But there are two packages that integrate network analysis much more nicely with the tidyverse: tidygraph and ggraph. These, I am going to show how to use for analyzing yet another network of characters from **A Song of Ice and Fire** / **Game of Thrones** (to be correct, this new network here is strictly based on the **A Song of Ice and Fire** books, NOT on the TV show).

**What can network analysis tell us?**

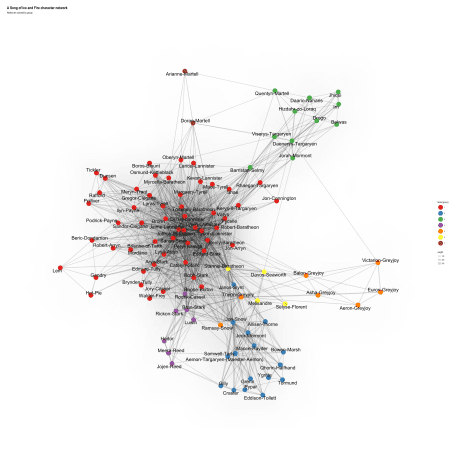
Network analysis can e.g. be used to explore relationships in social or professional networks. In such cases, we would typically ask questions like:

* How many connections does each person have?
* Who is the most connected (i.e. influential or “important”) person?
* Are there clusters of tightly connected people?
* Are there a few key players that connect clusters of people?
* etc.

These answers can give us a lot of information about the patterns of how people interact.

So, how do we find out who the most important characters are in this network? We consider a character “important” if he has connections to many other characters. There are a few network properties, that tell us more about this, like node centrality and which characters are key-players in the books.

**A word of caution before you read on: BEWARE of SPOILERS for all books!**



A Song of Ice and Fire character network across all five books; find out how I made it by following the code below…

library(readr) # fast reading of csv files

library(tidyverse) # tidy data analysis

library(tidygraph) # tidy graph analysis

library(ggraph) # for plotting

**The Data**

I obtained the following data by cloning [this Github repository](https://github.com/mathbeveridge/asoiaf) from Andrew Beveridge:

Character Interaction Networks for George R. R. Martin’s “A Song of Ice and Fire” saga These networks were created by connecting two characters whenever their names (or nicknames) appeared within 15 words of one another in one of the books in “A Song of Ice and Fire.” The edge weight corresponds to the number of interactions. You can use this data to explore the dynamics of the Seven Kingdoms using network science techniques. For example, community detection finds coherent plotlines. Centrality measures uncover the multiple ways in which characters play important roles in the saga.

Andrew already did a great job analyzing these character networks and you can read all his conclusions on his site [https://networkofthrones.wordpress.com](https://networkofthrones.wordpress.com/). Here, I don’t aim to replicate his analyses but I want to show how you could do this or similar analyses with tidygraph and ggraph. Thus, I am also not going to use all of his node and edge files.

path <- "/Users/shiringlander/Documents/Github/Data/asoiaf/data/"

files <- list.files(path = path, full.names = TRUE)

files

## [1] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-all-edges.csv"

## [2] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-all-nodes.csv"

## [3] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book1-edges.csv"

## [4] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book1-nodes.csv"

## [5] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book2-edges.csv"

## [6] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book2-nodes.csv"

## [7] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book3-edges.csv"

## [8] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book3-nodes.csv"

## [9] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book4-edges.csv"

## [10] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book4-nodes.csv"

## [11] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book45-edges.csv"

## [12] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book45-nodes.csv"

## [13] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book5-edges.csv"

## [14] "/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book5-nodes.csv"

**Characters across all books**

The first data set I am going to use are the character interactions in all five books. I am not using the node files here, because I find the edge names sufficient for this demonstration. If you wanted to have nice name labels, you could use the node files.

cooc\_all\_edges <- read\_csv(files[1])

Because there are so many characters in the books, many of them minor, I am subsetting the data to the 100 characters with the most interactions across all books.The edges are undirected, therefore there are no redundant Source-Target combination; because of this, I gathered Source and Target data before summing up the weights.

