The centrality lethality rule in the signed protein interaction network of Drosophila

Savas Paragamian

2017-05-26

Contents

Chapter 1

Data exploration and preparation

1.1 The network: first look

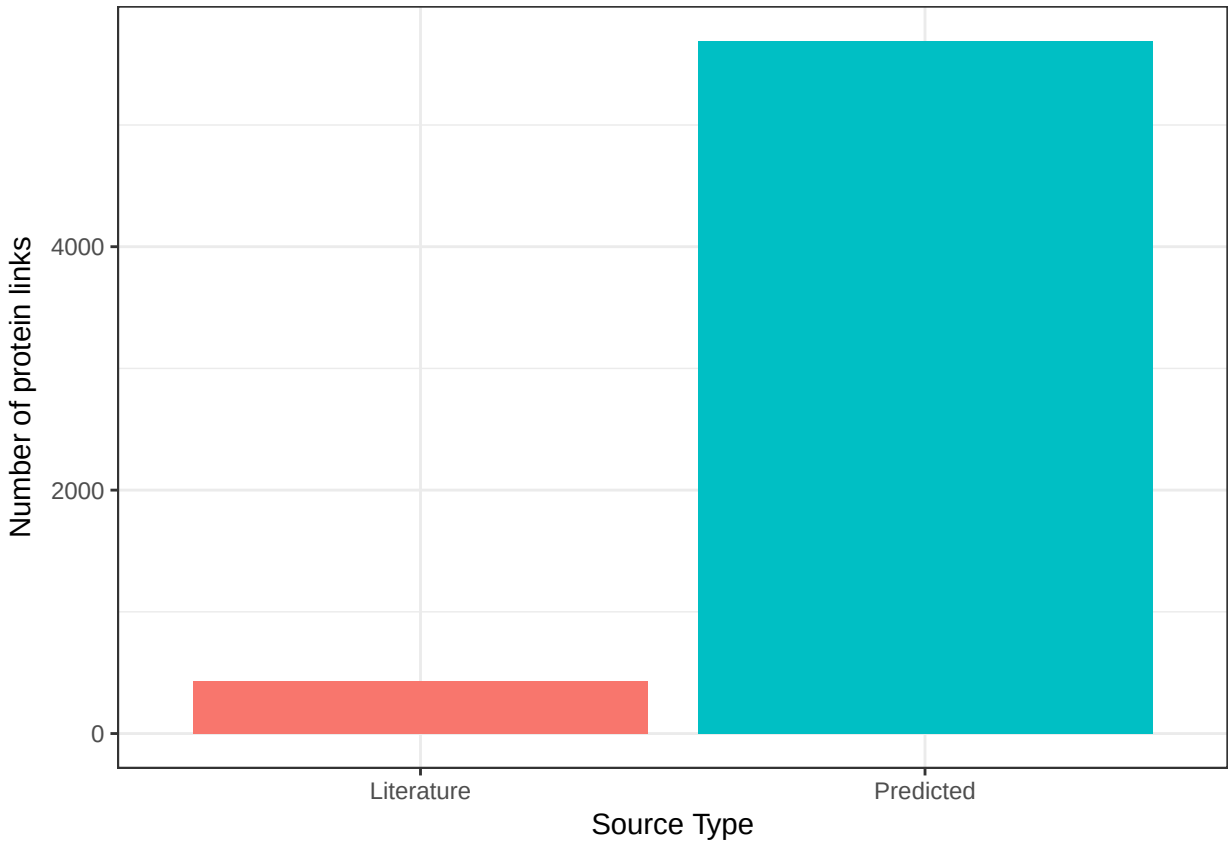
This is the analysis of the network data provided by the authors of the article (Arunachalam Vinayagam et al. 2014).

