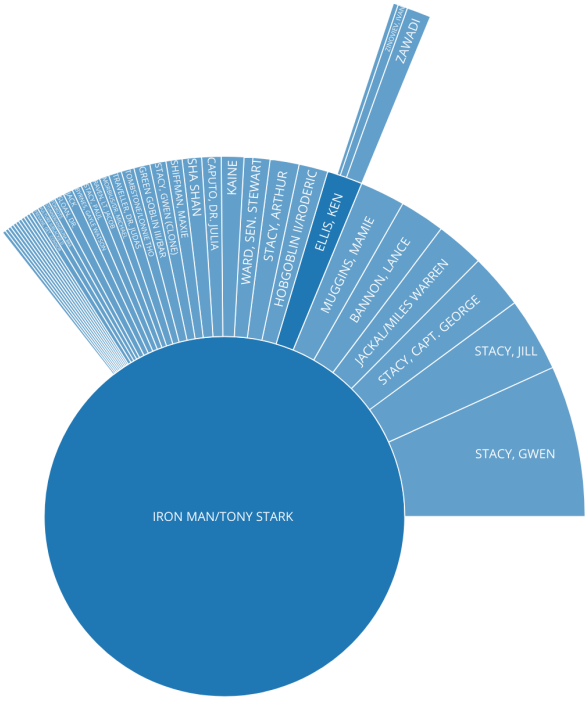

Figure 18: Iron Man's Relationships from the perspective of *Captain America*

```r
c <- 2  # Index of Most Social Character
socialability <- apply(data_adj, 2, sum)
(socialability <- socialability[order(-socialability)]) %>% head()
top <- names(socialability)[which(socialability ==
↪  max(socialability))]; i <- 1
top <- names(socialability)[c]

sd <- data_adj[1:600,1:600]
# Build a Data Frame and set up the root value
bfdf   <- data.frame(parents = "", labels = top, values =
↪  max(sd)+1) # reset to 1 later, it's easier than reordering

##### Loop
while (i <= nrow(bfdf)) {
  ## Identify matches
  matches <- sd[,top] # Extract column of interest
  matches <- matches[matches>0] #Only consider characters that have
  ↪  met
  matches <- matches[!(names(matches) %in% bfdf$labels)] #Remove
  ↪  Characters already used

  (matches <- matches[order(-matches)]) %>% head() # Arrange the
  ↪  matches

  ## Apend the value to the data frame
  bfrow  <- data.frame(parents = rep(top, length(matches)), labels
  ↪  = names(matches), values = matches)
  (bfdf <- rbind(bfdf, bfrow)) %>% head()

  ## Remove Duplicates in df (failsafe)
  # n <- which(duplicated(bfdf$labels))
  # bfdf <- bfdf[-n,]

  ## Set the next top value
  i <- i+1
  top <- bfdf$labels[i]
  while (is.na(top) && i <= nrow(bfdf)) {
    i <- i+1
    top <- bfdf$labels[i]
  }
}

bfdf$values[1] <- 1
(fig <- plot_ly( labels  = bfdf$labels, parents = bfdf$parents,
↪  values  = bfdf$values, type = "treemap" ))
```

```
##      CAPTAIN AMERICA SPIDER-MAN/PETER PAR IRON MAN/TONY STARK
##                16379                13717                11817
## THOR/DR. DONALD BLAK THING/BENJAMIN J. GR     WOLVERINE/LOGAN
##                11427                10681                10353
```