main\_ch <- cooc\_all\_edges %>%

select(-Type) %>%

gather(x, name, Source:Target) %>%

group\_by(name) %>%

summarise(sum\_weight = sum(weight)) %>%

ungroup()

main\_ch\_l <- main\_ch %>%

arrange(desc(sum\_weight)) %>%

top\_n(100, sum\_weight)

main\_ch\_l

## # A tibble: 100 x 2

## name sum\_weight

##

## 1 Tyrion-Lannister 2873

## 2 Jon-Snow 2757

## 3 Cersei-Lannister 2232

## 4 Joffrey-Baratheon 1762

## 5 Eddard-Stark 1649

## 6 Daenerys-Targaryen 1608

## 7 Jaime-Lannister 1569

## 8 Sansa-Stark 1547

## 9 Bran-Stark 1508

## 10 Robert-Baratheon 1488

## # ... with 90 more rows

cooc\_all\_f <- cooc\_all\_edges %>%

filter(Source %in% main\_ch\_l$name & Target %in% main\_ch\_l$name)

**tidygraph and ggraph**

Both tidygraph and ggraph have been developed by [Thomas Lin Pedersen](https://www.data-imaginist.com/):

With tidygraph I set out to make it easier to get your data into a graph and perform common transformations on it, but the aim has expanded since its inception. The goal of tidygraph is to empower the user to formulate complex questions regarding relational data as simple steps, thus enabling them to retrieve insights directly from the data itself. The central idea this all boils down to is this: you don’t have to plot a network to understand it. While I absolutely love the field of network visualisation, it is in many ways overused in data science — especially when it comes to extracting knowledge from a network. Just as you don’t need a plot to tell you which car in a dataset is the fastest, you don’t need a plot to tell you which pair of friends are the closest. What you do need, instead of a plot, is a tool that allow you to formulate your question into a logic sequence of operations. For many people in the world of rectangular data, this tool is increasingly dplyr (and friends), and I do hope that tidygraph can take on the same role in the world of relational data. <https://www.data-imaginist.com/2018/tidygraph-1-1-a-tidy-hope/>

The first step is to convert our edge table into a tbl\_graph object structure. Here, we use the as\_tbl\_graph() function from tidygraph; it can take many different types of input data, like data.frame, matrix, dendrogram, igraph, etc.

Underneath the hood of tidygraph lies the well-oiled machinery of igraph, ensuring efficient graph manipulation. Rather than keeping the node and edge data in a list and creating igraph objects on the fly when needed, tidygraph subclasses igraph with the tbl\_graph class and simply exposes it in a tidy manner. This ensures that all your beloved algorithms that expects igraph objects still works with tbl\_graph objects. Further, tidygraph is very careful not to override any of igraphs exports so the two packages can coexist quite happily. <https://www.data-imaginist.com/2017/introducing-tidygraph/>

A central aspect of tidygraph is that you can directly manipulate node and edge data from this tbl\_graph object by **activating** nodes or edges. When we first create a tbl\_graph object, the nodes will be activated. We can then directly calculate node or edge metrics, like centrality, using tidyverse functions.

as\_tbl\_graph(cooc\_all\_f, directed = FALSE)

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 1 (active)

## name

##

## 1 Aemon-Targaryen-(Maester-Aemon)

## 2 Aeron-Greyjoy

## 3 Aerys-II-Targaryen

## 4 Alliser-Thorne

## 5 Arianne-Martell

## 6 Arya-Stark

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 1 4 Undirected 43 7

## 2 1 13 Undirected 44 4

## 3 1 28 Undirected 52 3

## # ... with 795 more rows

We can change that with the activate() function. We can now, for example, remove multiple edges. When you are using RStudio, start typing ?edge\_is\_ and wait for the autocomplete function to show you what else is possible (or go to the tidygraph manual).

