# Exercises 2 Distances and Clustering Coefficient

*Roberto Alvarez*

*15 de junio de 2017*

## Contents



## 1 Paths, Shortest Paths, Distances and Diameter

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

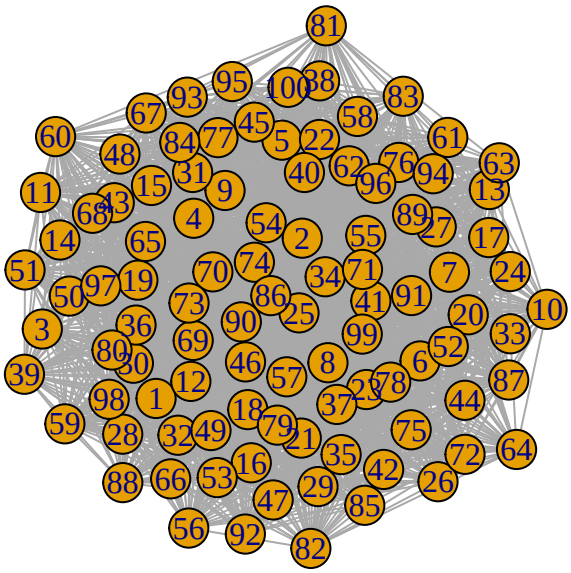
```
## The following object is masked from 'package:base':
##
##     union
```

The commands for Shortest Paths, Distances, Mean Distance and Diameter are, respectively,

`shortest_paths() ,shortest.path(),  distance(),diameter()`

- shortest_paths calculates a single shortest path (i.e. the path itself, not just its length) between the source vertex given in from, to the target vertices given in to.
- distances calculates the lengths of pairwise shortest paths from a set of vertices (from) to another set of vertices (to).
- mean_distance calculates the average path length in a graph, by calculating the shortest paths between all pairs of vertices (both ways for directed graphs).

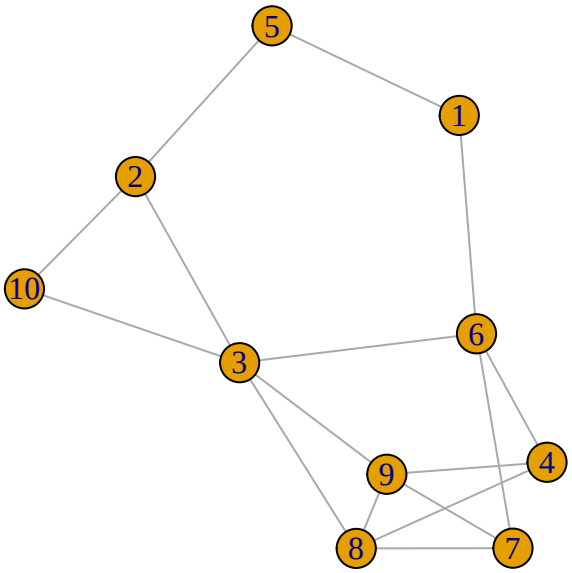
```
g1<-random.graph.game(100,0.5)
plot(g1)
```



```
g2<-barabasi.game(100,directed=FALSE)
```

```
g <- barabasi.game(1000, power=1)
layout <- layout.fruchterman.reingold(g)
plot(g, layout=layout, vertex.size=2,
  vertex.label=NA, edge.arrow.size=.2)
```

```
g<-random.graph.game(10,0.5)
plot(g)
```



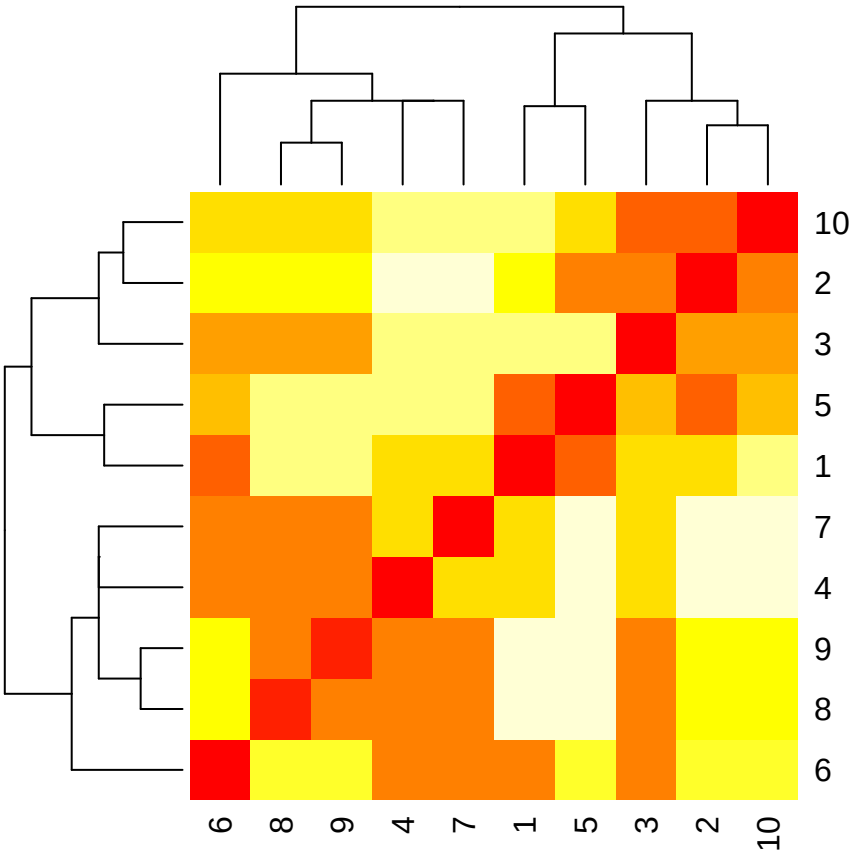

```
distancia<-shortest.paths(g)
distancia
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    2    2    2    1    1    2    3    3     3
##  [2,]    2    0    1    3    1    2    3    2    2     1
##  [3,]    2    1    0    2    2    1    2    1    1     1
##  [4,]    2    3    2    0    3    1    2    1    1     3
##  [5,]    1    1    2    3    0    2    3    3    3     2
##  [6,]    1    2    1    1    2    0    1    2    2     2
##  [7,]    2    3    2    2    3    1    0    1    1     3
##  [8,]    3    2    1    1    3    2    1    0    1     2
##  [9,]    3    2    1    1    3    2    1    1    0     2
## [10,]    3    1    1    3    2    2    3    2    2     0
```

```
class(distancia)
```

