Prepare metabolic graph

Anatoly Sorokin

Tue 31 Jul 2018

Introduction

In this tutorial we are going to work with graphs. Graphs, or networks are usually object consisting of several kind of nodes (or vertices) and links (or edges) connecting them. Analysis of network usually splitting into following parts

1. Loading network
2. Configuration of the network, structure changing
3. Estimation of network metrics
4. Analysis and visualisation
5. Extension, modification of the network

In general network is loaded only once, while other steps could be repeated in arbitrary order in line with analysis purpoces. In this tutorial we are going to consider steps in the order shown above.

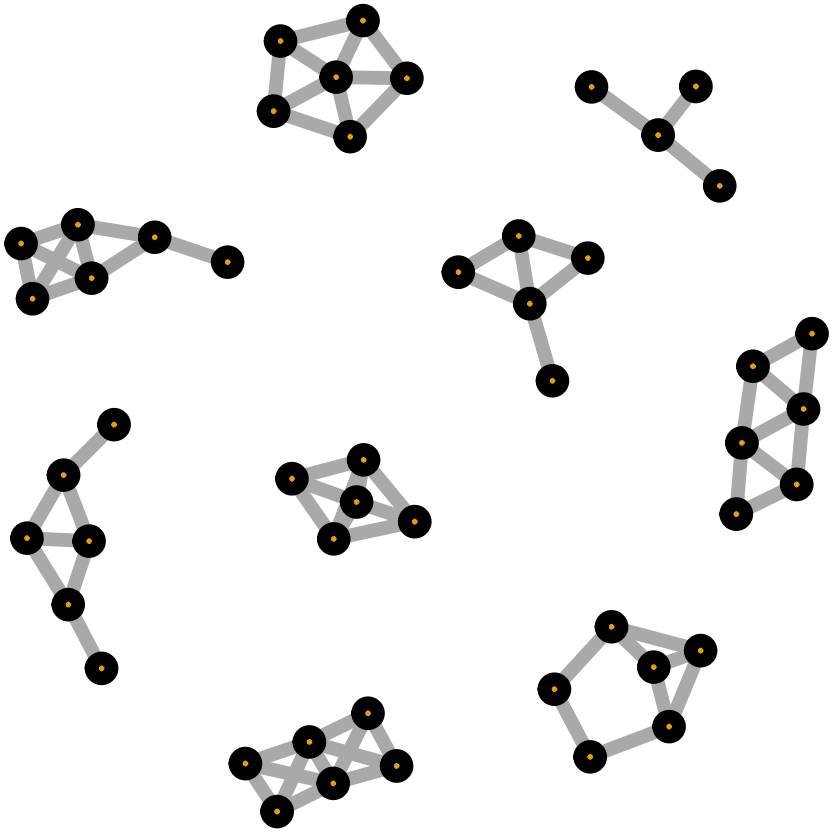
igraph

igraph – one of the most widely used network analysis package. It supports most of standard graph metrix, set of layout algorithms and rich visualisation framework. You have installed igraph already, but for the record for its installatition the following command should be executed:

```
install.packages('igraph',dependencies = c("Depends", "Imports", "Suggests"))
```

Let's test that installation is successfull:

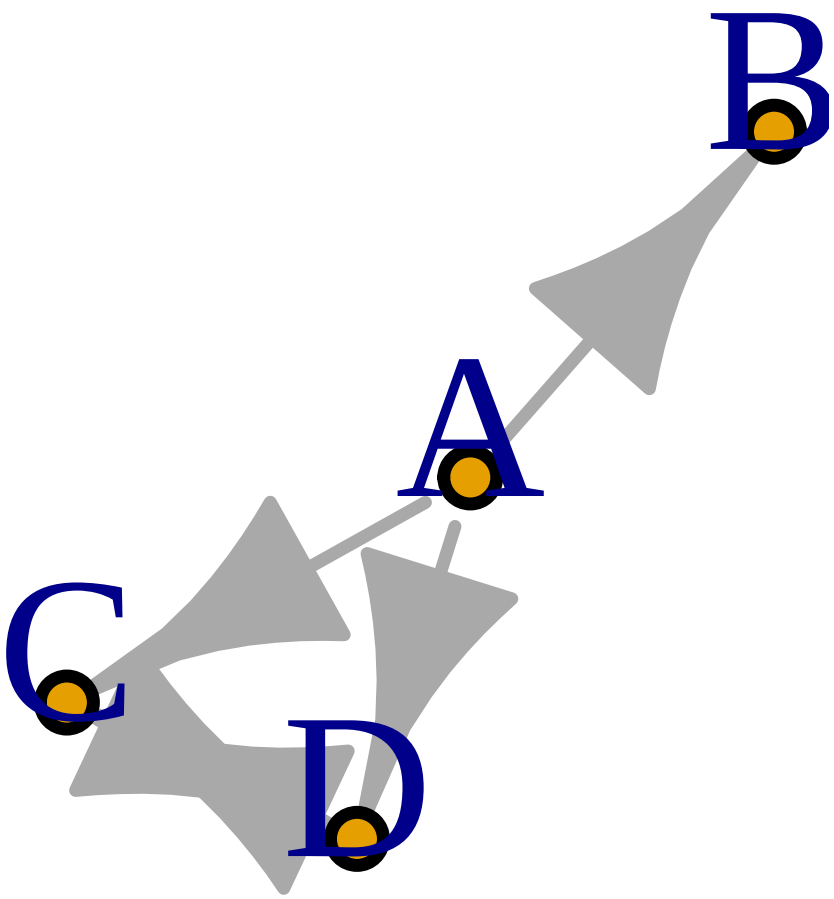
```
library(igraph)
library(ggplot2)
g<-make_graph('Nonline')
plot(g,vertex.label=NA,vertex.size=5)
```



Making a graph

As I said already graph is a combination of two sets: $G = (V, E)$, where V is set of all vertices, and E is a set of all edges linking those vertices. Visually vertises are drown as points or shapes, while edges as connecting lines or arrows. There are two types of graphs: directed and undirected. In undirected graph edge $A- > B$ is the same as edge $B- > A$, while in directed graph they are considered different. Let's make simple directed graph in `igrap`:

```
graph <- make_graph( ~ A-+B:C:D,C+-+D)
plot(graph,vertex.label.dist=2,vertex.label.degree=-pi/2)
```



By using this way of graph assembly, which is called “symbolic”, we explicitly define names of vertices and edges between them. Note sign ~ in the begining of definition string. That symbol marks all following as formula. Formulas are used in R widely, for example in regression and hypothesis testing, but their discussion is outside the scope of this tutorial.

Vertices in the previous example are named by letters from A to D, but any string could be used as a vertex name. The only restriction is that if name contains spaces or special symbols it should be quoted:

```
make_graph("this is" +- "a silly" -+ "graph here" )
```

Directed edges are represented by string `-+` and `+-`, bidirectional link in that case will be shown as `+-+`. Undirected edges are represented by string `-` or `--` or even `------`. The number of `-` letters is meaningless, but such representation is useful to make code more readable.

Function `V` returns list of graph vertices, and function `E` – list of all edges.

```
V(graph)
```

Practical task: make your own graph with 5 nodes and plot it. We will discuss parameters of `plot` function later.

```
## + 4/4 vertices, named, from 7f0958f:
## [1] A B C D
```

```
E(graph)
```

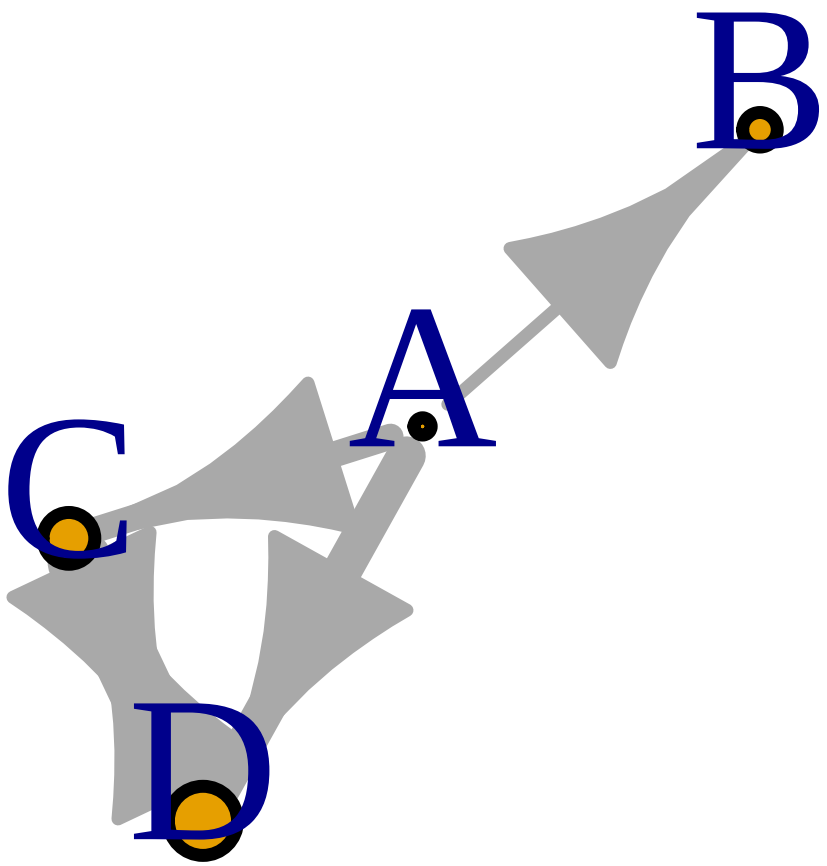
```
## + 5/5 edges from 7f0958f (vertex names):
## [1] A->B A->C A->D C->D D->C
```

All three kind of objects: graphs, vertices and edges could have attributes, required for analysis. For example, node could have name, reference id in some database, etc. Some of this attributes could influence plotting of the graph.

