The **qgraph** package for network visualizations of psychometric data in R

Sacha Epskamp, Angelique O. J. Cramer, Lourens J. Waldorp, Verena D. Schmittmann and Denny Borsboom

University of Amsterdam
Department of Psychological Methods

Psychoco 2011

All codes in these slides were run using R version 2.12.1 (2010-12-16) and **qgraph** version 0.4.8 and were made on Windows 7 x64 x86-64 build 7600.

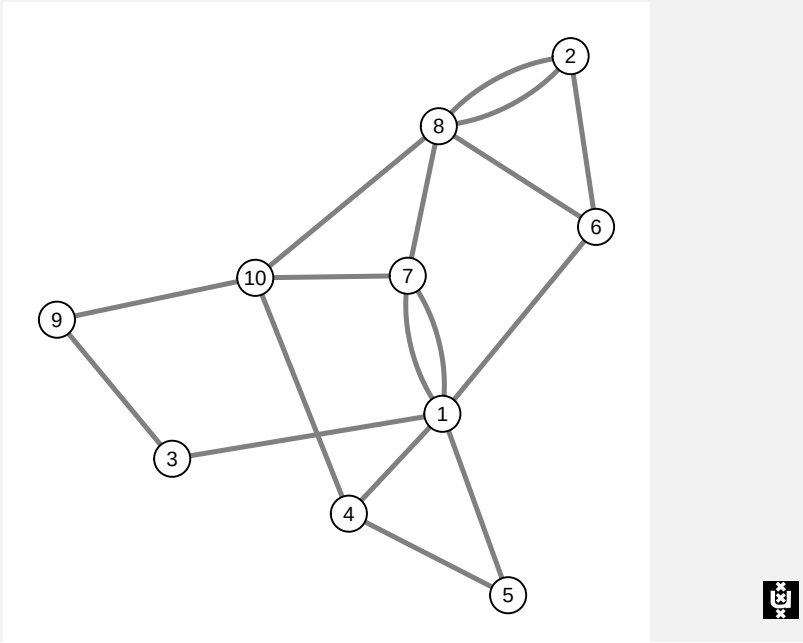
qgraph

- A R package (CRAN link)
- Can be used to plot various types of graphs
- Different from other R packages (e.g. **igraph** Csardi & Nepusz, 2006) in:
  - Focus on *weighted graphs*
  - Intended for visualization of data as graphs
  - Optimized for vector-type image files (e.g. PDF, SVG)
- Aims in **qgraph**
  - Simple input
  - Summarize a large amount of statistics without needing data reduction methods.
  - Visualize *relations between variables*
- Main idea: Show variables as nodes, relationships as edges

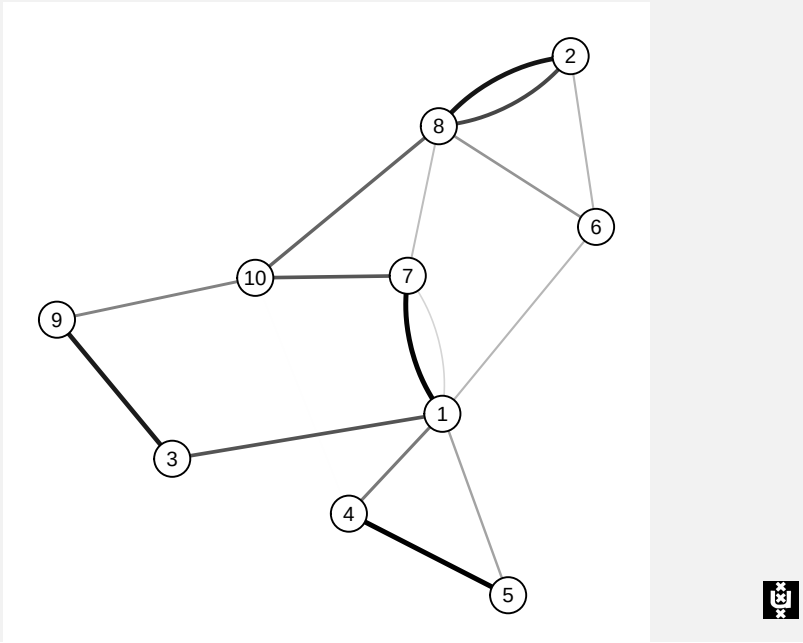
Graphs

- A graph is a *network* that consists of n nodes (or vertices) that are connected with m edges.
- Each edge has a *weight* indicating the strength of that connection
- An edge can be directed (have an arrow) or undirected

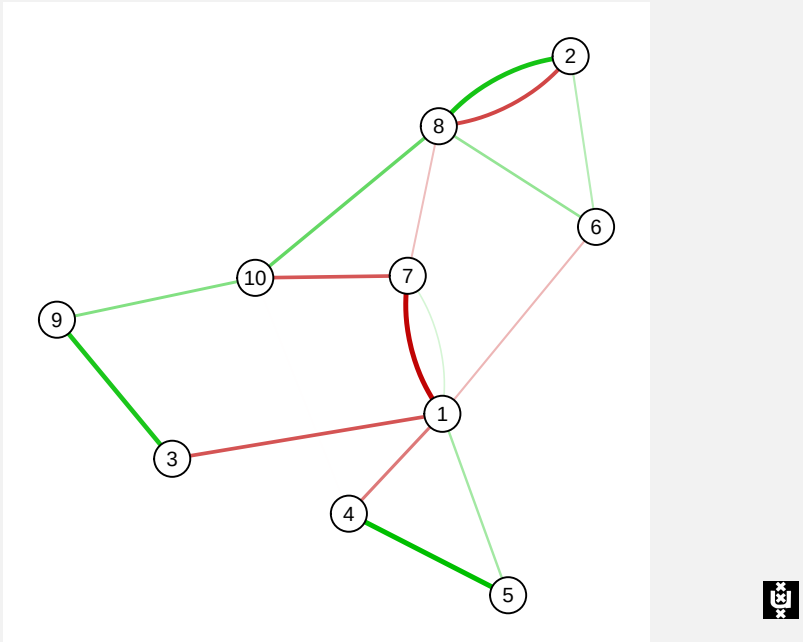
Unweighted graph



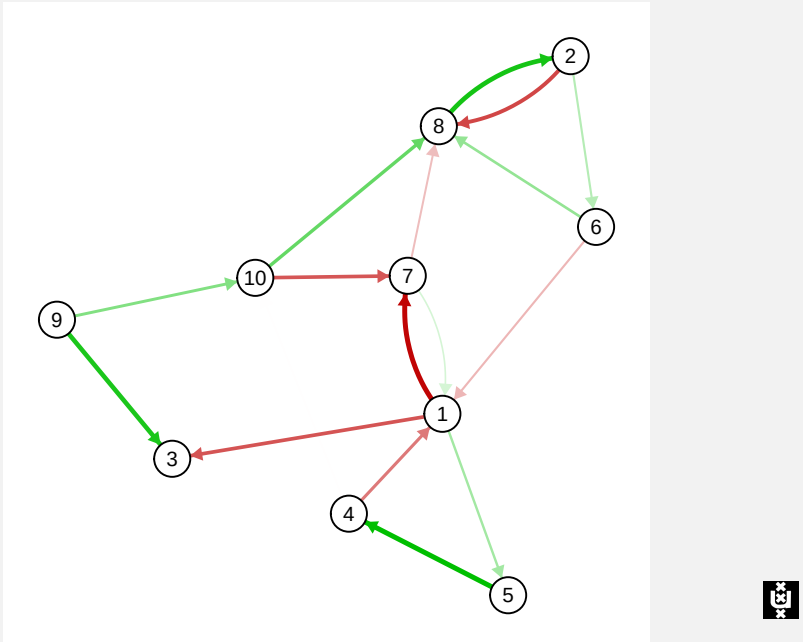
Weighted graph



Weighted graph



Directed graph



The qgraph() function

- The main function in **qgraph** is `qgraph()`
  - Most other functions are either wrapping functions using `qgraph()` or functions used in `qgraph()`
- The `qgraph()` function requires only one argument (`adj`)
- A lot of other arguments can be specified, but these are all optional

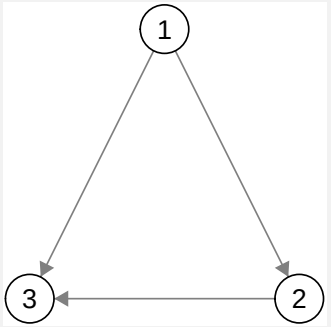
Usage:

```
qgraph( adj, ... )
```

The adjacency matrix

- The `adj` argument is the input. This can be an *adjacency matrix*
- An adjacency matrix is a square *n* by *n* matrix in which each element indicates the relationship between two variables
- Any relationship can be used as long as:
  - A 0 indicates no relationship
  - Absolute negative values are similar in strength to positive values
- Examples:
  - A 1 indicating a connection (unweighted graphs)
  - Correlations
  - Regression parameters
  - Factor loadings
- *Adjacency matrices occur naturally in statistics!*

```
     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    0    0    1
[3,]    0    0    0
```



Weighted graphs

$$\mathbf{Y} = \boldsymbol{\eta}\boldsymbol{\Lambda}^{\mathrm{T}} + \boldsymbol{\Theta}$$

```
> set.seed(2)
> eta <- matrix(rnorm(200 * 5), ncol = 5)
> lam <- matrix(rnorm(50 * 5, 0, 0.15), 50,
+     5)
> lam[apply(diag(5) == 1, 1, rep, each = 10)] <- rnorm(50,
+     0.7, 0.3)
> th <- matrix(rnorm(200 * 50), ncol = 50)
> Y <- eta %*% t(lam) + th
```