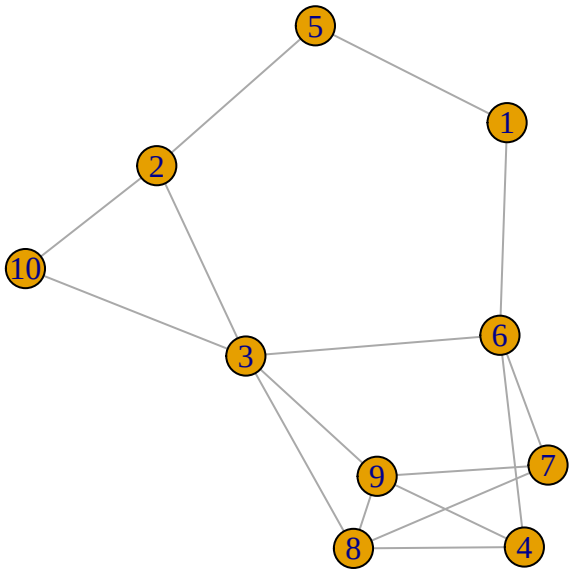
```
## [1] "matrix"
```

```
heatmap(distancia)
```




shortest_paths()

```
plot(g)
```




```
road<-shortest_paths(g, 1,3)
road
```

```
## $vpath
## $vpath[[1]]
## + 3/10 vertices, from 916212a:
```

```
## [1] 1 6 3
##
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

```
diameter(g)
```

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

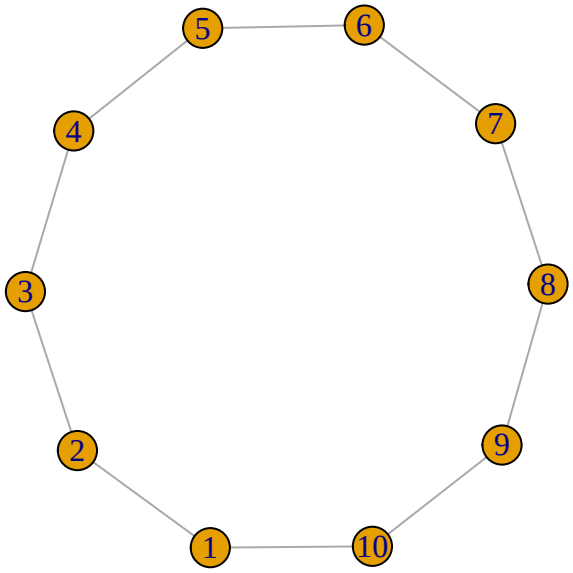
```
diameter(g1)
```

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

```
diameter(g2)
```

```
## [1] 10
```

```
g <- make_ring(10)
plot(g)
```




```
distances(g)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    1    2    3    4    5    4    3    2     1
##  [2,]    1    0    1    2    3    4    5    4    3     2
##  [3,]    2    1    0    1    2    3    4    5    4     3
##  [4,]    3    2    1    0    1    2    3    4    5     4
##  [5,]    4    3    2    1    0    1    2    3    4     5
##  [6,]    5    4    3    2    1    0    1    2    3     4
##  [7,]    4    5    4    3    2    1    0    1    2     3
##  [8,]    3    4    5    4    3    2    1    0    1     2
##  [9,]    2    3    4    5    4    3    2    1    0     1
```

```
## [10,]    1    2    3    4    5    4    3    2    1     0
```

```
shortest_paths(g, 5)
```

```
## $vpath
## $vpath[[1]]
## + 5/10 vertices, from 618b003:
## [1] 5 4 3 2 1
##
## $vpath[[2]]
## + 4/10 vertices, from 618b003:
## [1] 5 4 3 2
##
## $vpath[[3]]
## + 3/10 vertices, from 618b003:
## [1] 5 4 3
##
## $vpath[[4]]
## + 2/10 vertices, from 618b003:
## [1] 5 4
##
## $vpath[[5]]
## + 1/10 vertex, from 618b003:
## [1] 5
##
## $vpath[[6]]
## + 2/10 vertices, from 618b003:
## [1] 5 6
##
## $vpath[[7]]
## + 3/10 vertices, from 618b003:
## [1] 5 6 7
##
## $vpath[[8]]
## + 4/10 vertices, from 618b003:
## [1] 5 6 7 8
##
## $vpath[[9]]
## + 5/10 vertices, from 618b003:
## [1] 5 6 7 8 9
##
## $vpath[[10]]
## + 6/10 vertices, from 618b003:
## [1]  5  4  3  2  1 10
##
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

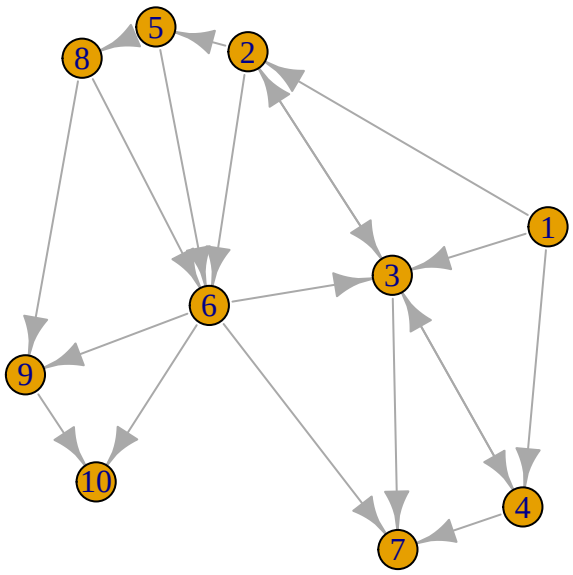
```
all_shortest_paths(g, 1, 6:8)
```

```
## $res
## $res[[1]]
## + 6/10 vertices, from 618b003:
## [1]  1 10  9  8  7  6
##
## $res[[2]]
## + 6/10 vertices, from 618b003:
## [1] 1 2 3 4 5 6
##
## $res[[3]]
## + 5/10 vertices, from 618b003:
## [1]  1 10  9  8  7
##
## $res[[4]]
## + 4/10 vertices, from 618b003:
## [1]  1 10  9  8
##
##
## $nrgeo
##  [1] 1 1 1 1 1 2 1 1 1 1
```

```
mean_distance(g)
```

```
## [1] 2.777778
```

```
## Weighted networks
el <- matrix(nc=3, byrow=TRUE,
             c(1,2,0, 1,3,2, 1,4,1, 2,3,0, 2,5,5, 2,6,2, 3,2,1, 3,4,1,
               3,7,1, 4,3,0, 4,7,2, 5,6,2, 5,8,8, 6,3,2, 6,7,1, 6,9,1,
               6,10,3, 8,6,1, 8,9,1, 9,10,4) )
g3 <- add_edges(make_empty_graph(10), t(el[,1:2]), weight=el[,3])
plot(g3)
```