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(edges) %>%

filter(!edge\_is\_multiple())

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Edge Data: 798 x 5 (active)

## from to Type id weight

##

## 1 1 4 Undirected 43 7

## 2 1 13 Undirected 44 4

## 3 1 28 Undirected 52 3

## 4 1 32 Undirected 53 20

## 5 1 34 Undirected 54 5

## 6 1 41 Undirected 56 5

## # ... with 792 more rows

## #

## # Node Data: 100 x 1

## name

##

## 1 Aemon-Targaryen-(Maester-Aemon)

## 2 Aeron-Greyjoy

## 3 Aerys-II-Targaryen

## # ... with 97 more rows

**Node ranking**

Often, especially when visualising networks with certain layouts, the order in which the nodes appear will have a huge influence on the insight you can get out (e.g. matrix plots and arc diagrams). The node\_rank\_\*() family of algorithms have been introduced to provide different ways of sorting nodes so that closely related nodes are positionally close. As there is often not a single correct answer to this endeavor, there’s a lot of different algorithms that may provide different insights into your network. Many of them are based on the seriation package, and the vignette provided therein serves as a nice introduction to the different algorithms. <https://www.data-imaginist.com/2018/tidygraph-1-1-a-tidy-hope/>

There are many options for node ranking (go to ?node\_rank for a full list); let’s try out **Minimize hamiltonian path length using a travelling salesperson solver**.

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(nodes) %>%

mutate(n\_rank\_trv = node\_rank\_traveller()) %>%

arrange(n\_rank\_trv)

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 2 (active)

## name n\_rank\_trv

##

## 1 Janos-Slynt 1

## 2 Aemon-Targaryen-(Maester-Aemon) 2

## 3 Jeor-Mormont 3

## 4 Samwell-Tarly 4

## 5 Qhorin-Halfhand 5

## 6 Ygritte 6

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 2 75 Undirected 43 7

## 2 2 76 Undirected 44 4

## 3 2 73 Undirected 52 3

## # ... with 795 more rows

**Centrality**

Centrality describes the number of edges that are in- or outgoing to/from nodes. High centrality networks have few nodes with many connections, low centrality networks have many nodes with similar numbers of edges. The centrality of a node measures the importance of it in the network.

This version adds 19(!) new ways to define the notion of centrality along with a manual version where you can mix and match different distance measures and summation strategies opening up the world to even more centrality scores. All of this wealth of centrality comes from the netrankr package that provides a framework for defining and calculating centrality scores. If you use centrality measures somewhere in your analysis I cannot recommend the vignettes provided by netrankr enough as they provide a fundamental intuition about the nature of such measures and how they can/should be used. <https://www.data-imaginist.com/2018/tidygraph-1-1-a-tidy-hope/>

Again, type ?centrality for an overview about all possible centrality measures you can use. Let’s try out centrality\_degree().

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(nodes) %>%

mutate(neighbors = centrality\_degree()) %>%

arrange(-neighbors)

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 2 (active)

## name neighbors

##

## 1 Tyrion-Lannister 54.

## 2 Cersei-Lannister 49.

## 3 Joffrey-Baratheon 49.

## 4 Robert-Baratheon 47.

## 5 Jaime-Lannister 45.

## 6 Sansa-Stark 44.

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 41 42 Undirected 43 7

## 2 41 60 Undirected 44 4

## 3 41 63 Undirected 52 3

## # ... with 795 more rows

**Grouping and clustering**

Another common operation is to group nodes based on the graph topology, sometimes referred to as community detection based on its commonality in social network analysis. All clustering algorithms from igraph is available in tidygraph using the group\_\* prefix. All of these functions return an integer vector with nodes (or edges) sharing the same integer being grouped together. <https://www.data-imaginist.com/2017/introducing-tidygraph/>

We can use ?group\_graph for an overview about all possible ways to cluster and group nodes. Here I am using group\_infomap(): **Group nodes by minimizing description length using**.