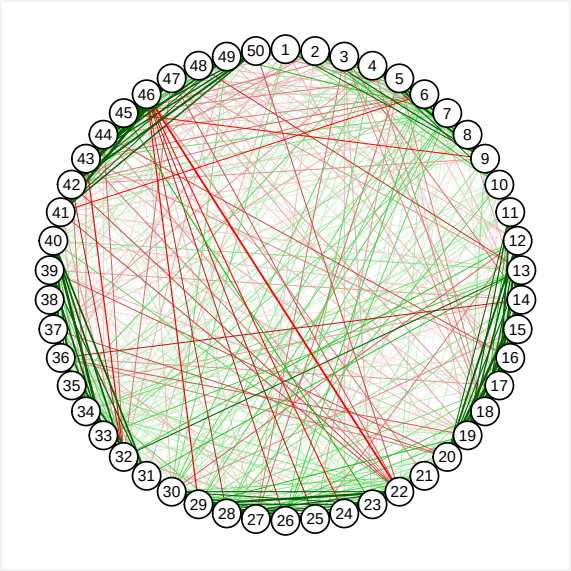
Weighted graphs

```
> cor(Y)[1:15, 1:3]
```

```
              [,1]         [,2]         [,3]
 [1,]  1.000000000  0.218987651  0.197325805
 [2,]  0.218987651  1.000000000  0.231696634
 [3,]  0.197325805  0.231696634  1.000000000
 [4,]  0.464897112  0.346780772  0.279845144
 [5,]  0.295912130  0.275030523  0.209220603
 [6,]  0.235201044  0.272947122  0.197676521
 [7,]  0.157314986 -0.001815960 -0.027551034
 [8,]  0.234392422  0.212721700  0.192401237
 [9,]  0.321680277  0.350685995  0.210808452
[10,]  0.204097076  0.277127339  0.148343574
[11,] -0.072734280 -0.000913891 -0.085440215
[12,]  0.052842181  0.105870583 -0.056247479
[13,]  0.001850306  0.025604291  0.081077133
[14,] -0.083391834 -0.088641129 -0.215127641
[15,] -0.055847218  0.007651554  0.004209916
```

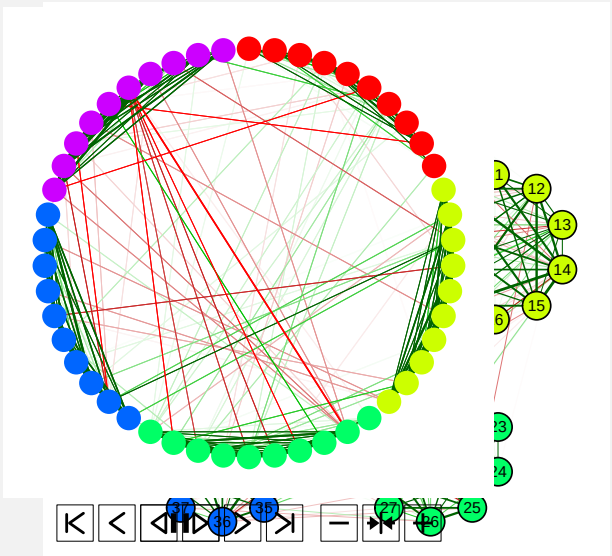
Weighted graphs

```
> qgraph(cor(Y))
```



Weighted graphs

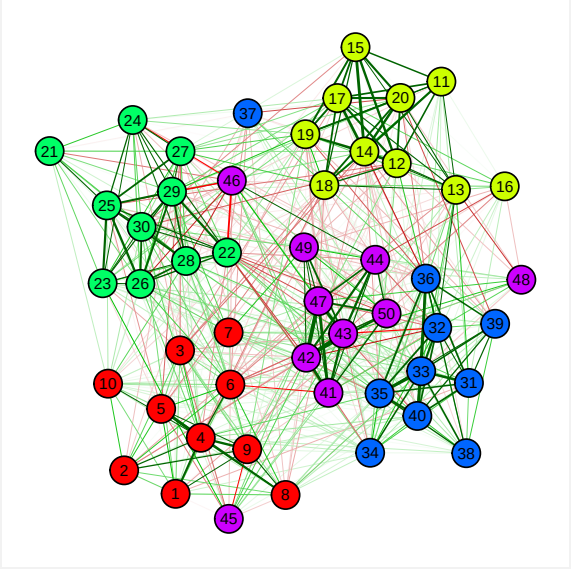
```
> gr <- list(1:10, 11:20, 21:30, 31:40, 41:50)
> qgraph(cor(Y), groups = gr)
```



Fruchterman-Reingold layout (20 iterations)

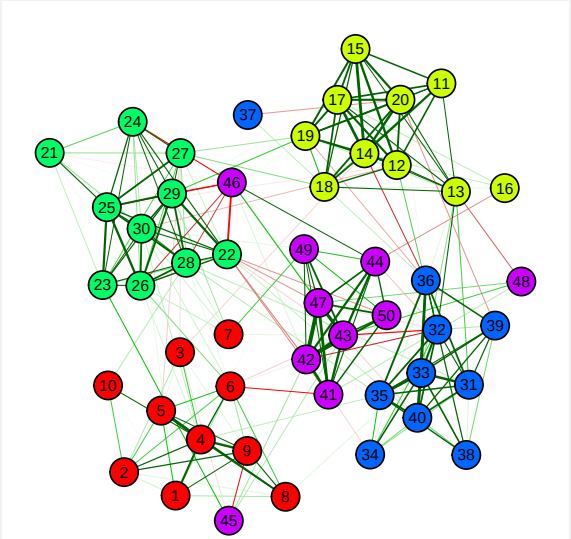
Fruchterman-Reingold layout (500 iterations)

```
> qgraph(cor(Y), groups = gr, layout = "spring")
```



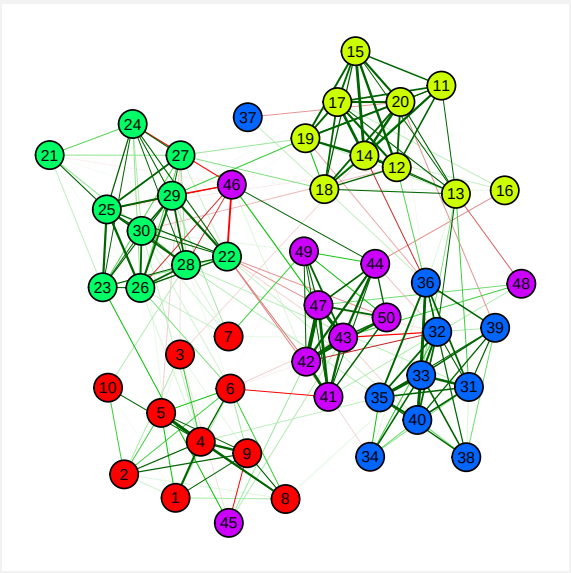
Saving arguments

```
> Q <- qgraph(cor(Y), groups = gr, layout = "spring",
+     minimum = 0.2)
```



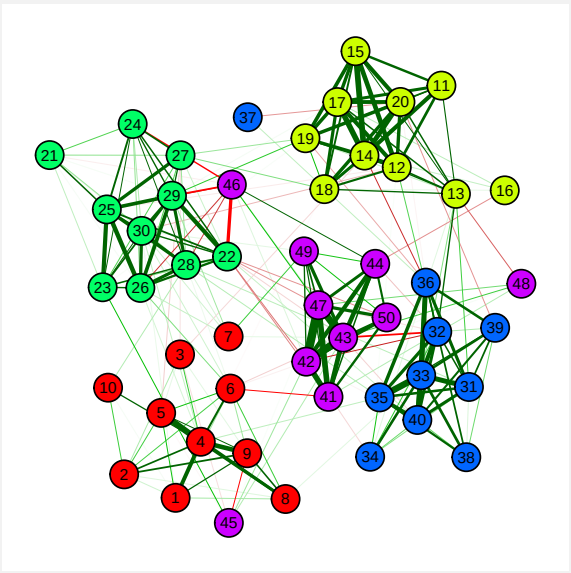
Saving arguments

```
> Q <- qgraph(Q)
```



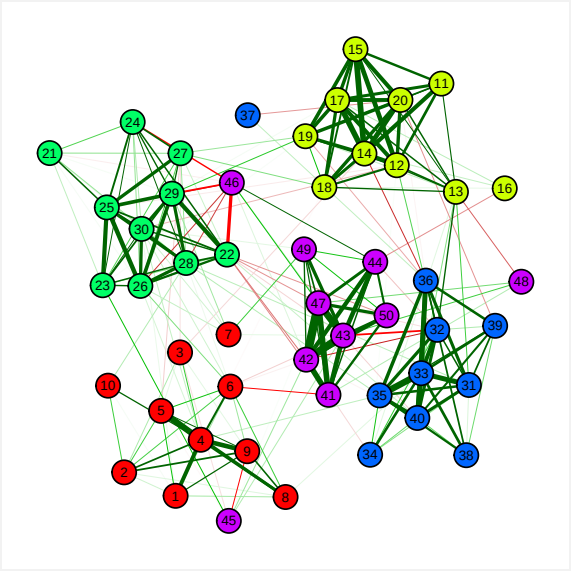
Graphical arguments

```
> Q <- qgraph(Q, esize = 10)
```



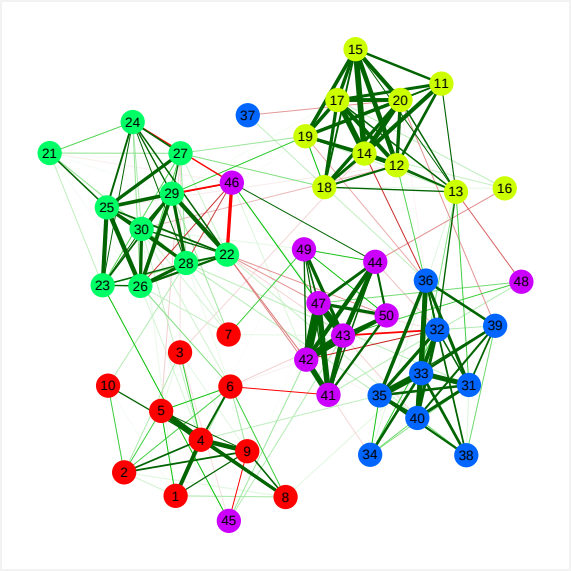
Graphical arguments

```
> Q <- qgraph(Q, vsize = 4)
```



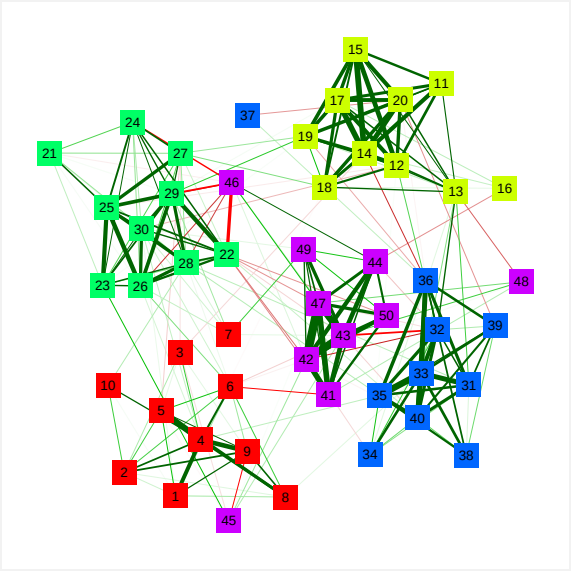
Graphical arguments

```
> Q <- qgraph(Q, borders = FALSE)
```