To get access to attributes standard R $ notation is used `V(g)$att_name`. To demonstrate we will change size of the vertex and width of the edge:

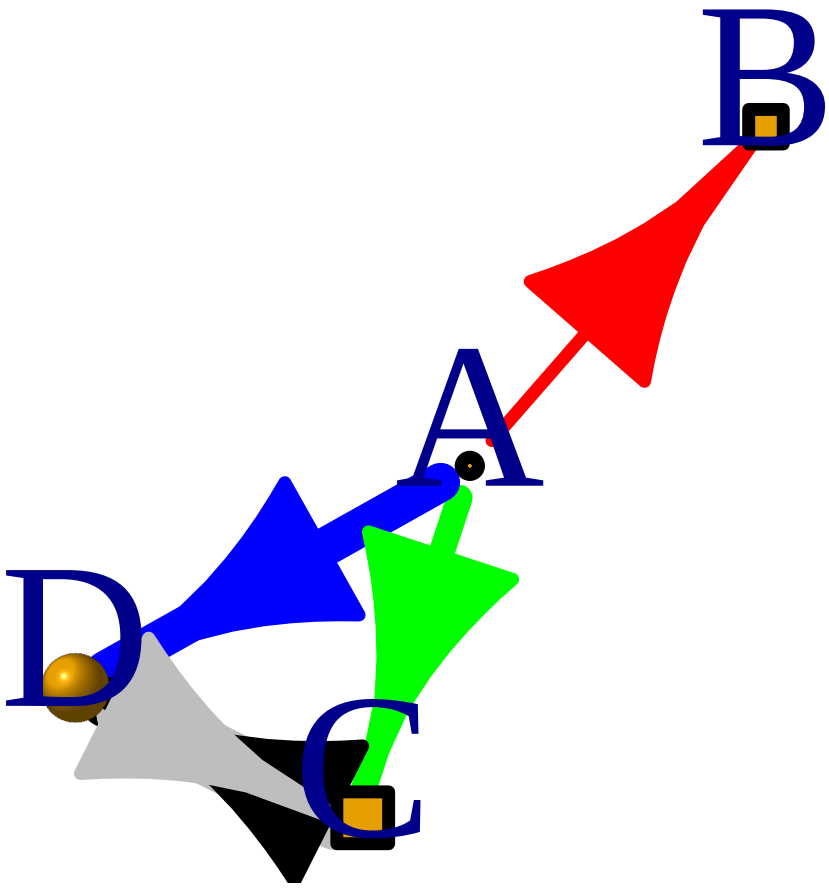
List of plotting attributes could be found in documentation:
`?igraph.plotting`
The most important we will show later.

```
V(graph)$size<-1:4*5
E(graph)$width<-1:5
plot(graph,vertex.label.dist=2,vertex.label.degree=-pi/2)
```



In the code above we have changed size of vertices and edge of edges. Let's assign different shapes to our graph nodes:

```
E(graph)$color <- c('red','green','blue','black','gray')
V(graph)$shape<-c('circle', 'square', 'csquare', 'sphere')
plot(graph,vertex.label.dist=2,vertex.label.degree=-pi/2)
```



Simple visualisation

Brilliant tutorial on igraph visualisation is available at `http://kateto.net/network-visualization`. We will consider most basic functions leaving color and font properties aside.

Graphical parameters

The most frequently used parameters to specify visualisation is shown below.

For vertices:

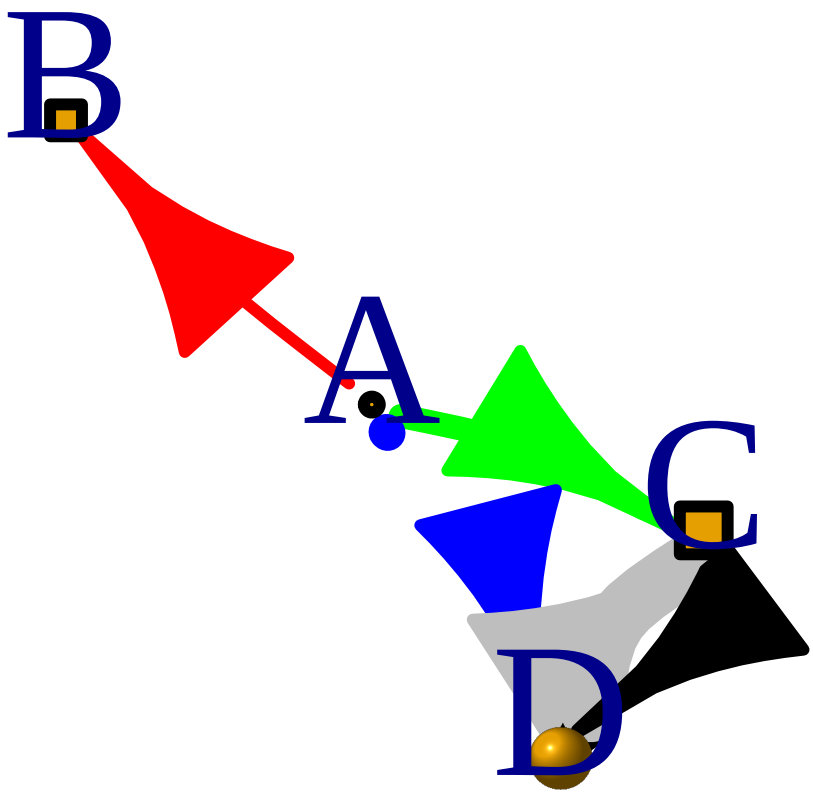
Attribute	Value
vertex.color	Fill color
vertex.frame.color	Line color
vertex.shape	Shape – “none”, “circle”, “square”, “csquare”, “rectangle”, “crectangle”, “vrectangle”, “pie”, “raster”, or “sphere”
vertex.size	Vertex size, the default is 15, for large networks 1-3
vertex.label	Label, to hide all labels place NA
vertex.label.dist	The distance between vertex and label
vertex.label.degree	Angular alignment of the label: 0 on the right, “pi” on the left, “pi/2” under, and “-pi/2” above

For edges

Attribute	Value
edge.color	Color
edge.width	Line width, by default is 1
edge.arrow.size	Arrow size by default is 1, better use 0.2-0.4
edge.lty	line type 0 or “blank” – no line, 1 or “solid” solid line, 2 or “dashed” dashed line, 3 or “dotted” dotted line, 4 or “dotdash” dot-dash line, 5 or “longdash” long dash line, 6 or “twodash” double dash line
edge.label	Edge label
edge.curved	Edge curvature, in a range 0-1 (0 – strait line)

Some of this attribute we have used already. Dealing with real life networks you should use vertex.size=1,vertex.label=NA to prevent hiding graph structure behind non necessary labels. You can specify these parameters at plot function call or set them as attributes to nodes and edges. In the latter case individual values will be used. If in the plot function parameter have length 1 than this value is applied to all elements of the graph, if length of the value is equal to the number of edges or nodes, than parameters applied individually:

```
plot(graph,
     vertex.label.dist=2,
     vertex.label.degree=-pi/2,
     edge.lty=1:5,
     edge.curved=1:5/10)
```

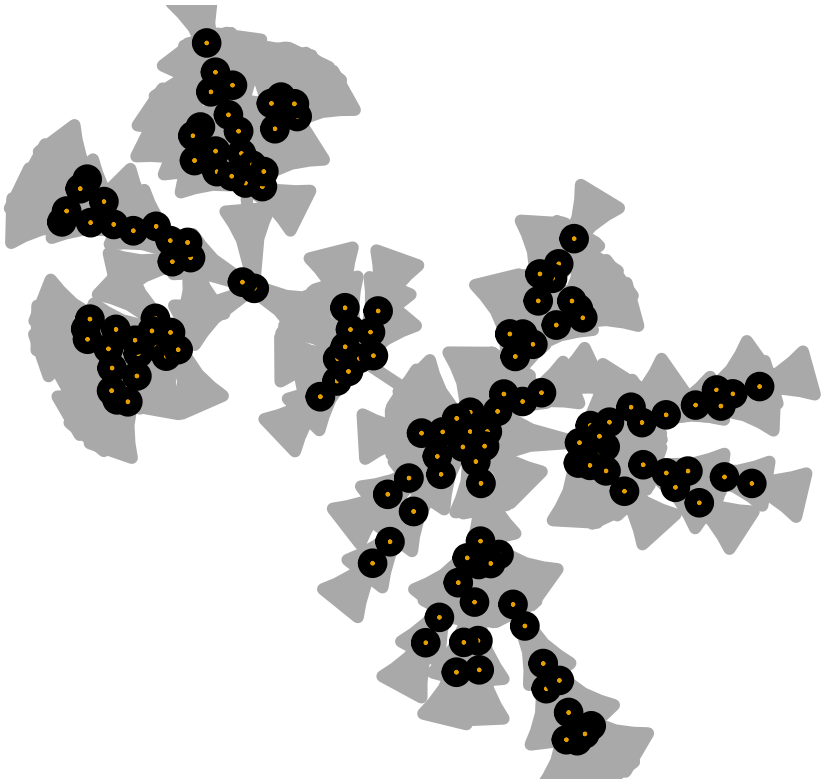


On the plot above each edge has its own curvature and line type.

