Tidygraph

Load library

```
library(dplyr)
library(tidygraph)
library(ggraph)
library(igraph)
```

Load data

```
df = read.table("data/data.tsv", header = T)
head(df)
```

```
##   node1 node2
## 1  POX1  FAA2
## 2  FAA1  POX1
## 3  TGL3  YJU3
## 4  TGL4  YJU3
## 5  TGL3  TGL4
## 6  FAA4  POX1
```

.

```
nodes = read.csv("data/dolphin_nodes.csv", header = T)
edges = read.csv("data/dolphin_edges.csv", header = T)
n = nrow(nodes)
m = nrow(edges)

# mutate edges and nodes
edge_type = sample(c("love", "friendship"), m, replace = TRUE)
edge_weight = runif(m, 1, 10)
edges = mutate(edges, type = edge_type, weight = edge_weight)
head(edges)
nodes = mutate(nodes, id = 1:n) %>% select(id, everything())

# degree of nodes (number of ties for each dolphin)
tb = tibble(v = c(1:n, edges$x, edges$y))
d = count(tb, v)$n - 1
nodes = mutate(nodes, degree = d)

# create graph from data frames
```

```
g = graph_from_data_frame(edges, directed = FALSE, nodes)
plot(g)
```

The tbl_graph object

From the definition [1]

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.

Basically a tbl_graph object consists of two tibbles/data frames

- Node Data
- Edge data

Create tbl_graph object

There are 2 key functions that can be used to create tbl_graph object.

- **tbl_graph()**. Creates a network object from nodes and edges data
- **as_tbl_graph()**. Converts network data and objects to a tbl_graph network. The following list gives the packages/classes that can currently be converted to tbl_graphs, using the as_tbl_graph function:
  - data.frame, list, matrix from base
  - igraph from igraph
  - network from network
  - dendrogram and hclust from stats
  - Node from data.tree
  - phylo and evonet from ape
  - graphNEL, graphAM, graphBAM from graph (in Bioconductor)

Convert igraph object to tbl_graph

```
g = graph_from_data_frame(df) %>% as_tbl_graph()
print(g)
```

```
## # A tbl_graph: 11 nodes and 32 edges
## #
## # A directed acyclic simple graph with 1 component
## #
## # Node Data: 11 x 1 (active)
##   name
##   <chr>
## 1 POX1
## 2 FAA1
## 3 TGL3
## 4 TGL4
## 5 FAA4
## 6 FAS1
## # ... with 5 more rows
## #
## # Edge Data: 32 x 2
##    from    to
##   <int> <int>
## 1     1     7
## 2     2     1
## 3     3     8
## # ... with 29 more rows
```

You can see that of the two tibbles associated with tbl_graph, node tibble is at present active. It means any manipulation will be applied to nodes. To make edge tibble as active, we have to activate the edge tibble. We will explain this in next few sections.

There are 11 nodes and 32 edges

Add vertex and node properties

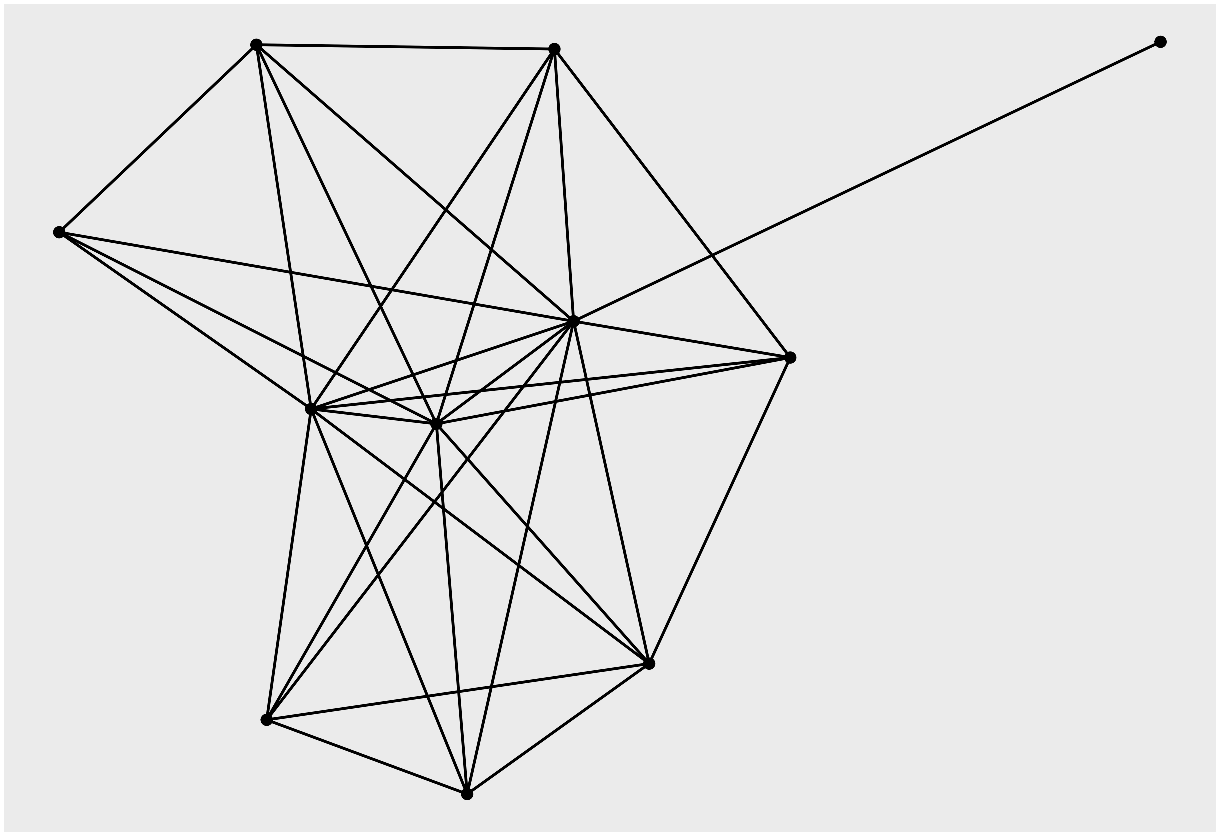
```
# vertex attributes
g = set_vertex_attr(graph = g, name = "type", value = c(rep("type1", 6), rep("type2", 3), rep("type3", 

# Edge attributes
g = set_edge_attr(graph = g, name = "type", value = c(rep("strong", 10), rep("weak", 22)))

print(g)
```

```
## # A tbl_graph: 11 nodes and 32 edges
## #
## # A directed acyclic simple graph with 1 component
## #
## # Node Data: 11 x 2 (active)
##   name  type
##   <chr> <chr>
## 1 POX1  type1
## 2 FAA1  type1
## 3 TGL3  type1
## 4 TGL4  type1
## 5 FAA4  type1
## 6 FAS1  type1
## # ... with 5 more rows
## #
## # Edge Data: 32 x 3
##    from    to type
##   <int> <int> <chr>
## 1     1     7 strong
## 2     2     1 strong
## 3     3     8 strong
## # ... with 29 more rows
```

Use ggraph to plot

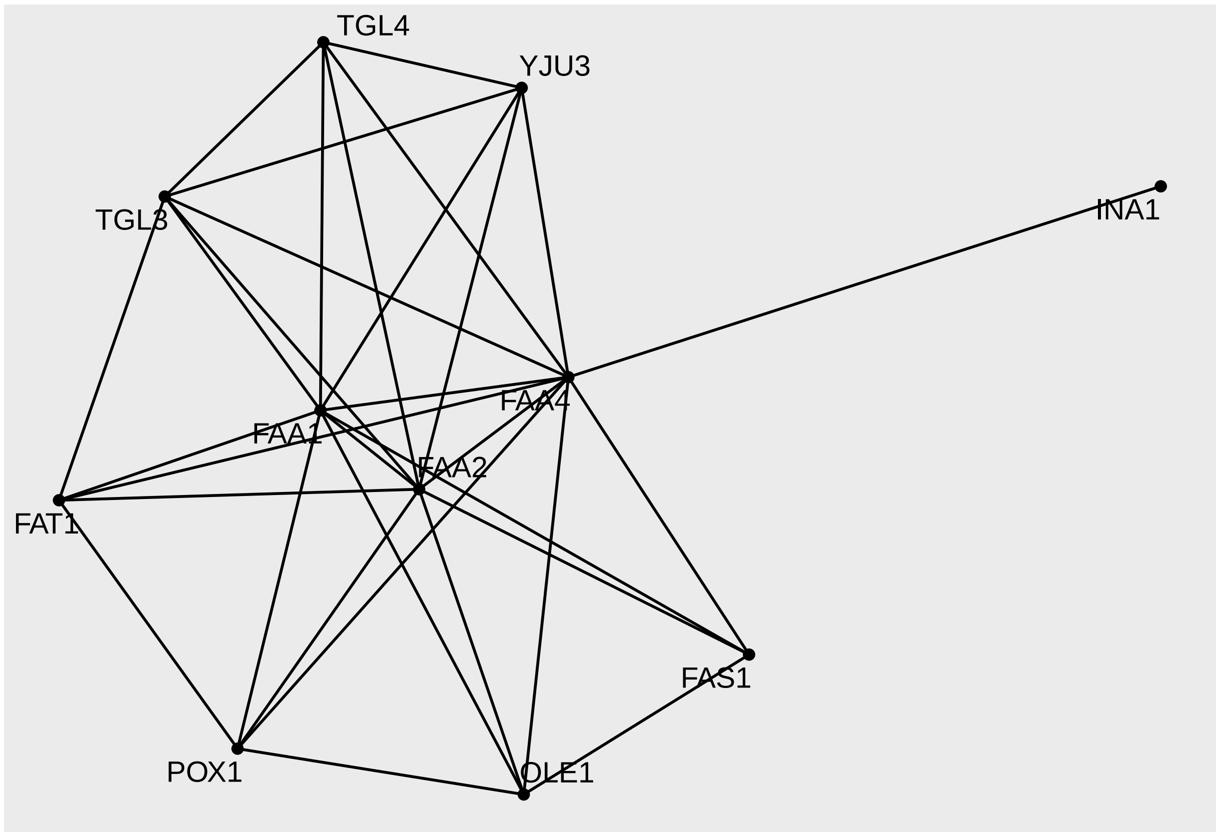
- geom_node_point(): Show nodes as points
- geom_edge_link(): Draw edges as straight lines between nodes

```
ggraph(graph = g, layout = "fr") +
  geom_node_point() +
  geom_edge_link()
```