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(nodes) %>%

mutate(group = group\_infomap()) %>%

arrange(-group)

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 2 (active)

## name group

##

## 1 Arianne-Martell 7

## 2 Doran-Martell 7

## 3 Davos-Seaworth 6

## 4 Melisandre 6

## 5 Selyse-Florent 6

## 6 Stannis-Baratheon 6

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 32 33 Undirected 43 7

## 2 32 34 Undirected 44 4

## 3 32 36 Undirected 52 3

## # ... with 795 more rows

**Querying node types**

We can also query different node types (?node\_types gives us a list of options):

These functions all lets the user query whether each node is of a certain type. All of the functions returns a logical vector indicating whether the node is of the type in question. Do note that the types are not mutually exclusive and that nodes can thus be of multiple types.

Here, I am trying out node\_is\_center() (does the node have the minimal eccentricity in the graph) and node\_is\_keyplayer() to identify the top 10 key-players in the network. You can read more about the node\_is\_keyplayer() function in the manual for the influenceR package:

The “Key Player” family of node importance algorithms (Borgatti 2006) involves the selection of a metric of node importance and a combinatorial optimization strategy to choose the set S of vertices of size k that maximize that metric. This function implements KPP-Pos, a metric intended to identify k nodes which optimize resource diffusion through the net … <https://cran.r-project.org/web/packages/influenceR/>

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(nodes) %>%

mutate(center = node\_is\_center(),

keyplayer = node\_is\_keyplayer(k = 10))

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 3 (active)

## name center keyplayer

##

## 1 Aemon-Targaryen-(Maester-Aemon) FALSE FALSE

## 2 Aeron-Greyjoy FALSE FALSE

## 3 Aerys-II-Targaryen FALSE FALSE

## 4 Alliser-Thorne FALSE FALSE

## 5 Arianne-Martell FALSE FALSE

## 6 Arya-Stark FALSE FALSE

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 1 4 Undirected 43 7

## 2 1 13 Undirected 44 4

## 3 1 28 Undirected 52 3

## # ... with 795 more rows

**Node pairs**

Some statistics are a measure between two nodes, such as distance or similarity between nodes. In a tidy context one of the ends must always be the node defined by the row, while the other can be any other node. All of the node pair functions are prefixed with node\_\* and ends with \_from/\_to if the measure is not symmetric and \_with if it is; e.g. there’s both a node\_max\_flow\_to() and node\_max\_flow\_from() function while only a single node\_cocitation\_with() function. The other part of the node pair can be specified as an integer vector that will get recycled if needed, or a logical vector which will get recycled and converted to indexes with which(). This means that output from node type functions can be used directly in the calls. <https://www.data-imaginist.com/2017/introducing-tidygraph/>

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(nodes) %>%

mutate(dist\_to\_center = node\_distance\_to(node\_is\_center()))

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Node Data: 100 x 2 (active)

## name dist\_to\_center

##

## 1 Aemon-Targaryen-(Maester-Aemon) 1.

## 2 Aeron-Greyjoy 2.

## 3 Aerys-II-Targaryen 1.

## 4 Alliser-Thorne 1.

## 5 Arianne-Martell 2.

## 6 Arya-Stark 1.

## # ... with 94 more rows

## #

## # Edge Data: 798 x 5

## from to Type id weight

##

## 1 1 4 Undirected 43 7

## 2 1 13 Undirected 44 4

## 3 1 28 Undirected 52 3

## # ... with 795 more rows

**Edge betweenness**

Similarly to node metrics, we can calculate all kinds of edge metrics. Betweenness, for example, describes the shortest paths between nodes. More about what you can do with edges can be found with ?edge\_types and in the [tidygraph manual](https://cran.r-project.org/web/packages/tidygraph/tidygraph.pdf).

as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

activate(edges) %>%

mutate(centrality\_e = centrality\_edge\_betweenness())