Layout

The most diffucult as in theory of graphs as in practical applications is the task of laying graph on the plane with minimal vertex overlapping and edge crossing. The layout task. To demonstrate various layout algorithms we will create small scale-free graph:

```
bg<-barabasi.game(150,0.7)
plot(bg,vertex.size=5,vertex.label=NA,edge.arrow.size=0.3)
```



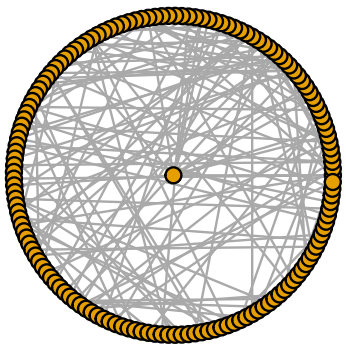
and layout our graph with various algorithms:

```
layouts <- grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|tree", layouts)]
par(mfrow=c(1,2), mar=c(1,1,1,1))
for (layout in layouts) {
  print(layout)
  l <- do.call(layout, list(bg))
  plot(bg,vertex.size=10,vertex.label=NA, edge.arrow.mode=0, layout=l, main=layout) }
```

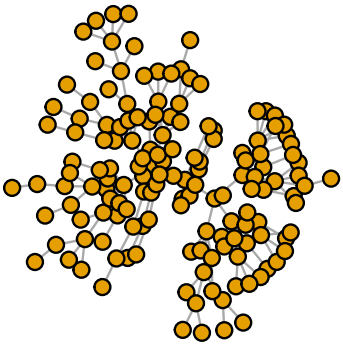
```
## [1] "layout_as_star"
```

```
## [1] "layout_components"
```

layout_as_star



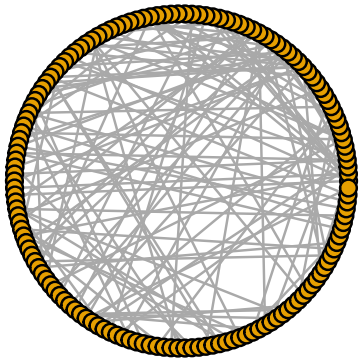
layout_components



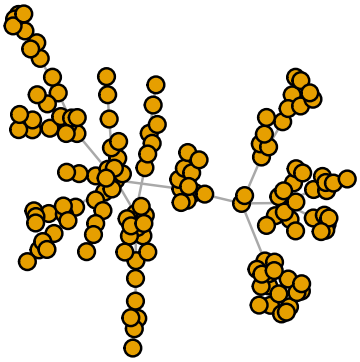
```
## [1] "layout_in_circle"
```

```
## [1] "layout_nicely"
```

layout_in_circle



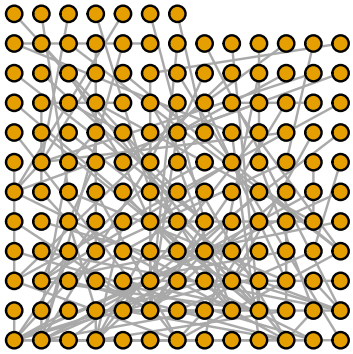
layout_nicely



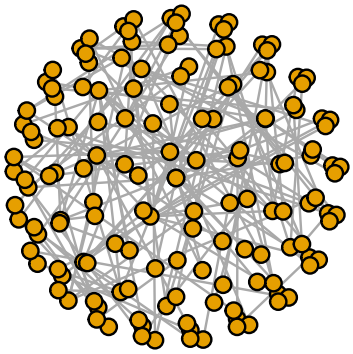
```
## [1] "layout_on_grid"
```

```
## [1] "layout_on_sphere"
```

layout_on_grid

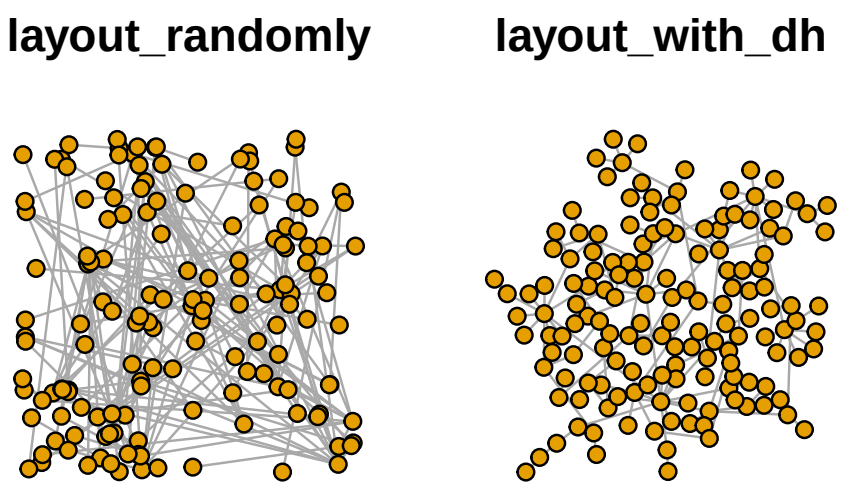


layout_on_sphere



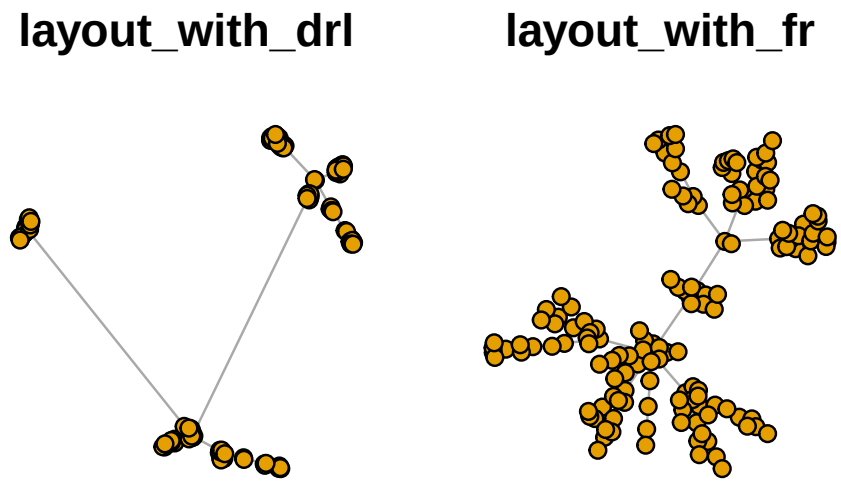
```
## [1] "layout_randomly"
```

```
## [1] "layout_with_dh"
```



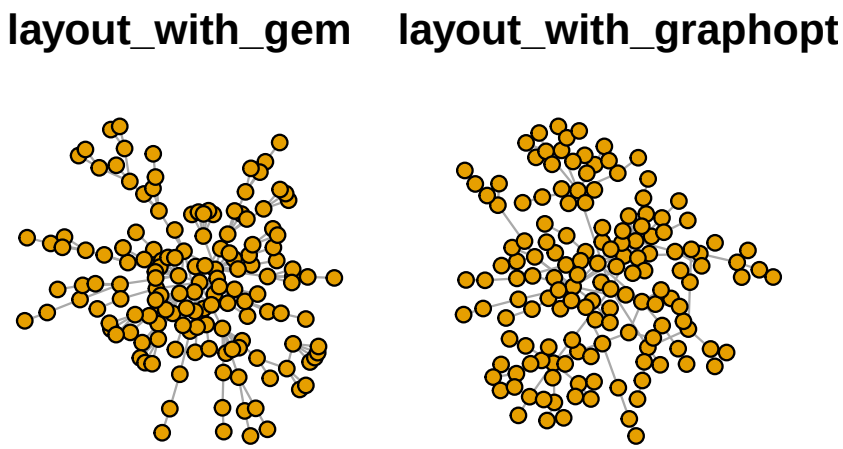

```
## [1] "layout_with_drl"
```

```
## [1] "layout_with_fr"
```



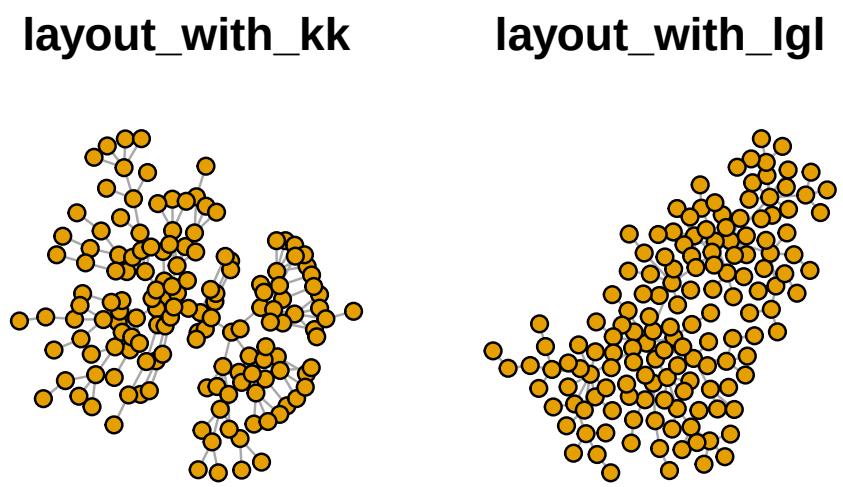

```
## [1] "layout_with_gem"
```

```
## [1] "layout_with_graphopt"
```



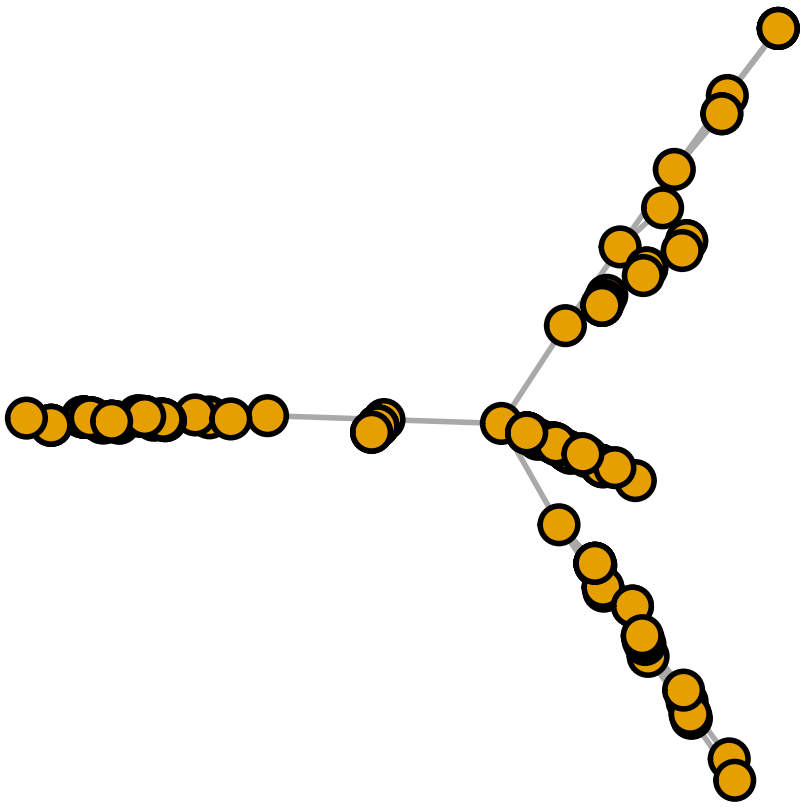

```
## [1] "layout_with_kk"
```

```
## [1] "layout_with_lgl"
```




```
## [1] "layout_with_mds"
```

layout_with_mds



`igraph` has function `layout_nicely`, which uses various heuristics to choose the best layout, but you should try manually. The `*_layout` functions returns coordinates of vertices in matrix with 2 (and for 3D layout 3) columns, so it is possible to create your own layout if needed:

```
pander(head(l))
```