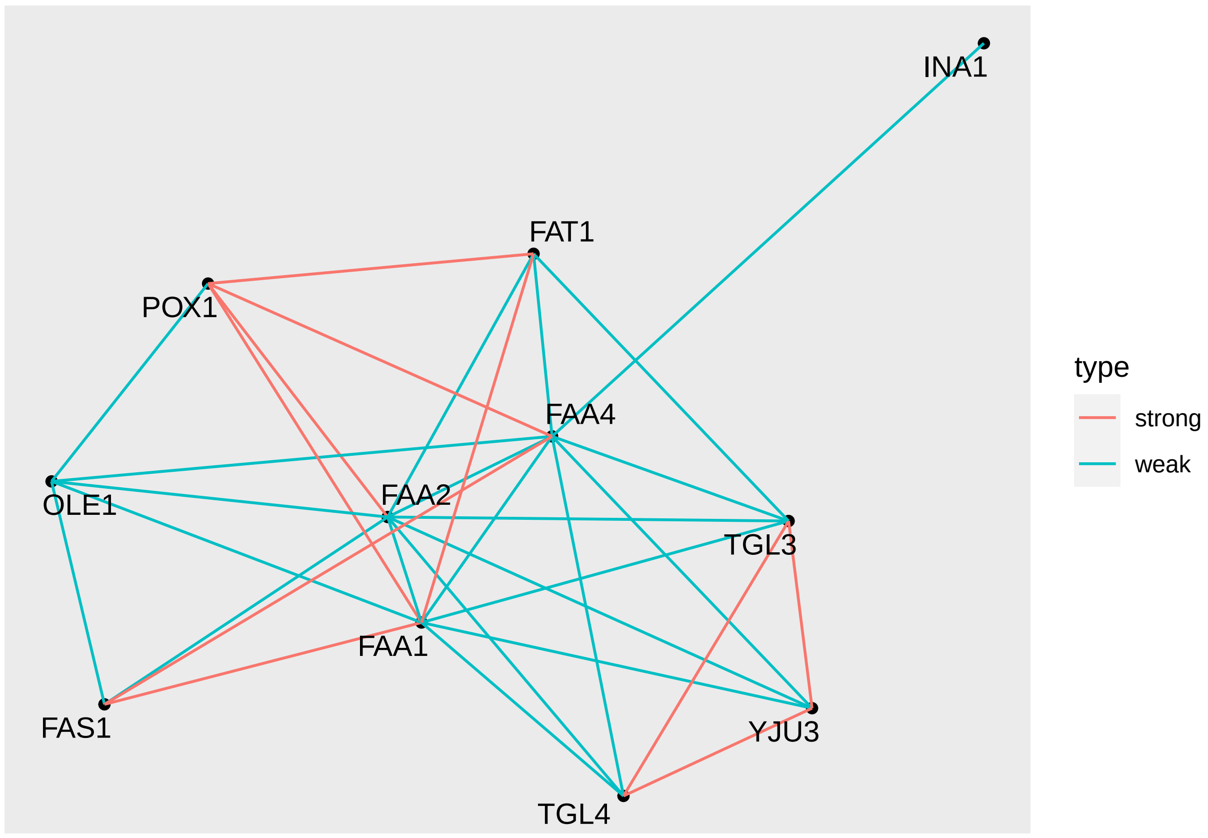
Add node names

```
ggraph(graph = g, layout = "fr") +
  geom_node_point() +
  geom_edge_link() +
  geom_node_text(aes(label = name), repel = T)
```



Color edges w.r.t type

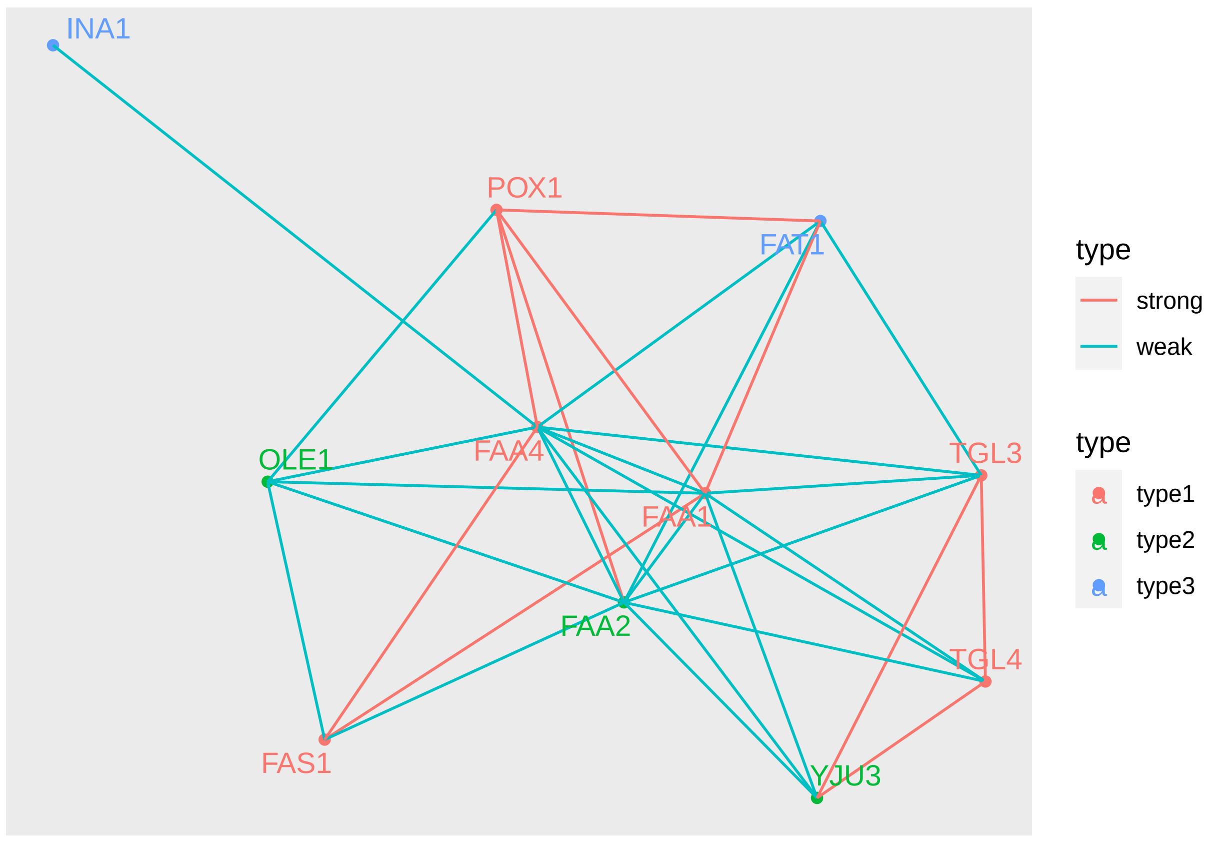
```
ggraph(graph = g, layout = "fr") +
  geom_node_point() +
  geom_edge_link(aes(color = type)) +
  geom_node_text(aes(label = name), repel = T)
```



Color nodes and text w.r.t type

Use color aes mapping in geom_node_point and geom_node_text

```
ggraph(graph = g, layout = "fr") +
  geom_node_point(aes(color = type)) +
  geom_edge_link(aes(color = type)) +
  geom_node_text(aes(label = name, color = type), repel = T)
```



.

```
nodes = read.csv("data/nodes.csv", header =T)
head(nodes)
```

```
##   id            name
## 1  1    Jamal Zougam
## 2  2 Mohamed Bekkali
## 3  3  Mohamed Chaoui
## 4  4     Vinay Kholy
## 5  5    Suresh Kumar
## 6  6 Mohamed Chedadi
```

```
dim(nodes)
```

```
## [1] 64  2
```

```
edges = read.csv("data/ties.csv", header =T)
head(edges)
```

```
##   from to weight
## 1    1  2      1
## 2    1  3      3
## 3    1  4      1
## 4    1  5      1
## 5    1  6      1
## 6    1  7      4
```

```
dim(edges)
```

```
## [1] 243   3
```

```
g = graph_from_data_frame(d = edges, directed = FALSE, vertices = nodes) %>% as_tbl_graph()
g$name = "Madrid network"

# Add id attributes. Set a node attribute id as the sequence from 1 to the number of nodes of the netwo
V(g)$id <- 1:V(g)
```

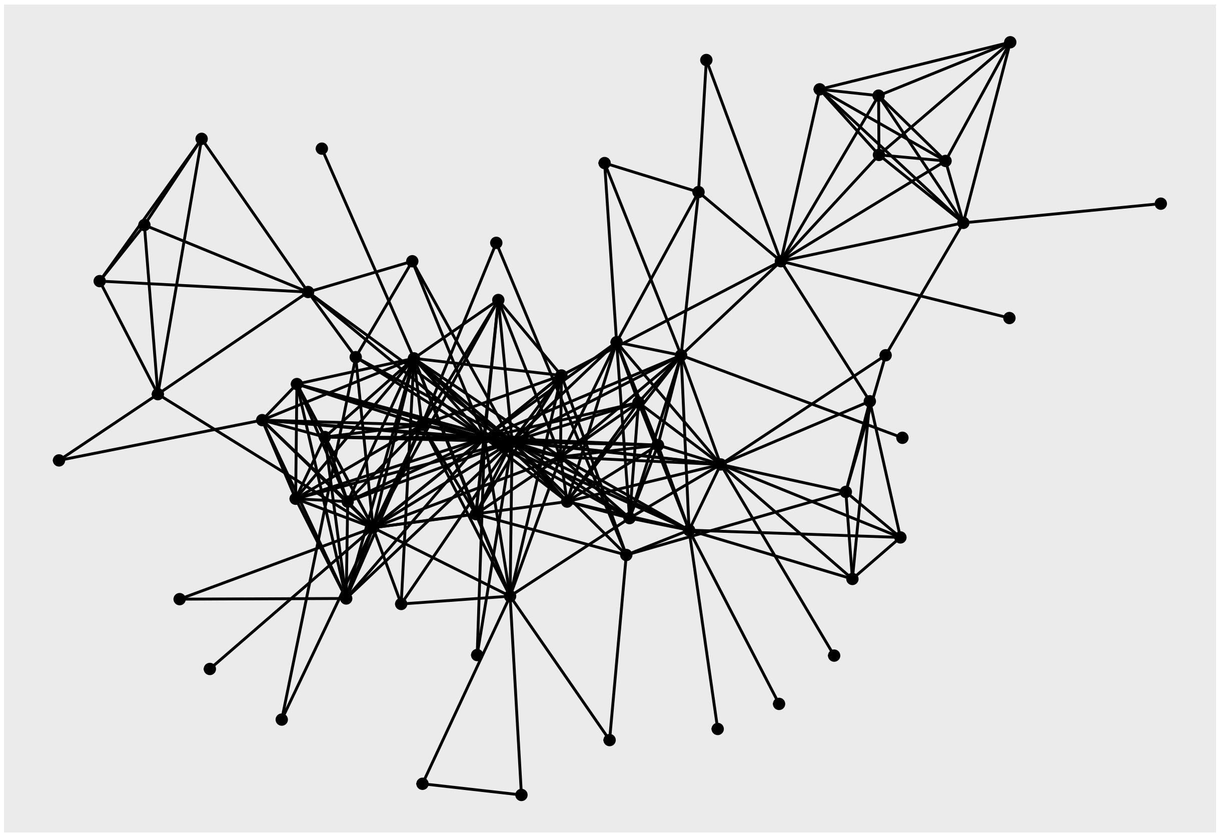
```
## Warning in 1:V(g): numerical expression has 64 elements: only the first used
```

```
# Add node attributes
V(g)$nodetype = sample(c("type1", "type2"), size = vcount(g), replace = T)

print(g)
```

```
## # A tbl_graph: 64 nodes and 243 edges
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 64 x 3 (active)
##   name                 id nodetype
##   <chr>             <int> <chr>
## 1 Jamal Zougam          1 type1
## 2 Mohamed Bekkali       1 type1
## 3 Mohamed Chaoui        1 type1
## 4 Vinay Kholy           1 type1
## 5 Suresh Kumar          1 type1
## 6 Mohamed Chedadi       1 type2
## # ... with 58 more rows
## #
## # Edge Data: 243 x 3
##    from    to weight
##   <int> <int>  <int>
## 1     1     2      1
## 2     1     3      3
## 3     1     4      1
## # ... with 240 more rows
```

visualize the network

```
ggraph(g, layout = "with_kk") +
    geom_edge_link() +
    geom_node_point()
```