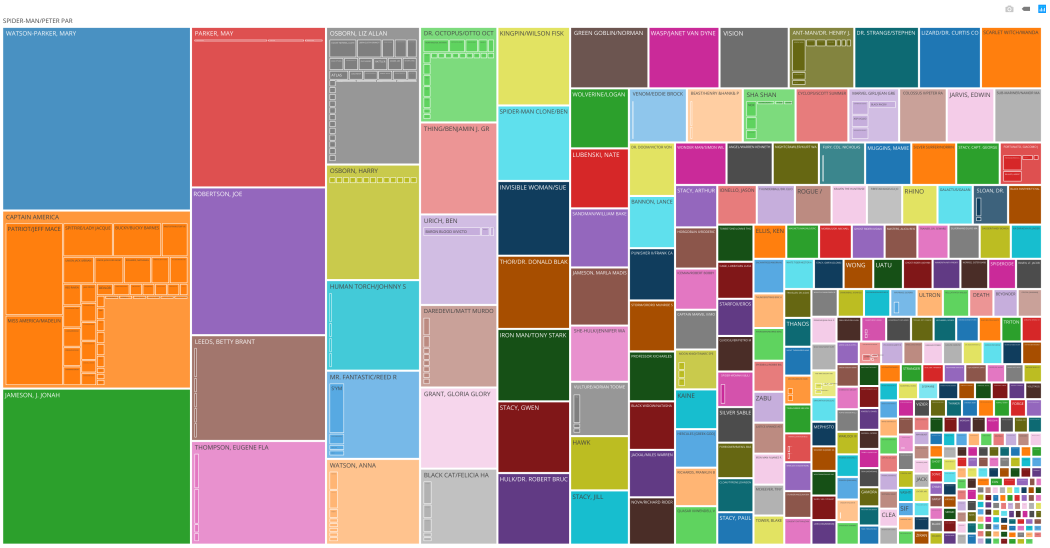

Figure 19: Treemap of *Spiderman*'s Relationships

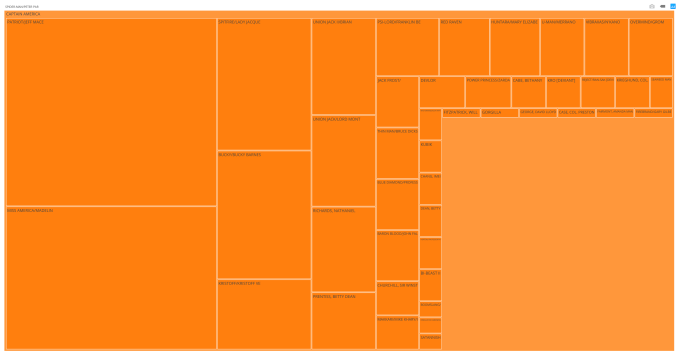

Figure 20: Zooming in on the Interactive Treemap

2. Thor
   Repeating the process shown previously in listing for the third most connected character (*Thor*) the visualisation shown in 21 may be produced.

Results

Despite sampling the characters for interconnectivity, the characters appear to show a very low *degree of seperation* indicating that for the most part, the *Marvel Universe* is very inter connected (as opposed to the *DC Comics* universe which is fragmented by different imprints of comics targeted towards different demographics, this fragmentation is something that the publisher has been trying to overcome [25]

Figure 21: Treemap of *Iron Man*'s Relationships

Tree

For print media, a more appropriate representation of heirarchical data would be a graph in the form of a tree, as opposed to a sunburst visualisation, because a sunburst or treemap will be too small to readily visualise and the rigid structure of a tree will be easy to interpret without any interactive aids on a page for which the scale cannot be adjusted.

1. Build the Heirarchical Data Set

   As before the process here will be to choose a character that has the highest number of inter-relations and then match the next 2-8 edges that have the highest weight (total number of interactions), this can be repeated for a depth of 2-8, however there is a computational limit as the number of iterations the loop will perform is $\sum_{k=0}^{l} \left[n^k \right]$, moreover because the edges can still interact in this algorithm (a property that I found to be desirable for want of visualisation) the structure of the tree may become somewhat distorted (e.g. higher nodes being pushed down).

   This distortion may make the tree somewhat misleading by moving nodes between levels but is an acceptable compromise given that it provides an effective way to illustrate the interconnected-ness of characters as opposed to a purely linear descent.

   First extract the character that socialises with the most people on average (from before we know this to be *Captain America*) as shown in listing 19

   ```
   ## [1] "CAPTAIN AMERICA"
   ```

   Next execute the algorithm to create the hierarchical structure described previously as shown in listing 20

2. Build the Graph

   The tree graph, as before can be constructed by using the `igraph` library as shown in listing 21:

```r
# load(file = "./hero_adjacency.Rdata")
# data <- hero_adj
data <- data_adj
most_social <- apply(data, 2, mean)
most_social <- most_social[order(-most_social)]


treedf <- data.frame()
n <- 2 # number of children # I really liked 2
l <- 5 # number of levels   # I liked 3 (actually n=3, l = 5 was good too)
en <- sum(n^(1:l)) #number of edges for corresponding level number

(top <- names(most_social)[1])
```

Listing 19: Create the data-frame for the tree data.