1.649	-1.135
1.005	0.05993
1.294	-0.05375
-0.3106	0.1059
1.288	-0.05184
-0.3784	0.03147

Loading data from external resources

In real life we would never create graphs by hand. There are packages to load data from a standard formats: for example NetPathMiner is able to load data from SBML, BioPAX etc. For educational purposes we will load graph from SIF (simple interaction format) file. That format was developed by Cytoscape and used quite widely. SIF file contains three columsn separated byt tabulation: initial vertex, edge type, source vertex. We can load this data as data.frame:

```
sif.df<-read.table('BINDhuman.sif',
                   header=FALSE,
                   sep='\t',
                   quote = '')
names(sif.df)<-c('A','type','B')
```

Parameters to `read.table` function are following: `header = FALSE` tells that first line is not name of the column and have to be considered as data, `sep='\t'` defines tabulation as separation symbol.

Let's check structure of loaded data:

```
summary(sif.df)
```

```
##           A                    type
##  Unknown   : 2582   interactsWith:10235
##  HNF4-alpha: 2314   pd           :12740
##  TAF1      : 1776   pp           :15740
##  c-Myc     : 1640
##  TAFII250  : 1159
##  Max       :  933
##  (Other)   :28311
##        B
##  F2     :  291
##  Unknown:  249
##  Insulin:  207
##  CDK2   :  123
##  HLA-A  :  101
##  AFP    :   96
##  (Other):37648
```

```
pander(head(sif.df))
```

A	type	B
LAT	pp	Grb2
LAT	pp	PI3K_p85-alpha

A	type	B
LAT	pp	PLC-gamma
LAT	pp	Grap
LAT	pp	Gads
SLP-76	pp	Vav

Task: remove from graph nodes with names Unknown and "" (empty string). Count number of edges deleted. Repeat analysis with clean network.

It is interesting that most frequent initial node is Unknown.

Let's build graph with data.frame obtained:

```
g<-graph_from_data_frame(sif.df[,c(1,3,2)],directed = FALSE)
g
```

```
## IGRAPH c2a5487 UN-- 19906 38715 --
## + attr: name (v/c), type (e/c)
## + edges from c2a5487 (vertex names):
##  [1] LAT   --Grb2
##  [2] LAT   --PI3K_p85-alpha
##  [3] LAT   --PLC-gamma
##  [4] LAT   --Grap
##  [5] LAT   --Gads
##  [6] SLP-76--Vav
##  [7] SLP-76--Nck
##  [8] SLP-76--SLAP-130
## + ... omitted several edges
```

First line shows that graph is undirected (UN) and consists of 19K vertices and 37K edges. Size of graph and its type could be obtained explicitly:

```
vcount(g)
```

```
## [1] 19906
```

```
ecount(g)
```

```
## [1] 38715
```

```
is.directed(g)
```

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

Most graph algorithms assumes that there is no parallel edges and loops, so we will use finction `simplify` to remove unnecessary edges:

```
agg<-function(.x)toString(unique(.x))
sg<-simplify(g,
             remove.multiple = TRUE,
```

```
            remove.loops = TRUE,
            edge.attr.comb = agg)
sg
```

```
## IGRAPH 2eea7d7 UN-- 19906 30243 --
## + attr: name (v/c), type (e/c)
## + edges from 2eea7d7 (vertex names):
##  [1] LAT   --PI3K_p85-alpha
##  [2] LAT   --Grb2
##  [3] LAT   --ITK
##  [4] LAT   --Unknown
##  [5] LAT   --Grap
##  [6] LAT   --PLC-gamma
##  [7] LAT   --Gads
##  [8] LAT   --PLC-gamma-1
## + ... omitted several edges
```

we can see that about 8K edges were removed

```
table(E(g)$type)
```

```
##
## interactsWith            pd            pp
##         10235         12740         15740
```

```
table(E(sg)$type)
```

```
##
##     interactsWith                pd
##              6972             12452
##                pp pp, interactsWith
##             10818                 1
```

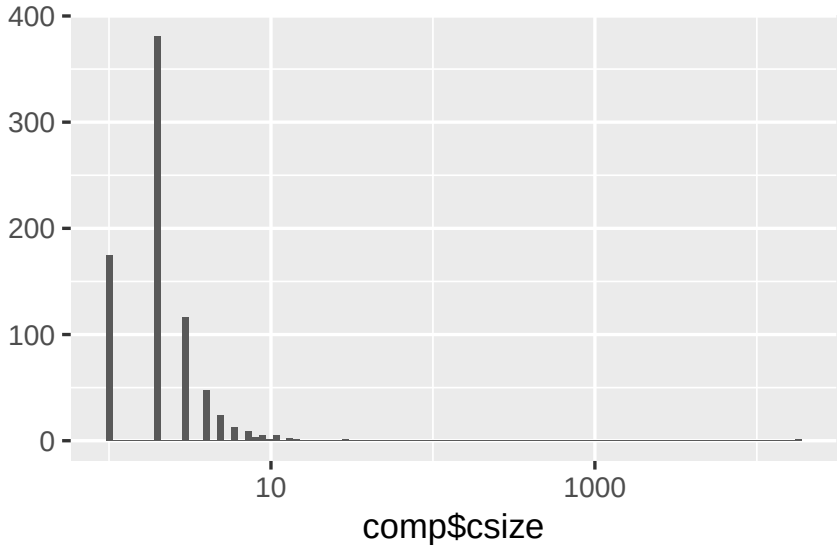
edges `interactsWith` were most abundant in parallel edges, and `pp` were least abundant.

Connectivity is the most important property of the network. Usually biological network contains one or two large cluster and many small ones. It is often enough to analyse properties of the main componet. So let's see how many components are in our network:

```
comp<-components(sg)
max(comp$no)
```

```
## [1] 785
```

```
qplot(comp$csize,log='x',bins=100)
```



We can see that there is 785 components and most of them consists of one to three verties only. Largest component contains 17963 vertices (90.2391239% of 19906 total graph nodes). Let's extract main component from the graph:

```
i<-which.max(comp$csize)
vg<-groups(comp)
csg<-induced_subgraph(sg,vg[[i]])
c(vcount(csg),max(comp$csize))
```

```
## [1] 17963 17963
```

```
table(E(csg)$type)
```

```
##
##     interactsWith                pd
##              6586             12354
##                pp pp, interactsWith
##             10056                 1
```

Function `groups` split vertices into corresponding component lists, while `induced_subgraph` create subgraph from those lists. Edge type frequency analysis shows that most of small components contains edges of type `pp`.

Basic graph metrix

Vertex degree and scale free networks

Number of incident edges for the vertex is called degree:

```
d<-igraph::degree(csg)
summary(d)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.
##    1.000    1.000    1.000    3.228    2.000
##     Max.
## 2314.000
```

It could be seen that half of nodes have degree 1 and about 75% degree 2. At the same time the highest degree is 2314. Let's look at this vertex:

```
topD<-which.max(d)
topD
```

```
## HNF4-alpha
##       1535
```

```
V(csg)[topD]
```

```
## + 1/17963 vertex, named, from 4ace70d:
## [1] HNF4-alpha
```

```
table(incident_edges(csg,topD)[[1]]$type)
```

```
##
##   pd
## 2314
```

all its edges have type `pd`.

Task: find position in the degree list for the node which have 50% of `pp` edges.

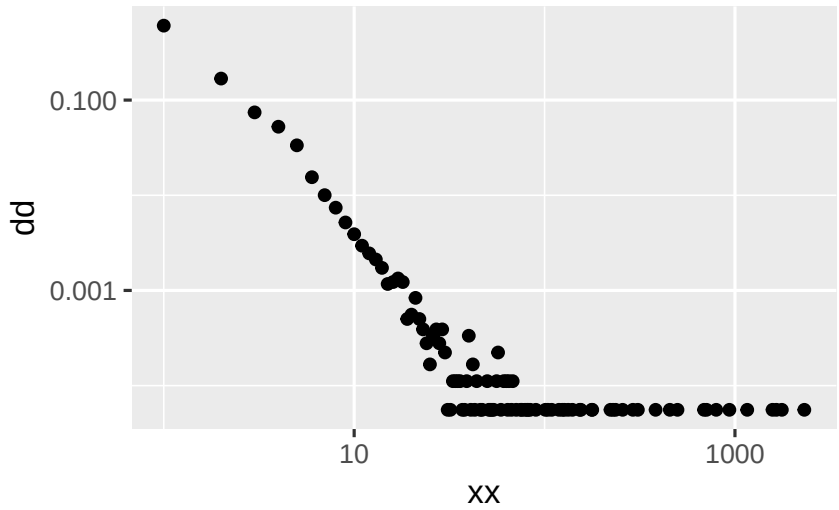
Scale-free networks

The probability to find a vertex with degree k in scale free network have power law with α between 2 and 3:

$$P(x = k) = \frac{x^{-\alpha}}{\zeta(\alpha)}, \quad 2 \leq \alpha \leq 3$$

The naive approach to check if the network is scale-free would be log-log plot of the degree distribution:

```
dd<-degree_distribution(csg)
xx<-1:length(dd)-1
ind<-which(dd>0)
dd<-dd[ind]
xx<-xx[ind]
qplot(xx,dd,log='xy')
```