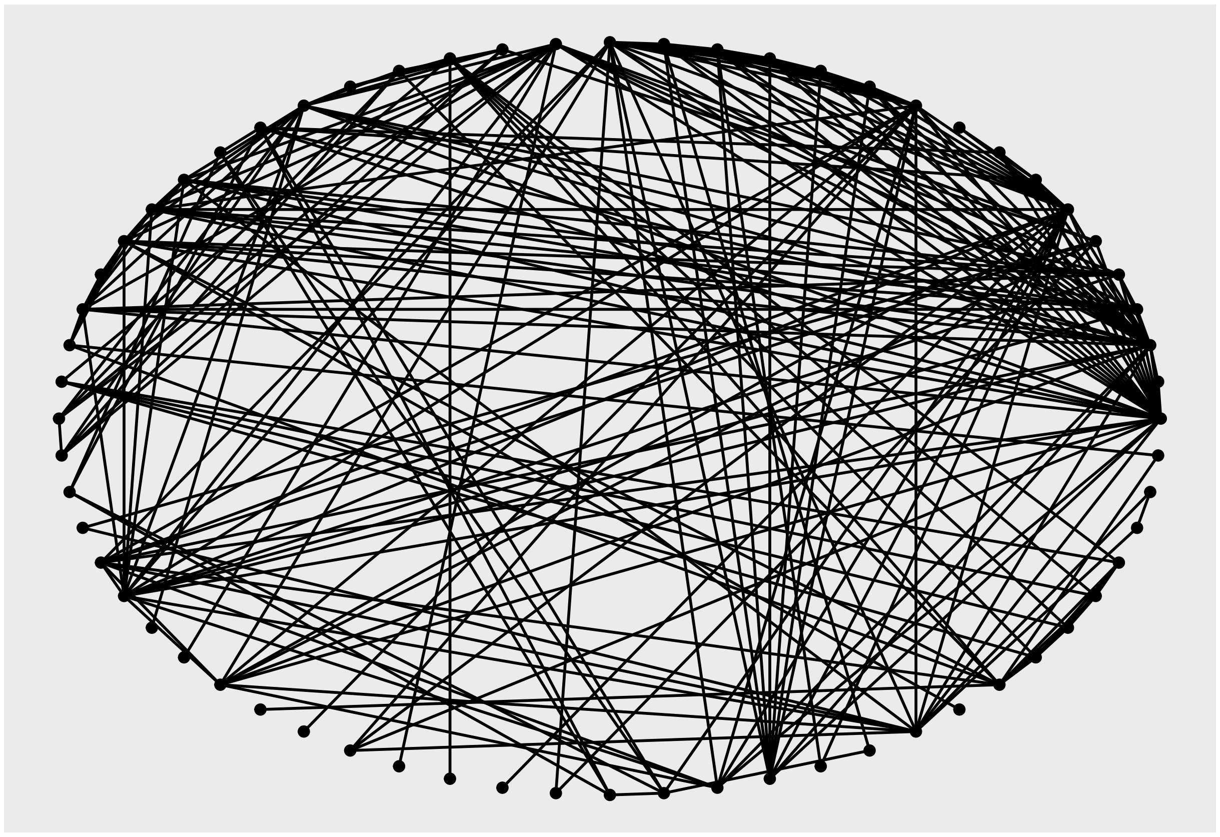


Layout

Explore igraph layout functions. Type ?layout_ for more help. Ignore the layout_ in the function name and use this as value e.g. use in_circle instead of layout_in_circle

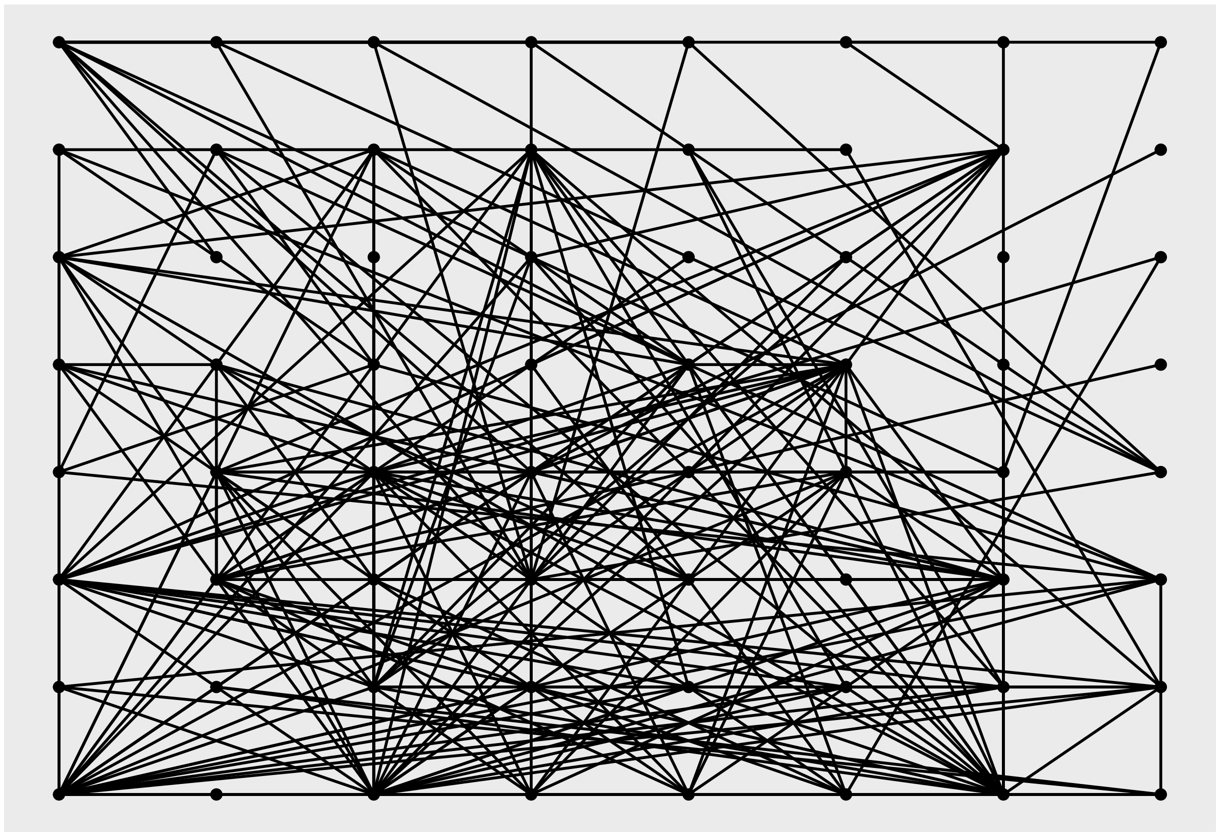
in_circle

```
ggraph(g, layout = "in_circle") +
    geom_edge_link() +
    geom_node_point()
```



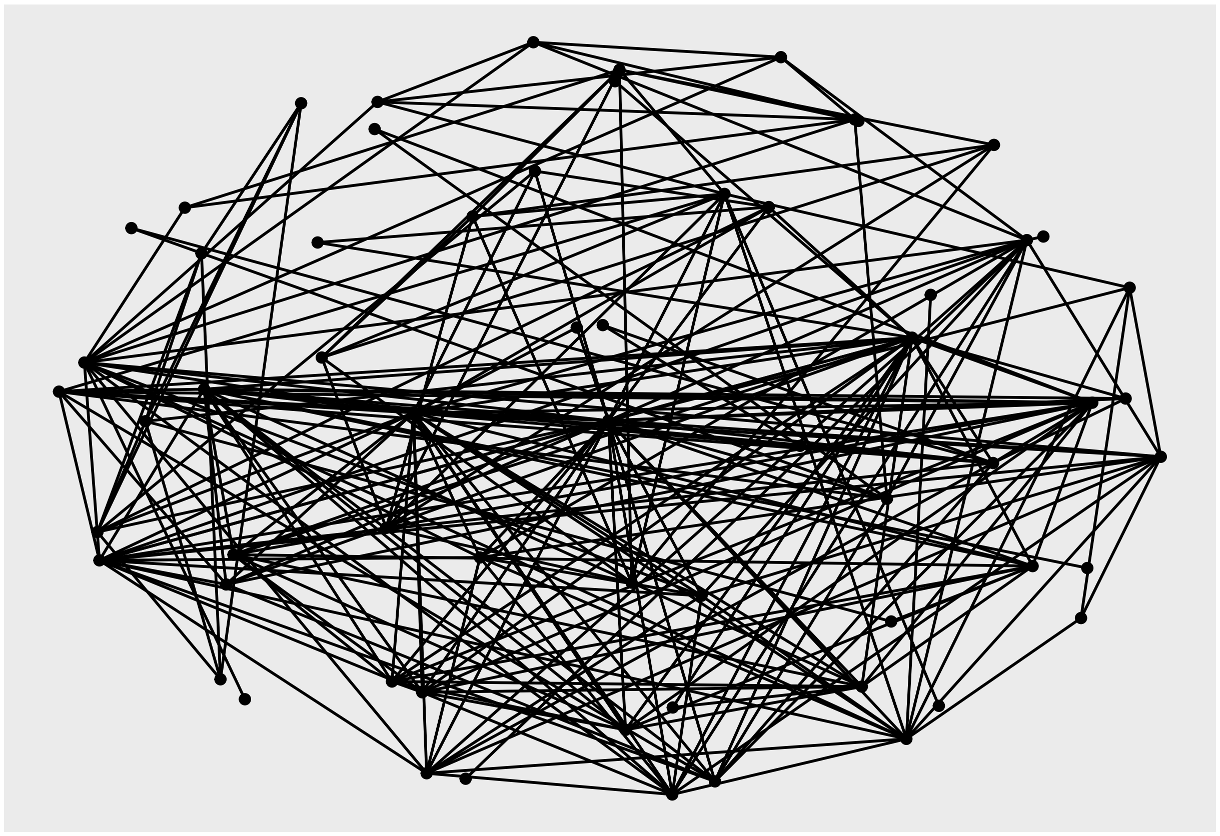
on_grid

```
ggraph(g, layout = "on_grid") +
    geom_edge_link() +
    geom_node_point()
```