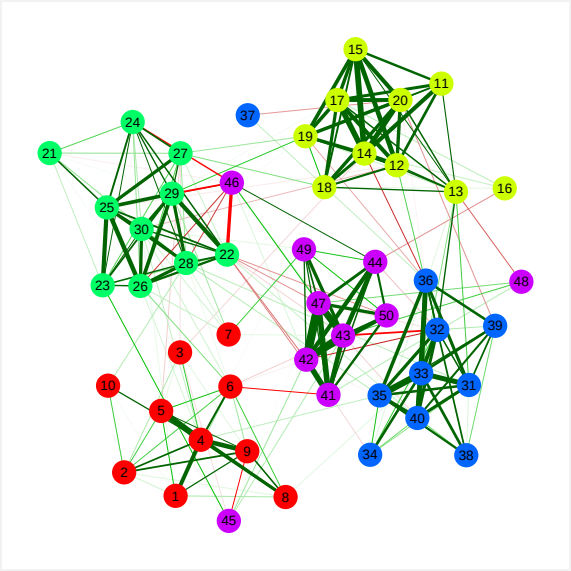
Graphical arguments

```
> Q <- qgraph(Q, shape = "square")
```



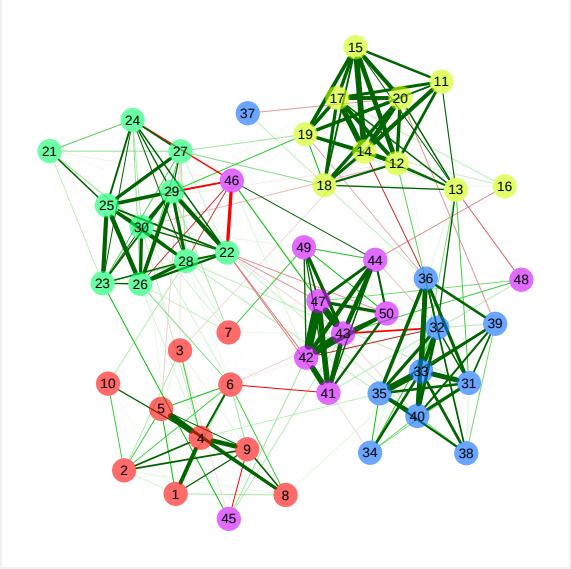
Graphical arguments

```
> Q <- qgraph(Q, shape = "circle")
```



Graphical arguments

```
> Q <- qgraph(Q, vTrans = 150)
```

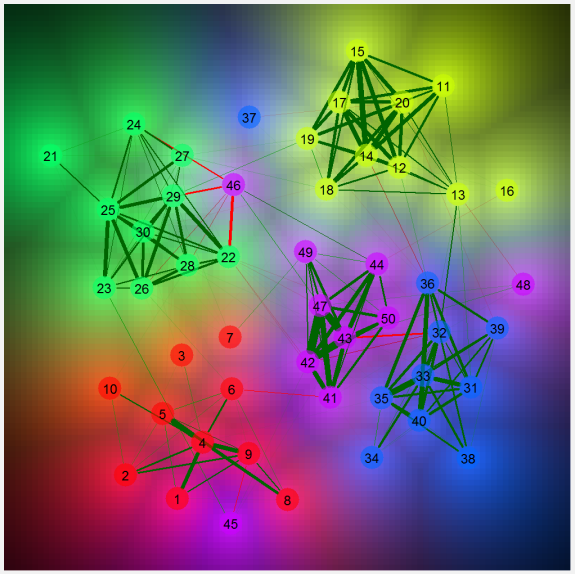


Graphical arguments

```
> qgraph(Q, transparency = T, bg = T, bgcontrol = 5,
+     filetype = "png", filename = "bg", res = 144,
+     width = 7, height = 7)
```

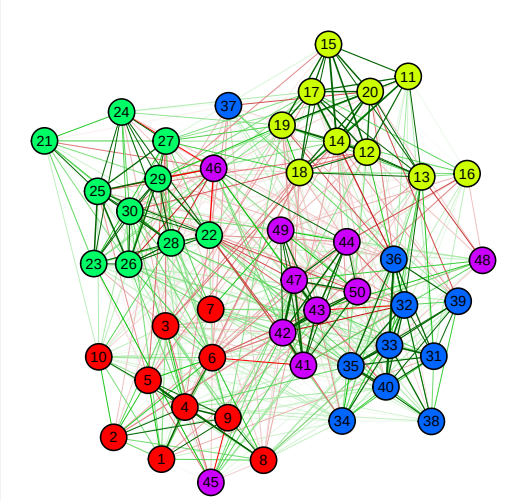
```
[1] "Output stored in C:/Users/Sacha/Documents/Work/qgraph/Psych
```

Graphical arguments



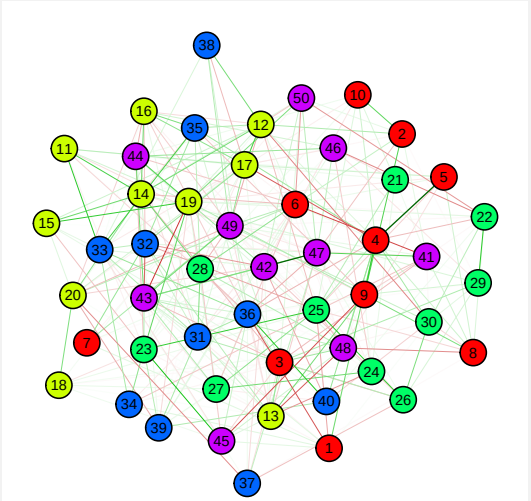
Correlations

```
> qgraph(cor(Y), layout = "spring", groups = gr,
+     cut = 0.3, minimum = 0.1, maximum = 1,
+     graph = "association")
```



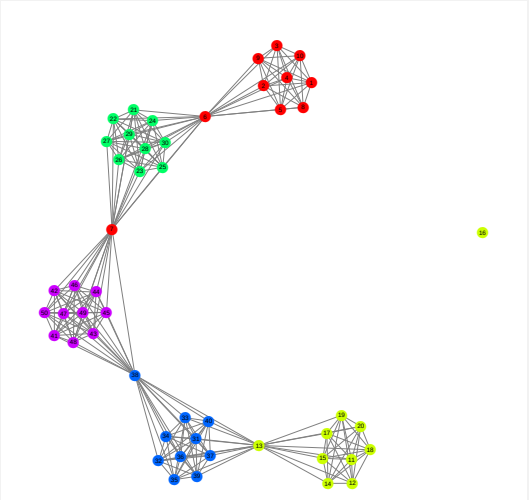
Partial correlations

```
> qgraph(cor(Y), layout = "spring", groups = gr,
+      cut = 0.3, minimum = 0.1, maximum = 1,
+      graph = "concentration")
```



Factorial graph

```
> qgraph(cor(Y), layout = "spring", groups = gr,
+      cut = 0.2, vsize = 2, esize = 1, borders = F,
+      graph = "factorial")
```

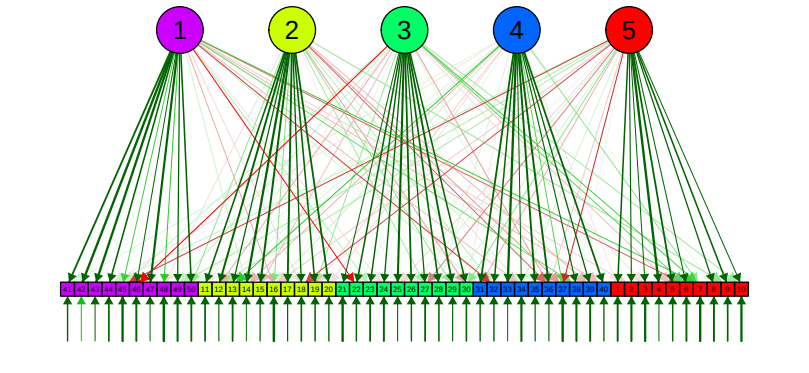


Factor loadings

- A factor loadings matrix can be visualized using `qgraph.loadings()`
- There are two wrapper functions that perform an analysis and send the results to `qgraph.loadings()`:
  - `qgraph.efa()` performs an exploratory factor analysis (EFA) using `stats:::factanal`
  - `qgraph.pca()` performs a principal component analysis (PCA) using `psych:::principal` (Revelle, 2010)
- These functions use a correlation or covariance matrix as input

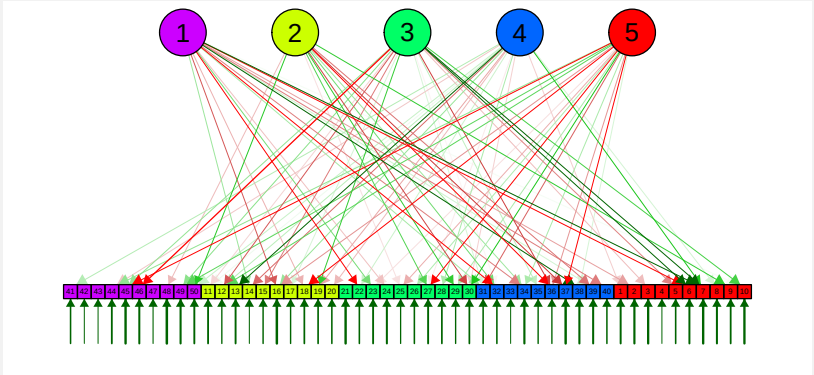
Factor loadings: EFA

```
> qgraph.efa(cor(Y), 5, rotation = "promax",
+      layout = "tree", vsize = c(3, 10), groups = gr)
```



