Graph Measurements Part 2

Load libraries

```
library(dplyr)
library(igraph)
library(ggplot2)
library(tidygraph)
library(networkD3)
library(visNetwork)
library(knitr) # For table rendering
```

Graph object

```
df = read.table("./data/data.tsv", header = T)
veccol = c(rep("pink",5), rep("light blue",6))
g = graph_from_data_frame(df)
net = g
```

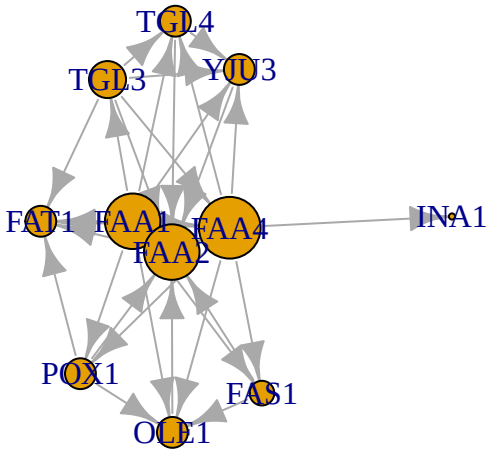
Node degrees

The function degree() has a mode of in for in-degree, out for out-degree, and all or total for total degree

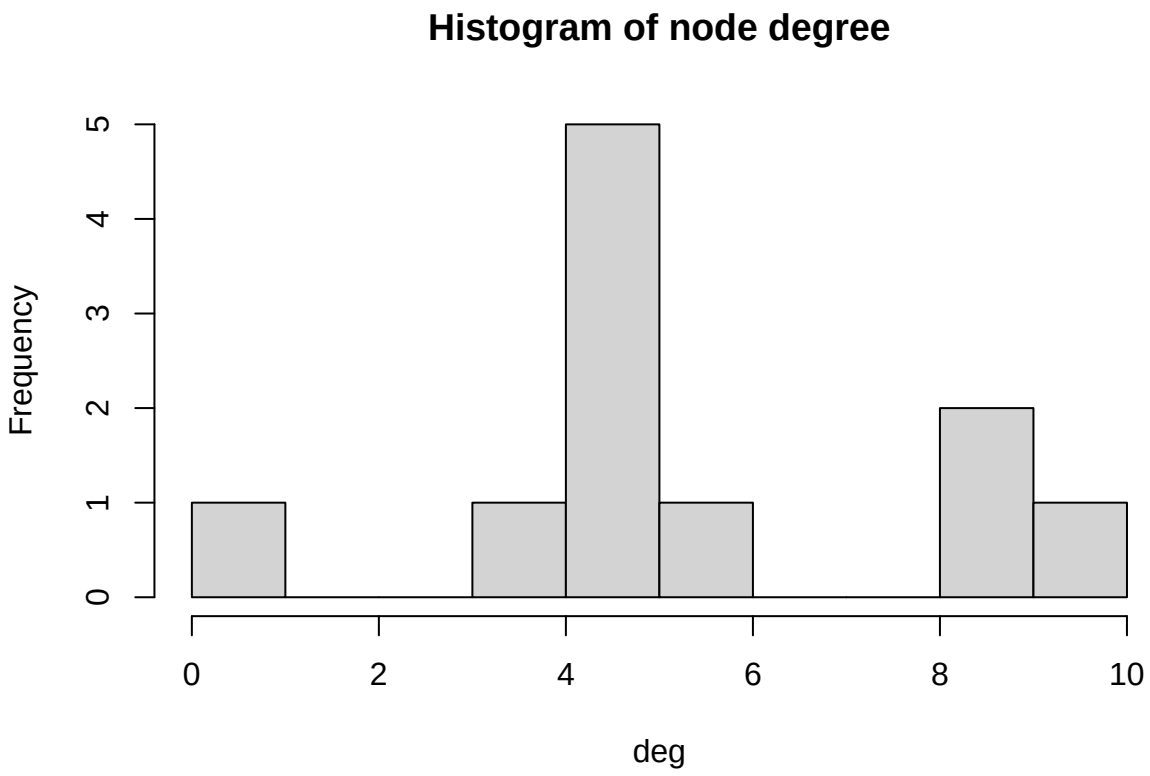
```
deg <- degree(net, mode="all")
print(sort(deg))
```

```
## INA1 FAS1 POX1 TGL4 YJU3 OLE1 FAT1 TGL3 FAA1 FAA2 FAA4
##    1    4    5    5    5    5    5    6    9    9   10
```

```
plot(net, vertex.size=deg*3)
```




```
hist(deg, breaks=1:vcount(net)-1, main="Histogram of node degree")
```