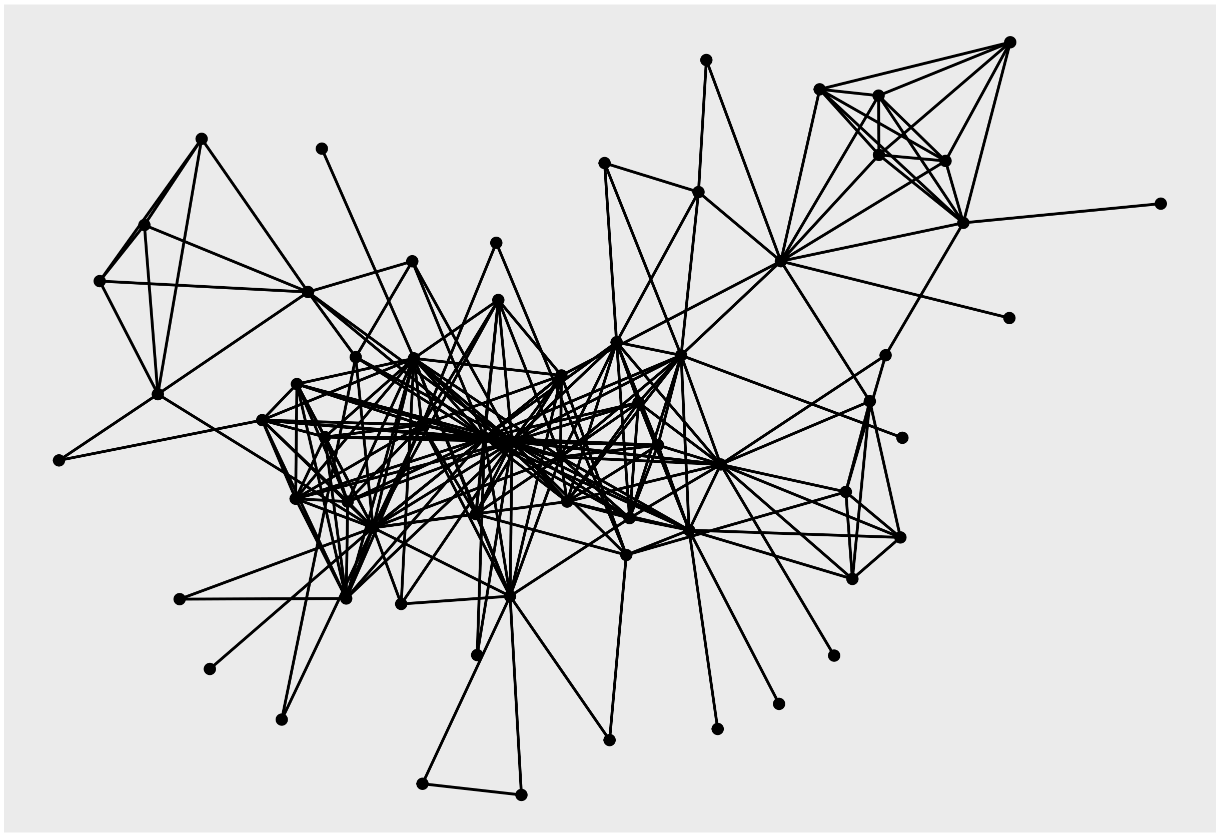
on sphere

```
ggraph(g, layout = "on_sphere") +
    geom_edge_link() +
    geom_node_point()
```



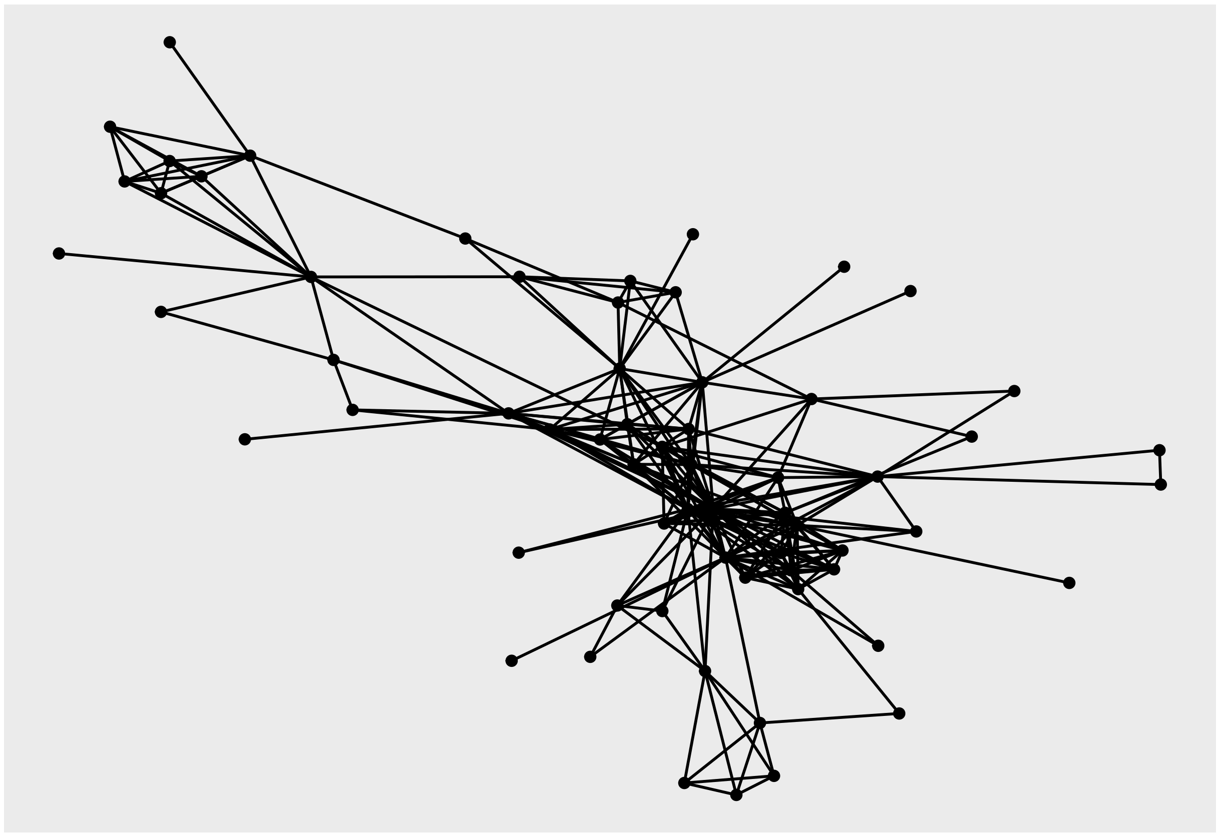
with_kk

```
ggraph(g, layout = "with_kk") +
    geom_edge_link() +
    geom_node_point()
```



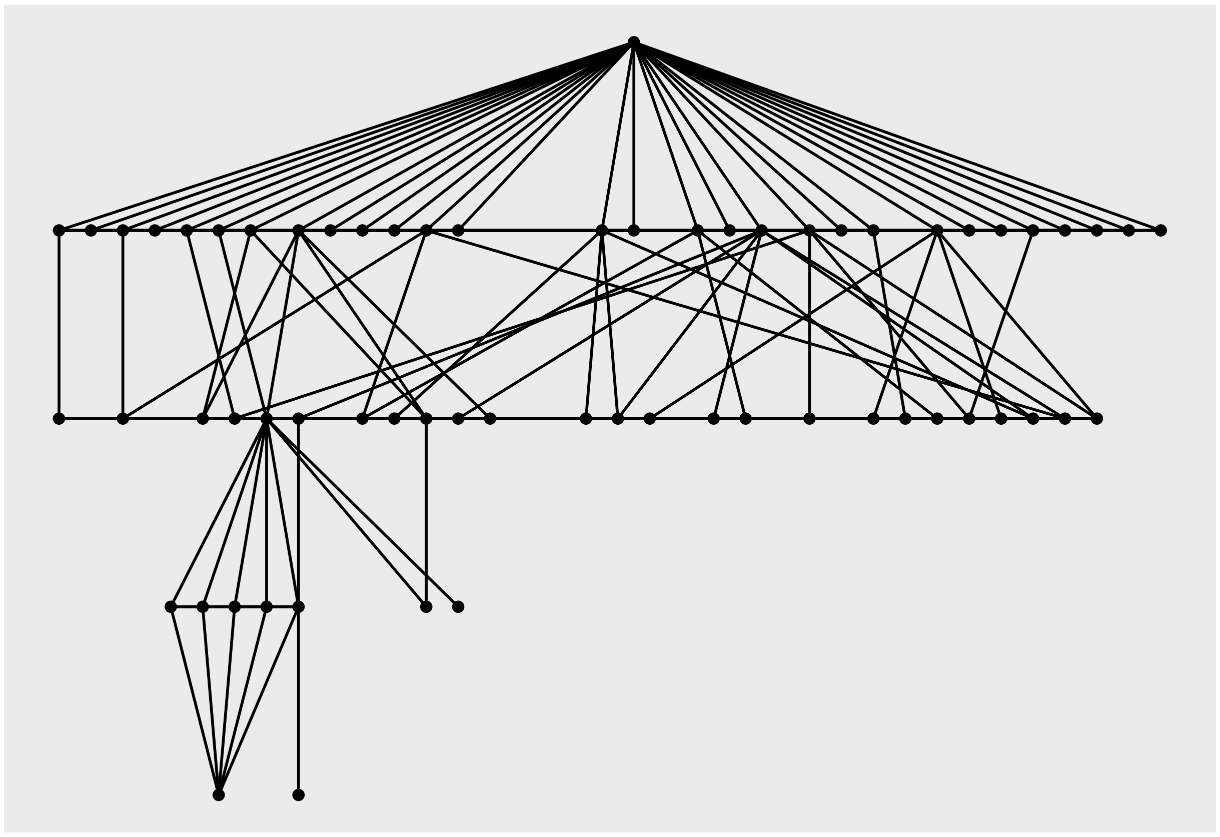
with_fr

```
ggraph(g, layout = "with_fr") +
    geom_edge_link() +
    geom_node_point()
```



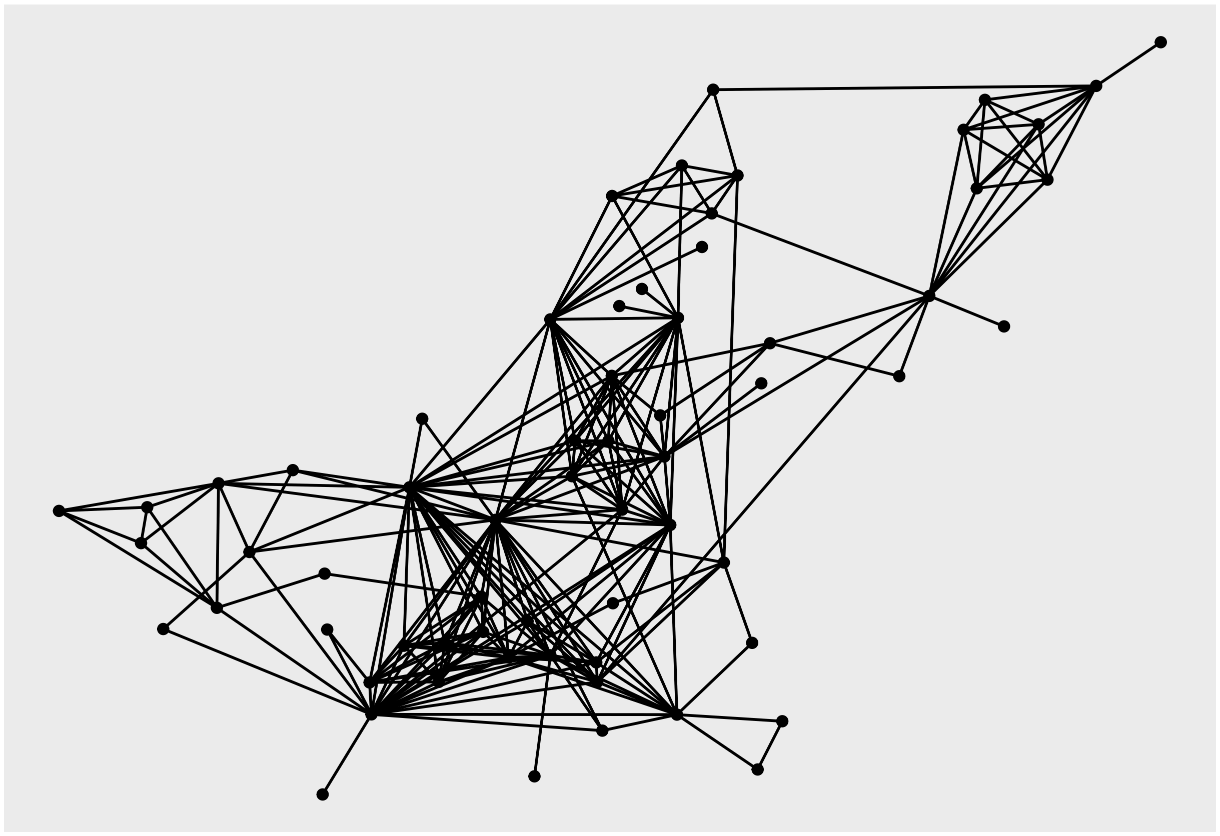
with_sugiyama

```
ggraph(g, layout = "with_sugiyama") +
    geom_edge_link() +
    geom_node_point()
```