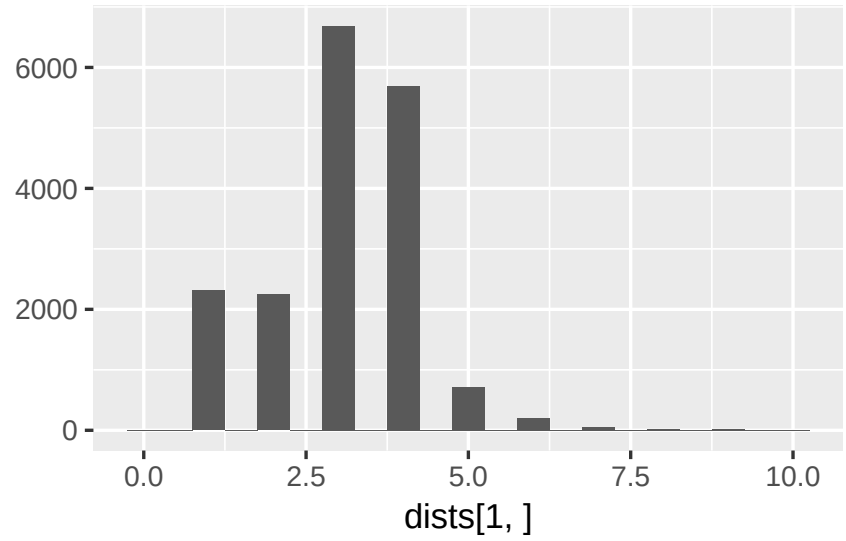
It is seen that the plot have wide hail and the slope of linear function is defined by the nodes with smallest degree. At the same time α parameter of the distribution is mainly influenced by "hubs", vertices with highest degree. It should be also noted that due to accuracy of experimental data in most real networks power law starts working at some threshold value k_0 (A. Clauset, C. R. Shalizi, and M. Newman, "Power-law distributions in empirical data," SIAM review, 2009.). The `igraph` package allow us to estimate α, k_0 and the p-value of hypothesis that the network is scale-free:

```
fit1 <- fit_power_law(d)
fit1
```

```
## $continuous
## [1] FALSE
##
## $alpha
## [1] 2.464437
##
## $xmin
## [1] 6
##
## $logLik
## [1] -3574.683
##
## $KS.stat
## [1] 0.02843181
##
## $KS.p
## [1] 0.2930298
```

Average path length, diameter and centrality

```
distances(csg,v=topD)->dists
qplot(dists[1,],binwidth=0.5)
```




```
table(dists)
```

```
## dists
##    0    1    2    3    4    5    6    7    8
##    1 2314 2246 6690 5696  712  200   59   26
##    9   10
##   18    1
```

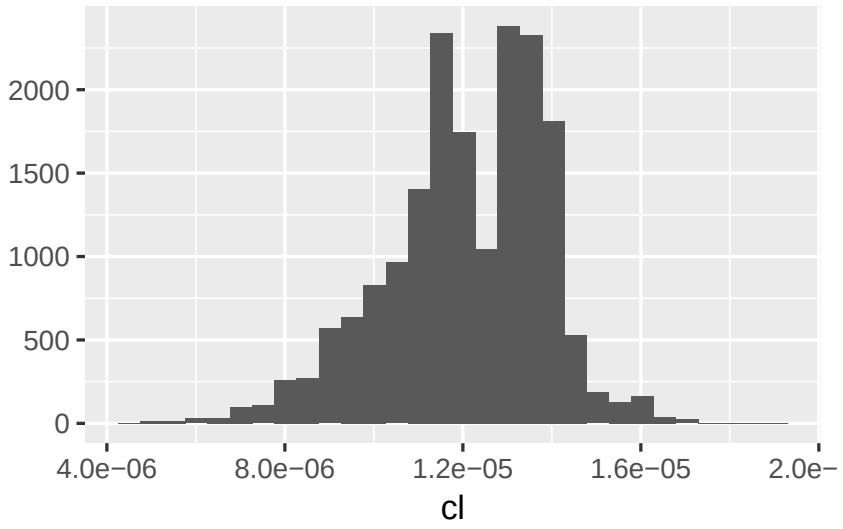
```
summary(dists[1])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##       3       3       3       3       3
##    Max.
##       3
```

it could be seen that most vertices are located in HNF4-alpha steps from the main hub and average shortest path length is also 3, the diameter of the graph – 18 also shows that the main hub located in the center of the graph. Lets calculate centrality measure for the graph:

```
cl<-closeness(csg)
topCl<-which.max(cl)
qplot(cl)
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```




```
summary(cl)
```

```
##      Min.   1st Qu.    Median      Mean
## 4.631e-06 1.106e-05 1.212e-05 1.209e-05
##   3rd Qu.      Max.
## 1.367e-05 1.917e-05
```

The bimodal distribution of the centrality could indicate presence of several communities in the graph.

```
cl[order(cl,decreasing = TRUE)[1:7]]
```

```
##      Unknown          Tat   HNF4-alpha
## 1.916517e-05 1.833853e-05 1.811168e-05
##         E2F4          p53          TBP
## 1.733553e-05 1.721585e-05 1.718449e-05
##         CDK2
## 1.718449e-05
```

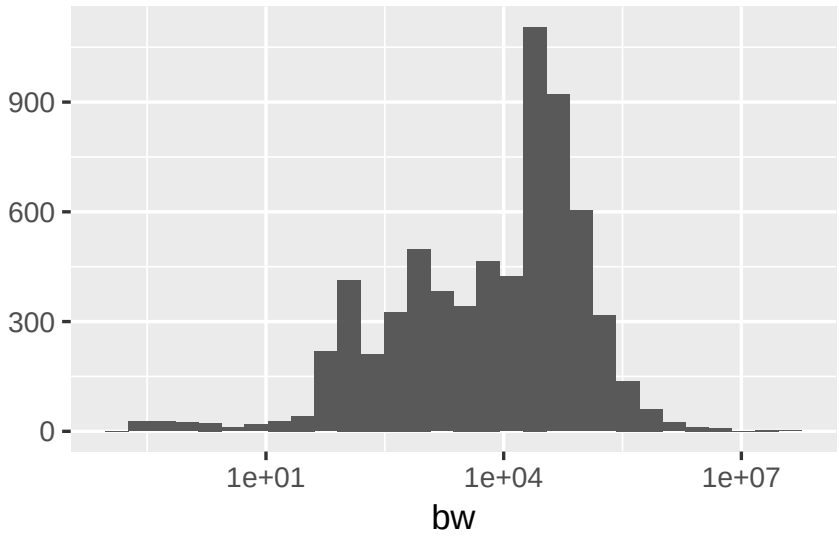
```
d[order(cl,decreasing = TRUE)[1:7]]
```

```
##    Unknown        Tat HNF4-alpha       E2F4
##       1575        709       2314        688
##        p53        TBP       CDK2
##        155         43         81
```

Betweenness

```
bw<-betweenness(csg,directed = FALSE)
qplot(bw,log = 'x')
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```




```
summary(bw)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.
##        0        0        0    33464     1921
##     Max.
## 51644173
```

```
bw[order(bw,decreasing = TRUE)[1:7]]
```

```
##    Unknown HNF4-alpha        Tat       TAF1
##   51644173   50973366   28886614   27695538
##      c-Myc       E2F4        Sp1
##   18949027   14879454    7121987
```

```
d[order(bw,decreasing = TRUE)[1:7]]
```

```
##    Unknown HNF4-alpha        Tat       TAF1
##       1575       2314        709       1763
##      c-Myc       E2F4        Sp1
##       1650        688        383
```

```
cl[order(bw,decreasing = TRUE)[1:7]]
```

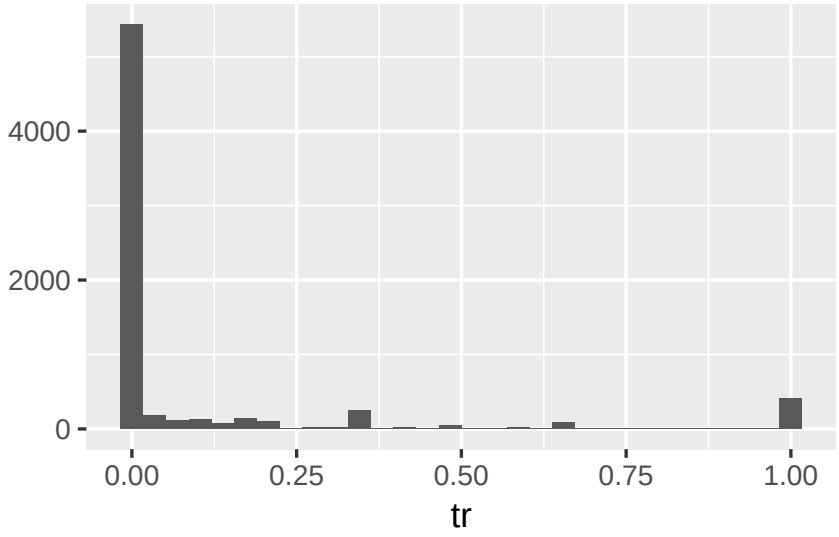
```
##      Unknown   HNF4-alpha          Tat
## 1.916517e-05 1.811168e-05 1.833853e-05
##         TAF1        c-Myc         E2F4
## 1.667473e-05 1.480538e-05 1.733553e-05
##          Sp1
## 1.613320e-05
```

Cluster coefficient

In the `igraph` package cluster coefficient is known by its other name transitivity:

```
tr<-transitivity(csg,type = 'local',vids = V(csg))
qplot(tr)
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```




```
summary(tr)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##   0.000   0.000   0.000   0.101   0.000
##    Max.    NA's
##   1.000   10877
```

The cluster coefficient for the main hub

