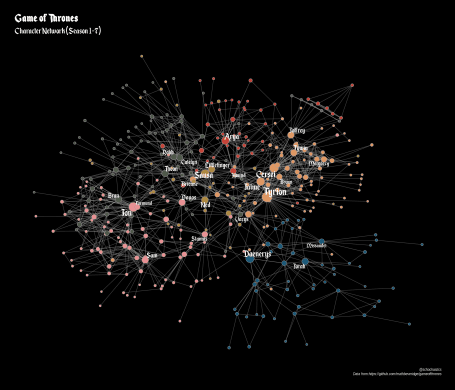
This post introduces the new R package graphlayouts which is available on CRAN since a few days. We will use network data from the *Game of Thrones* TV series (seemed timely at the time of writing)  
to illustrate the core layout algorithms of the package. Most of the algorithms use  
stress majorization as its basis. Here, I will only focus on the practical aspects of the package.



library(tidyverse)

library(igraph)

library(graphlayouts)

library(ggraph)

library(extrafont)

loadfonts()

**Preparing the Data**

To illustrate the functionality of graphlayouts we will use data compiled from the  
TV Show *Game of Thrones*, specifically the character interaction networks. We do this using the map() function of purrr.

edges <- paste0("https://raw.githubusercontent.com/mathbeveridge/gameofthrones/master/data/got-s",1:7,"-edges.csv")

edges\_tbl <- map(edges,read\_csv)

Next, we transform the raw edgelists into igraph objects, again using map.

got\_graphs <- map(1:7,function(x) {

g <- graph\_from\_data\_frame(edges\_tbl[x],directed = F)

g$title <- paste0("Season ",x)

g

})

Lastly, we transform and add some node variables. First, we change the character names from  
upper case to title case. Then we compute a clustering, and the total number of interactions per character.

mutate\_graph <- function(x){

V(x)$name <- str\_replace\_all(V(x)$name,"\\\_"," ") %>%

str\_to\_title()

clu <- cluster\_louvain(x)

V(x)$clu <- as.character(clu$membership)

V(x)$size <- graph.strength(x)

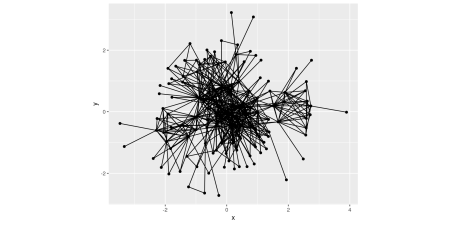
x

}

got\_graphs <- map(got\_graphs,mutate\_graph)

got\_graphs now contains all seven season networks in a list. graphlayouts contains  
the function qgraph() which can be used to get a very rough visualization of the network,  
without the need of lengthy ggraph code.

qgraph(got\_graphs[[1]])

  
The function is very similar to qplot() from ggplot2, yet it does not take any additional arguments.  
This is planned for a later release.

**Visualizing each Season**

The core layout algorithm of graphlayouts is implemented in the function layout\_with\_stress(), which is also called in qgraph by the way. The package also contains a convenience function to work smoothly with ggraph. Instead of

xy <- layout\_with\_stress(x)

ggraph(x,layout="manual",node.positions=data.frame(x=xy[,1],y=xy[,2]))+...

you can do

ggraph(x,layout = "stress")+...

Creating nice plots is now “just” a matter of stitching some ggraph code together.  
For our character networks, we define a function that does that for all seasons simultaneously.  
I usually would not use hard to read fonts for visualizations but I thought *Enchanted Land* (available [here](https://www.dafont.com/enchanted-land.font)) kind of fits here.

got\_palette <- c("#1A5878","#C44237","#AD8941","#E99093",

"#50594B","#DE9763","#5D5F5D","#E2E9E7")

plot\_graph <- function(x){

ggraph(x,layout = "stress")+

geom\_edge\_link0(aes(width=Weight),edge\_colour="grey66")+

geom\_node\_point(aes(fill=clu,size=size),shape=21,col="grey25")+

geom\_node\_text(aes(size=size,label=name),family = "Enchanted Land",repel=F)+

scale\_edge\_width\_continuous(range=c(0.1,1.5))+

scale\_size\_continuous(range=c(1,8))+

scale\_fill\_manual(values=got\_palette)+

theme\_graph(title\_family = "Enchanted Land",

title\_size = 20)+

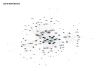
labs(title=paste0("Game of Thrones (",x$title,")"))+

theme(legend.position = "none")