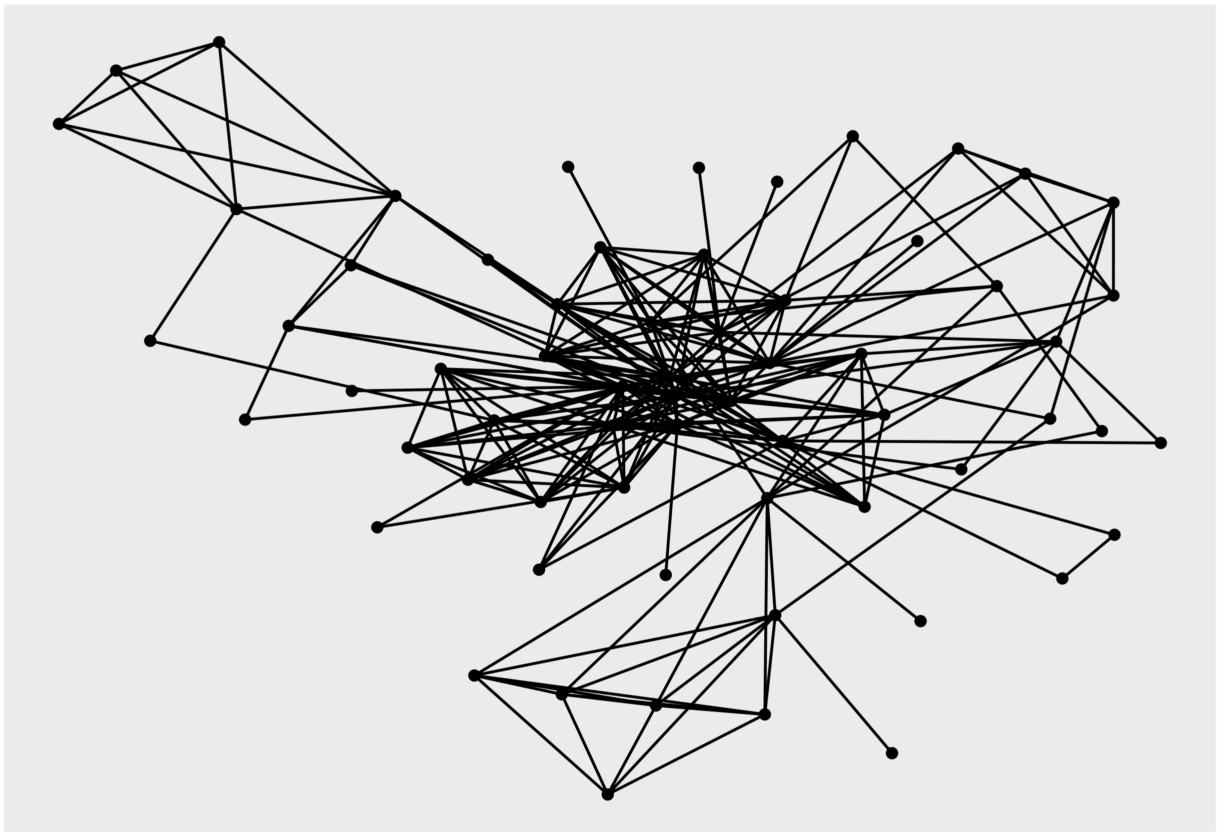
with_dh

```
ggraph(g, layout = "with_dh") +
    geom_edge_link() +
    geom_node_point()
```



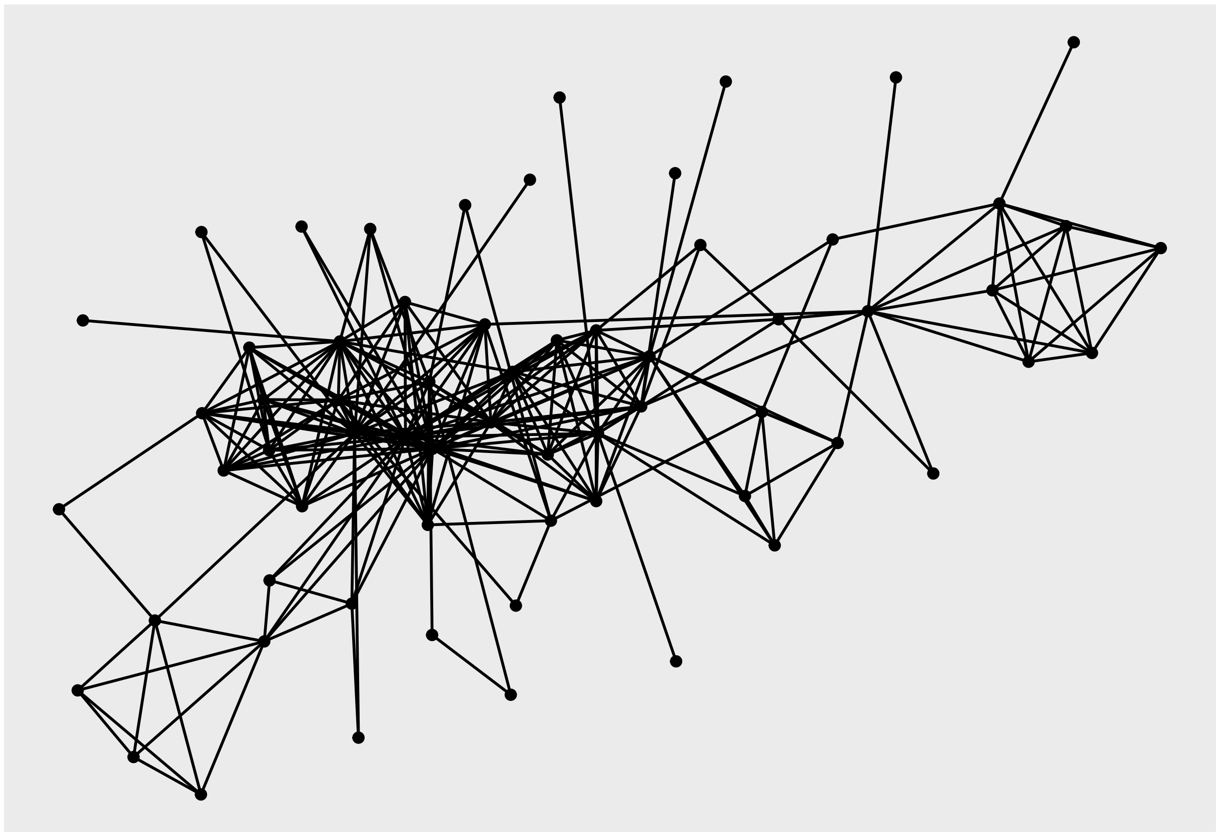
with gem

```
ggraph(g, layout = "with_gem") +
    geom_edge_link() +
    geom_node_point()
```



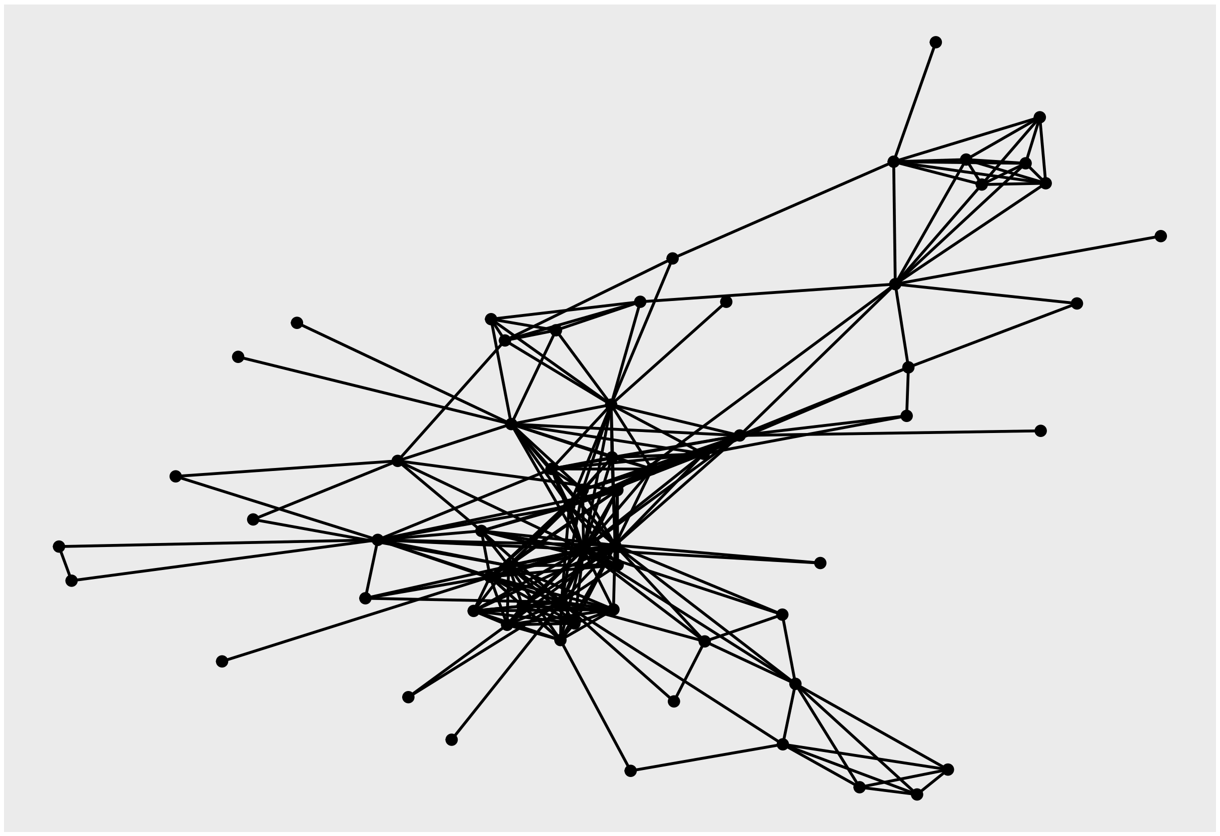
with lgl

```
ggraph(g, layout = "with_lgl") +
    geom_edge_link() +
    geom_node_point()
```