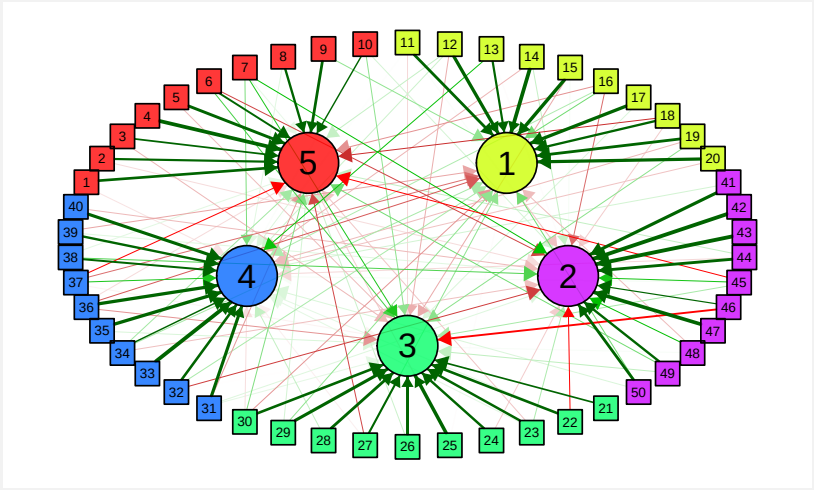
Factor loadings: EFA crossloadings

```
> qgraph.efa(cor(Y), 5, rotation = "promax",
+     layout = "tree", crossloadings = TRUE,
+     vsize = c(3, 10), groups = gr, cut = 0.2)
```



Factor loadings: PCA

```
> qgraph.pca(cor(Y), 5, rotation = "promax",
+     vsize = c(4, 10), groups = gr, vTrans = 200)
```

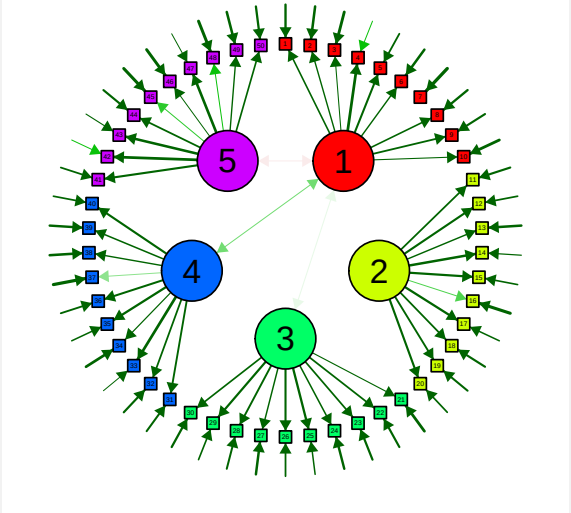


Confirmatory Factor Analysis

- `qgraph.cfa()` can be used to fit a simple confirmatory factor model
  - Each variable loads on only one factor
  - Factors are correlated
  - Scaling by fixing first loading of each factor to 1
- This is done with the **sem** (Fox, 2010) package
- Returns a `"sem"` object
- Results can be send to `qgraph.sem()` for a full report

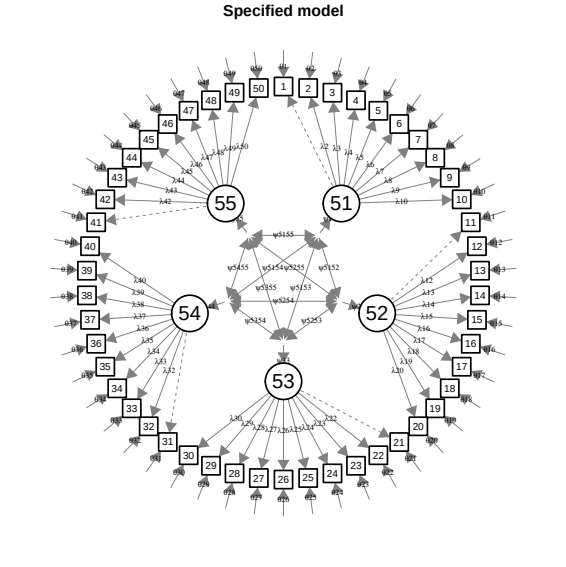
Confirmatory Factor Analysis

```
> res <- qgraph.cfa(cov(Y), N = 200, groups = gr,
+     vsize = c(2, 10))
```



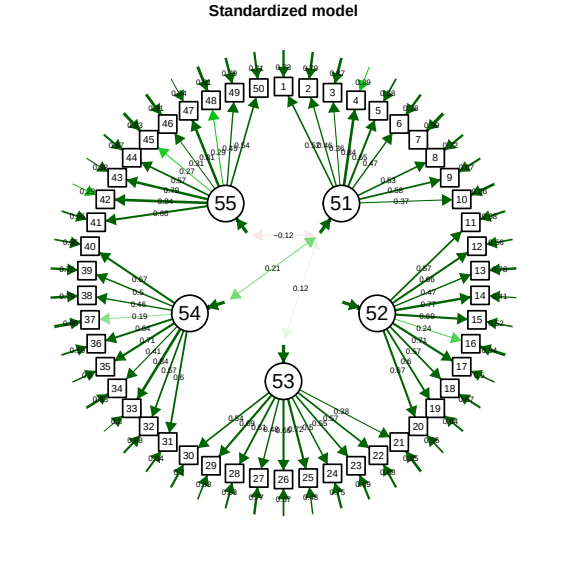
Confirmatory Factor Analysis

```
> qgraph.semModel(res, edge.label.cex = 0.6)
```



Confirmatory Factor Analysis

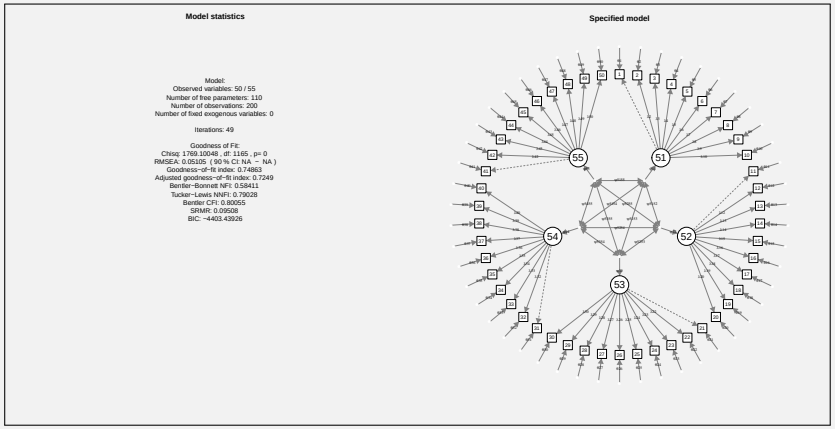
```
> qgraph(res, edge.label.cex = 0.6)
```



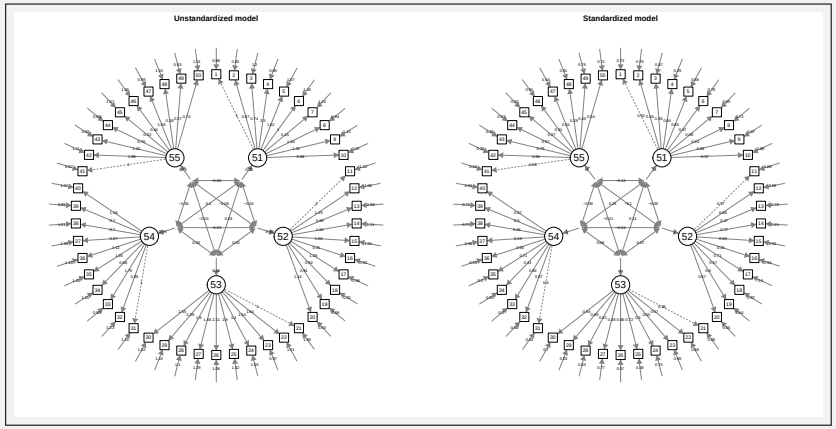
Confirmatory Factor Analysis

```
> qgraph.sem(res, filename = "sem%03d", onefile = F,
+     panels = 2, legend = FALSE, groups = gr,
+     edge.label.cex = 0.6)
```