3. Plot the Graph

   Finally the plot can be constructed by using `ggplot` to plot the nodes and map the lines relative to the edge group as shown in listing 22, the output of which is illustrated in figure 22

4. Effectiveness of Tree Graph

   This visualisation is less descriptive than the ring chart or treemaps but it is easier to interpret because of the clear spacing, and smaller number of visual elements. Compared to the preceeding 2D/3D graphs this hierarchical chart may appear at first a little misleading, because connections between characters are only shown for a characters strongest relationship that hasn't already been made for a preceeding character (from the perspective of the initial charater/node, i.e. *Captain America*), the clear starting point of the tree is also only true from the perspective of the initial character.

   Nonetheless, this is an effective visualisation because the initial character is a character with the highest number of mean connections and this can show a heirarchical layout of the universe from the most inter-related characters to the outskirts of isolated characters.

Discussion

Details of Techniques

The techniques used to build the visualisations have been included in code dialog boxes and explained previously, but the the reason these partucluar methods were chosen is because:

- Relational data can be readily visualised as a network graph
- Analysing the data to observe heirarchical relations can be insighful

```
matches <- data[top,]
data <- data[-which(rownames(data)==top), -which(rownames(data)==top)]
matches <- matches[matches>0]
matches <- matches[order(-matches)]
matchdf <- data.frame(parent = rep(top,n ), labels =
 ↪  names(matches)[1:n], values = matches[1:n])
treedf <- rbind(treedf,matchdf)

for (i in 1:en) {
  #  i <- 1
  top <- treedf$labels[i]
  matches <- data[top,]    # This for which filter sometimes returns
   ↪  null and I don't know why
  #  data <- data[-which(rownames(data)==top),
   ↪  -which(rownames(data)==top)]
  matches <- matches[matches>0]
  matches <- matches[matches != top] # This is a simpler fix because R
   ↪  is slow
  matches <- matches[order(-matches)]
  if (length(matches)<n) {
    matchdf <- data.frame(parent = rep(top,length(matches)), labels =
     ↪  names(matches)[1:length(matches)], values =
     ↪  matches[1:length(matches)])
  } else if(length(matches)==0) {
    print("No matches for this iterance")
  } else {
    matchdf <- data.frame(parent = rep(top,n ), labels =
     ↪  names(matches)[1:n], values = matches[1:n])
  }
  treedf <- rbind(treedf,matchdf)
#  print(i/en)
  #  i <- i+1
}
top <- treedf$labels[2]
matches <- data[top,]
matches <- matches[matches>0]
matches <- matches[order(-matches)]
matchdf <- data.frame(parent = rep(top,n ), labels =
 ↪  names(matches)[1:n], values = matches[1:n])
treedf <- rbind(treedf,matchdf)
```

Listing 20: Use a loop to build a tree structure

```
Edges <- treedf
names(Edges) <- c("Source", "Target", "Value")


#{

# GGPlot (# Because of the way that I made the graph, the nodes come from the edges, hence, every node must be connected.)
G <- igraph::graph_from_data_frame(d = Edges, directed = FALSE)
laytg <- igraph::layout_as_tree(G, root = 1)
colnames(laytg) <- c("xval", "yval")
laytg <- as_tibble(laytg)
laytg$node <- vertex_attr(G)[[1]]


ne <- nrow(Edges)
ys <- xe <- ye <- xs <- vector(length = ne)
for (i in seq_len(length(xs))) {
  xs[i] <- laytg$xval[laytg$node==Edges$Source[i]]
  ys[i] <- laytg$yval[laytg$node==Edges$Source[i]]
  xe[i] <- laytg$xval[laytg$node==Edges$Target[i]]
  ye[i] <- laytg$yval[laytg$node==Edges$Target[i]]
}

starts    <- data.frame("xval" = xs, "yval" = ys, edgenum = 1:ne) # TODO Make a factor
ends      <- data.frame("xval" = xe, "yval" = ye, edgenum = 1:ne)
Edges_val <- as_tibble(rbind(starts,ends))
```

Listing 21: Make the tree structure using the `igraph`

```r
library(ggrepel)
p <-  ggplot(rbind(starts,ends), aes(x = xval, y = yval)) +
  geom_line(aes(group = edgenum), lty = 3, col = "darkgrey", size = 0.3) +
  geom_point(data = laytg, aes(x = xval, y = yval, col = node), size = 4) +
  labs(x = "", y = "") +
  geom_label_repel(data = laytg, aes(x = xval, y = yval, label = node, col = node), size = 1.5, nudge_x = 0, nudge_y = 0) +
  guides(col = FALSE) +
  theme_classic() +
  theme(axis.line = element_blank(),  # https://stackoverflow.com/a/6542792/12843551
    axis.text.y=element_blank(),axis.ticks=element_blank(),
    axis.text.x=element_blank())
 p
```