The file *nmeth.2733-S2.xlsx*, download from supplementary information, is an edge list for the network. It has 6125 rows (links) and the following 20 columns.

```
##  [1] "ID1"                   "Symbol1"
##  [3] "ID2"                   "Symbol2"
##  [5] "Source_Type"           "Edge_Sign"
##  [7] "Sign_Score"            "NumberOfScreens"
##  [9] "Coexpress_Development" "Literature_SignSource"
## [11] "PPI_Assay"             "PPI_PubMed"
## [13] "PPI_Database"          "Interolog_Organisms"
## [15] "ID1_HumanOrtholog"     "ID1_DIOPTScore"
## [17] "ID1_HumanDisease"      "ID2_HumanOrtholog"
## [19] "ID2_DIOPTScore"        "ID2_HumanDisease"
```

The first 2 columns are the proteins that interact with the proteins in the 3-4 columns. So the **ID** and **Symbol** columns refer to an individual protein. The **ID** is the id of the protein in the FlyBase database (e.g *FBgn0041604*) and the **Symbol** is the name of the protein which provides some information about it's function (e.g *dlp*).

The 5th column, **Source_Type**, is a link attribute of whether the **Sign_Score** of the link was *Predicted* from the authors framework or was added from *Literature*.




```
## Saving 6.5 x 4.5 in image
```

1.2 Duplicates removal

Next, we want to check the uniqueness of the links. In order to check this we first concatenate the node IDs for each link and then we check for uniqueness.

```
## [1] 6094
```

There are 6094 unique links but the number of links is 6125.

```
## [1] 6125
```

So there are some duplicates. Next we want to label each link to see if it's duplicated in order to find the reasons of the duplicates.

```
dros_values$duplicate <- duplicated(dros_values$IDS) | duplicated(dros_values$IDS, fromLast = TRUE)  #
```

Looking through the data is ease to recognize that the origin of the duplicates comes from the overlap of some links of the *Literature* and the *Predicted* methods for the *Sign Score*. Now we have to select from the duplicates. I will disregard the *Literature* duplicates.

```
dros_values <- dros_values[!(dros_values$duplicate == "TRUE" & dros_values$Source_Type == "Literature")
    ]  # Selection of the duplicates

print(paste0("Number of rows after the removal of duplicates = ", nrow(dros_values)))
```

```
## [1] "Number of rows after the removal of duplicates = 6094"
```

```
print(paste0("Number of unique links after the removal of duplicates = ", length(unique(dros_values$IDS
```

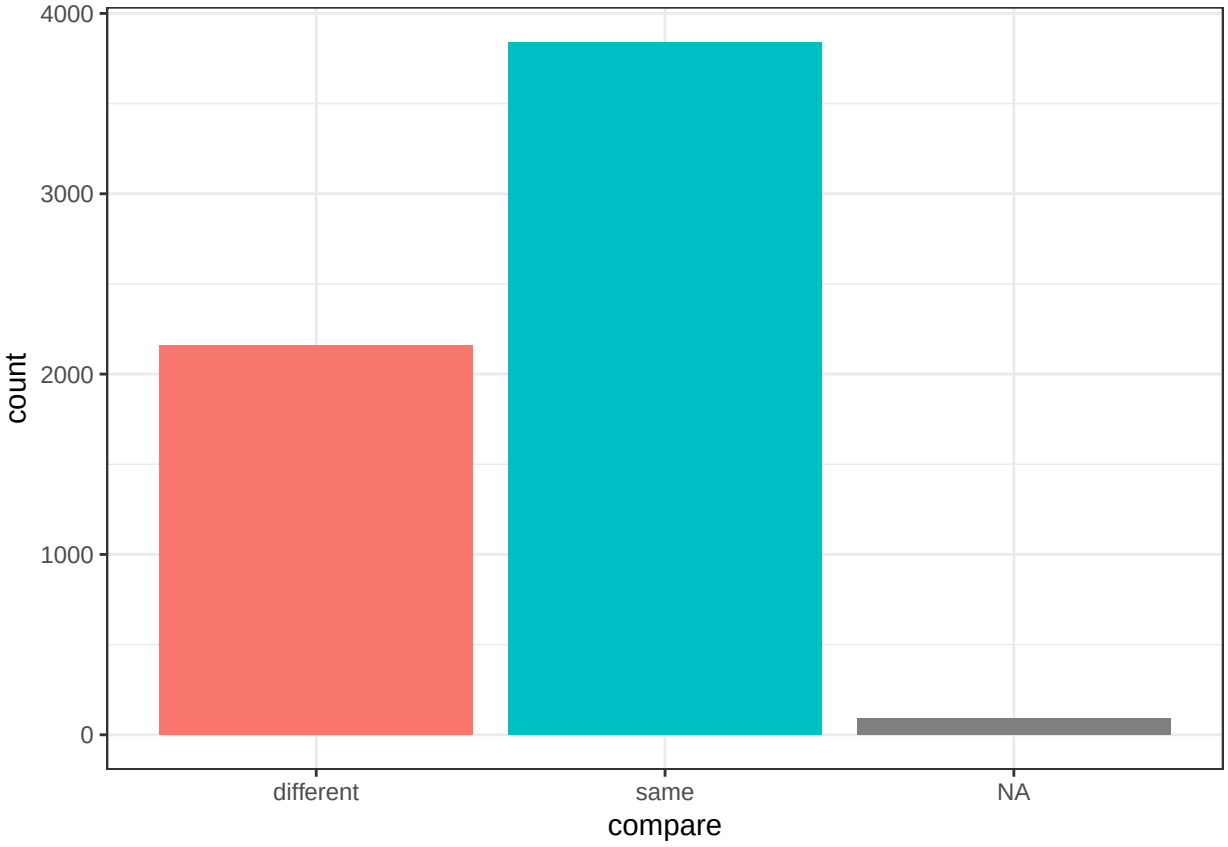
```
## [1] "Number of unique links after the removal of duplicates = 6094"
```