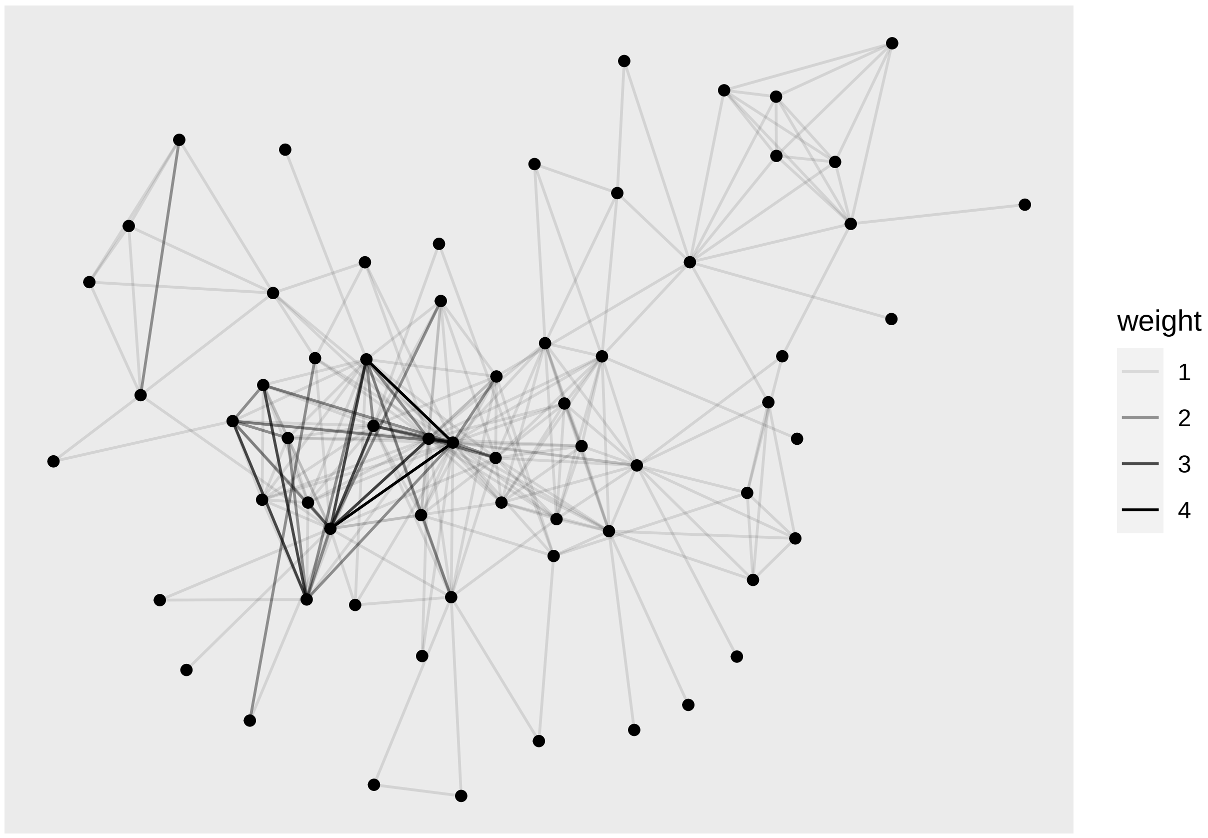
layout nicely

```
ggraph(g, layout = "nicely") +
    geom_edge_link() +
    geom_node_point()
```



Change edge transparency using alpha

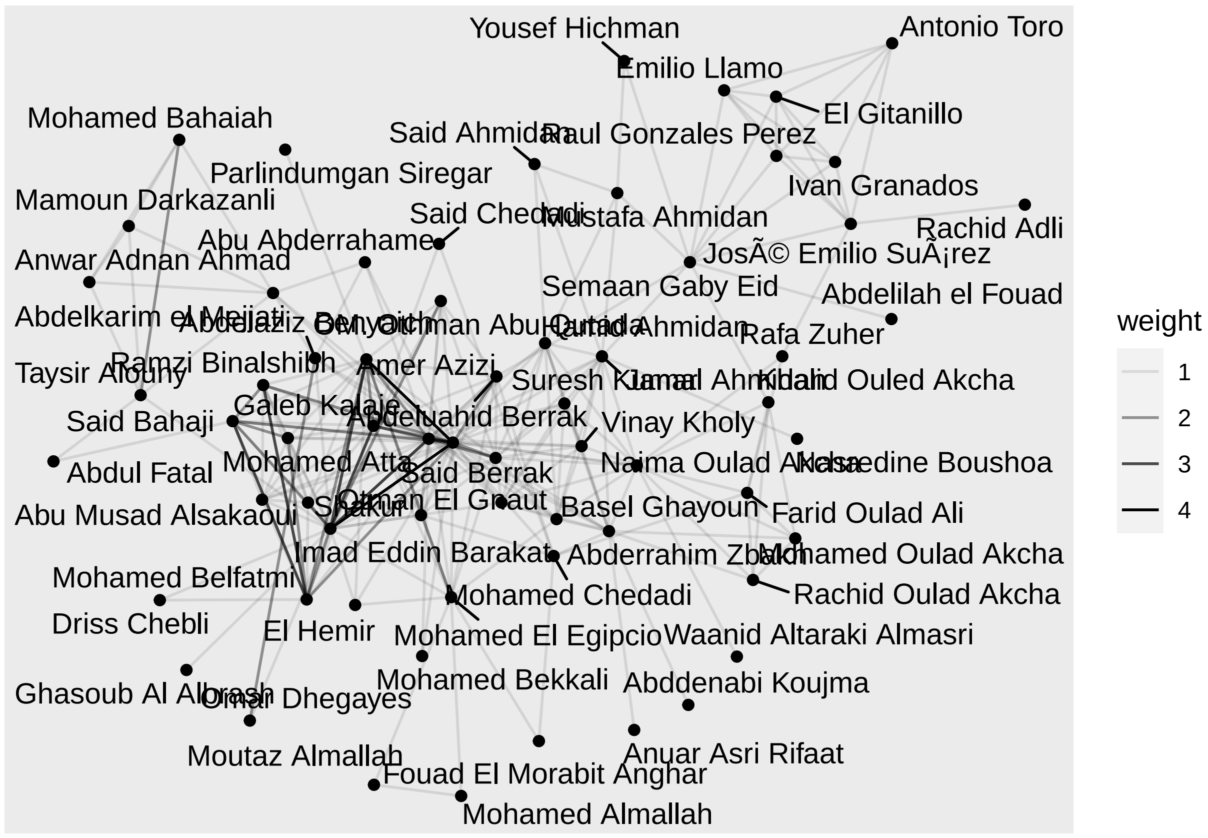
```
ggraph(g, layout = "with_kk") +
    geom_edge_link(aes(alpha = weight)) +
    geom_node_point()
```



Add text to nodes

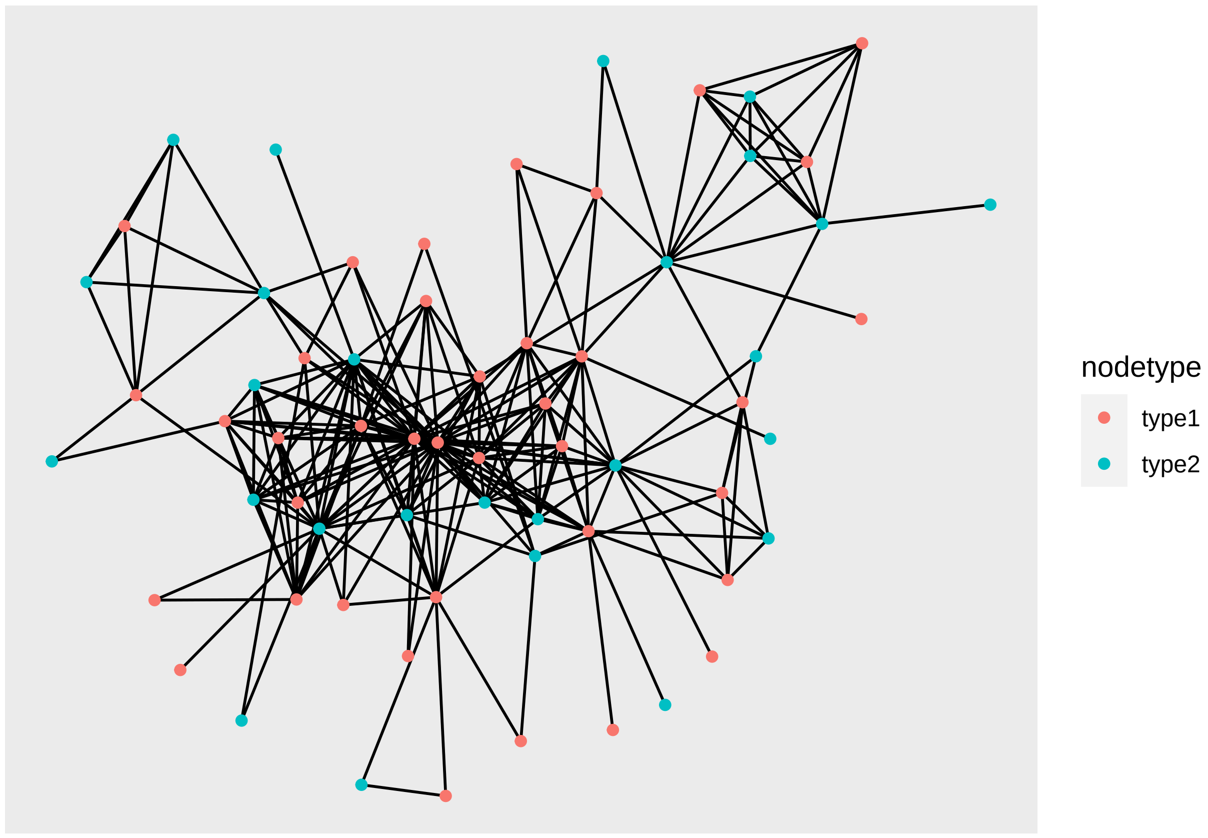
```
ggraph(g, layout = "with_kk") +
    geom_edge_link(aes(alpha = weight)) +
    geom_node_point() +
    geom_node_text(aes(label = name), repel = T)
```

```
## Warning: ggrepel: 3 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



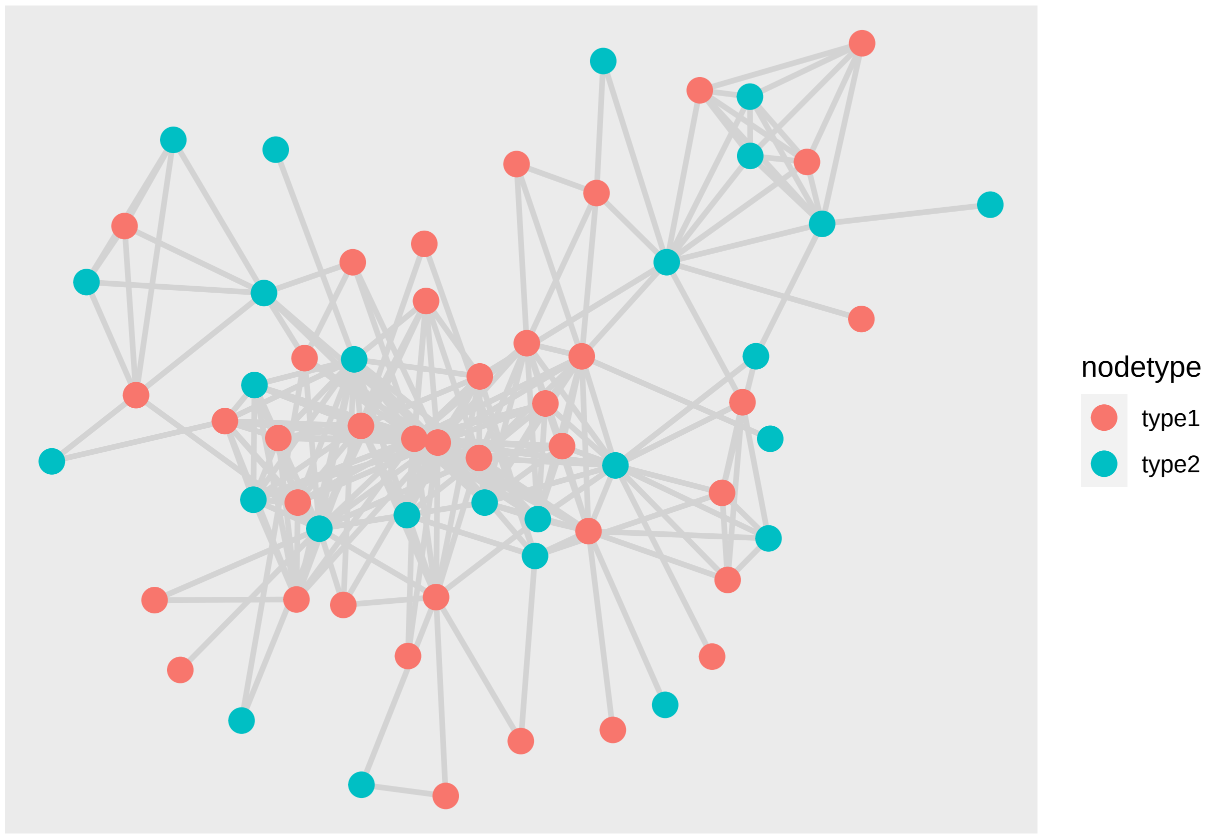
Color