Centrality

Read https://towardsdatascience.com/graph-analytics-introduction-and-concepts-of-centrality-8f5543b55de3

Centrality applies to node level while Centralization applies to graph level

Degree centrality

Degree centrality measures how connected an entity is by counting the number of direct links each entity has to others in the network. This centrality measure can reveal how much activity is going on and who are its most active members.[1]

```
# mode: in/out/all or total
degree(net, mode="in")
```

```
## POX1 FAA1 TGL3 TGL4 FAA4 FAS1 FAA2 YJU3 OLE1 FAT1 INA1
##    2    0    1    3    2    2    8    4    4    5    1
```

Closeness centrality

Centrality is based on the distance to others in the graph. Calculated as inverse of the node's average geodesic distance to others in the network.

Closeness centrality measures the proximity of an entity to the other entities in the social network [1]. An entity with a high measure of closeness centrality has the shortest paths to the other entities. This measure allows them to pass on, and receive communications more quickly than anybody else in the organization. Information travels further to and from an entity on the edge of a network that is attached to few other entities. They have a lower measure of closeness centrality.

Closeness centrality measures both direct and indirect closeness: [1]

- Direct closeness is when two entities are connected by a link.
- Indirect closeness exists when information can pass only from one entity to another by way of a path that runs through one or more entities.

```
closeness(net, mode="all", weights=NA)
```

```
##       POX1       FAA1       TGL3       TGL4       FAA4       FAS1       FAA2
## 0.06666667 0.09090909 0.07142857 0.06666667 0.10000000 0.06250000 0.09090909
##       YJU3       OLE1       FAT1       INA1
## 0.06666667 0.06666667 0.06666667 0.05263158
```

```
centr_clo(net, mode="all", normalized=T)
```

```
## $res
##  [1] 0.6666667 0.9090909 0.7142857 0.6666667 1.0000000 0.6250000 0.9090909
##  [8] 0.6666667 0.6666667 0.6666667 0.5263158
##
## $centralization
## [1] 0.6297198
##
## $theoretical_max
## [1] 4.736842
```

Betweenness centrality

Centrality based on a broker position connecting others. Number of geodesics that pass through the node or the edge. The vertex and edge betweenness are (roughly) defined by the number of geodesics (shortest paths) going through a vertex or an edge.

Betweenness centrality measures the number of paths that pass through each entity. This measure might identify entities with the ability to control information flow between different parts of the network. These entities are called gatekeeper entities. Gatekeepers might have many paths that run through them that allows them to channel information to most of the others in the network. Alternatively, they might have few paths that run through them, but still play a powerful communication role if they exist between different network clusters [1].

```
betweenness(net, directed=T, weights=NA)
```

```
## POX1 FAA1 TGL3 TGL4 FAA4 FAS1 FAA2 YJU3 OLE1 FAT1 INA1
##    0    0    0    0    5    0    4    0    0    0    0
```

```
edge_betweenness(net, directed=T, weights=NA)
```

```
##  [1] 1 1 1 1 1 2 1 1 2 1 1 2 2 2 1 5 1 5 1 1 1 2 1 1 3 1 1 1 2 1 2 1
```

```
centr_betw(net, directed=T, normalized=T)
```

```
## $res
##  [1] 0 0 0 0 5 0 4 0 0 0 0
##
## $centralization
## [1] 0.05111111
##
## $theoretical_max
## [1] 900
```

Eigenvector Centrality

This metric measures the importance of a node in a graph as a function of the importance of its neighbors. If a node is connected to highly important nodes, it will have a higher Eigen Vector Centrality score as compared to a node which is connected to lesser important nodes.