Listing 22: use ggplot to build the graph from the tree data

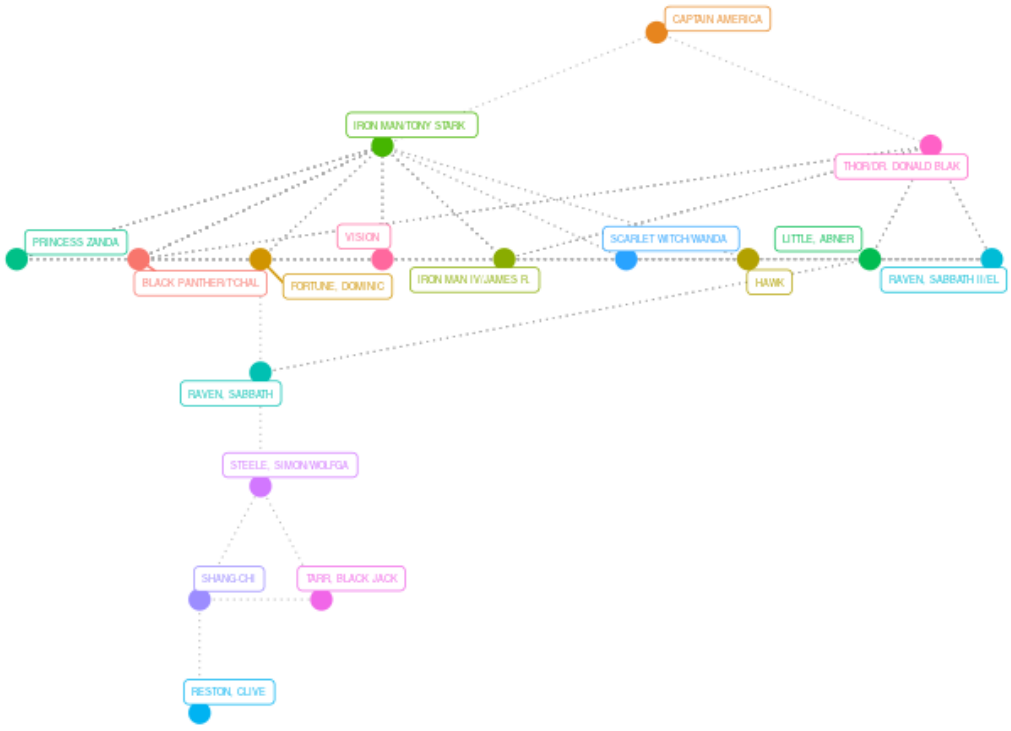

Figure 22: Heirarchical Tree Constructed From the Most Social Character through the closest relation

- Sunburst and treemap visualisations help explain these relations relative to the total number of character interactions (the weight) and relative to other characters.

Advantages and Disadvantages of Utilised Methods

These methods are good because they allow much of the data to be visualised in a very raw form, however with so much data the plots can be resource intensive to generate and manimpulate and also hard to interpret.

1. Other methods in literature
   A potential improvement could be categorising nodes as subsets of a parent node (e.g. joining all the *X-men*, Norse Gods and all the *Avengers* together) or simply doing a type of proximitly clustering merely for want of visualisation (although this would compromise the capacity to interpret the labels of the nodes, it would be easier than manually clustering observations).

   Some force directed methods that allow the force to be interacted with show more information by allowing the user to interact with the nodes and edges, this provides a 'sense' of the force which corresponds to the weight of the edge (in this case the number of interactions between characters). Such a technique provides another method to express the dimension of weight; this provides a potential improvement to the methods utilised, unforutunately this isn't a method that can be readily implemented in the *Plotly* graphing library and would require a different approach.

2. Can They be used for large sets of relational data

   The 3D Network graph can be used to visualise moderately sized data, up to about 1,500 edges, this is because it's interactive and it can scale to still be useful (depending on the data set of course). Unfourtunately however it is very resource intensive for many edges owing to the way the library draws lins and a more appropriate limit might be closer to 500 Edges. Data sets larger than that simply cannot be visualised by a nework graph because they become too visually unwieldly and resource intensvie to be practically visualised and a type of reduction technique would need to be implemented.