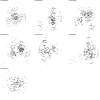
}

got\_plot <- map(got\_graphs,plot\_graph)

got\_plot[[1]]

  
*(Open image in a new tab to view it in full size)*

The advantages of stress based layouts are outlined. It has been my goto  
layout algorithm since years and I’d wish that more SNA software would implement it.  
The below figure shows all seven network in one plot.



**Focus on Characters**

While “stress” is the key graph layout in the package, there are other, more specialized layouts  
that can be used for different purposes. layout\_with\_focus() for instance allows you to focus  
the network on a specific character and order all other nodes in concentric circles (depending on distance) around it. Here is Season one with focus on Ned Stark.

ggraph(got\_graphs[[1]],layout="focus",v=1)+

geom\_edge\_link0(aes(width=Weight),edge\_colour="grey66")+

geom\_node\_point(aes(fill=clu,size=size),shape=21,col="grey25")+

geom\_node\_text(aes(filter=(name=="Ned"),size=size,label=name),family = "Enchanted Land",repel=F)+

scale\_edge\_width\_continuous(range=c(0.2,0.9))+

scale\_size\_continuous(range=c(1,8))+

scale\_fill\_manual(values=got\_palette)+

theme\_graph(title\_family = "Enchanted Land",

subtitle\_family = "Enchanted Land",

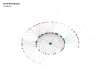
title\_size = 20,

subtitle\_size = 16)+

labs(title=paste0("Game of Thrones (Season 1)"),

subtitle = "Focus on Ned Stark")+

theme(legend.position = "none")



Based on a similar principle is layout\_with\_centrality(). You can specify any centrality index (or numeric vector for that matter), and create a concentric layout where the most central nodes are put in the center.

ggraph(got\_graphs[[1]],layout="centrality",cent=graph.strength(got\_graphs[[1]]))+

geom\_edge\_link0(aes(width=Weight),edge\_colour="grey66")+

geom\_node\_point(aes(fill=clu,size=size),shape=21,col="grey25")+

geom\_node\_text(aes(size=size,label=name),family = "Enchanted Land",repel=F)+

scale\_edge\_width\_continuous(range=c(0.2,0.9))+

scale\_size\_continuous(range=c(1,8))+

scale\_fill\_manual(values=got\_palette)+

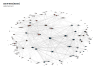
theme\_graph(title\_family = "Enchanted Land",

title\_size = 20)+

labs(title=paste0("Game of Thrones (Season 1)"),

subtitle = "weighted degree layout")+

theme(legend.position = "none")



To get someone else in the center than Ned Stark, here is season seven.

ggraph(got\_graphs[[7]],layout="centrality",cent=graph.strength(got\_graphs[[7]]))+

geom\_edge\_link0(aes(width=Weight),edge\_colour="grey66")+

geom\_node\_point(aes(fill=clu,size=size),shape=21,col="grey25")+

geom\_node\_text(aes(size=size,label=name),family = "Enchanted Land",repel=F)+

scale\_edge\_width\_continuous(range=c(0.2,0.9))+

scale\_size\_continuous(range=c(1,8))+

scale\_fill\_manual(values=got\_palette)+

theme\_graph(title\_family = "Enchanted Land",

title\_size = 20)+

labs(title=paste0("Game of Thrones (Season 7)"),

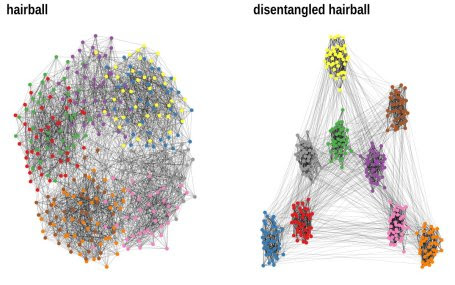
subtitle = "weighted degree layout")+

theme(legend.position = "none")



**Combining all Seasons**

The last important layout algorithm is layout\_as\_backbone() which is tailored to work  
with “hairball” networks that may contain a hidden group structure. The below plot shows its impressive performance.