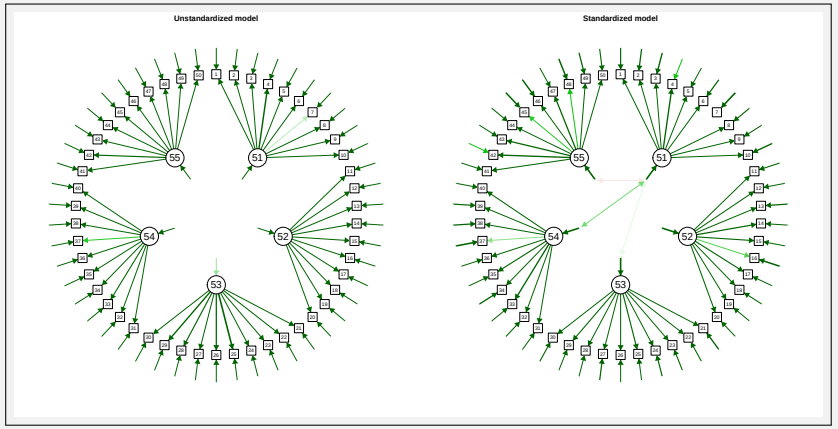
Confirmatory Factor Analysis



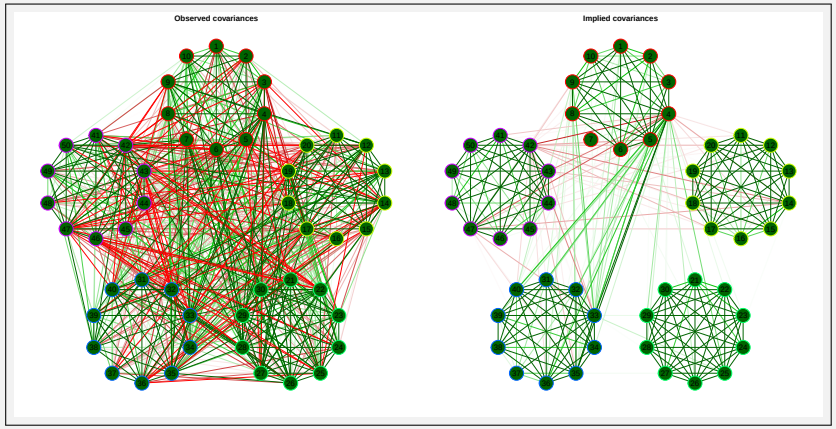
Confirmatory Factor Analysis



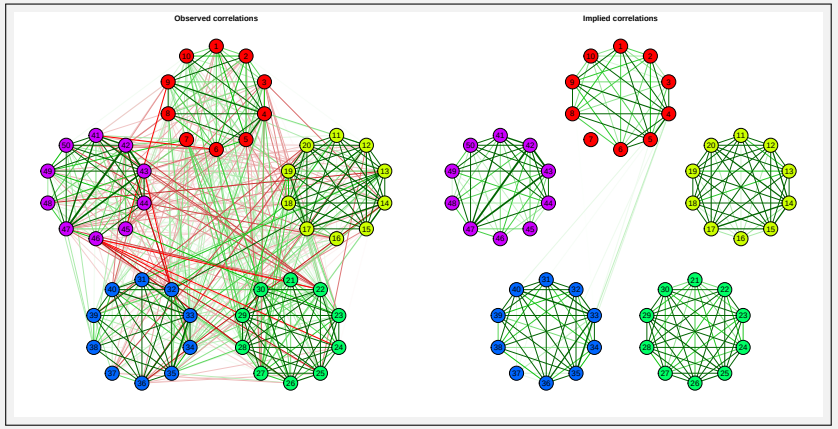
Confirmatory Factor Analysis



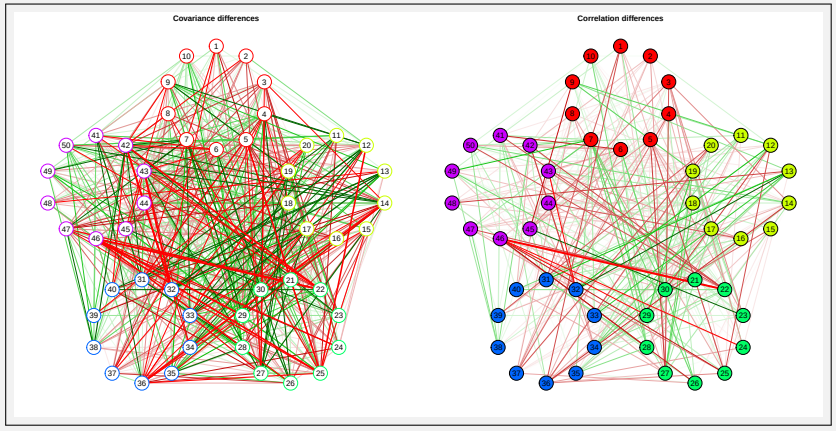
Confirmatory Factor Analysis



Confirmatory Factor Analysis



Confirmatory Factor Analysis



Structural Equation Modelling

- **qgraph** comes with a function that extends output from **sem** (Fox, 2010) with path diagrams and graphs visualizing the parameter estimates
- This is done with the `qgraph.sem()` function
- The output of `qgraph.sem()` is a multi-page pdf file
- We can use **sem** as usual and pass the output to `qgraph.sem()` with only two things to note:
  - It is best to limit variable names in the model to three characters
  - **qgraph** supports Greek letters, by adding an asterisk a label is printed in the symbol font

Structural Equation Modelling

```
> library('sem')
> R.thur <- read.moments(diag=FALSE, names=c('Sen','Voc',
+                   'SC','FL','4LW','Suf','LS','Ped','LG'))
1:     .828
2:     .776   .779
4:     .439   .493    .46
7:     .432   .464    .425   .674
11:    .447   .489    .443   .59    .541
16:    .447   .432    .401   .381   .402   .288
22:    .541   .537    .534   .35    .367   .32    .555
29:    .38    .358    .359   .424   .446   .325   .598   .452
37:
Read 36 items
```

Structural Equation Modelling

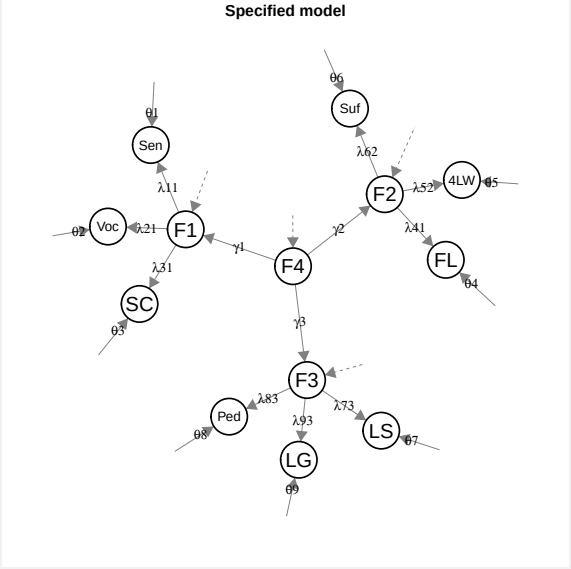
```
> model.thur <- specify.model()
1:     F1 -> Sen,    *l11, NA
2:     F1 -> Voc,    *l21, NA
3:     F1 -> SC,     *l31, NA
4:     F2 -> FL,     *l41, NA
5:     F2 -> 4LW,    *l52, NA
6:     F2 -> Suf,    *l62, NA
7:     F3 -> LS,     *l73, NA
8:     F3 -> Ped,    *l83, NA
9:     F3 -> LG,     *l93, NA
10:    F4 -> F1,     *g1,  NA
11:    F4 -> F2,     *g2,  NA
12:    F4 -> F3,     *g3,  NA
13:    Sen <-> Sen,  q*1,  NA
14:    Voc<-> Voc,   q*2,  NA
15:    SC <-> SC,    q*3,  NA
```

Structural Equation Modelling

```
16:    FL <-> FL,   q*4,  NA
17:    4LW <-> 4LW, q*5,  NA
18:    Suf<-> Suf,  q*6,  NA
19:    LS <-> LS,   q*7,  NA
20:    Ped<-> Ped,  q*8,  NA
21:    LG <-> LG,   q*9,  NA
22:    F1 <-> F1,   NA,    1
23:    F2 <-> F2,   NA,    1
24:    F3 <-> F3,   NA,    1
25:    F4 <-> F4,   NA,    1
26:
Read 25 records
> sem.thur <- sem(model.thur, R.thur, 213)
```

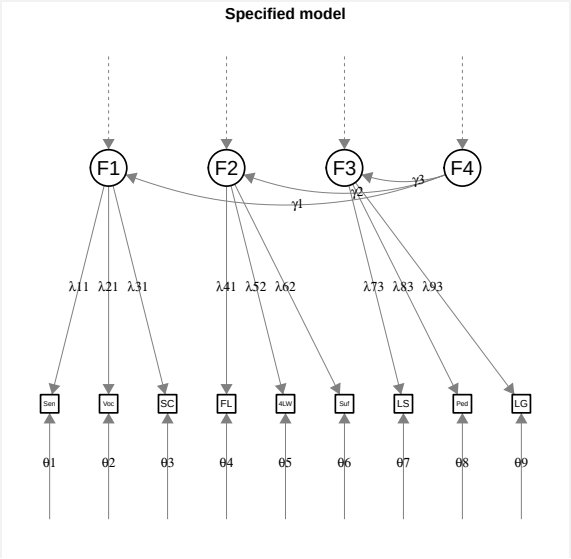
Structural Equation Modelling

```
> qgraph(model.thur)
```



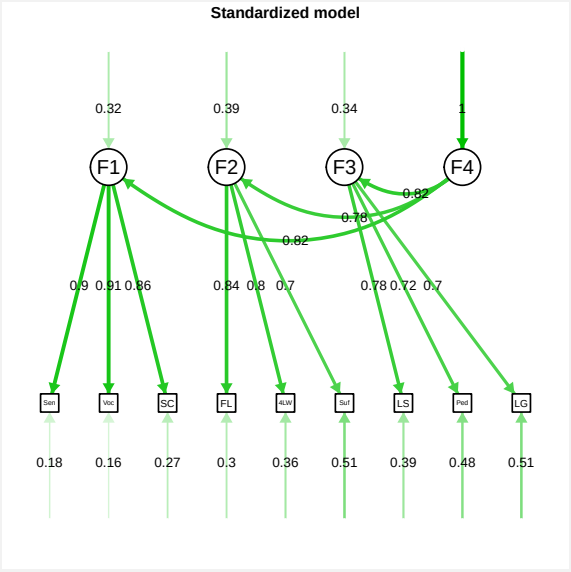
Structural Equation Modelling

```
> qgraph(model.thur, manifest = rownames(R.thur),
+     layout = "tree")
```



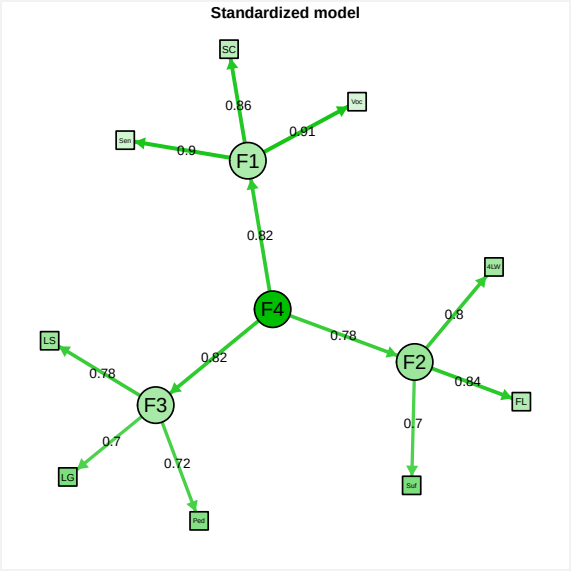
Structural Equation Modelling

```
> qgraph(sem.thur, layout = "tree", curve = 0.4)
```



Structural Equation Modelling

```
> qgraph(sem.thur, layout = "spring", residuals = FALSE)
```