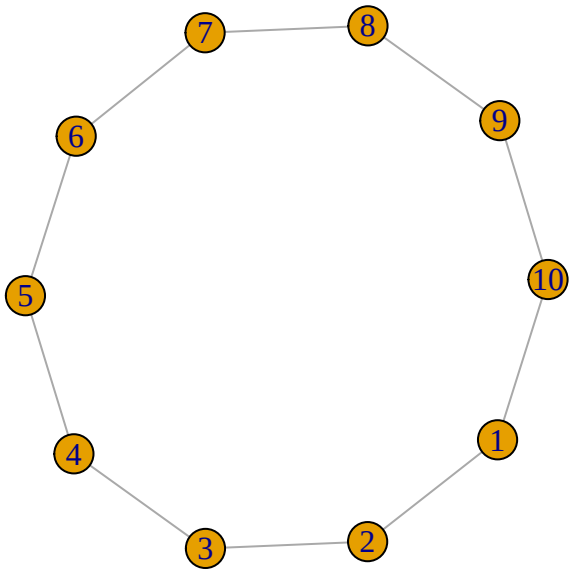

```
distances(g3, mode="out")
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    0    0    1    5    2    1   13    3     5
##  [2,]  Inf    0    0    1    5    2    1   13    3     5
##  [3,]  Inf    1    0    1    6    3    1   14    4     6
##  [4,]  Inf    1    0    0    6    3    1   14    4     6
##  [5,]  Inf    5    4    5    0    2    3    8    3     5
##  [6,]  Inf    3    2    3    8    0    1   16    1     3
##  [7,]  Inf  Inf  Inf  Inf  Inf  Inf    0  Inf  Inf   Inf
##  [8,]  Inf    4    3    4    9    1    2    0    1     4
##  [9,]  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0     4
## [10,]  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf     0
```

## 2 Clustering coefficent

The command for calculating the clustering coefficient is `transitivity()`

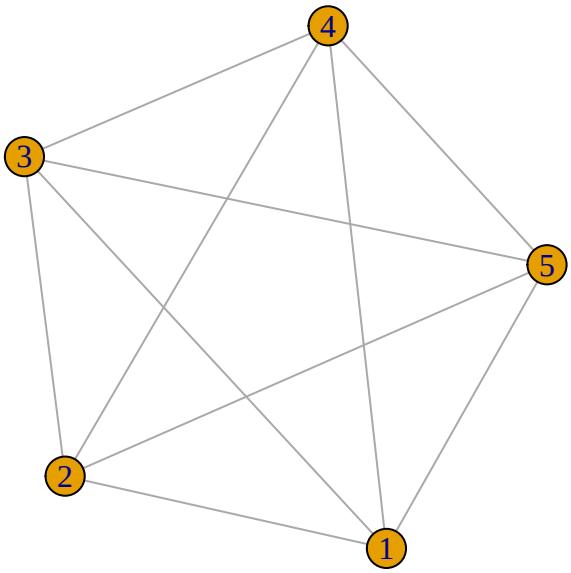
```
g <- make_ring(10)
plot(g)
```




```
transitivity(g)
```

```
## [1] 0
```

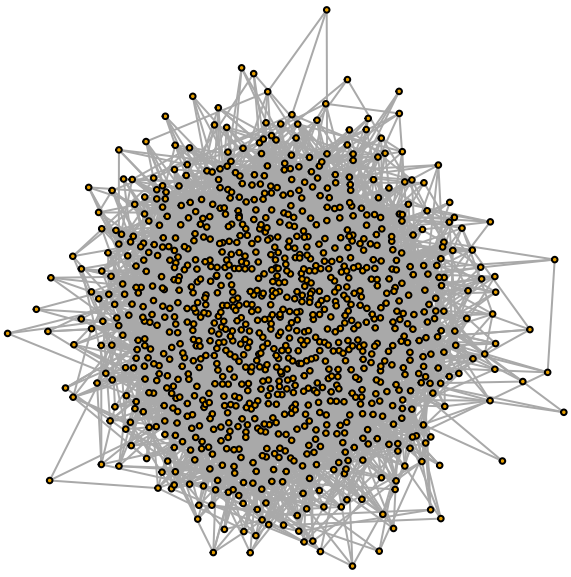
```
g1<-make_full_graph(5)
plot(g1)
```




```
transitivity(g1)
```

```
## [1] 1
```

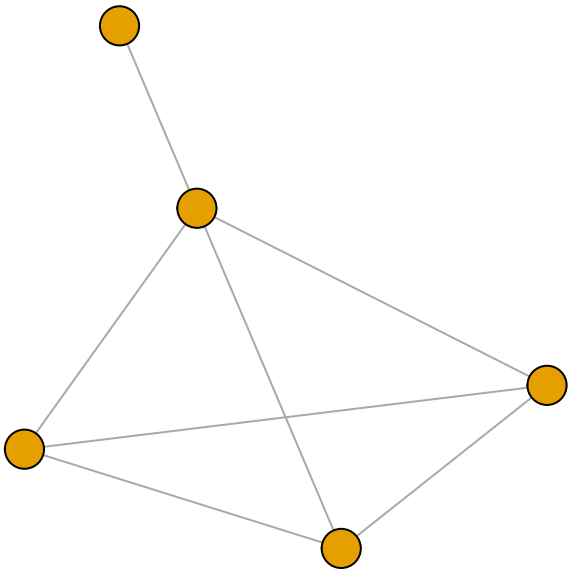
```
g2 <- sample_gnp(1000, 10/1000)
layout <- layout.fruchterman.reingold(g2)
plot(g2, layout=layout, vertex.size=2,
  vertex.label=NA, edge.arrow.size=.2)
```



```
transitivity(g2)    # this is about 10/1000
```

```
## [1] 0.009510478
```

```
# Weighted version, the figure from the Barrat paper
gw <- graph_from_literal(A-B:C:D:E, B-C:D, C-D)
E(gw)$weight <- 1
E(gw)[ V(gw)[name == "A"] %--% V(gw)[name == "E" ] ]$weight <- 5
plot(gw,vertex.label=NA)
```



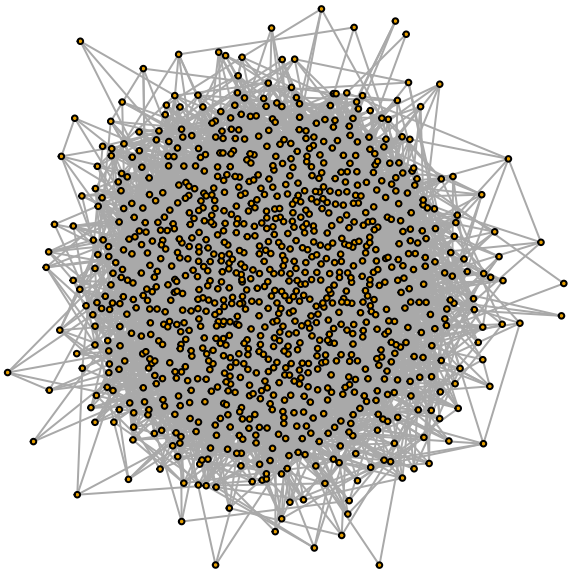
```
transitivity(gw, vids="A", type="local")
```

```
## [1] 0.5
```

```
transitivity(gw, vids="A", type="weighted")
```

```
## [1] 0.25
```

```
# Weighted reduces to "local" if weights are the same
gw2 <- sample_gnp(1000, 10/1000)
E(gw2)$weight <- 1
t1 <- transitivity(gw2, type="local")
t2 <- transitivity(gw2, type="weighted")
layout <- layout.fruchterman.reingold(gw2)
plot(gw2, layout=layout, vertex.size=2,
  vertex.label=NA, edge.arrow.size=.2)
```



```
all(is.na(t1) == is.na(t2))
```

```
## [1] TRUE
```

```
all(na.omit(t1 == t2))
```