```
ggraph(g, layout = "with_kk")+
    geom_edge_link() +
    geom_node_point(aes(color = nodetype))
```



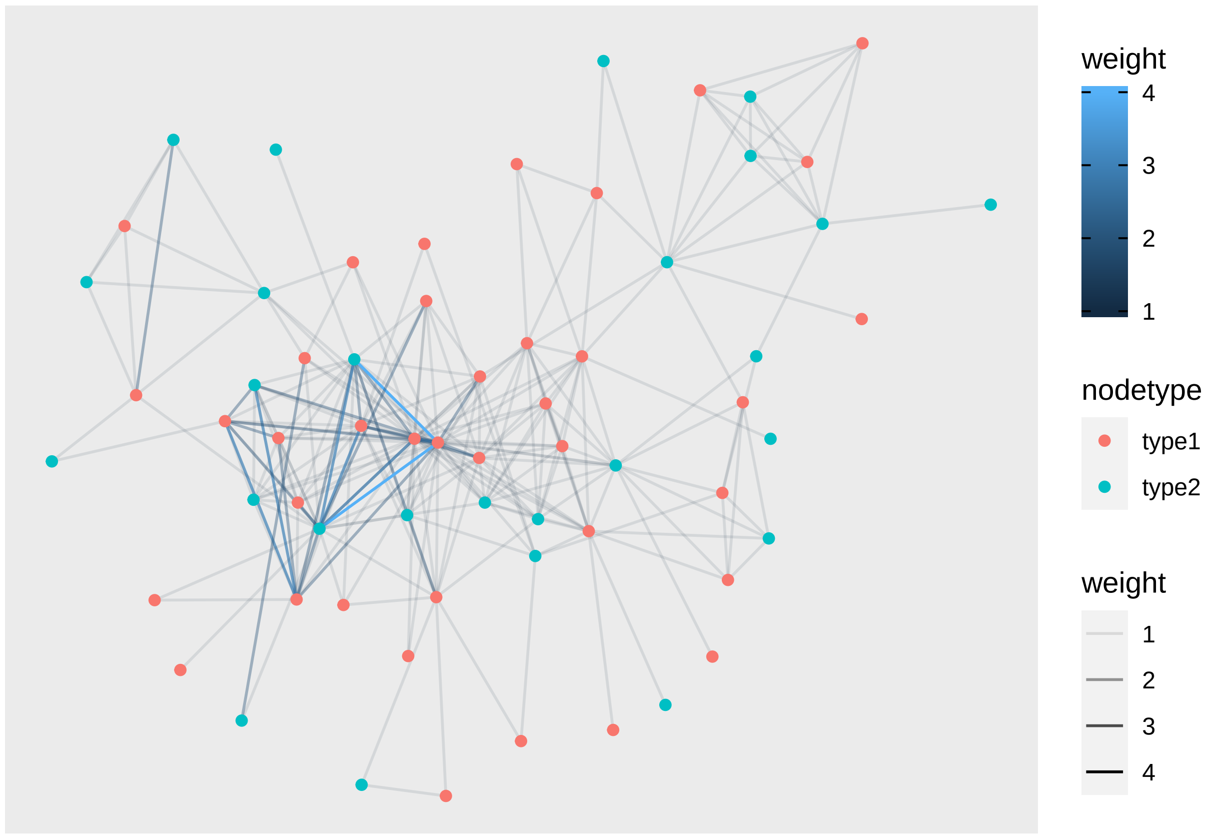
uniform color, width, size

```
ggraph(g, layout = "with_kk")+
    geom_edge_link(colour = "lightgray", width = 1) +
    geom_node_point(aes(color = nodetype), size = 4)
```



Edge color w.r.t weight

```
ggraph(g, layout = "with_kk")+
    geom_edge_link(aes(colour = weight, alpha = weight)) +
    geom_node_point(aes(color = nodetype))
```

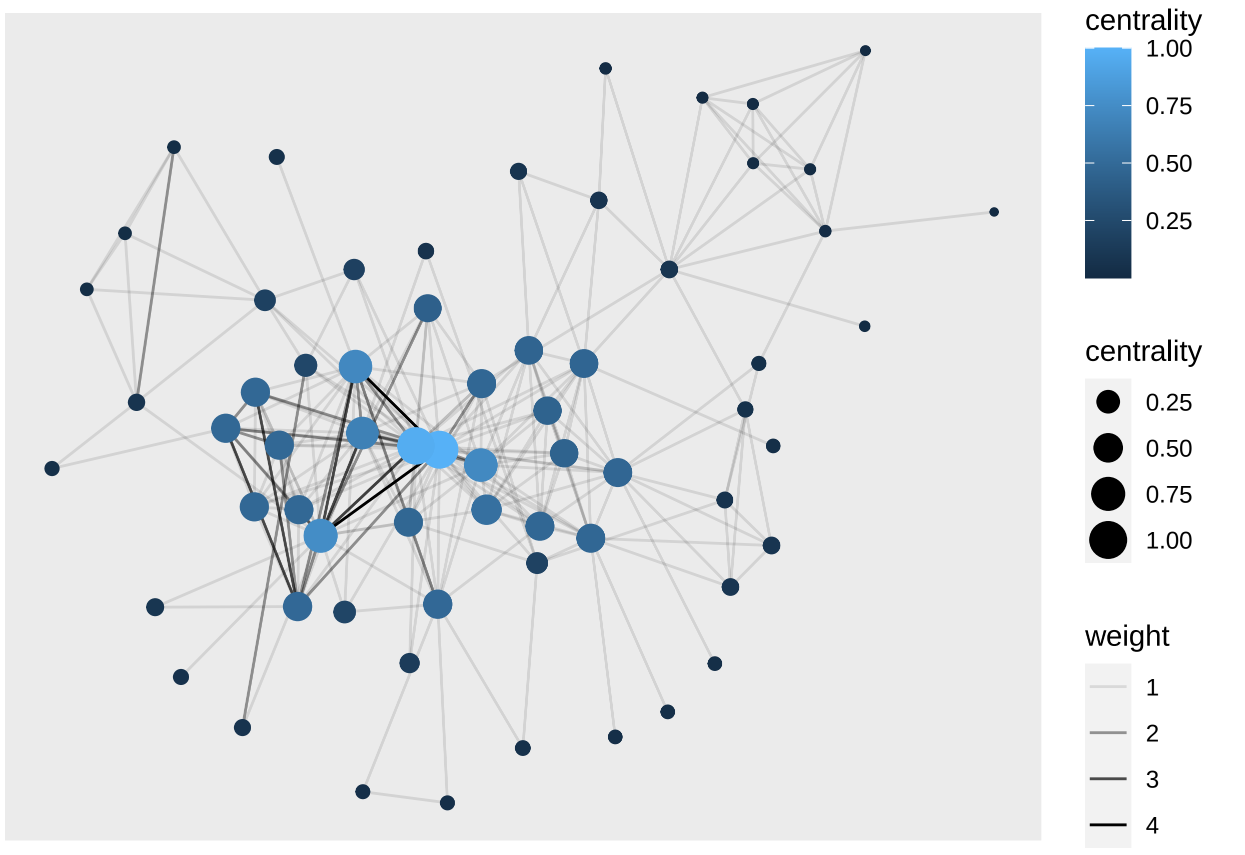


Change node attributes

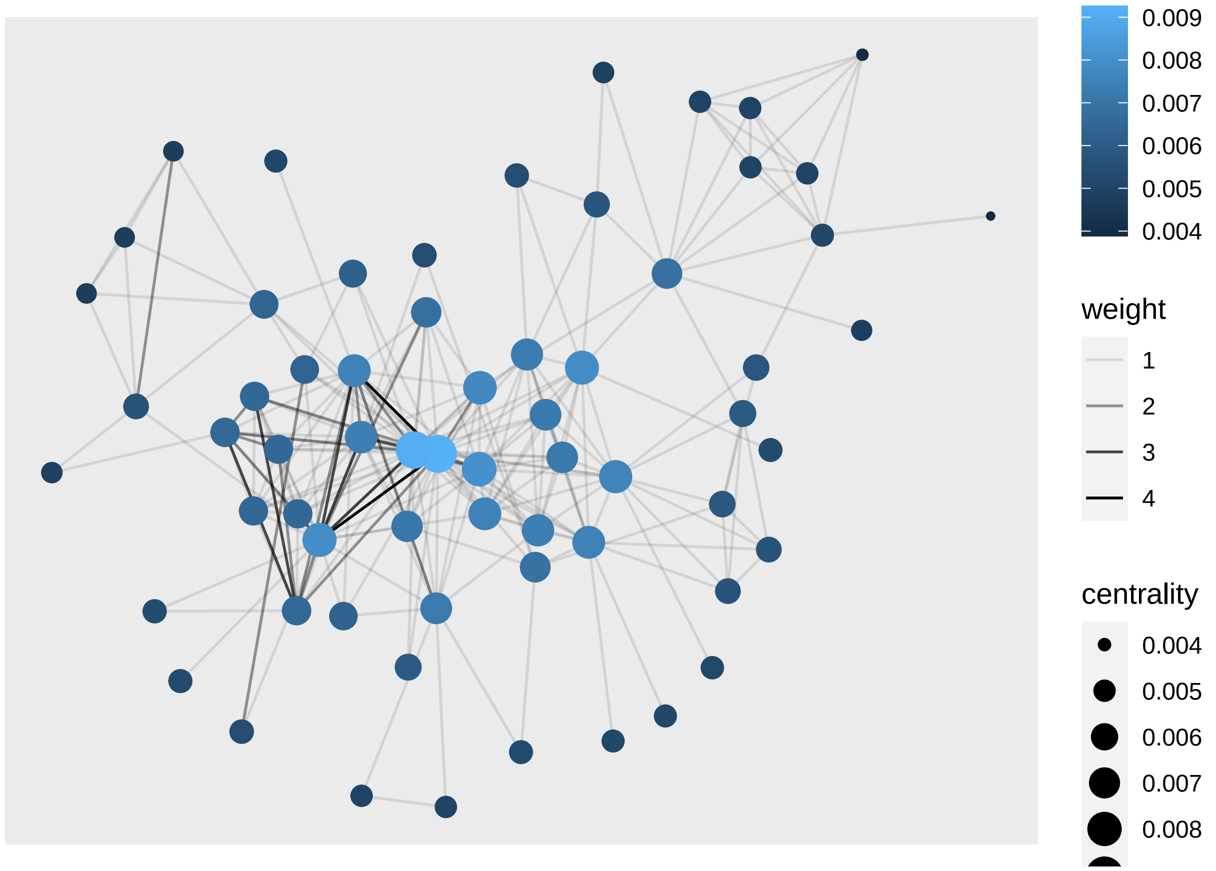
```r
nodes_with_centrality <- nodes %>%
  mutate(
    degree = degree(g),
    # Add a column containing the strength of each node
    strength = strength(g)) %>%
  # Arrange rows by descending strength
  arrange(desc(strength))