The big 5

```
> library(qgraph)
> data(big5)
> str(big5)
```

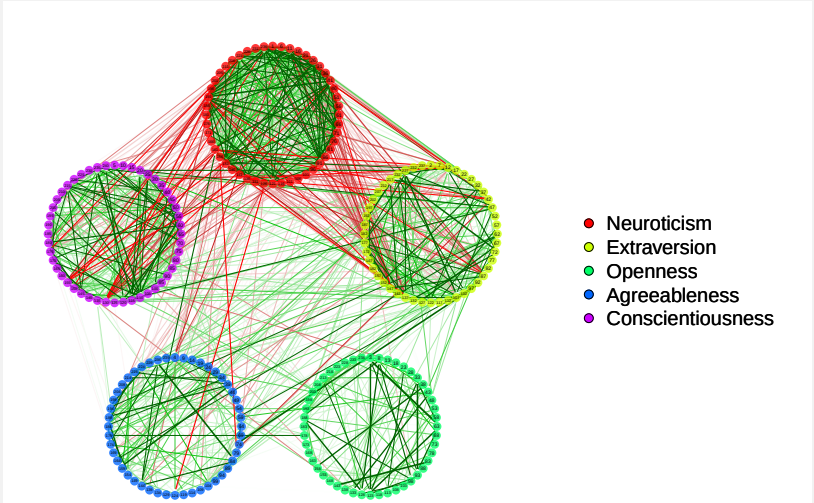
```
 num [1:500, 1:240] 2 3 4 4 5 2 2 1 4 2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:240] "N1" "E2" "O3" "A4" ...
```

```
> data(big5groups)
> str(big5groups)
```

```
List of 5
 $ Neuroticism      : num [1:48] 1 6 11 16 21 26 31 36 41 46 ...
 $ Extraversion     : num [1:48] 2 7 12 17 22 27 32 37 42 47 ...
 $ Openness         : num [1:48] 3 8 13 18 23 28 33 38 43 48 ...
 $ Agreeableness    : num [1:48] 4 9 14 19 24 29 34 39 44 49 ...
 $ Conscientiousness: num [1:48] 5 10 15 20 25 30 35 40 45 50 ..
```

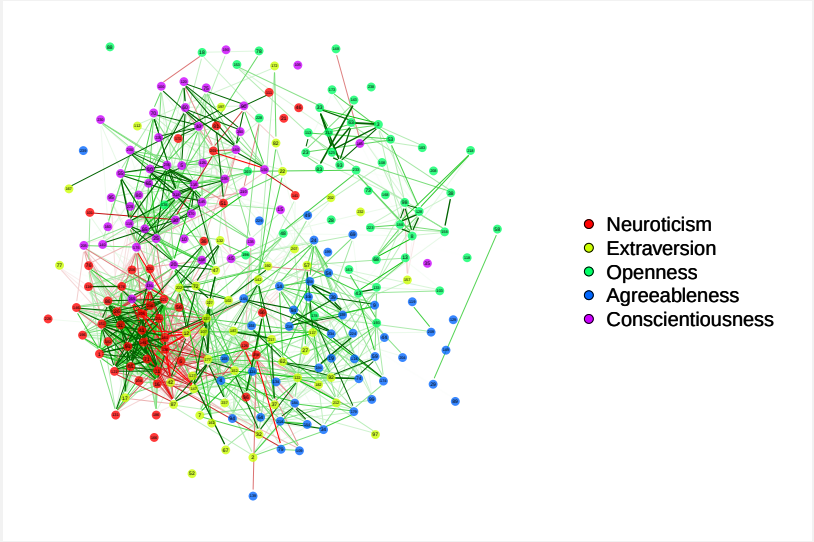
The big 5

```
> Q <- qgraph(cor(big5), minimum = 0.25, cut = 0.4,
+     vsize = 2, groups = big5groups, legend = T,
+     borders = F, vTrans = 200)
```



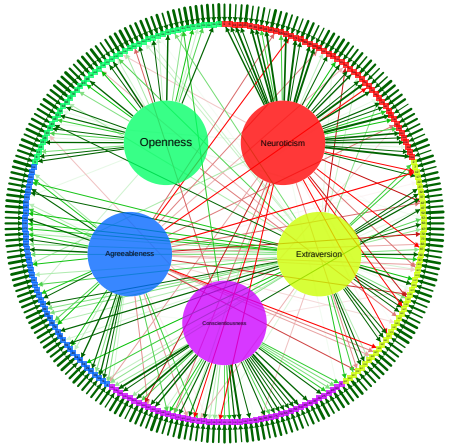
The big 5

```
> qgraph(cor(big5), Q, layout = "spring")
```



EFA

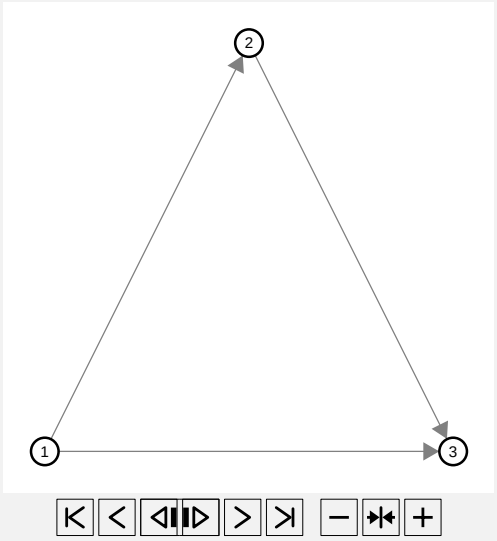
```
> qgraph.efa(cor(big5), factors = 5, Q, layout = "circle",
+     vsize = c(1, 15), borders = F, asize = 0.07,
+     esize = 4, rotation = "promax", residSize = 0.1)
```



Concluding comments

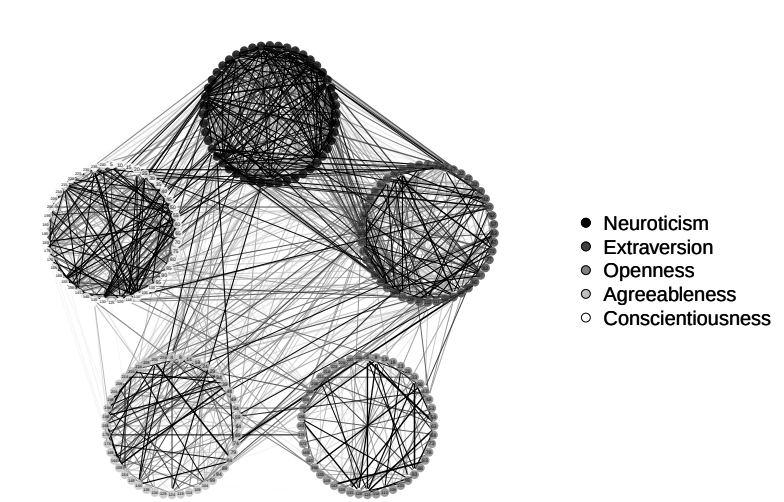
- **qgraph** is still work in progress
- Plans for the future:
  - More wrapper functions for different statistics (e.g. IRT)
  - More layout modes
  - Estimating and fitting causal models
- Some things I couldn't describe...

Layout constraints



Grayscale colors

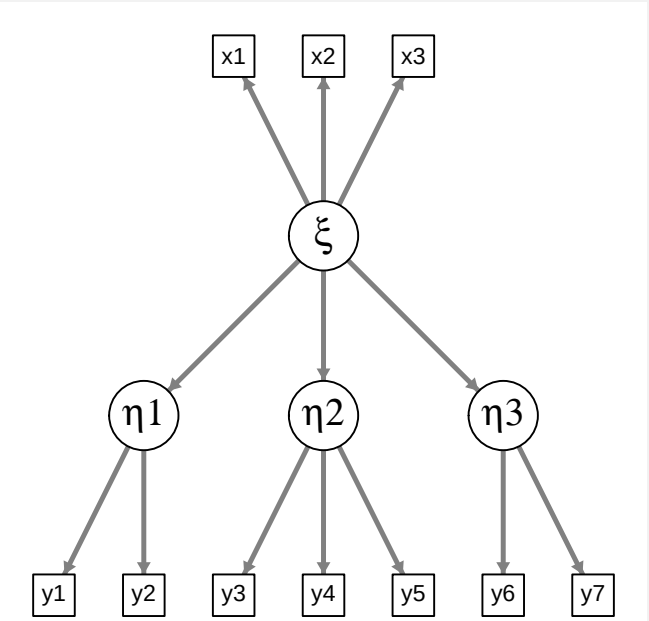
```
> Q <- qgraph(cor(big5), minimum = 0.25, cut = 0.4,
+     vsize = 2, groups = big5groups, legend = T,
+     borders = F, vTrans = 200, gray = TRUE)
```

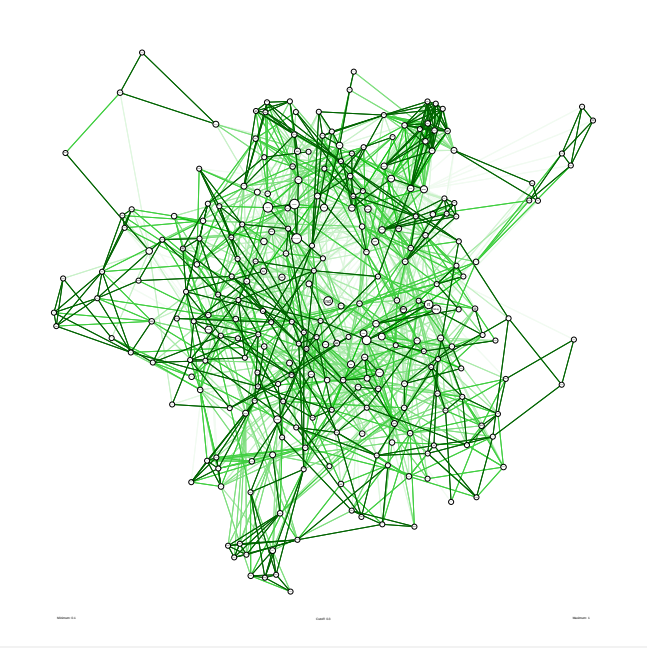


Tooltips

Link

Modelling





Concluding comments

Thank you for your attention!

References

Butts, C. T. (2010). **sna**: Tools for social network analysis [Computer software manual]. Available from `http://CRAN.R-project.org/package=sna` (R package version 2.2-0)

Csardi, G., & Nepusz, T. (2006). The **igraph** software package for complex network research. *InterJournal*, *Complex Systems*, 1695. Available from `http://igraph.sf.net`

Fox, J. (2010). **sem**: Structural equation models [Computer software manual]. Available from `http://CRAN.R-project.org/package=sem` (R package version 0.9-21)

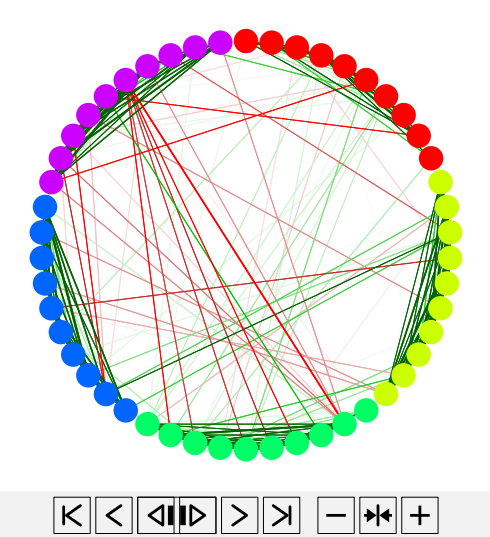
Fruchterman, T., & Reingold, E. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, *21*(11), 1129–1164.