```
tr[order(d,decreasing = TRUE)[1:7]]
```

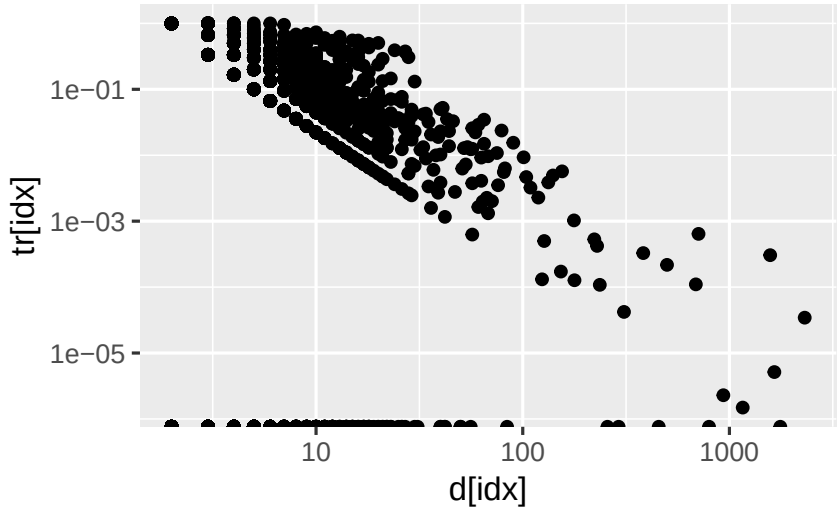
```
## [1] 3.437786e-05 0.000000e+00 5.145451e-06
## [4] 3.049555e-04 1.487608e-06 2.290190e-06
## [7] 0.000000e+00
```

```
tr[order(cl,decreasing = TRUE)[1:7]]
```

```
## [1] 3.049555e-04 6.414700e-04 3.437786e-05
## [4] 1.100166e-04 5.697528e-03 3.543743e-02
## [7] 5.555556e-03
```

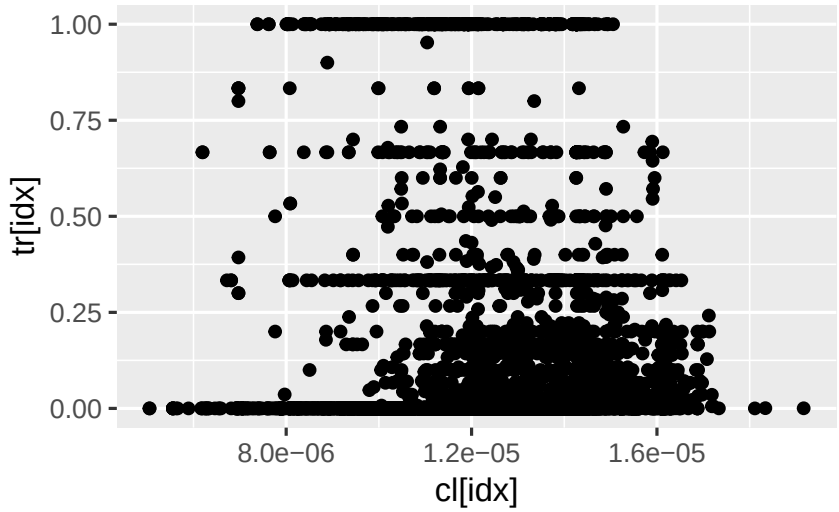
Cluster coefficient vs degree

```
idx<-which(!is.na(tr))
qplot(d[idx],tr[idx],log='xy')
```



Cluster coefficient vs centrality

```
idx<-which(!is.na(tr))
qplot(cl[idx],tr[idx])
```



Assortativity degree

```
assortativity_degree(csg,directed = FALSE)
```

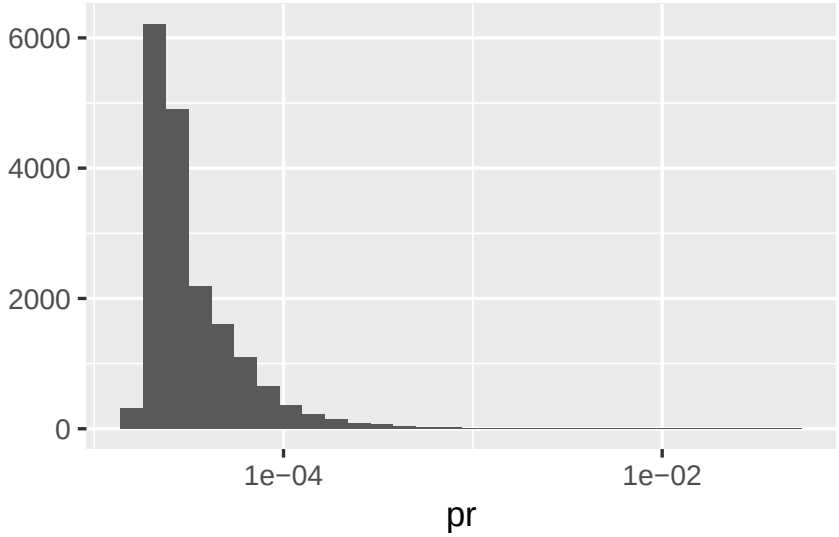
```
## [1] -0.257744
```

negative value shows that hubs are repel each other.

Page rank

```
pr<-page_rank(csg,directed = FALSE)$vector
qplot(pr,log = 'x')
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```




```
summary(pr)
```

```
##      Min.   1st Qu.    Median      Mean
## 1.391e-05 2.282e-05 2.660e-05 5.567e-05
##   3rd Qu.      Max.
## 3.971e-05 4.208e-02
```

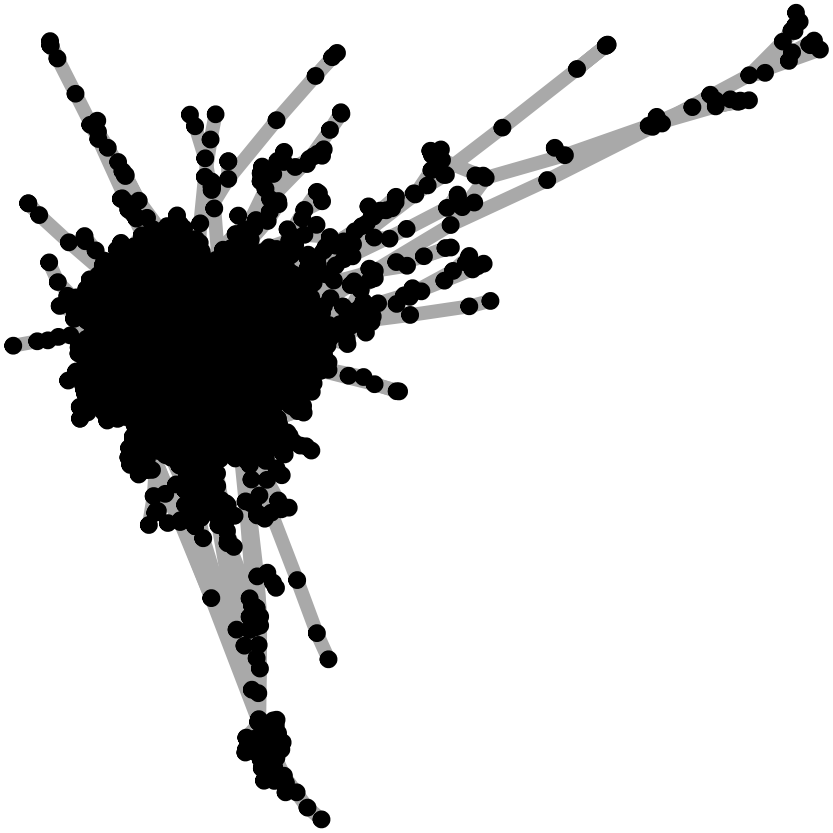
```
d[order(pr,decreasing = TRUE)[1:7]]
```

```
## HNF4-alpha       TAF1    Unknown      c-Myc
##       2314       1763       1575       1650
##   TAFII250        Tat        Max
##       1160        709        935
```

Clusterisation

Let's keep only `pp` edges and extract the main connected component:

```
pprm<-which(E(csg)$type!='pp')
t<-delete_edges(csg,pprm)
ct<-components(t)
ppg<-induced_subgraph(t,groups(ct)[[1]])
plot(ppg,vertex.size=1,vertex.label=NA,layout=layout.fruchterman.reingold)
```



Most vertices are located in the dense core of the graph. Let's try to split graph into modules in such a way that there will be more edges within module that between modules:

$$Q = \frac{1}{2m} \sum_{i,j} \frac{A_{ij} - k_i \cdot k_j}{2m} \delta(c_i, c_j)$$

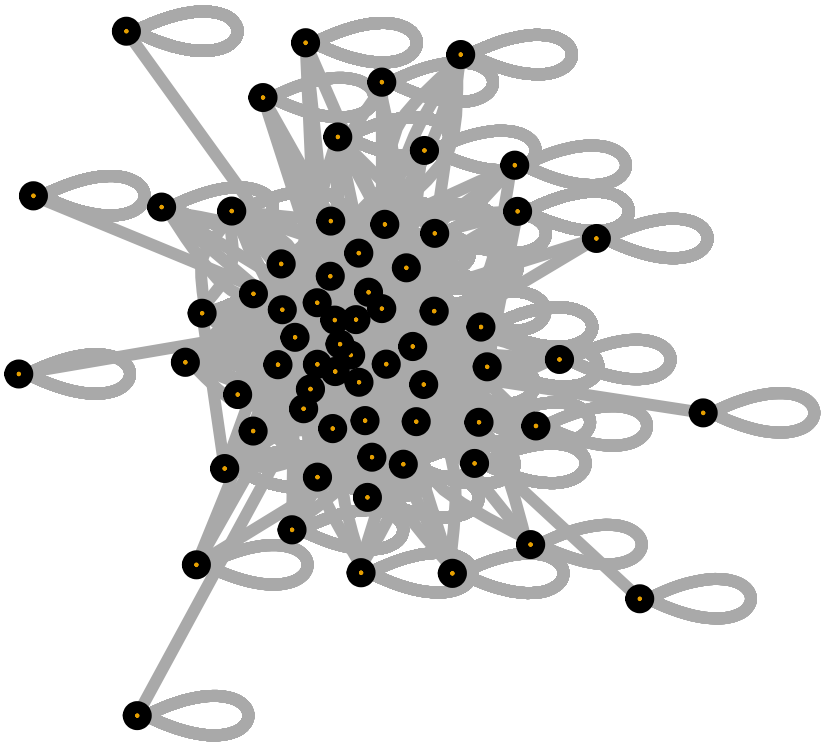
, where m – number of edges, i and j vertex indices, k_i degree of vertex i, A_{ij} element of the connectivity matrix, c_i index of the module.