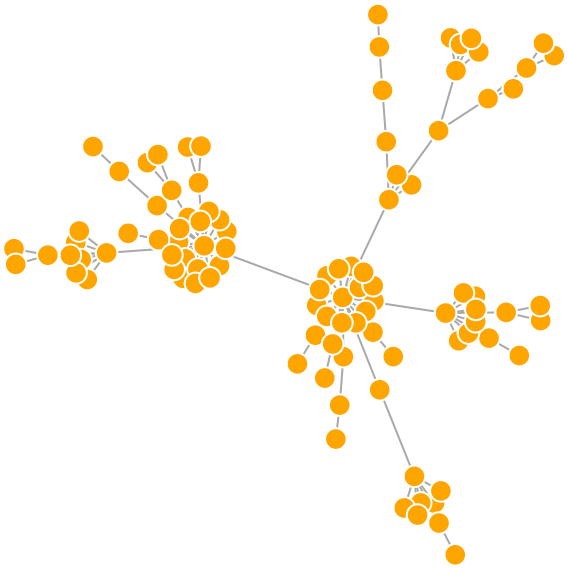
```
## [1] FALSE
```

# 3 Network layouts

Network layouts are simply algorithms that return coordinates for each node in a network.

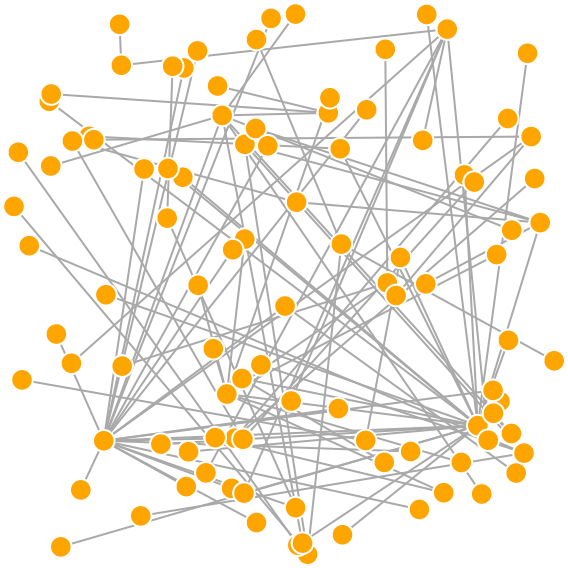
For the purposes of exploring layouts, we will generate a slightly larger 100-node graph. We use the sample_pa() function which generates a simple graph starting from one node and adding more nodes and links based on a preset level of preferential attachment (Barabasi-Albert model).

```
net.bg <- sample_pa(100)
V(net.bg)$size <- 8
V(net.bg)$frame.color <- "white"
V(net.bg)$color <- "orange"
V(net.bg)$label <- ""
E(net.bg)$arrow.mode <- 0
plot(net.bg)
```



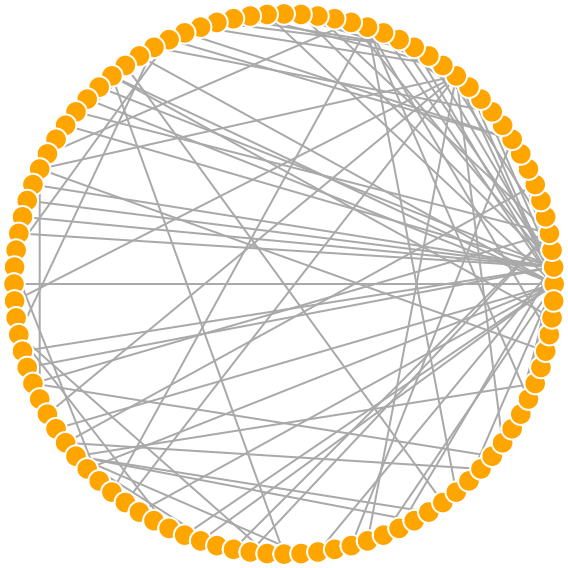
You can set the layout in the plot function:

```
plot(net.bg, layout=layout_randomly)
```



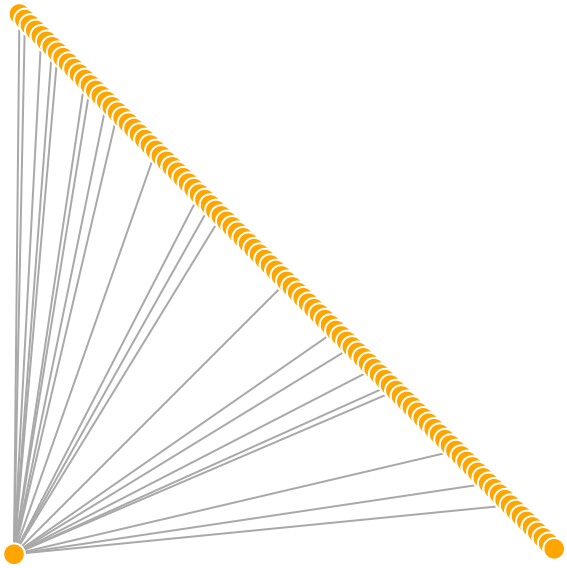
Or you can calculate the vertex coordinates in advance:

```
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



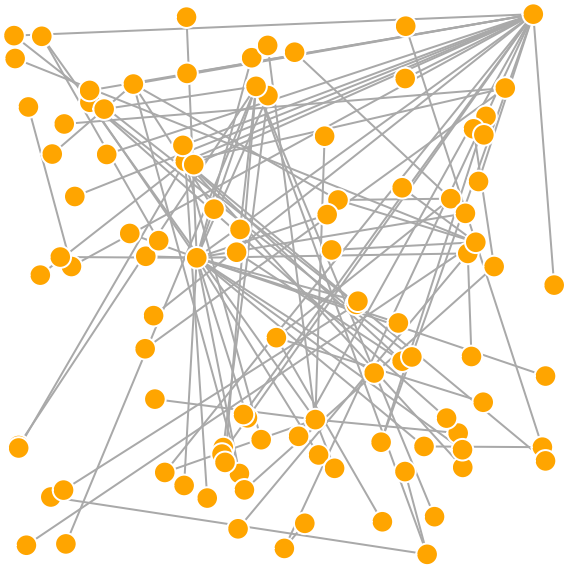
l is simply a matrix of x, y coordinates (N x 2) for the N nodes in the graph. You can easily generate your own:

```
l <- cbind(1:vcount(net.bg), c(1, vcount(net.bg):2))
plot(net.bg, layout=l)
```

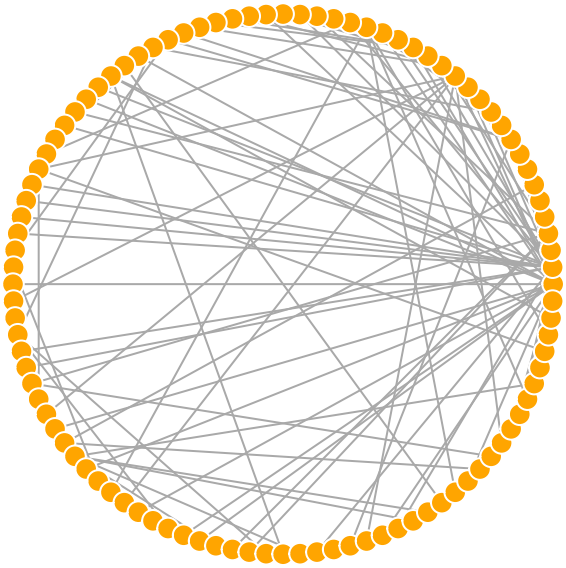


This layout is just an example and not very helpful - thankfully igraph has a number of built-in layouts, including:

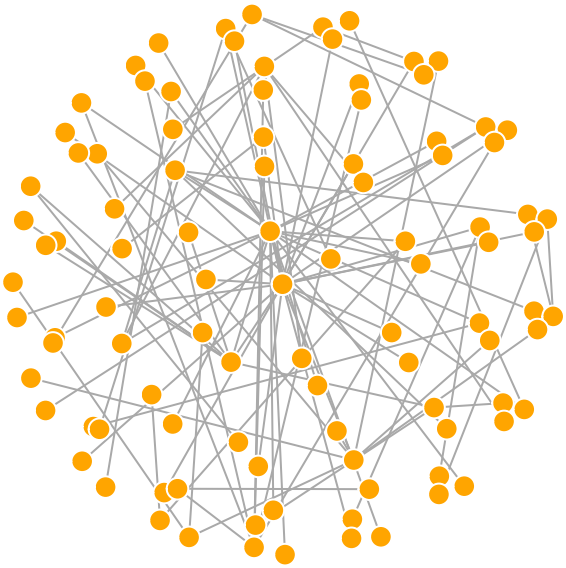
```
# Randomly placed vertices
l <- layout_randomly(net.bg)
plot(net.bg, layout=l)
```



```
# Circle layout
l <- layout_in_circle(net.bg)
plot(net.bg, layout=l)
```



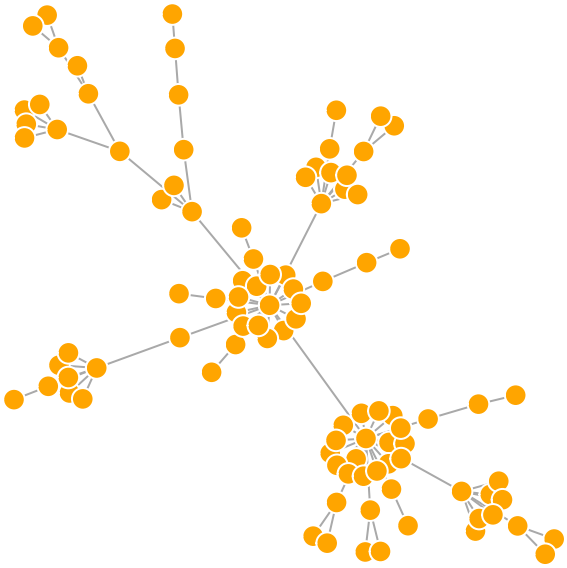
```
# 3D sphere layout
l <- layout_on_sphere(net.bg)
plot(net.bg, layout=l)
```



Fruchterman-Reingold is one of the most used force-directed layout algorithms out there.

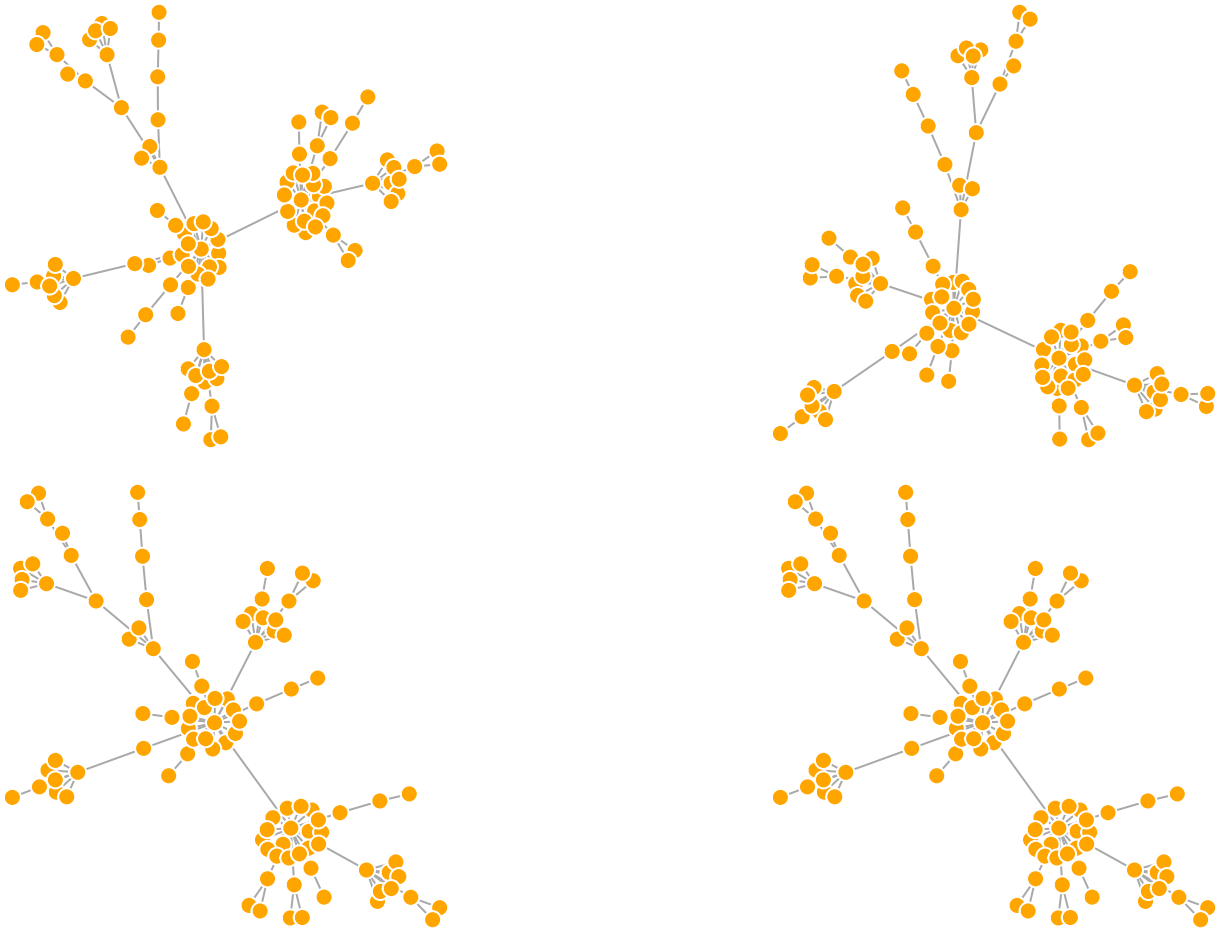
Force-directed layouts try to get a nice-looking graph where edges are similar in length and cross each other as little as possible. They simulate the graph as a physical system. Nodes are electrically charged particles that repulse each other when they get too close. The edges act as springs that attract connected nodes closer together. As a result, nodes are evenly distributed through the chart area, and the layout is intuitive in that nodes which share more connections are closer to each other. The disadvantage of these algorithms is that they are rather slow and therefore less often used in graphs larger than ~1000 vertices. You can set the "weight" parameter which increases the attraction forces among nodes connected by heavier edges.