   If the graph is printed there is a very hard limit to what can be practically visualised, perhaps around 50 nodes depending on the size of the printout and structure of the edge-node relationships, a limit beyond which it is not feasible to interpret a visualisation.

3. Results Analysis
   Analysis results were discussed below each plot, but generally the visualisation allows for observations like the relative isolation of the *Fantastic Four*; as well as the way that the Norse gods of the Aesir and Vanir hover above the rest of the *Marvel* characters almost as if they were Gods of another realm looking over *Midgard*.

   Ultimately the characters of the *Marvel Universe* are very interconnected but certain worlds are still quite distinct, for example the *X-men* and the Norse Gods which are very isolated when compared to, for example, *The Avengers*.

Conclusion

The inter-relations of marvel characters creates a very interesting set of relations that can be effectively visualised using techniques such as network graphs, 3D netowrk graphs, treemaps, sunburst diagrams and tree graphs.

References

[1] Murray Aitkin, Duy Vu, and Brian Francis. "Statistical Modelling of the Group Structure of Social Networks". en. In: *Social Networks* 38 (July 2014), pp. 74–87. ISSN: 0378-8733. DOI: 10.1016/j.socnet.2014.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0378873314000161 (visited on 04/08/2020) (cit. on p. 2).

[2] R. Alberich, J. Miro-Julia, and F. Rossello. "Marvel Universe Looks Almost like a Real Social Network". en. In: (Feb. 2002). URL: https://arxiv.org/abs/cond-mat/0202174v1 (visited on 04/06/2020) (cit. on p. 2).

[3] Patricia M. Amburgy. "Review of Comic Book Nation: The Transformation of Youth Culture in America". In: *History of Education Quarterly* 42.3 (2002), pp. 452–457. ISSN: 0018-2680. URL: https://www.jstor.org/stable/3217991 (visited on 04/07/2020) (cit. on p. 1).

[4] Luis M. Camarinha-Matos and Hamideh Afsarmanesh. "Roots of Collaboration: Nature-Inspired Solutions for Collaborative Networks". In: *IEEE Access* 6 (2018). Conference Name: IEEE Access, pp. 30829–30843. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2845119 (cit. on p. 2).

[5] Anna Chaimani and Georgia Salanti. "Visualizing Assumptions and Results in Network Meta-Analysis: The Network Graphs Package". en. In: *The Stata Journal* 15.4 (Dec. 2015), pp. 905–950. ISSN: 1536-867X. DOI: 10.1177/1536867X1501500402. URL: https://doi.org/10.1177/1536867X1501500402 (visited on 04/08/2020) (cit. on p. 2).

[6] Russ Chappell. *Marvel Chronology Project - Main*. Database. Dec. 2019. URL: http://www.chronologyproject.com/ (visited on 04/08/2020) (cit. on p. 2).

[7] Sanhueza Claudio. *The Marvel Universe Social Network*. en. Library Catalog: www.kaggle.com. 2017. URL: https://kaggle.com/csanhueza/the-marvel-universe-social-network (visited on 04/06/2020) (cit. on p. 2).

[8] 1943- Daniels Les. *Marvel : Five Fabulous Decades of the World's Greatest Comics*. English. Marvel Comics (New York, N.Y.) ; 1. New York: H.N. Abrams, 1991. ISBN: 0-8109-3821-9 (cit. on p. 1).

[9] Tom DeFalco. *Spder-Man: The Ultimate Guide*. New York: DK, 2001. ISBN: 978-0-7894-7946-4. URL: https://archive.org/details/spidermanultimat00defa (cit. on p. 2).

[10] Gerrit Sander van Doorn and Michael Taborsky. "The Evolution of Generalized Reciprocity on Social Interaction Networks". en. In: *Evolution* 66.3 (2012). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1558-5646.2011.01479.x, pp. 651–664. ISSN: 1558-5646. DOI: 10.1111/j.1558-5646.2011.01479.x. URL: http://onlinelibrary.wiley.com/doi/abs/10.1111/j.1558-5646.2011.01479.x (visited on 04/08/2020) (cit. on p. 2).

[11] Cody Dunne and Ben Shneiderman. "Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: Association for Computing Machinery, Apr. 2013, pp. 3247–3256. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2466444. URL: http://doi.org/10.1145/2470654.2466444 (visited on 04/07/2020) (cit. on p. 2).