We will remove edges with highest betweenness untill graph will be split into disconnected components9

```
ebcl<-cluster_edge_betweenness(ppg)
```

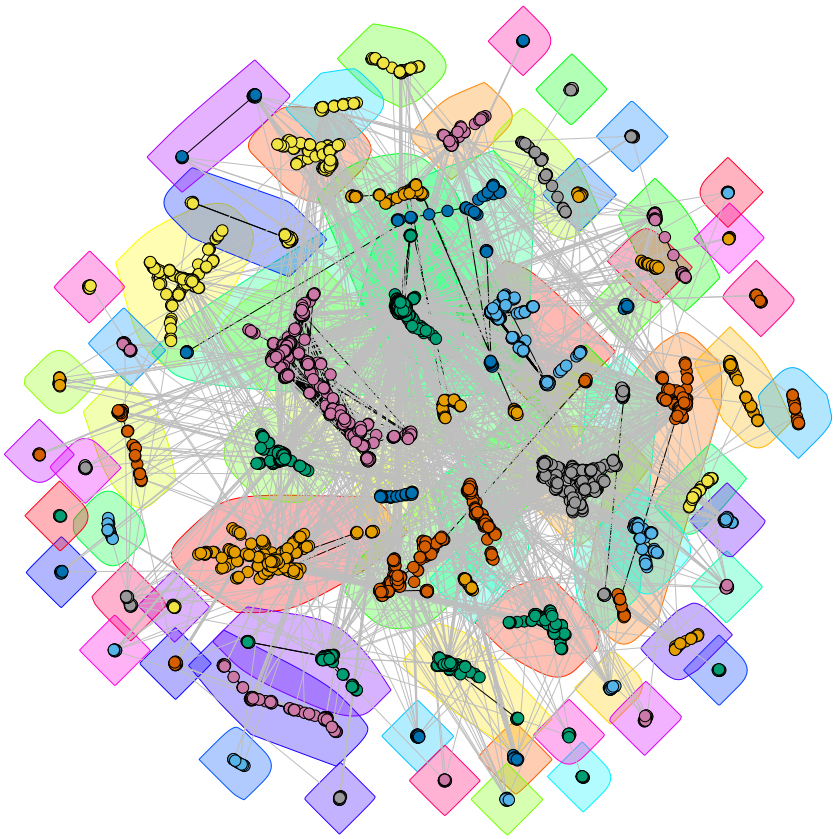
As a result we have got 67 (`length(unique(ebcl$membership))`) modules by deletion of 5992 (`length(ebcl$bridges)`) intermodules edges, the modularity value is – 0.8130693. The structure of the graph in modules:

```
eb.comm.graph <- contract.vertices(ppg, ebcl$membership, vertex.attr.comb=list(size="sum", "ignore"))
plot(eb.comm.graph,vertex.size=5,vertex.label=NA,layout=layout.fruchterman.reingold)
```



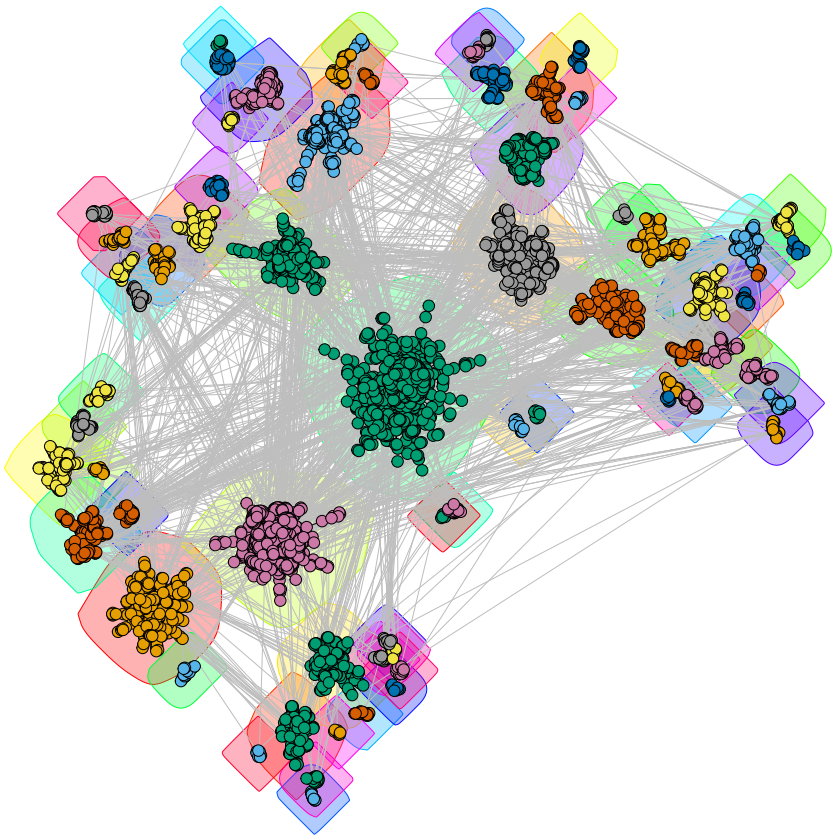
Structure of the graph taking modules into account:

```
cg<-delete_edges(ppg,which(crossing(ebcl,ppg)))
l<-layout_nicely(cg)
plot(ebcl,ppg,
     vertex.size=3,
     vertex.label=NA,
     layout=l,
     edge.color=c("black", "gray")[crossing(ebcl,ppg) + 1])
```

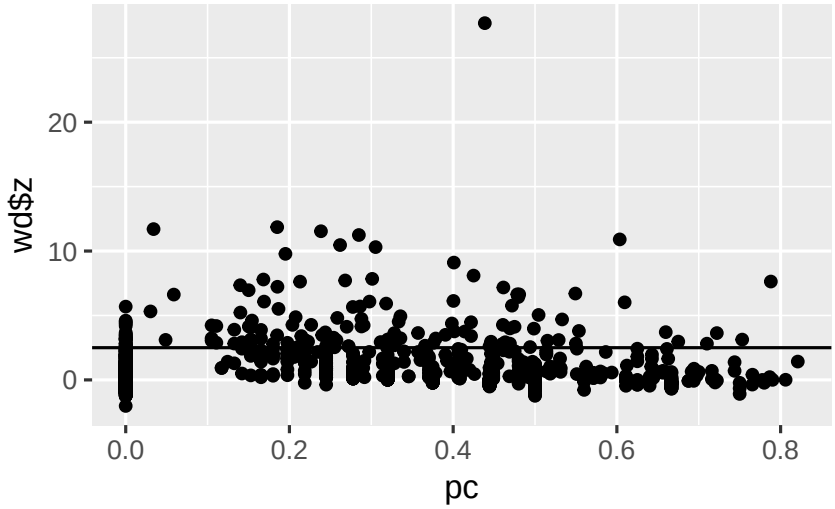


We can make our own layout

```
l<-memb_layout(ppg,ebcl)
plot(ebcl,ppg,
     vertex.size=3,
     vertex.label=NA,
     layout=l,
     edge.color=c("black", "gray")[crossing(ebcl,ppg) + 1])
```

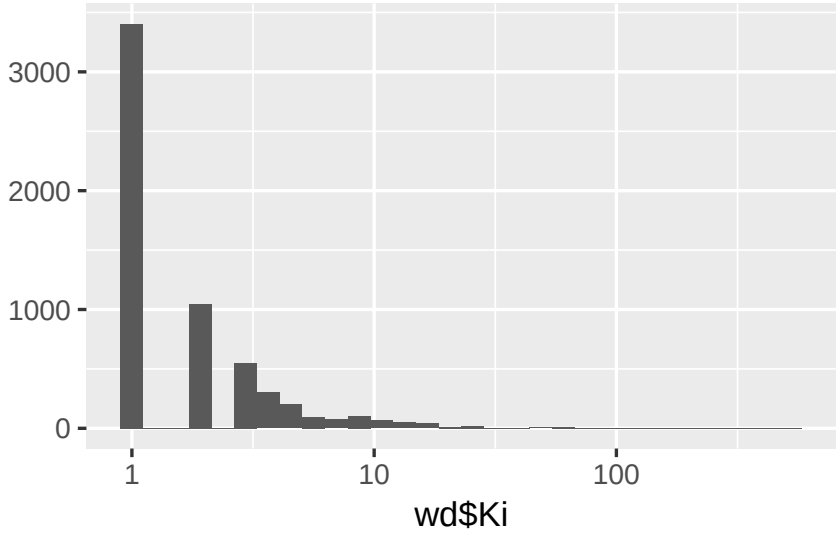


```
pc<-part.coeff(ppg,ebcl$membership)
wd<-within_module_deg_z_score(ppg,ebcl$membership)
qplot(pc,wd$z)+geom_hline(yintercept = 2.5)
```