Eigenvector measures how connected an entity is and how much direct influence it might have over other connected entities in the network [1]. The eigenvector scores of the entities it is connected to, is considered. For example, a person with a high eigenvector score is likely to be at the center of a cluster of key entities that themselves have high eigenvector scores. That person can communicate directly with those key entities compared with a person with a low eigenvector score on the periphery of the network.

Hubs and authorities are the terms that are used to describe the two eigenvector scores that are calculated in networks that contain directed links. Hubs see the scores for outbound links, and authorities see the scores for inbound links. A high-scoring hub has many outbound links to high-scoring authorities, and a high-scoring authority has many inbound links from high-scoring hubs. [1]

```
eigen_centrality(g)$vector
```

```
##      POX1      FAA1      TGL3      TGL4      FAA4      FAS1      FAA2      YJU3
## 0.6394758 0.9802349 0.7437332 0.6578800 1.0000000 0.5406743 0.9802349 0.6578800
##      OLE1      FAT1      INA1
## 0.6244794 0.6551043 0.1508179
```

Pagerank centrality

Calculates the Google PageRank for the specified vertices. For the explanation of the PageRank algorithm, see the following webpage: http://infolab.stanford.edu/~backrub/google.html

```
page_rank(g)
```

```
## $vector
##       POX1       FAA1       TGL3       TGL4       FAA4       FAS1       FAA2
## 0.04714358 0.03831262 0.04193103 0.05427186 0.04905930 0.04714358 0.24492396
##       YJU3       OLE1       FAT1       INA1
## 0.07733739 0.08053695 0.27581457 0.04352517
##
## $value
## [1] 1
##
## $options
## NULL
```

Centralization

Centralization is a method for creating a graph level centralization measure from the centrality scores of the vertices.

```
# Returns res -
# vertex centrality,
# centralization, and
# theoretical_max - maximum centralization score for a graph of that size.

# Centralize a graph according to the degrees of vertices
centr_degree(net, mode="in", normalized=T)$centralization
```

```
## [1] 0.5090909
```

```
# Centralize a graph according to the betweenness of vertices
centr_betw(net)$centralization
```

```
## [1] 0.05111111
```

```
# Centralize a graph according to the eigenvector centrality of vertices
centr_eigen(net)$centralization
```

```
## [1] 0.3743873
```

Hubs and authorities

- Hubs: large number of outgoing links
- Authorities would get many incoming links from hubs

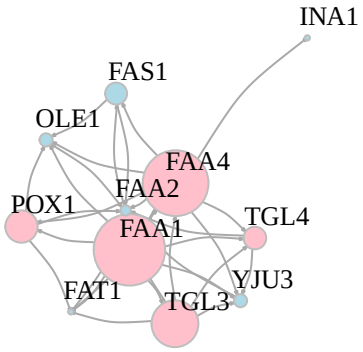
Hubs and authorities are the terms that are used to describe the two eigenvector scores that are calculated in networks that contain directed links. Hubs see the scores for outbound links, and authorities see the scores for inbound links. A high-scoring hub has many outbound links to high-scoring authorities, and a high-scoring authority has many inbound links from high-scoring hubs. [1]

Hub: A measure of how well-connected an entity is, based on its outbound links. Hub is one of two eigenvector centrality measures used in social network analysis.