## # A tbl\_graph: 100 nodes and 798 edges

## #

## # An undirected simple graph with 1 component

## #

## # Edge Data: 798 x 6 (active)

## from to Type id weight centrality\_e

##

## 1 1 4 Undirected 43 7 1.00

## 2 1 13 Undirected 44 4 30.2

## 3 1 28 Undirected 52 3 42.1

## 4 1 32 Undirected 53 20 0.

## 5 1 34 Undirected 54 5 35.2

## 6 1 41 Undirected 56 5 18.9

## # ... with 792 more rows

## #

## # Node Data: 100 x 1

## name

##

## 1 Aemon-Targaryen-(Maester-Aemon)

## 2 Aeron-Greyjoy

## 3 Aerys-II-Targaryen

## # ... with 97 more rows

**The complete code**

Now let’s combine what we’ve done above in true tidyverse fashion:

cooc\_all\_f\_graph <- as\_tbl\_graph(cooc\_all\_f, directed = FALSE) %>%

mutate(n\_rank\_trv = node\_rank\_traveller(),

neighbors = centrality\_degree(),

group = group\_infomap(),

center = node\_is\_center(),

dist\_to\_center = node\_distance\_to(node\_is\_center()),

keyplayer = node\_is\_keyplayer(k = 10)) %>%

activate(edges) %>%

filter(!edge\_is\_multiple()) %>%

mutate(centrality\_e = centrality\_edge\_betweenness())

We can also convert our active node or edge table back to a tibble:

cooc\_all\_f\_graph %>%

activate(nodes) %>% # %N>%

as.tibble()

## # A tibble: 100 x 7

## name n\_rank\_trv neighbors group center dist\_to\_center keyplayer

##

## 1 Aemon-Targa… 45 13. 2 FALSE 1. FALSE

## 2 Aeron-Greyj… 21 5. 5 FALSE 2. FALSE

## 3 Aerys-II-Ta… 11 12. 1 FALSE 1. FALSE

## 4 Alliser-Tho… 48 13. 2 FALSE 1. FALSE

## 5 Arianne-Mar… 29 4. 7 FALSE 2. FALSE

## 6 Arya-Stark 79 37. 1 FALSE 1. FALSE

## 7 Asha-Greyjoy 20 7. 5 FALSE 1. FALSE

## 8 Balon-Greyj… 18 11. 5 FALSE 2. FALSE

## 9 Barristan-S… 54 23. 3 FALSE 1. FALSE

## 10 Belwas 52 6. 3 FALSE 2. FALSE

## # ... with 90 more rows

cooc\_all\_f\_graph %>%

activate(edges) %>% # %E>%

as.tibble()

## # A tibble: 798 x 6

## from to Type id weight centrality\_e

##

## 1 1 4 Undirected 43 7 1.00

## 2 1 13 Undirected 44 4 30.2

## 3 1 28 Undirected 52 3 42.1

## 4 1 32 Undirected 53 20 0.

## 5 1 34 Undirected 54 5 35.2

## 6 1 41 Undirected 56 5 18.9

## 7 1 42 Undirected 57 25 0.

## 8 1 48 Undirected 58 110 0.

## 9 1 58 Undirected 60 5 24.5

## 10 1 71 Undirected 62 5 17.0

## # ... with 788 more rows

**Plotting with ggraph**

For plotting our graph object, we can make good use of the ggraph package:

ggraph is an extension of ggplot2 aimed at supporting relational data structures such as networks, graphs, and trees. While it builds upon the foundation of ggplot2 and its API it comes with its own self-contained set of geoms, facets, etc., as well as adding the concept of layouts to the grammar.

First, I am going to define a layout. There are lots of [options for layouts](https://www.data-imaginist.com/2017/ggraph-introduction-layouts/), here I am using a [Fruchterman-Reingold](http://igraph.org/r/doc/layout_with_fr.html) algorithm.

layout <- create\_layout(cooc\_all\_f\_graph,

layout = "fr")

The rest works like any ggplot2 function call, just that we use special geoms for our network, like geom\_edge\_density() to draw a shadow where the edge density is higher, geom\_edge\_link() to connect edges with a straight line, geom\_node\_point() to draw node points and geom\_node\_text() to draw the labels.