```
l <- layout_with_fr(net.bg)
plot(net.bg, layout=l)
```



You will notice that this layout is not deterministic - different runs will result in slightly different configurations. Saving the layout in l allows us to get the exact same result multiple times, which can be helpful if you want to plot the time evolution of a graph, or different relationships – and want nodes to stay in the same place in multiple plots.

```
par(mfrow=c(2,2), mar=c(0,0,0,0))    # plot four figures - 2 rows, 2 columns
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=layout_with_fr)
plot(net.bg, layout=l)
plot(net.bg, layout=l)
```



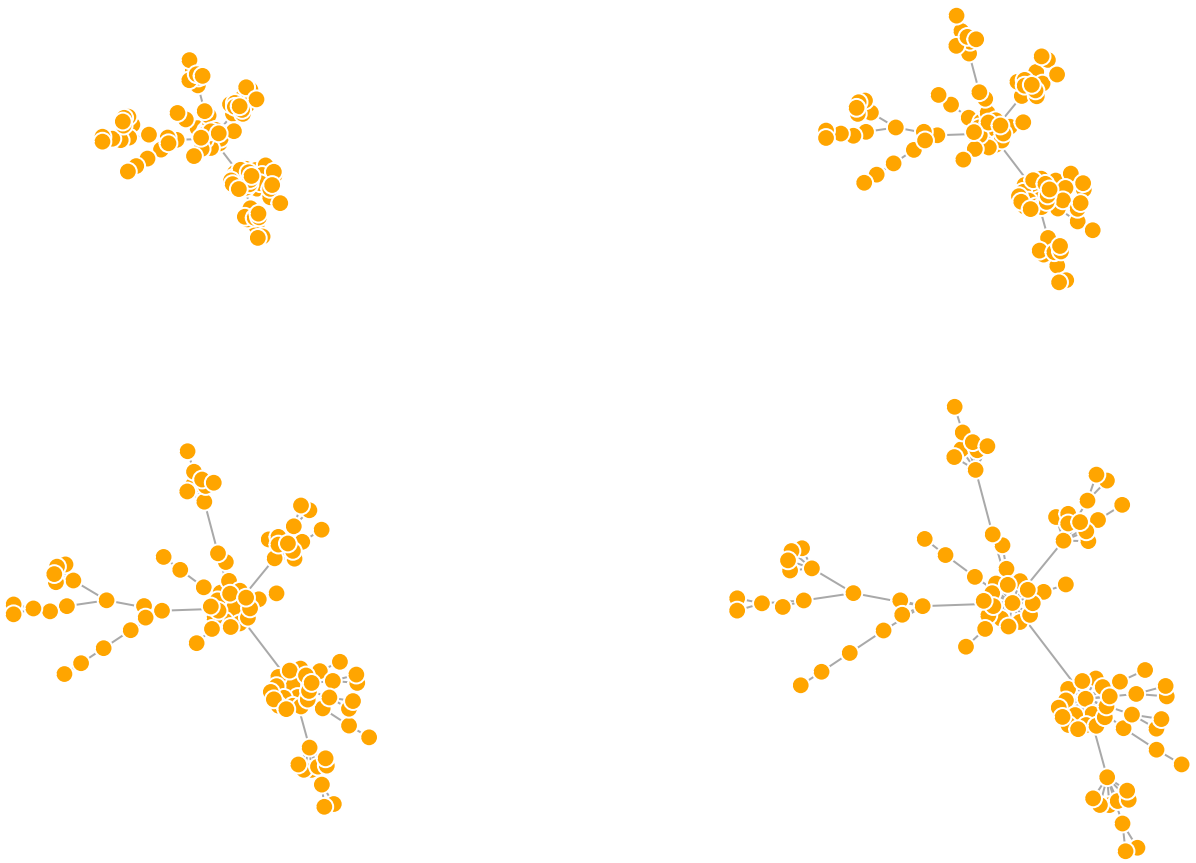
```
dev.off()
```

```
## null device
##           1
```

By default, the coordinates of the plots are rescaled to the [-1,1] interval for both x and y. You can change that with the parameter rescale=FALSE and rescale your plot manually by multiplying the coordinates by a scalar. You can use norm_coords to normalize the plot with the boundaries you want. This way you can create more compact or spread out layout versions.