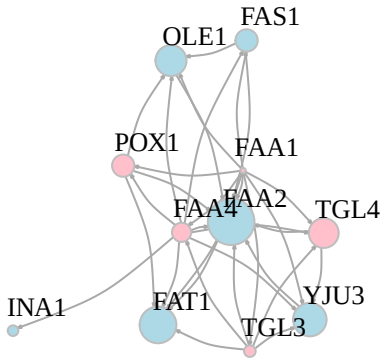
Authorities: A measure of how well-connected an entity is, based on its inbound links. Authority is one of two eigenvector centrality measures used in social network analysis. [1]

```
hs <- hub_score(net, weights=NA)$vector
as <- authority_score(net, weights=NA)$vector
par(mfrow=c(1,2))
plot(net, vertex.size=hs*50, main="Hubs", edge.arrow.size=0.1, vertex.color=veccol, vertex.frame.color=
plot(net, vertex.size=as*30, main="Authorities",,edge.arrow.size=0.1, vertex.color=veccol,vertex.frame.
```

Hubs

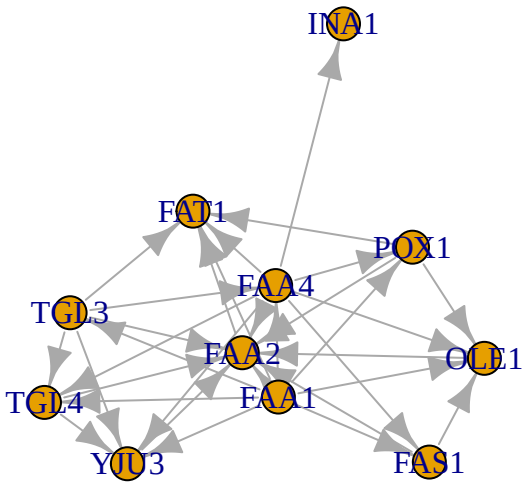


Authorities



Combine all centrality measures

```
# Directed graph
plot(g)
```




```
d1 = degree(g)
cl = closeness(g)
```

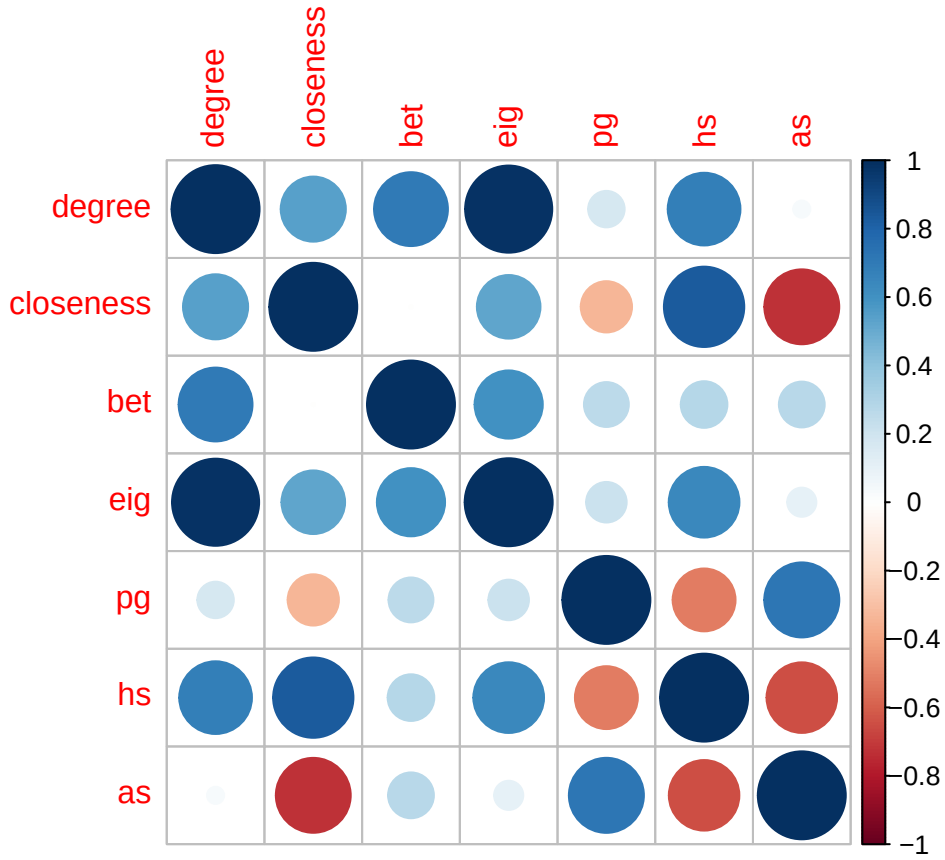
```
## Warning in closeness(g): At centrality.c:2784 :closeness centrality is not well-
## defined for disconnected graphs
```

```
bet = betweenness(g)
eig = eigen_centrality(g)$vector
pg = page_rank(g)$vector
hs <- hub_score(g, weights=NA)$vector
as <- authority_score(g, weights=NA)$vector

centdf = data.frame(degree = d1, closeness = cl, bet = bet, eig= eig, pg = pg, hs = hs, as = as)

cor(centdf)
```

```
##                degree    closeness          bet       eig         pg         hs
## degree     1.00000000  0.547679634  0.705203395 0.9808557  0.1718962  0.6833756
## closeness 0.54767963  1.000000000 -0.001191608 0.5229275 -0.3395854  0.8341511
## bet        0.70520339 -0.001191608  1.000000000 0.6008687  0.2604394  0.2810249
## eig        0.98085571  0.522927541  0.600868661 1.0000000  0.2124385  0.6458662
## pg         0.17189620 -0.339585365  0.260439404 0.2124385  1.0000000 -0.5112477
## hs         0.68337563  0.834151055  0.281024869 0.6458662 -0.5112477  1.0000000
## as         0.03779731 -0.725117504  0.271285846 0.1098315  0.7276602 -0.6430467
##                   as
```

```
## degree      0.03779731
## closeness -0.72511750
## bet         0.27128585
## eig         0.10983153
## pg          0.72766016
## hs         -0.64304672
## as          1.00000000
```