Revelle, W. (2010). **psych**: Procedures for psychological, psychometric, and personality research [Computer software manual]. Evanston, Illinois. Available from `http://personality-project.org/r/psych.manual.pdf` (R package version 1.0-93)

Layout modes

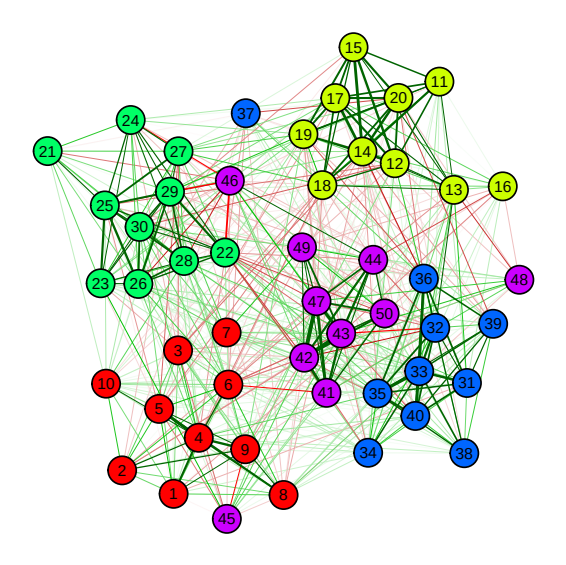
- The placement of the nodes is specified with the `layout` argument in `qgraph()`
- This can be a n by 2 matrix indicating the x and y position of each node
- `layout` can also be given a character indicating one of the two default layouts
- If `layout="circular"` the nodes are placed in circles per group (if the `groups` list is specified)
- If `layout="spring"` a force-embedded algorithm (Fruchterman & Reingold, 1991) is used for the placement
  - This is an iterative algorithm that clusters the nodes so that the length of the edges correspond to the absolute strength of the edges

Fruchterman-Reingold layout (20 iterations)



Fruchterman-Reingold layout (500 iterations)

```
> qgraph(cor(Y), groups = gr, layout = "spring")
```



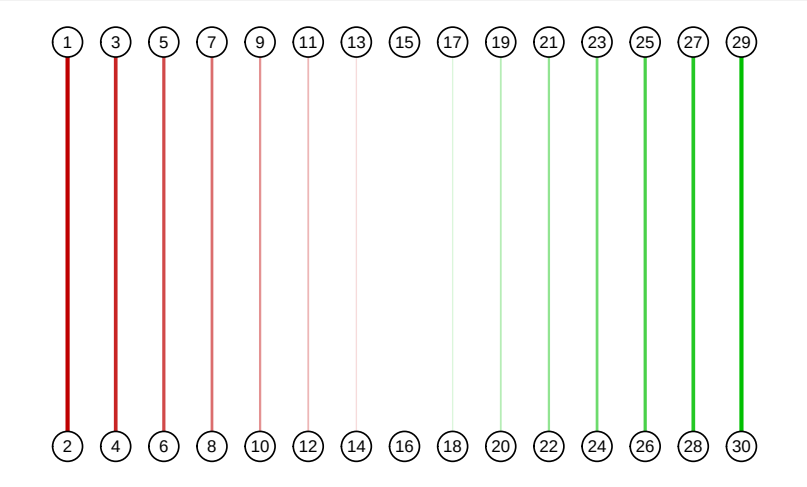
Grid layout

- A final option is to specify a grid as layout
- This can be done by specifying a matrix to `layout` with more than two columns
- This matrix contains zeros and a number for each node

```
> dat.3 <- matrix(c(1:15 * 2 - 1, 1:15 * 2),
+     , 2)
> dat.3 <- cbind(dat.3, round(seq(-0.7, 0.7,
+     length = 15), 1))
> L.3 <- matrix(1:30, nrow = 2)
> L.3
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    3    5    7    9   11   13   15   17
[2,]    2    4    6    8   10   12   14   16   18
     [,10] [,11] [,12] [,13] [,14] [,15]
[1,]    19    21    23    25    27    29
[2,]    20    22    24    26    28    30
```

Grid layout

```
> qgraph(dat.3, layout = L.3, directed = FALSE)
```



Fruchterman-Reingold layout

- `layout="spring"` uses a force-embedded algorithm that was proposed by Fruchterman and Reingold (1991)
  - This layout was ported from the **sna** package (Butts, 2010)
  - A solution for weighted graphs was taken from **igraph** (Csardi & Nepusz, 2006)
- This is an iterative algorithm.
- The initial layout is a circle
- Then in each iteration:
  - Each node is repulsed by all other nodes
  - Connected nodes are also attracted to each other
  - The maximum displacement weakens each iteration
- After this process the layout is rescaled to fit the -1 to 1 xy-plane
- The unscaled layout is returned as `layout.orig`

Fruchterman-Reingold layout

- The Fruchterman-Reingold algorithm can be controlled with the `layout.par` argument
- This must be a list containing other arguments:

niter	Number of iterations, default is 500
max.delta	Maximum displacement, default is n
area	The area of the plot, default is n^2
cool.exp	Cooling exponent, default is 1.5
repulse.rad	Repulse radius, default is $n \cdot \text{area}$
init	Matrix indicating initial layout

Constraints in the Fruchterman-Reingold layout

- The Fruchterman-Reingold algorithm behaves like a chaotic system
- A small difference in initial setup can result in a completely different layout
- In **qgraph** the layout can be constrained to compensate this
- Hard constraints
  - Hard constraints can be used to fix the x and y position of certain nodes
  - This can be done with the `constraints` argument in `layout.par`
- Soft constraints
  - Soft constraints can be used to limit the displacement of certain nodes
  - This can be done with the `max.delta` and `init` arguments in `layout.par`

No constraints

```
> dat.3 <- matrix(c(1, 2, 1, 3, 2, 3), 3, 2,
+     byrow = T)
> L <- matrix(c(1:3, 1, 3, 1), 3, 2)
> Q <- qgraph(dat.3, layout = L, vsize = 3,
+     esize = 1)
> par <- list(init = L)
> set.seed(1)
> for (i in 3:20) {
+     dat.3 <- rbind(dat.3, c(sample(c(i, sample(1:i,
+         1)), 1), i + 1))
+     L <- rbind(L, 0)
+     par$init <- L
+     L <- qgraph(dat.3, Q, layout.par = par,
+         layout = "spring")$layout.orig
+ }
```

No constraints

Hard constraints

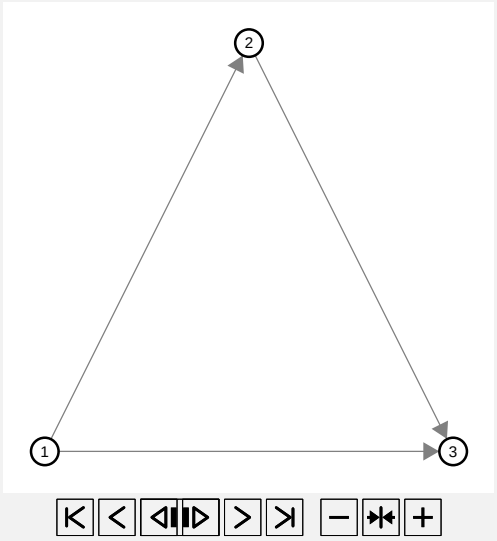
```
> dat.3 <- matrix(c(1, 2, 1, 3, 2, 3), 3, 2,
+     byrow = T)
> L <- matrix(c(1:3, 1, 3, 1), 3, 2)
> par <- list(max.delta = 10, area = 10^2,
+     repulse.rad = 10^3)
> Q <- qgraph(dat.3, layout = L, vsize = 3,
+     esize = 1, layout.par = par)
> set.seed(1)
> for (i in 3:20) {
+     dat.3 <- rbind(dat.3, c(sample(c(i, sample(1:i,
+         1)), 1), i + 1))
+     par$init <- rbind(L, 0)
+     L <- rbind(L, NA)
+     par$constraints <- L
+     L <- qgraph(dat.3, Q, layout.par = par,
+         layout = "spring")$layout.orig
+ }
```

Hard constraints

Soft constraints

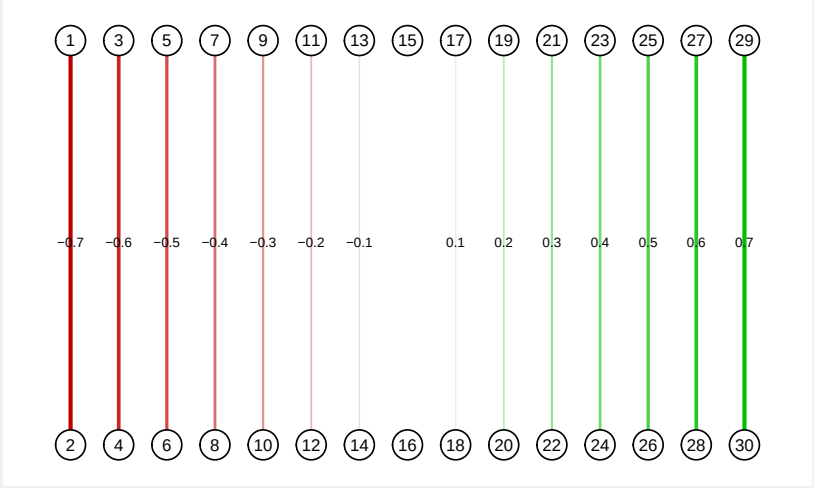
```
> dat.3 <- matrix(c(1, 2, 1, 3, 2, 3), 3, 2,
+     byrow = T)
> L <- matrix(c(1:3, 1, 3, 1), 3, 2)
> par <- list(max.delta = 10, area = 10^2,
+     repulse.rad = 10^3)
> Q <- qgraph(dat.3, layout = L, vsize = 3,
+     esize = 1, layout.par = par)
> set.seed(1)
> for (i in 3:20) {
+     dat.3 <- rbind(dat.3, c(sample(c(i, sample(1:i,
+         1)), 1), i + 1))
+     par$init <- rbind(L, 0)
+     L <- rbind(L, NA)
+     par$max.delta <- 10/(i + 1):1
+     L <- qgraph(dat.3, Q, layout.par = par,
+         layout = "spring")$layout.orig
+ }
```