Here are three options of plotting the network with the metrics we just calculated:

ggraph(layout) +

geom\_edge\_density(aes(fill = weight)) +

geom\_edge\_link(aes(width = weight), alpha = 0.2) +

geom\_node\_point(aes(color = factor(group)), size = 10) +

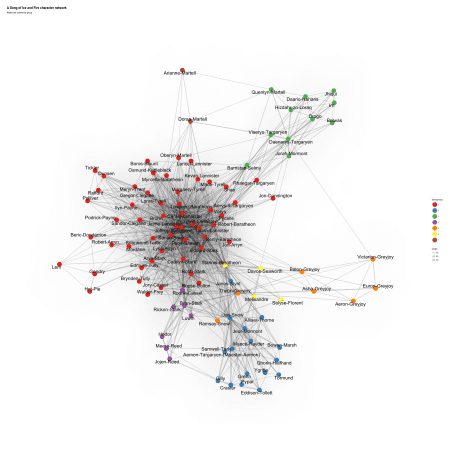
geom\_node\_text(aes(label = name), size = 8, repel = TRUE) +

scale\_color\_brewer(palette = "Set1") +

theme\_graph() +

labs(title = "A Song of Ice and Fire character network",

subtitle = "Nodes are colored by group")



Interestingly, many of the groups reflect the narrative perfectly: the men from the Night’s Watch are grouped together with the Wildlings, Stannis, Davos, Selyse and Melisandre form another group, the Greyjoys, Bran’s group in Winterfell before they left for the North, Dany and her squad and the Martells (except for Quentyn, who “belongs” to Dany – just like in the books ;-)). The big group around the remaining characters is the only one that’s not split up very well.

For the next graphs, I want specific colors form the RColorBrewer palette “Set1”:

cols <- RColorBrewer::brewer.pal(3, "Set1")

ggraph(layout) +

geom\_edge\_density(aes(fill = weight)) +

geom\_edge\_link(aes(width = weight), alpha = 0.2) +

geom\_node\_point(aes(color = factor(center), size = dist\_to\_center)) +

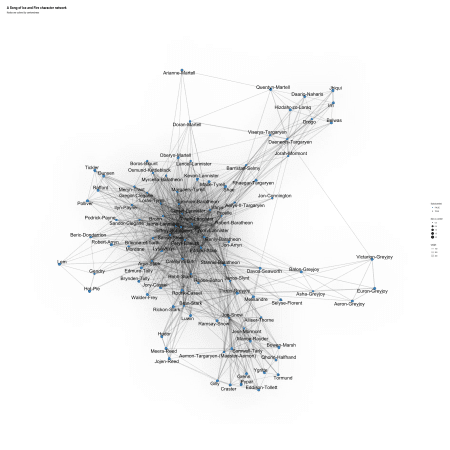
geom\_node\_text(aes(label = name), size = 8, repel = TRUE) +

scale\_colour\_manual(values = c(cols[2], cols[1])) +

theme\_graph() +

labs(title = "A Song of Ice and Fire character network",

subtitle = "Nodes are colored by centeredness")



In the next graph I plotted the center-most characters in red and the distance to center as node size. The two center characters across all books are Robert Baratheon and Tyrion Lannister. I had not expected Robert, since he dies pretty much right away but I guess he and his rebellion following Lyanna’s “abduction” is the main trigger for most of what happens in the books, so why not… And that Tyrion is the best character (and George RR Martin’s favorite) is a given, anyways! 

**Characters devided by books**

The second data set I am going to use is a comparison of character interactions in the five books.

**A little node on the side:** My original plan was to loop over the separate edge files for each book, concatenate them together with the information from which book they are and then plot them via faceting. This turned out to be a bad solution because I wanted to show the different key-players in each of the five books. So, instead of using one joined graph, I created separate graphs for every book and used the bind\_graphs() and facet\_nodes() functions to plot them together.

for (i in 1:5) {

cooc <- read\_csv(paste0("/Users/shiringlander/Documents/Github/Data/asoiaf/data//asoiaf-book", i, "-edges.csv")) %>%

mutate(book = paste0("book\_", i)) %>%

filter(Source %in% main\_ch\_l$name & Target %in% main\_ch\_l$name)

assign(paste0("coocs\_book\_", i), cooc)