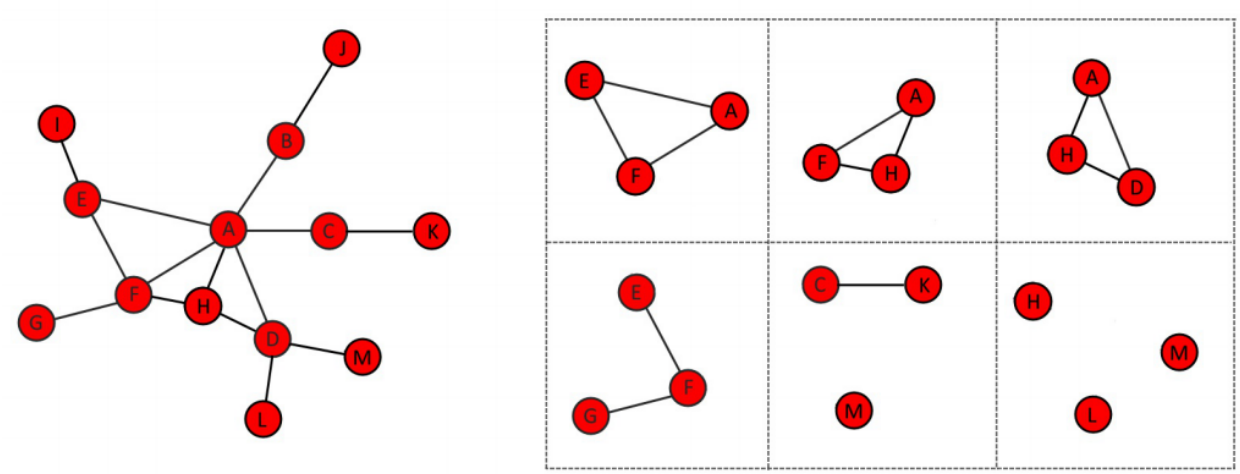
```
corrplot::corrplot(cor(centdf))
```



Transitivity

Transitivity measures the probability that the adjacent vertices of a vertex are connected. This is sometimes also called the clustering coefficient.