[12] Robert Genter. ""With Great Power Comes Great Responsibility": Cold War Culture and the Birth of Marvel Comics". en. In: *The Journal of Popular Culture* 40.6 (2007). _eprint: https://onlinelibrary.wiley.com/doi 5931.2007.00480.x, pp. 953–978. ISSN: 1540-5931. DOI: 10.1111/j.1540-5931.2007.00480.x. URL: http://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5931.2007.00480.x (visited on 04/07/2020) (cit. on p. 1).

[13] Yifan Hu and Lei Shi. "Visualizing Large Graphs". en. In: *WIREs Computational Statistics* 7.2 (2015). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1343, pp. 115–136. ISSN: 1939-0068. DOI: 10.1002/wics.1343. URL: http://www.onlinelibrary.wiley.com/doi/abs/10.1002/wics.1343 (visited on 04/08/2020) (cit. on p. 2).

[14] *Igraph R Package*. URL: https://igraph.org/r/%5C#contribute (visited on 04/08/2020) (cit. on p. 2).

[15] Stephan Kitchovitch and Pietro Liò. "Community Structure in Social Networks: Applications for Epidemiological Modelling." English. In: *PloS one* 6.7 (2011). Num Pages: 1, e22220. DOI: http://dx.doi.org.ezproxy.uws.edu.au/10.1371/journal.pone.0022220. URL: http://search.proquest.com/docview/879484085?rfr_id=info%5C%3Axri%5C%2Fsid%5C%3Aprimo (visited on 04/08/2020) (cit. on p. 2).

[16] Stan Lee. *Origins of Marvel Comics*. New York: Simon & Schuster/Fireside Books, 1974. ISBN: 978-0-671-21863-8 (cit. on p. 1).

[17] Stan Lee and Danny Fingeroth. *Superman on the Couch: What Superheroes Really Tell Us about Ourselves and Our Society*. English. 2004. ISBN: 0-8264-1539-3. URL: http://search.proquest.com/docview/38120802/ (cit. on p. 2).

[18] Mikhail Lyubansky. "Prejudice Lesons from the Xavier Institute". In: *The Psychology of Superheroes: An Unauthorized Exploration*. Benbella Books, 2008, pp. 75–90. ISBN: 978-1-933771-31-1 (cit. on p. 2).

[19] Lei Shi et al. "Scalable Network Traffic Visualization Using Compressed Graphs". In: *2013 IEEE International Conference on Big Data*. Oct. 2013, pp. 606–612. DOI: 10.1109/BigData.2013.6691629 (cit. on p. 2).

[20] Joe Simon, ed. *GCD :: Issue :: Captain America Comics #1*. Timely Publications, Dec. 1940. URL: https://www.comics.org/issue/1313/ (visited on 04/07/2020) (cit. on p. 1).

[21] Robert Spence. *Information Visualization*. London, UK: Springer, 2014. ISBN: 978-3-319-07340-8. URL: https://www.springer.com/gp/book/9783319073408 (cit. on p. 4).

[22] Frank van Ham, Martin Wattenberg, and Fernanda B. Viegas. "Mapping Text with Phrase Nets". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 1169–1176. ISSN: 1941-0506. DOI: 10.1109/TVCG.2009.165 (cit. on p. 2).

[23] Mathew Ward. *Interactive Data Visualization*. Second. CRC Press, June 2015. ISBN: 1-4822-5737-8. URL: https://www.routledge.com/Interactive-Data-Visualization-Foundations-Techniques-and-Applications/Ward-Grinstein-Keim/p/book/9781482257373 (cit. on p. 3).

[24] Eric W. Weisstein. *Disconnected Graph*. en. Text. Library Catalog: mathworld.wolfram.com. URL: https://mathworld.wolfram.com/DisconnectedGraph.html (visited on 04/06/2020) (cit. on p. 8).

[25] James Whitbrook. *What The Hell Did Superman Just Do To The DC Comics Universe?* en. Library Catalog: www.kotaku.com.au. Mar. 2017. URL: https://www.kotaku.com.au/2017/03/what-the-hell-did-superman-just-do-to-the-dc-comics-universe/ (visited on 04/06/2020) (cit. on p. 27).

[26] rohola zandie. *Network Plot with Plotly and Graphviz*. en. Library Catalog: medium.com. Dec. 2017. URL: https://medium.com/@hilbert.cantor/network-plot-with-plotly-and-graphviz-ebd7778073b (visited on 04/08/2020) (cit. on p. 2).