Now there only unique links.

1.3 Weights examination

The 7th, **Sign_Score** and the 9th, **Coexpress_Development**, columns are different signed weights of the PPI network generated from different experiments. **Sign_Score** is predicted from RNAi screens in Drosophila and **Coexpress_Development** is predicted from time-course gene expression data, microarrays. **Coexpress_Development** has **94** missing values and **Sign_Score** doesn't have, missing values= **0**.

First we can see the sign similarity of each method.




```
## Saving 6.5 x 4.5 in image
```

So it is obvious that the different methods produce different weights for the interactions.

Some statistics of the Sign Scores:

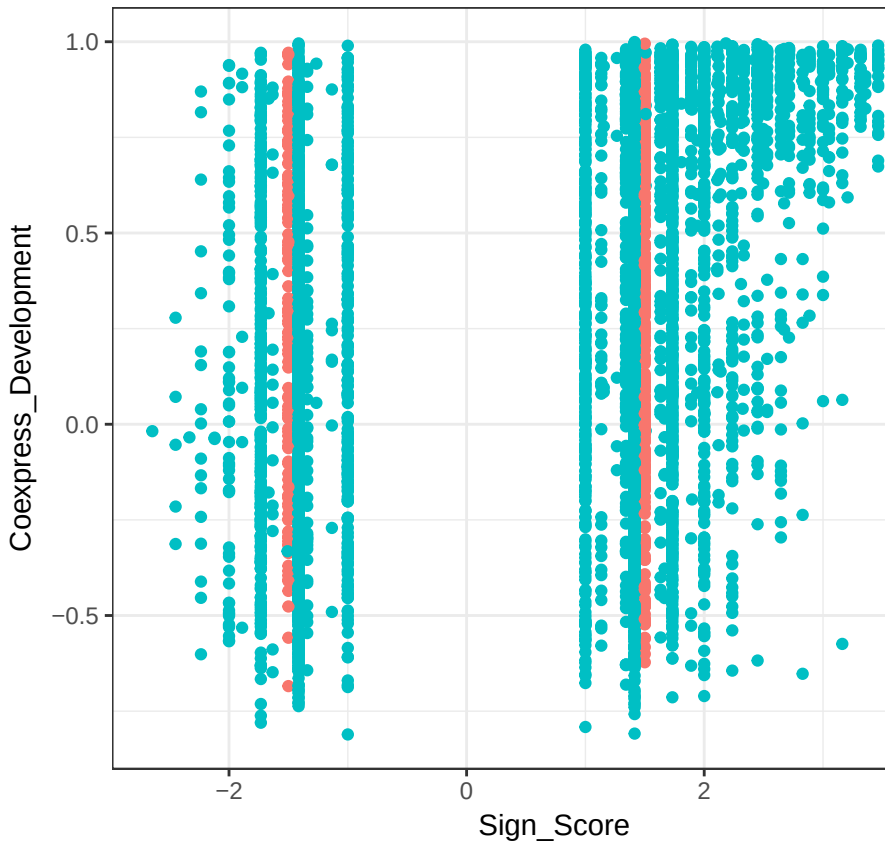
```
summary(dros_values$Sign_Score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.6460 -1.4140  1.4140  0.6446  1.6330  4.1230
```

and for the Coexpress Correlations

```
summary(dros_values$Coexpress_Development)
```