It calculates the fraction of connected **triplets** that are **closed triangles** for a given network. A connected triplet is three nodes, where at least two pairs of them are connected; a **triangle means** that every pair of the three nodes is connected. A triangle is a triplet but not the vice versa. For example, if A is friend of B and B is a friend of C, then A-B-C is a triplet; if furthermore, A and C are also friends, then A-B-C is a triangle. Hence, **a large value of transitivity means that most triplets are triangles.** ***In other words, your friends are also likely friends with each other.***



There are two versions of transitivity: **global and local**. For a given network, the global transitivity calculates the ratio between the total number of triangles and the total number of triplets, while the local transitivity is defined for each node and calculates the ratio between the number of triangles and the number of triplets centered at each node.

- global - ratio of triangles (direction disregarded) to connected triples.
- local - ratio of triangles to connected triples each vertex is part of.

```
V(net)
```

```
## + 11/11 vertices, named, from d79d5e6:
##  [1] POX1 FAA1 TGL3 TGL4 FAA4 FAS1 FAA2 YJU3 OLE1 FAT1 INA1
```

```
transitivity(net, type='global')
```

```
## [1] 0.7021277
```

```
transitivity(net, type='local')
```

```
##  [1] 0.9000000 0.6111111 0.8666667 1.0000000 0.4888889 1.0000000 0.6111111
##  [8] 1.0000000 0.9000000 0.9000000       NaN
```

```
# Count how many triangles a vertex is part of, in a graph, or just list the triangles of a graph.
count_triangles(g, vids = 'FAA4')
```

```
## [1] 22
```

Neighbors

Neighboring (adjacent) vertices in a graph

A vertex is a neighbor of another one (in other words, the two vertices are adjacent), if they are incident to the same edge.

```
nt1 = neighbors(g, "POX1")
print(nt1)
```

```
## + 3/11 vertices, named, from d79d5e6:
## [1] FAA2 OLE1 FAT1
```

```
nt2 = neighbors(g, "TGL3")
print(nt2)
```

```
## + 5/11 vertices, named, from d79d5e6:
## [1] TGL4 FAA4 FAA2 YJU3 FAT1
```

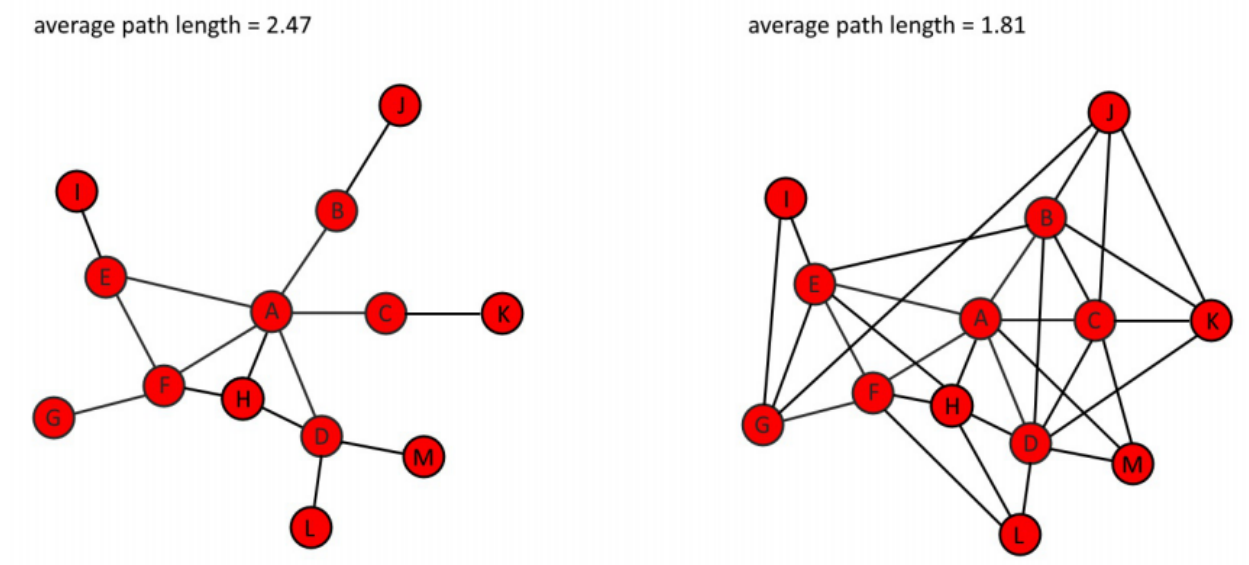
```
intersection(nt1, nt2)
```

```
## + 2/11 vertices, named, from d79d5e6:
## [1] FAA2 FAT1
```