```
l <- layout_with_fr(net.bg)
l <- norm_coords(l, ymin=-1, ymax=1, xmin=-1, xmax=1)

par(mfrow=c(2,2), mar=c(0,0,0,0))
plot(net.bg, rescale=F, layout=l*0.4)
plot(net.bg, rescale=F, layout=l*0.6)
plot(net.bg, rescale=F, layout=l*0.8)
plot(net.bg, rescale=F, layout=l*1.0)
```

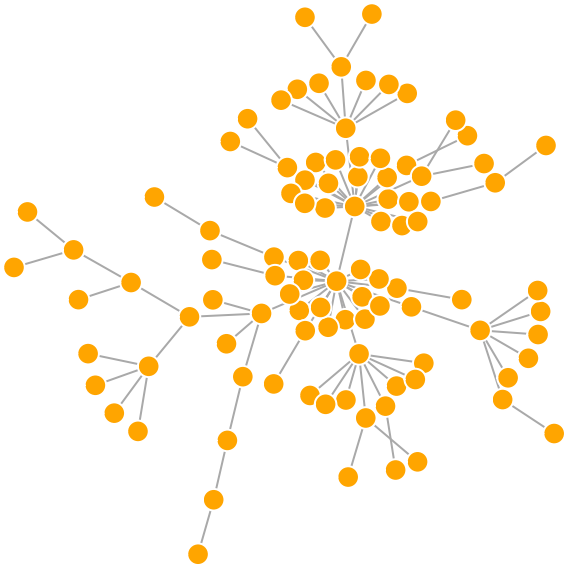


```
dev.off()
```

```
## null device
##           1
```

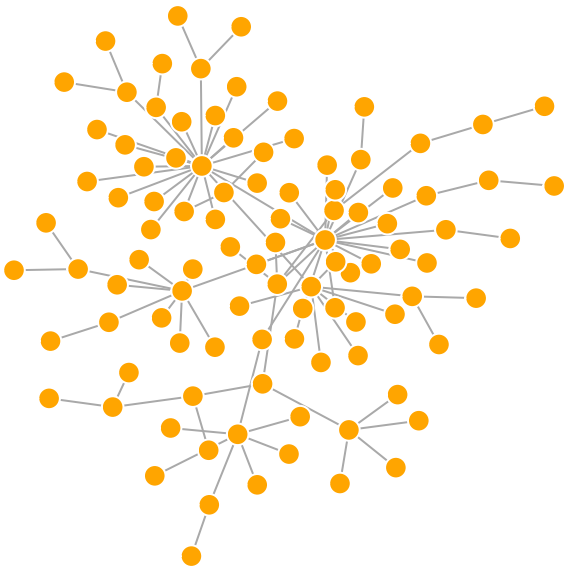
Another popular force-directed algorithm that produces nice results for connected graphs is Kamada Kawai. Like Fruchterman Reingold, it attempts to minimize the energy in a spring system.

```
l <- layout_with_kk(net.bg)
plot(net.bg, layout=l)
```



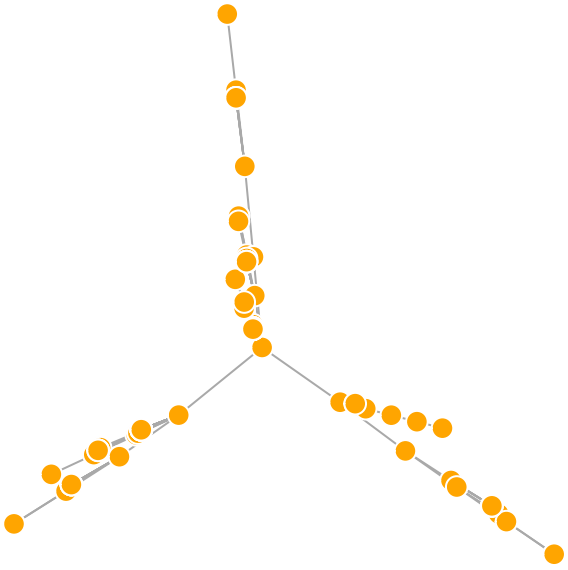
The LGL algorithm is meant for large, connected graphs. Here you can also specify a root: a node that will be placed in the middle of the layout.

```
plot(net.bg, layout=layout_with_lgl)
```



The MDS (multidimensional scaling) algorithm tries to place nodes based on some measure of similarity or distance between them. More similar nodes are plotted closer to each other. By default, the measure used is based on the shortest paths between nodes in the network. We can change that by using our own distance matrix (however defined) with the parameter dist. MDS layouts are nice because positions and distances have a clear interpretation. The problem with them is visual clarity: nodes often overlap, or are placed on top of each other.