Soft constraints



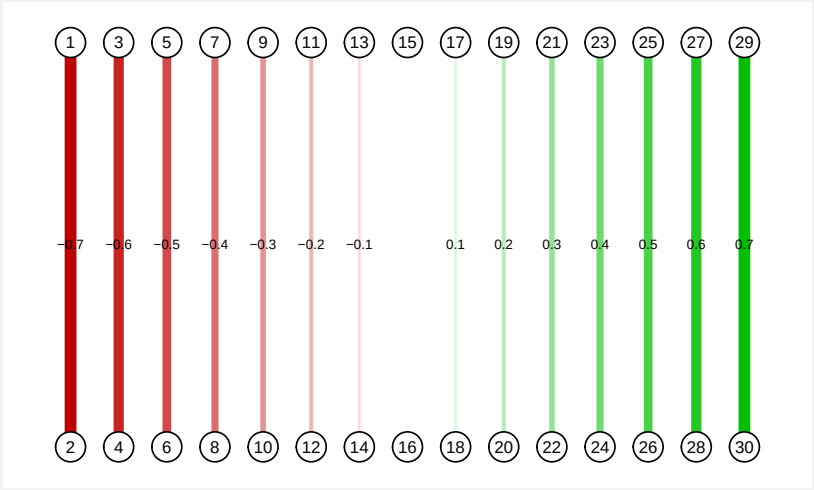
Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+     edge.labels = T)
```



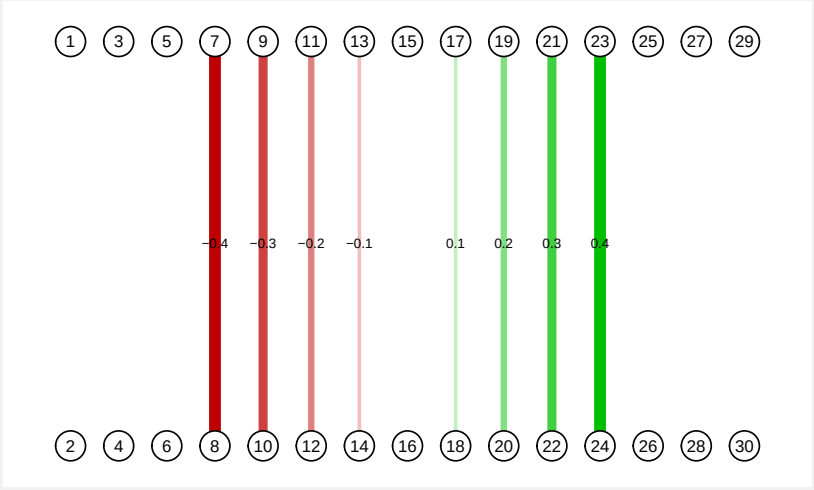
Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+     edge.labels = T, esize = 14)
```



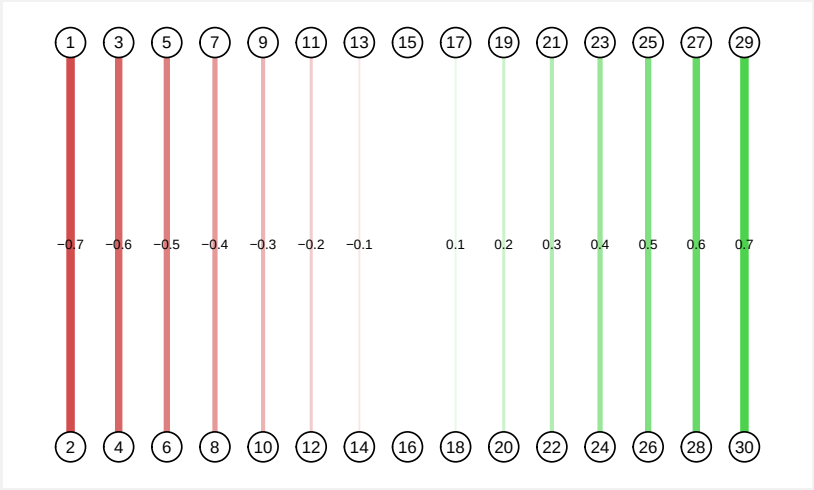
Interpreting weighted graphs

```
> qgraph(dat.3[-c(1:3, 13:15), ], layout = L.3,
+     nNodes = 30, directed = FALSE, edge.labels = T,
+     esize = 14)
```



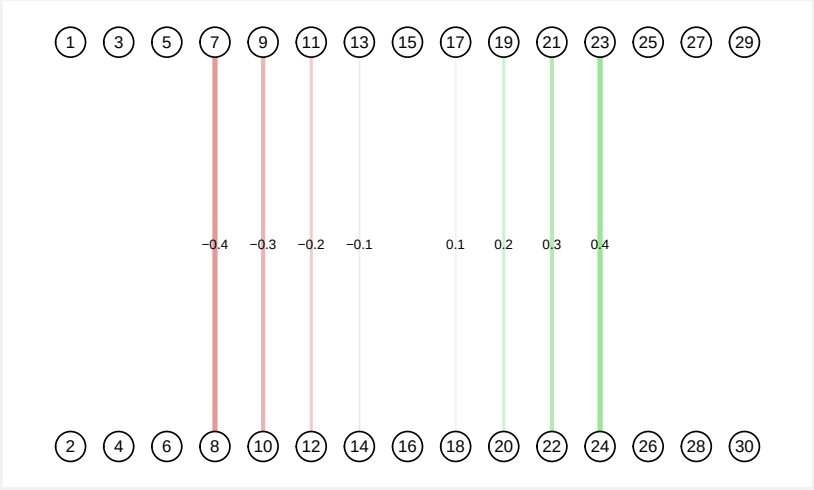
Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+     edge.labels = T, esize = 14, maximum = 1)
```



Interpreting weighted graphs

```
> qgraph(dat.3[-c(1:3, 13:15), ], layout = L.3,
+     nNodes = 30, directed = FALSE, edge.labels = T,
+     esize = 14, maximum = 1)
```

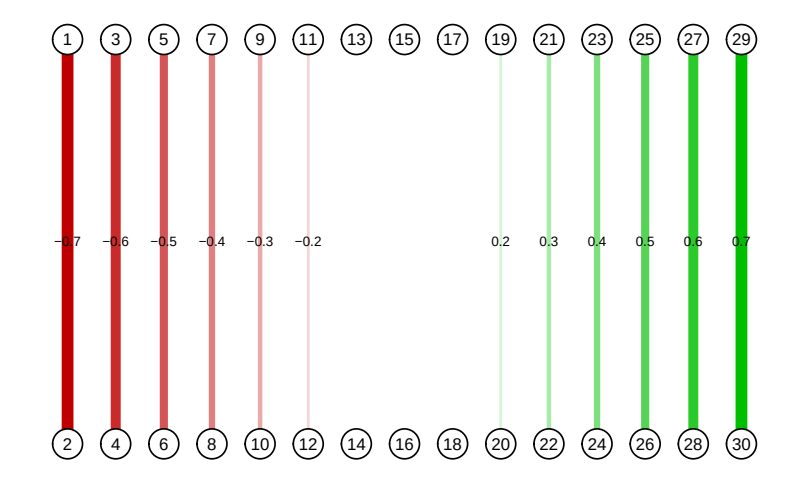


Interpreting weighted graphs

`maximum` **must be set to be able to compare multiple graphs!**

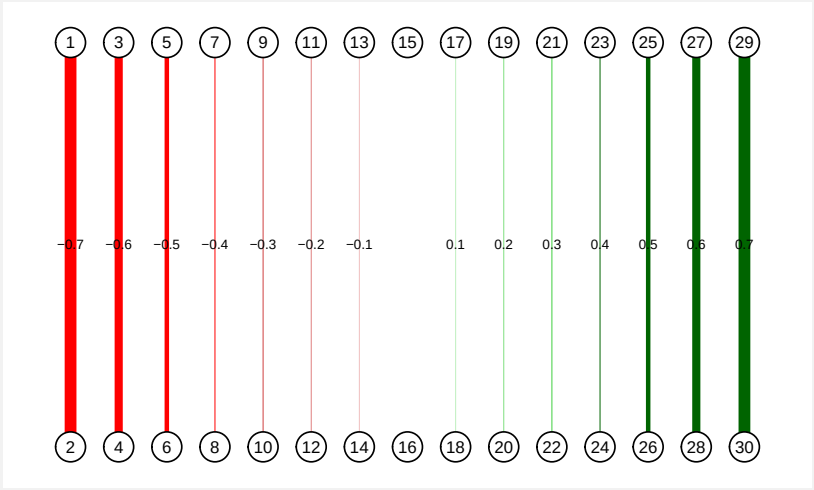
Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+      edge.labels = T, esize = 14, minimum = 0.1)
```



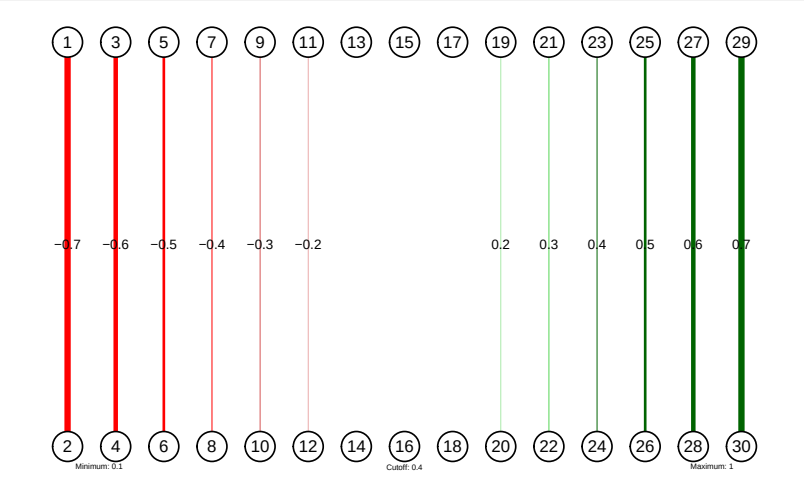
Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+      edge.labels = T, esize = 14, cut = 0.4)
```



Interpreting weighted graphs

```
> qgraph(dat.3, layout = L.3, directed = FALSE,
+      edge.labels = T, esize = 14, minimum = 0.1,
+      maximum = 1, cut = 0.4, details = TRUE)
```



Interpreting weighted graphs

Graphs can not be interpreted without knowing `minimum`, `cut` **and** `maximum`**!**