Distances and paths

Average path length

Mean of the shortest distance between each pair of nodes in the network




```
mean_distance(net, directed=F)
```

```
## [1] 1.418182
```

```
mean_distance(net, directed=T)
```

```
## [1] 1.219512
```

```r
# We can also find the length of all shortest paths in the graph:
distances(net)
```

```
##      POX1 FAA1 TGL3 TGL4 FAA4 FAS1 FAA2 YJU3 OLE1 FAT1 INA1
## POX1    0    1    2    2    1    2    1    2    1    1    2
## FAA1    1    0    1    1    1    1    1    1    1    1    2
## TGL3    2    1    0    1    1    2    1    1    2    1    2
## TGL4    2    1    1    0    1    2    1    1    2    2    2
## FAA4    1    1    1    1    0    1    1    1    1    1    1
## FAS1    2    1    2    2    1    0    1    2    1    2    2
## FAA2    1    1    1    1    1    1    0    1    1    1    2
## YJU3    2    1    1    1    1    2    1    0    2    2    2
## OLE1    1    1    2    2    1    1    1    2    0    2    2
## FAT1    1    1    1    2    1    2    1    2    2    0    2
## INA1    2    2    2    2    1    2    2    2    2    2    0
```

```r
# Extract the distances to a node or set of nodes we are interested in
distances(net, v = c('POX1','FAA1'), to = c('TGL3','TGL4'))
```

```
##      TGL3 TGL4
## POX1    2    2
## FAA1    1    1
```

```r
# Find the shortest path between specific nodes.
shortest_paths(net, from = "POX1", to = "TGL4", output = "both")
```

```
## Warning in shortest_paths(net, from = "POX1", to = "TGL4", output = "both"): At
## structural_properties.c:745 :Couldn't reach some vertices
```

```
## $vpath
## $vpath[[1]]
## + 0/11 vertices, named, from d79d5e6:
##
##
## $epath
## $epath[[1]]
## + 0/32 edges from d79d5e6 (vertex names):
##
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

```r
# Identify the edges going into or out of a vertex
incident(net,v = "OLE1",mode = 'in')
```

```
## + 4/32 edges from d79d5e6 (vertex names):
## [1] POX1->OLE1 FAA1->OLE1 FAA4->OLE1 FAS1->OLE1
```

```
incident(net,v = "OLE1",mode = 'out')
```

```
## + 1/32 edge from d79d5e6 (vertex names):
## [1] OLE1->FAA2
```

```
incident(net,v = "OLE1",mode = 'all')
```

```
## + 5/32 edges from d79d5e6 (vertex names):
## [1] OLE1->FAA2 POX1->OLE1 FAA1->OLE1 FAA4->OLE1 FAS1->OLE1
```

```
# For a single node, use incident(), for multiple nodes use incident_edges()
incident_edges(net, v=c("OLE1","FAA4"), mode="in");
```

```
## $OLE1
## + 4/32 edges from d79d5e6 (vertex names):
## [1] POX1->OLE1 FAA1->OLE1 FAA4->OLE1 FAS1->OLE1
##
## $FAA4
## + 2/32 edges from d79d5e6 (vertex names):
## [1] FAA1->FAA4 TGL3->FAA4
```