Of course the network used in the above example is specifically tailored to show this power. So  
what about real networks?

We will put all seasons together and create one big GoT network that contains all  
character interactions from Season 1-7 to illustrate its performance with our example.

got\_all <- map\_dfr(edges\_tbl,bind\_rows) %>%

group\_by(Source,Target) %>%

summarise(Weight=sum(Weight)) %>%

ungroup() %>%

rename(weight=Weight) %>%

mutate\_if(is.character,function(x) str\_replace\_all(x,"\\\_"," ") %>%

str\_to\_title()) %>%

graph\_from\_data\_frame(directed=FALSE)

clu <- cluster\_louvain(got\_all)

V(got\_all)$clu <- as.character(clu$membership)

V(got\_all)$size <- graph.strength(got\_all)

First, let’s check what it looks like using layout\_with\_stress().

ggraph(got\_all,layout="stress")+

geom\_edge\_link0(aes(width=weight),edge\_colour="grey66",alpha=0.5)+

geom\_node\_point(aes(fill=clu,size=size),shape=21)+

scale\_edge\_width\_continuous(range=c(0.8,1.8))+

scale\_size\_continuous(range=c(2,10))+

scale\_fill\_manual(values=got\_palette)+

theme\_graph(title\_family = "Enchanted Land",

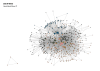
subtitle\_family = "Enchanted Land",

title\_size = 20,

subtitle\_size = 16)+

labs(title=paste0("Game of Thrones"),subtitle = "Character Network (Season 1-7)")+

theme(legend.position = "none")



It is clearly hard to see anything here, since the network is too dense. Enter,  
layout\_as\_backbone(). The algorithm itself is rather involved but some of the key points are:

* Find strongly embedded edges depending on graph motifs
* Compute the union of all maximum spanning trees to “hold the network together”

layout\_as\_backbone() takes a parameter to\_keep which determines the percentage of edges  
to keep for the layout calculation. In our case, we will the 20% most embedded edges.  
(The parameter always requires some experimenting to find out what works best). Note that this does not mean that we throw away the rest of the edges. They are just not used to calculate the layout.  
But you can still do that afterwards (as we will do here) since the function returns a logical  
vector which indicates if an edge belongs to the backbone or not.

bb <- layout\_as\_backbone(got\_all,keep=0.2)

E(got\_all)$backbone <- F

E(got\_all)$backbone[bb$backbone] <- T

ggraph(got\_all,layout="manual",node.positions=data.frame(x=bb$xy[,1],y=bb$xy[,2]))+

geom\_edge\_link0(aes(filter=backbone,width=weight),edge\_colour="grey66",alpha=0.5)+

geom\_node\_point(aes(fill=clu,size=size),shape=21)+

geom\_node\_text(aes(filter=(size>=800),size=size,label=name),

family = "Enchanted Land",repel=T)+

scale\_edge\_width\_continuous(range=c(0.8,1.8))+

scale\_size\_continuous(range=c(2,10))+

scale\_fill\_manual(values=got\_palette)+

theme\_graph(title\_family = "Enchanted Land",

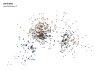
subtitle\_family = "Enchanted Land",

title\_size = 20,

subtitle\_size = 16)+

labs(title=paste0("Game of Thrones"),subtitle = "Character Network (Season 1-7)")+

theme(legend.position = "none")



There is definitely not as much structure going on as in the contrived example. The algorithm can’t  
uncover a hidden structure if there is no such thing to uncover. Yet, the layout still  
reveals some structure and clearly enhances readability over the stress based layout.

**Addendum**

This post introduced only the core layout algorithms of graphlayouts. Now that graphlayouts is out, I will continue working on that package.