}

The concepts are the same as above, here I want to know the key-players in each book:

cooc\_books\_1\_graph <- as\_tbl\_graph(coocs\_book\_1, directed = FALSE) %>%

mutate(book = "Book 1: A Game of Thrones",

keyplayer = node\_is\_keyplayer(k = 10))

cooc\_books\_2\_graph <- as\_tbl\_graph(coocs\_book\_2, directed = FALSE) %>%

mutate(book = "Book 2: A Clash of Kings",

keyplayer = node\_is\_keyplayer(k = 10))

cooc\_books\_3\_graph <- as\_tbl\_graph(coocs\_book\_3, directed = FALSE) %>%

mutate(book = "Book 3: A Storm of Swords",

keyplayer = node\_is\_keyplayer(k = 10))

cooc\_books\_4\_graph <- as\_tbl\_graph(coocs\_book\_4, directed = FALSE) %>%

mutate(book = "Book 4: A Feast for Crows",

keyplayer = node\_is\_keyplayer(k = 10))

cooc\_books\_5\_graph <- as\_tbl\_graph(coocs\_book\_5, directed = FALSE) %>%

mutate(book = "Book 5: A Dance with Dragons",

keyplayer = node\_is\_keyplayer(k = 10))

And let’s combine and plot the key-players:

cooc\_books\_1\_graph %>%

bind\_graphs(cooc\_books\_2\_graph) %>%

bind\_graphs(cooc\_books\_3\_graph) %>%

bind\_graphs(cooc\_books\_4\_graph) %>%

bind\_graphs(cooc\_books\_5\_graph) %>%

ggraph(layout = "fr") +

facet\_nodes( ~ book, scales = "free", ncol = 1) +

geom\_edge\_density(aes(fill = weight)) +

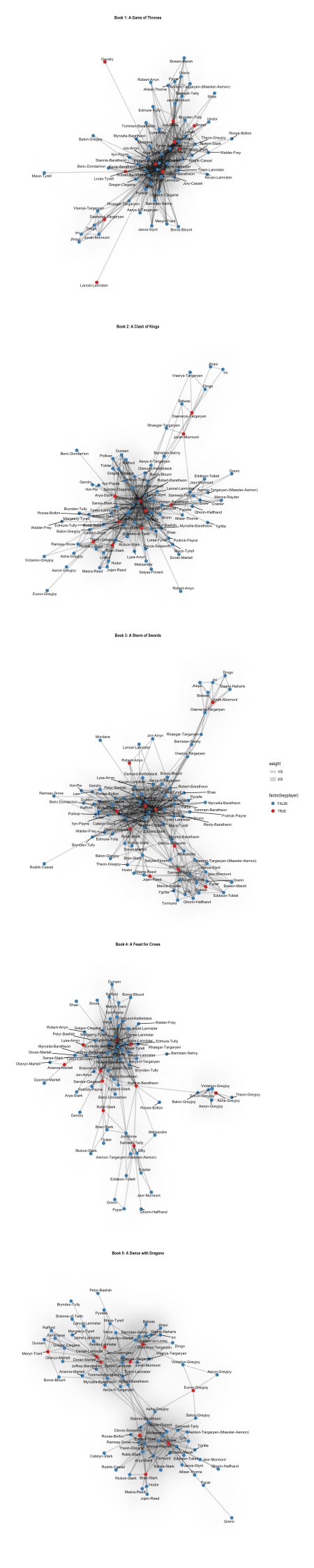
geom\_edge\_link(aes(edge\_width = weight), alpha = 0.2) +

geom\_node\_point(aes(color = factor(keyplayer)), size = 3) +

geom\_node\_text(aes(label = name), color = "black", size = 3, repel = TRUE) +

theme\_graph() +

scale\_colour\_manual(values = c(cols[2], cols[1]))



The networks and key-players of the five different books also offer a few surprises but also a lot that reflects the narrative quite well. I’m not going to go into details here as that would go a bit too far for an R-related blog – but if you are interested in in-depth discussions about the books, email me… 