```
plot(net.bg, layout=layout_with_mds)
```

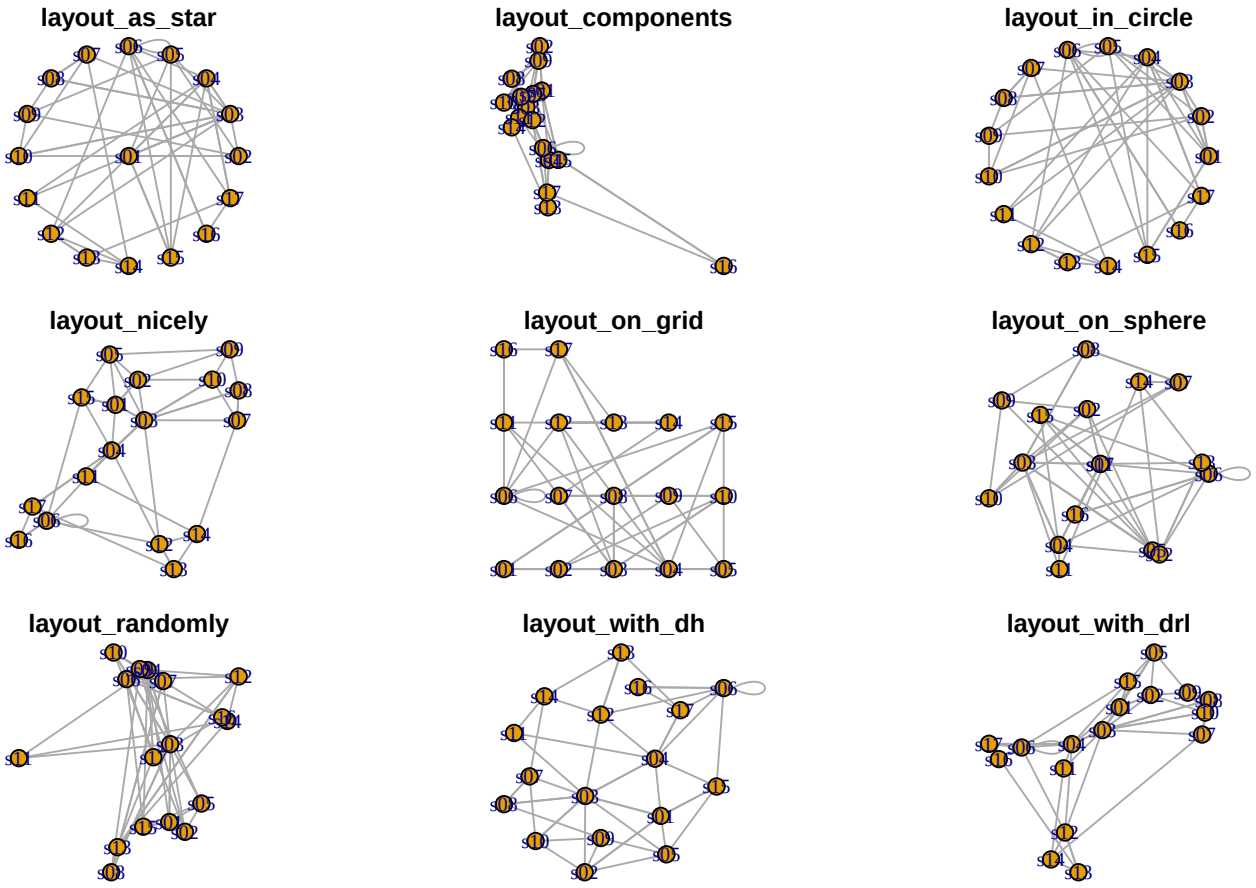


Let's take a look at all available layouts in igraph:

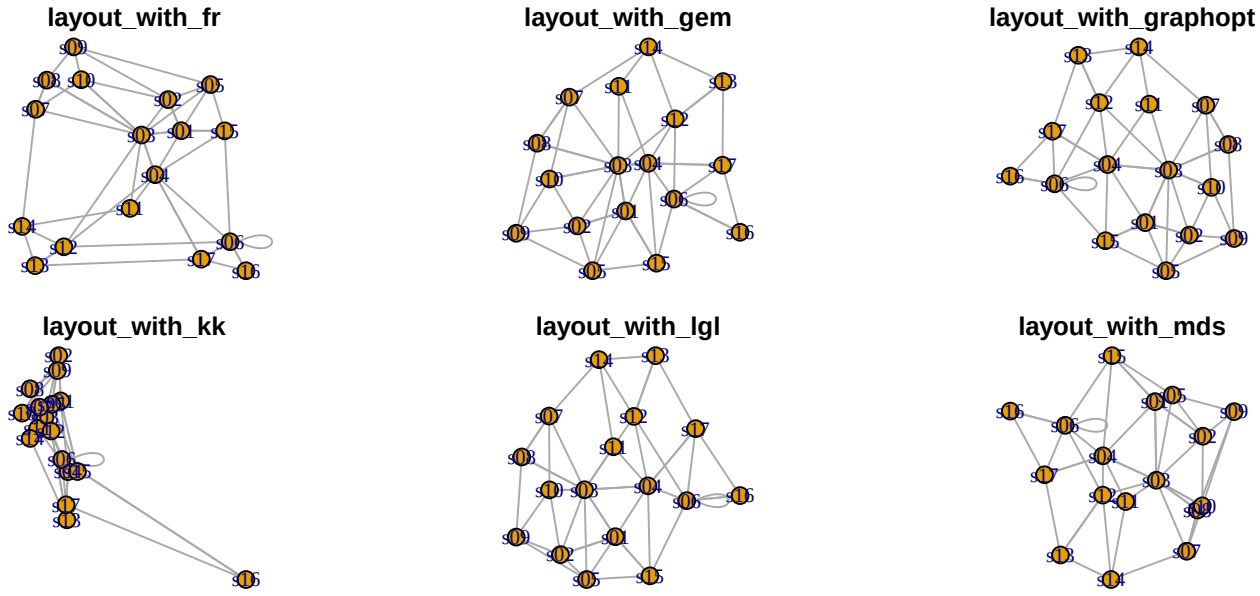
```
layouts <- grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
# Remove layouts that do not apply to our graph.
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|tree", layouts)]

complete_address<-"~/Dropbox/BioInfo2017/polnet2018/Data files/"
nodes <- read.csv(paste0(complete_address,"Dataset1-Media-Example-NODES.csv"), header=T, as.is=T)
links <- read.csv(paste0(complete_address,"Dataset1-Media-Example-EDGES.csv"), header=T, as.is=T)
net <- graph_from_data_frame(d=links, vertices=nodes, directed=T)
par(mfrow=c(3,3), mar=c(1,1,1,1))
for (layout in layouts) {
  print(layout)
  l <- do.call(layout, list(net))
  plot(net, edge.arrow.mode=0, layout=l, main=layout) }
```

```
## [1] "layout_as_star"
## [1] "layout_components"
## [1] "layout_in_circle"
## [1] "layout_nicely"
## [1] "layout_on_grid"
## [1] "layout_on_sphere"
## [1] "layout_randomly"
## [1] "layout_with_dh"
## [1] "layout_with_drl"
```




```
## [1] "layout_with_fr"
## [1] "layout_with_gem"
## [1] "layout_with_graphopt"
## [1] "layout_with_kk"
## [1] "layout_with_lgl"
## [1] "layout_with_mds"
```



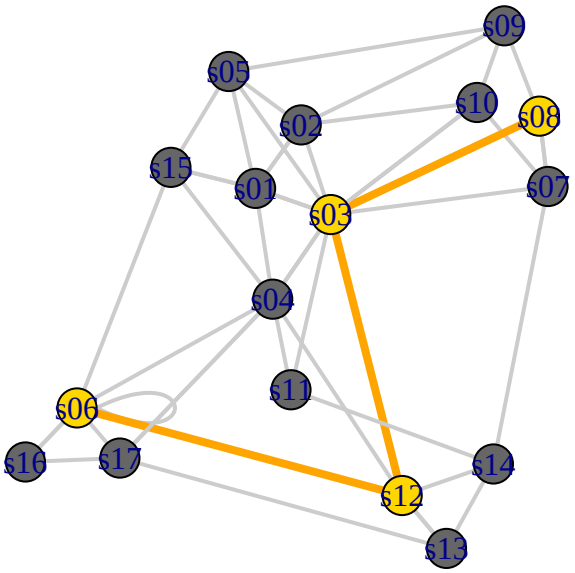

## 3.1 Highlight elements of a network

We can also highlight a path in the network:

```
news.path <- shortest_paths(net,
                            from = V(net)[media=="MSNBC"],
                             to  = V(net)[media=="New York Post"],
                             output = "both") # both path nodes and edges

# Generate edge color variable to plot the path:
ecol <- rep("gray80", ecount(net))
ecol[unlist(news.path$epath)] <- "orange"
# Generate edge width variable to plot the path:
ew <- rep(2, ecount(net))
ew[unlist(news.path$epath)] <- 4
# Generate node color variable to plot the path:
vcol <- rep("gray40", vcount(net))
vcol[unlist(news.path$vpath)] <- "gold"

plot(net, vertex.color=vcol, edge.color=ecol,
     edge.width=ew, edge.arrow.mode=0)
```




### 3.1.1 Exercises

1. Using the networks from slide number calculate the distances, mean of distances,the path from the most distant nodes and the clustering coefficient
2. Using the next networks calculate the distances, mean of distances, the path from the most distant nodes and the clustering coefficient and highlight this path.

```
g1<-barabasi.game(100,directed = FALSE)
g2<-random.graph.game(100,0.20)
g3<-sample_smallworld(1,100,p=0.2,nei=3)
```

3. From the network you previous (nodes=students, links= neighbour states), alculate the distances, mean of distances, the path from the most distant nodes and the clustering coefficient and highlight this path.