# Calculate the reciprocal of the tie weights
dist_weight <- 1 / E(g)$weight

ties_with_betweenness <- edges %>%
  # Add an edge betweenness column weighted by dist_weight
  mutate(betweenness = edge_betweenness(g, weights = dist_weight))

g = g %>%  activate(nodes) %>% left_join(nodes_with_centrality, by = "name") %>%  activate(edges) %>% le


ggraph(g, layout = "with_kk")+
    geom_edge_link(colour="lightgray", aes(alpha = betweenness)) +
    geom_node_point(aes(size = degree, colour = strength))
```

Direct

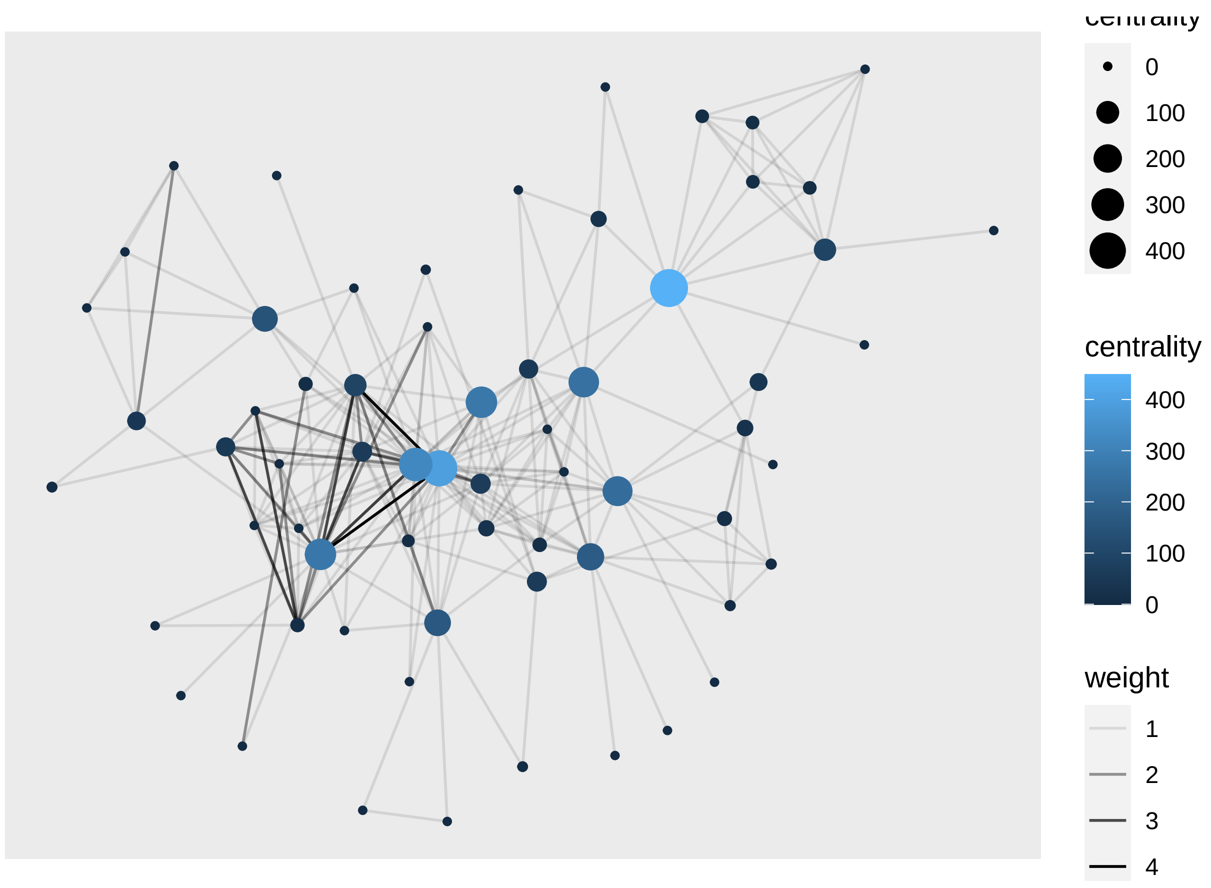
```r
g = graph_from_data_frame(d = edges, directed = FALSE, vertices = nodes) %>% as_tbl_graph()

g %>% activate(nodes) %>% mutate(centrality = centrality_authority()) %>%
  ggraph(layout = 'kk') +
    geom_edge_link(aes(alpha = weight)) +
    geom_node_point(aes(size = centrality, colour = centrality))
```



```r
# closeness centrality
g %>% activate(nodes) %>% mutate(centrality = centrality_closeness()) %>%
  ggraph(layout = 'kk') +
    geom_edge_link(aes(alpha = weight)) +
    geom_node_point(aes(size = centrality, colour = centrality))
```



```r
# betweenness centrality
g %>% activate(nodes) %>% mutate(centrality = centrality_betweenness()) %>%
  ggraph(layout = 'kk') +
    geom_edge_link(aes(alpha = weight)) +
    geom_node_point(aes(size = centrality, colour = centrality))
```

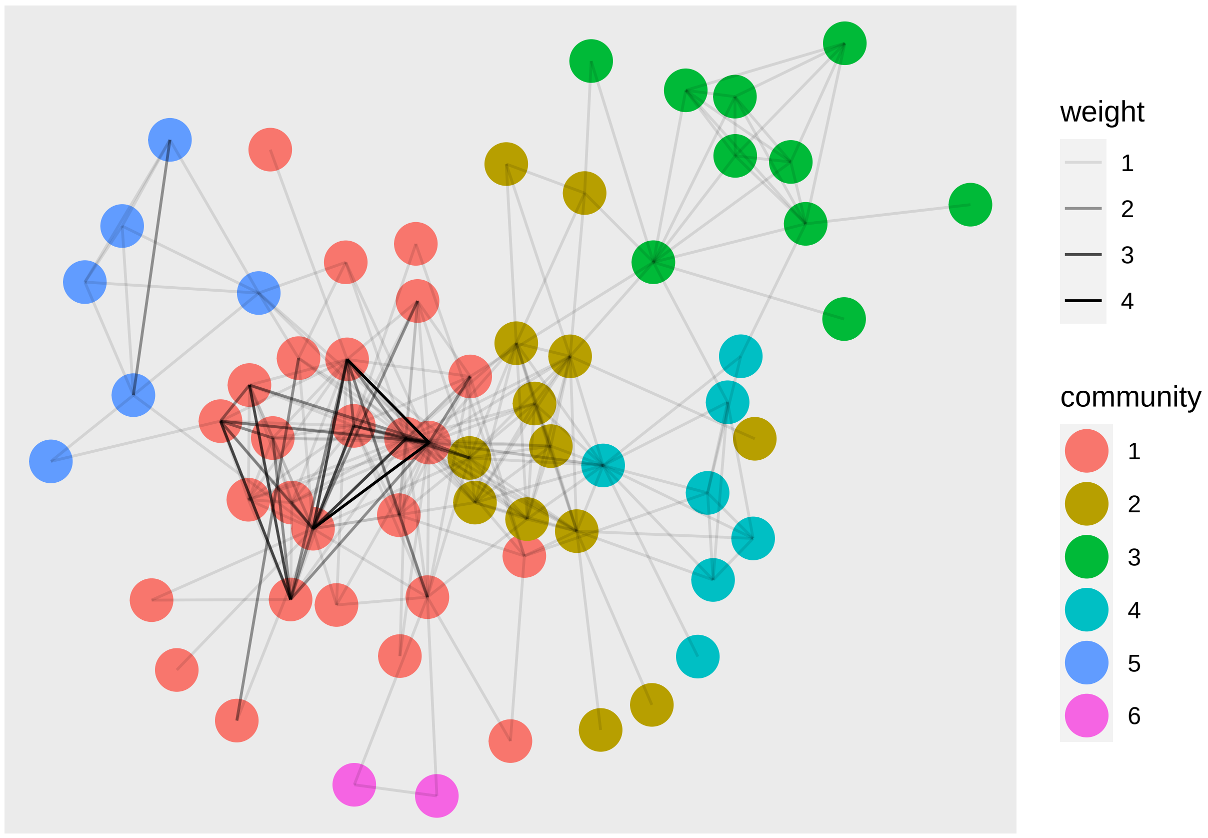


Filter important edges

```r
median_betweenness = median(E(g)$betweenness)

ggraph(g, layout = "with_kk") +
  # Filter ties for betweenness greater than the median
  geom_edge_link(aes(filter = betweenness > median_betweenness, alpha = betweenness), size = 2) +
  theme(legend.position="none")
```

Clustering

```r
g %>% activate(nodes) %>% mutate(community = as.factor(group_infomap())) %>%
  ggraph(layout = 'kk') +
  geom_node_point(aes(colour = community), size = 7)+
  geom_edge_link(aes(alpha = weight))
```



References

1. https://www.data-imaginist.com/2017/introducing-tidygraph/