**More info**

You can find more info about

* tidygraph [here](https://cran.r-project.org/web/packages/tidygraph)
* ggraph [here](https://cran.r-project.org/web/packages/ggraph)
* influenceR [here](https://cran.r-project.org/web/packages/influenceR)
* and DataCamp has a Python project for the same data set [here](https://www.datacamp.com/projects/76?tap_a=5644-dce66f&tap_s=10907-287229)

sessionInfo()

## R version 3.4.3 (2017-11-30)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.3

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

##

## locale:

## [1] de\_DE.UTF-8/de\_DE.UTF-8/de\_DE.UTF-8/C/de\_DE.UTF-8/de\_DE.UTF-8

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] bindrcpp\_0.2 ggraph\_1.0.1 tidygraph\_1.1.0

## [4] forcats\_0.3.0 stringr\_1.3.0 dplyr\_0.7.4

## [7] purrr\_0.2.4 tidyr\_0.8.0 tibble\_1.4.2

## [10] ggplot2\_2.2.1.9000 tidyverse\_1.2.1 readr\_1.1.1

##

## loaded via a namespace (and not attached):

## [1] nlme\_3.1-131.1 bitops\_1.0-6 lubridate\_1.7.3

## [4] RColorBrewer\_1.1-2 httr\_1.3.1 prabclus\_2.2-6

## [7] rprojroot\_1.3-2 tools\_3.4.3 backports\_1.1.2

## [10] utf8\_1.1.3 R6\_2.2.2 KernSmooth\_2.23-15

## [13] lazyeval\_0.2.1 colorspace\_1.3-2 trimcluster\_0.1-2

## [16] nnet\_7.3-12 withr\_2.1.1.9000 tidyselect\_0.2.4

## [19] gridExtra\_2.3 mnormt\_1.5-5 compiler\_3.4.3

## [22] cli\_1.0.0 rvest\_0.3.2 TSP\_1.1-5

## [25] influenceR\_0.1.0 xml2\_1.2.0 labeling\_0.3

## [28] bookdown\_0.7 diptest\_0.75-7 caTools\_1.17.1

## [31] scales\_0.5.0.9000 DEoptimR\_1.0-8 robustbase\_0.92-8

## [34] mvtnorm\_1.0-7 psych\_1.7.8 digest\_0.6.15

## [37] foreign\_0.8-69 rmarkdown\_1.8 pkgconfig\_2.0.1

## [40] htmltools\_0.3.6 rlang\_0.2.0.9000 readxl\_1.0.0

## [43] rstudioapi\_0.7 bindr\_0.1 jsonlite\_1.5

## [46] mclust\_5.4 gtools\_3.5.0 dendextend\_1.7.0

## [49] magrittr\_1.5 modeltools\_0.2-21 Rcpp\_0.12.15

## [52] munsell\_0.4.3 viridis\_0.5.0 stringi\_1.1.6

## [55] whisker\_0.3-2 yaml\_2.1.17 MASS\_7.3-49

## [58] flexmix\_2.3-14 gplots\_3.0.1 plyr\_1.8.4

## [61] grid\_3.4.3 parallel\_3.4.3 gdata\_2.18.0

## [64] ggrepel\_0.7.0 crayon\_1.3.4 udunits2\_0.13

## [67] lattice\_0.20-35 haven\_1.1.1 hms\_0.4.1

## [70] knitr\_1.20 pillar\_1.2.1 igraph\_1.1.2

## [73] fpc\_2.1-11 stats4\_3.4.3 reshape2\_1.4.3

## [76] codetools\_0.2-15 glue\_1.2.0 gclus\_1.3.1

## [79] evaluate\_0.10.1 blogdown\_0.5 modelr\_0.1.1

## [82] tweenr\_0.1.5 foreach\_1.4.4 cellranger\_1.1.0

## [85] gtable\_0.2.0 kernlab\_0.9-25 assertthat\_0.2.0

## [88] xfun\_0.1 ggforce\_0.1.1 broom\_0.4.3

## [91] class\_7.3-14 viridisLite\_0.3.0 seriation\_1.2-3

## [94] iterators\_1.0.9 registry\_0.5 units\_0.5-1

## [97] cluster\_2.0.6