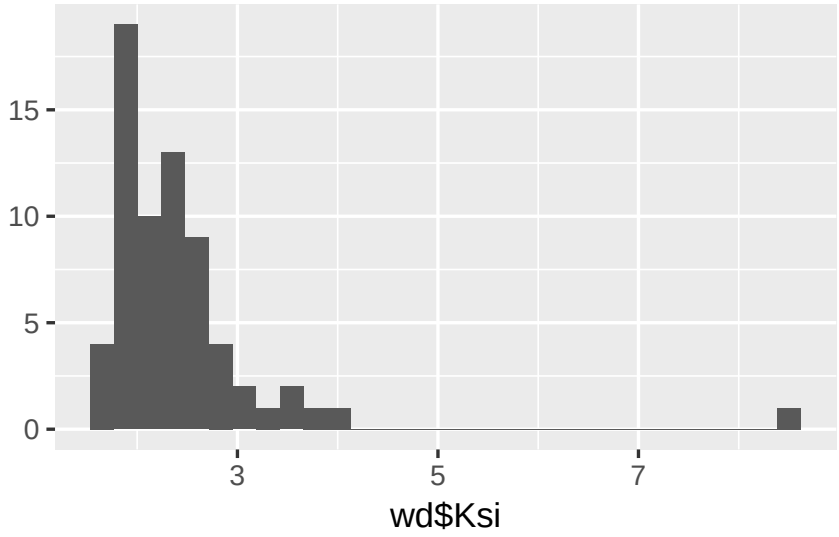

```
qplot(wd$Ki,log='x')
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```




```
qplot(wd$Ksi)
```

```
## 'stat_bin()' using 'bins = 30'. Pick
## better value with 'binwidth'.
```



R and Cytoscape

To work with graphs interactively it is easier to use Cytoscape rathe then `igrap` package. To set up communication between R and Cytoscape we will need to install `RCy3` package:

```
source("https://bioconductor.org/biocLite.R")
biocLite("RCy3")
```

Appendix

Custom Functions

```
# @author Christopher G. Watson, \email{cgwatson@@bu.edu}
# @references Guimera, R. and Amaral, L.A.N. (2005) Cartography of complex
# networks: modules and universal roles, Journal of Statistical Mechanics:
# Theory and Experiment, 02, P02001.

part.coeff <- function(g, memb) {
  i <- NULL
  if ('degree' %in% vertex_attr_names(g)) {
    degs <- V(g)$degree
  } else {
    degs <- degree(g)
  }
  es <- E(g)
  vs <- which(degs > 0)

  PC <- rep(0, length(degs))
  for (i in vs) {
    Kis <- vapply(seq_len(max(memb)), function(x)
      sum(neighbors(g, i) %in% which(memb == x)),
      integer(1))
    Ki <- degs[i]
    PC[i] <- 1 - sum((Kis/Ki)^2)
  }


  return(PC)
}
# @author Christopher G. Watson, \email{cgwatson@@bu.edu}
# @references Guimera, R. and Amaral, L.A.N. (2005) Cartography of complex
# networks: modules and universal roles, Journal of Statistical Mechanics:
# Theory and Experiment, 02, P02001.

within_module_deg_z_score <- function(g, memb) {
  i <- NULL
  stopifnot(is_igraph(g))
  if ('degree' %in% vertex_attr_names(g)) {
    degs <- V(g)$degree
  } else {
    degs <- degree(g)
  }
```

```
  vs <- which(degs > 0)
  es <- E(g)
  z <- Ki <- rep(0, length(degs))

  for (i in vs) {
    Ki[i] <- length(es[i %--% which(memb == memb[i])])
  }


  di <- lapply(seq_len(max(memb)), function(x) Ki[memb == x])
  Ksi <- vapply(di, mean, numeric(1))
  sigKsi <- vapply(di, sd, numeric(1))

  z[vs] <- (Ki[vs] - Ksi[memb[vs]]) / sigKsi[memb[vs]]
  z <- ifelse(!is.finite(z), 0, z)
  return(list(z=z,Ki=Ki,Ksi=Ksi,sigKsi=sigKsi))
}
memb_layout<-function(graph,comm){
  graphs<-list()
  layouts<-list()
  for(i in unique(membership(comm))){
    gi<-induced_subgraph(graph,which(membership(comm)==i))
    li<-layout.fruchterman.reingold(gi)
    graphs[[i]]<-gi
    layouts[[i]]<-li

  }
  g <- disjoint_union(graphs)
  lay <- merge_coords(graphs, layouts)
  l<-lay[match(V(graph)$name,V(g)$name),]
  return(l)
}
```

Document version

Tue Jul 31 18:15:21 2018

Session Info

Platform

	name	value
- version	R version 3.5.1 (2018-07-02)	
- system	x86_64, darwin15.6.0	
- ui	X11	

name	value
- language	(EN)
- collate	en_US.UTF-8
- tz	Asia/Tokyo
- date	2018-07-31

Packages

package	*	version	date	source
assertthat		0.2.0	2017-04-11	CRAN (R 3.5.0)
backports		1.1.2	2017-12-13	CRAN (R 3.5.0)
base	*	3.5.1	2018-07-05	local
bindr		0.1.1	2018-03-13	CRAN (R 3.5.0)
bindrcpp		0.2.2	2018-03-29	CRAN (R 3.5.0)
Biobase		2.40.0	2018-05-01	Bioconductor
BiocGenerics		0.26.0	2018-05-01	Bioconductor
biomformat	*	1.8.0	2018-05-01	Bioconductor
Biostrings		2.48.0	2018-05-01	Bioconductor
bitops		1.0-6	2013-08-17	CRAN (R 3.5.0)
codetools		0.2-15	2016-10-05	CRAN (R 3.5.1)
colorspace		1.3-2	2016-12-14	CRAN (R 3.5.0)
compiler		3.5.1	2018-07-05	local
crayon		1.3.4	2017-09-16	CRAN (R 3.5.0)
data.table		1.11.4	2018-05-27	CRAN (R 3.5.0)
datasets	*	3.5.1	2018-07-05	local
devtools		1.13.6	2018-06-27	CRAN (R 3.5.0)
digest		0.6.15	2018-01-28	CRAN (R 3.5.0)
dplyr		0.7.6	2018-06-29	CRAN (R 3.5.1)
evaluate		0.11	2018-07-17	CRAN (R 3.5.0)
flexmix		2.3-14	2017-04-28	CRAN (R 3.5.0)
formatR		1.5	2017-04-25	CRAN (R 3.5.0)
ggplot2		3.0.0	2018-07-03	CRAN (R 3.5.0)
glue		1.3.0	2018-07-17	CRAN (R 3.5.0)
graphics	*	3.5.1	2018-07-05	local
grDevices	*	3.5.1	2018-07-05	local
grid		3.5.1	2018-07-05	local
gtable		0.2.0	2016-02-26	CRAN (R 3.5.0)
htmltools		0.3.6	2017-04-28	CRAN (R 3.5.0)
httr		1.3.1	2017-08-20	CRAN (R 3.5.0)
igraph	*	1.2.1	2018-03-10	CRAN (R 3.5.0)
IRanges		2.14.10	2018-05-16	Bioconductor
jsonlite		1.5	2017-06-01	CRAN (R 3.5.0)
KEGGREST		1.20.1	2018-06-27	Bioconductor

package	*	version	date	source
knitr	*	1.20	2018-02-20	CRAN (R 3.5.0)
lattice		0.20-35	2017-03-25	CRAN (R 3.5.1)
lazyeval		0.2.1	2017-10-29	CRAN (R 3.5.0)
magrittr		1.5	2014-11-22	CRAN (R 3.5.0)
Matrix		1.2-14	2018-04-13	CRAN (R 3.5.1)
memoise		1.1.0	2017-04-21	CRAN (R 3.5.0)
methods	*	3.5.1	2018-07-05	local
mmnet	*	1.15.0-2	2018-07-31	Bioconductor
modeltools		0.2-22	2018-07-16	CRAN (R 3.5.0)
munsell		0.5.0	2018-06-12	CRAN (R 3.5.0)
nnet		7.3-12	2016-02-02	CRAN (R 3.5.1)
pander	*	0.6.2	2018-07-13	Github (`Rapporter/pander@843907d`)
parallel		3.5.1	2018-07-05	local
pillar		1.3.0	2018-07-14	CRAN (R 3.5.0)
pkgconfig		2.0.1	2017-03-21	CRAN (R 3.5.0)
plyr		1.8.4	2016-06-08	CRAN (R 3.5.0)
png		0.1-7	2013-12-03	CRAN (R 3.5.0)
purrr		0.2.5	2018-05-29	CRAN (R 3.5.0)
R6		2.2.2	2017-06-17	CRAN (R 3.5.0)
Rcpp		0.12.18	2018-07-23	CRAN (R 3.5.0)
RCurl		1.95-4.11	2018-07-15	CRAN (R 3.5.0)
rhdf5		2.24.0	2018-05-01	Bioconductor
Rhdf5lib		1.2.1	2018-05-17	Bioconductor
RJSONIO	*	1.3-0	2014-07-28	CRAN (R 3.5.0)
rlang		0.2.1	2018-05-30	CRAN (R 3.5.0)
rmarkdown		1.10	2018-06-11	CRAN (R 3.5.0)
rprojroot		1.3-2	2018-01-03	CRAN (R 3.5.0)
rstudioapi		0.7	2017-09-07	CRAN (R 3.5.0)
S4Vectors		0.18.3	2018-06-08	Bioconductor
scales		0.5.0	2017-08-24	CRAN (R 3.5.0)
stats	*	3.5.1	2018-07-05	local
stats4		3.5.1	2018-07-05	local
stringi		1.2.4	2018-07-20	CRAN (R 3.5.0)
stringr		1.3.1	2018-05-10	CRAN (R 3.5.0)
tibble		1.4.2	2018-01-22	CRAN (R 3.5.0)
tidyselect		0.2.4	2018-02-26	CRAN (R 3.5.0)
tools		3.5.1	2018-07-05	local
tufte	*	0.4	2018-07-15	CRAN (R 3.5.0)
utils	*	3.5.1	2018-07-05	local
withr		2.1.2	2018-03-15	CRAN (R 3.5.0)
XML		3.98-1.12	2018-07-15	CRAN (R 3.5.0)
XVector		0.20.0	2018-05-01	Bioconductor

package	*	version	date	source
yaml		2.2.0	2018-07-25	CRAN (R 3.5.0)
zlibbioc		1.26.0	2018-05-01	Bioconductor