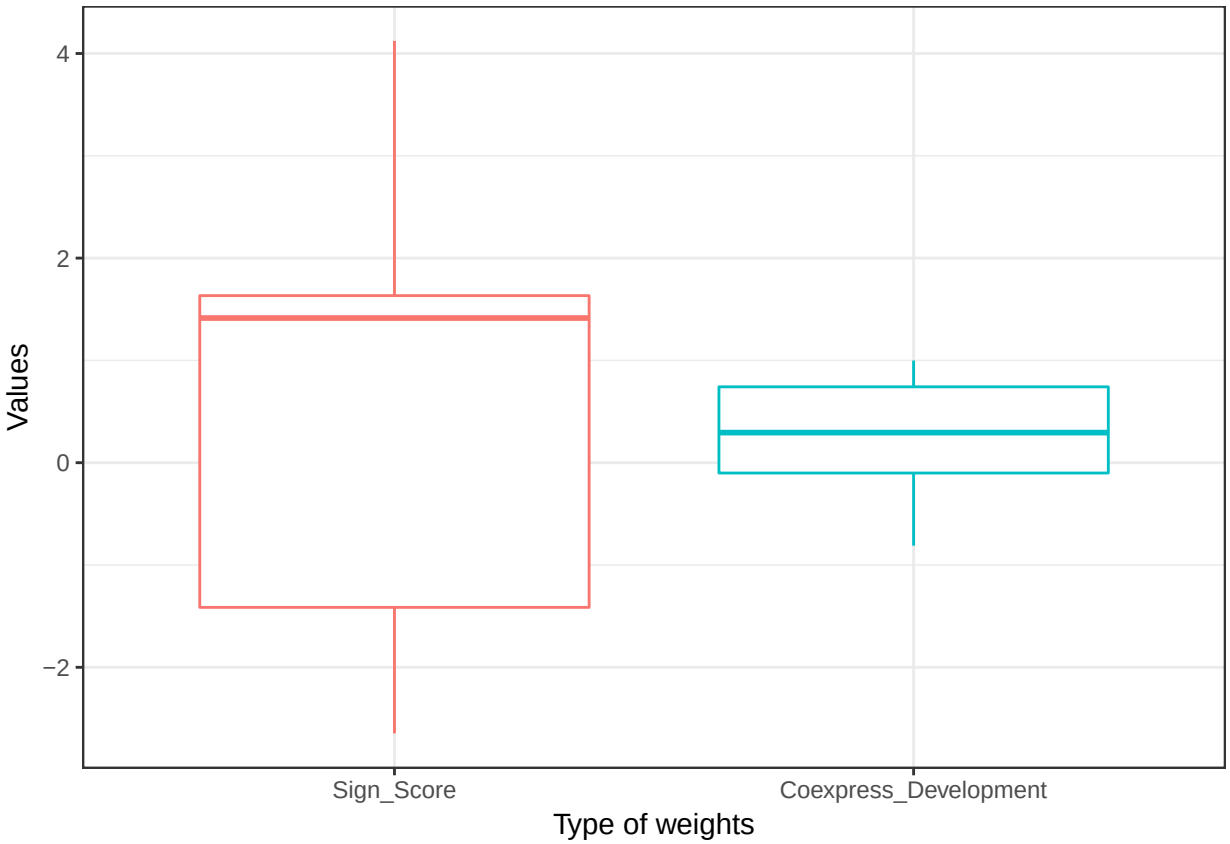
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -0.8109 -0.1008  0.2942  0.2899  0.7416  0.9993      94
```



Here we can see the different density plots of each:

```
## Saving 6.5 x 4.5 in image
```

```
## No id variables; using all as measure variables
```



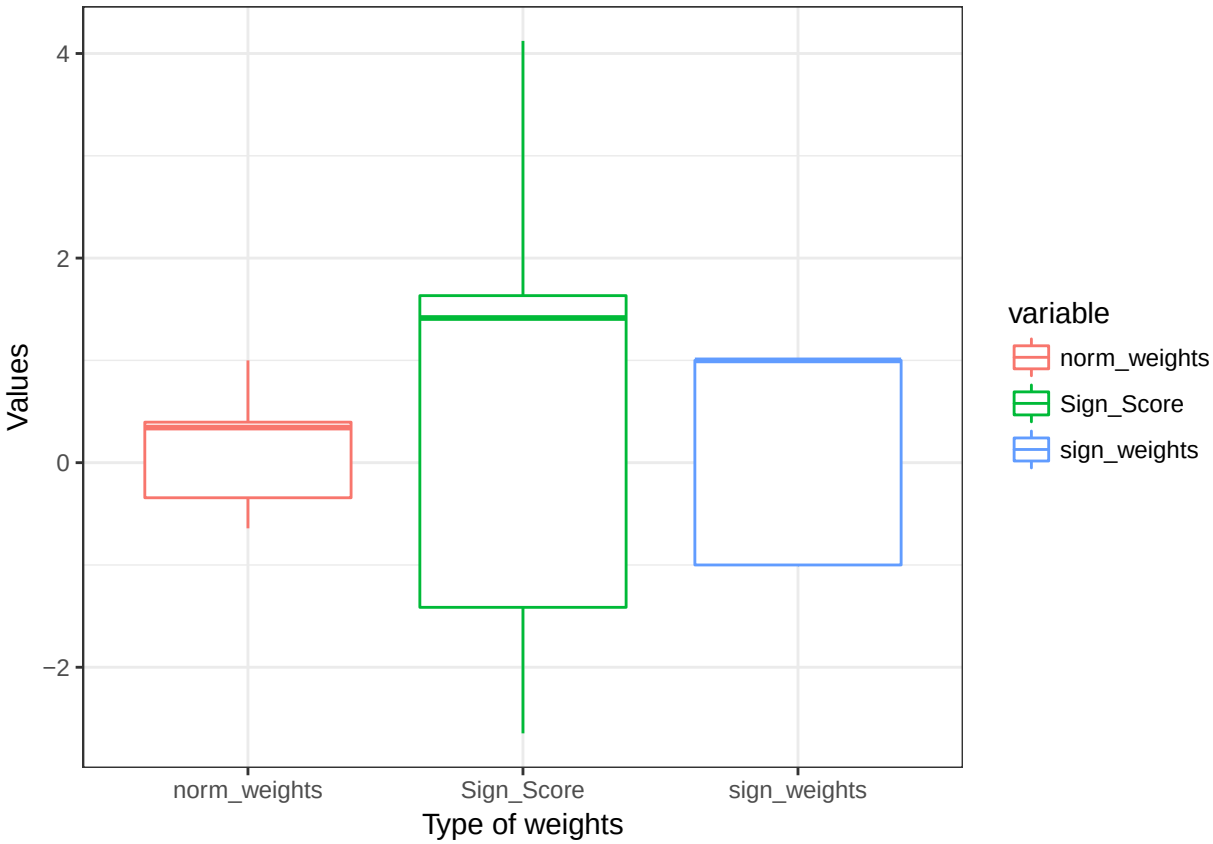

```
## Saving 6.5 x 4.5 in image
```

1.4 Weights normalization

```
dros_values$norm_weights <- sapply(dros_values$Sign_Score, function(x) x/max(abs(dros_values$Sign_Score)

dros_values$sign_weights <- with(dros_values, ifelse(Sign_Score < 0, -1, 1))
dros_values$abs_weights <- abs(dros_values$norm_weights)

# boxplot different weights:sign scores, normalized with max value and just signs

signs_norm <- dros_values[, c("norm_weights", "Sign_Score", "sign_weights")]

melt_sign_norm <- melt(data = signs_norm, na.rm = F)
```

```
## No id variables; using all as measure variables
```

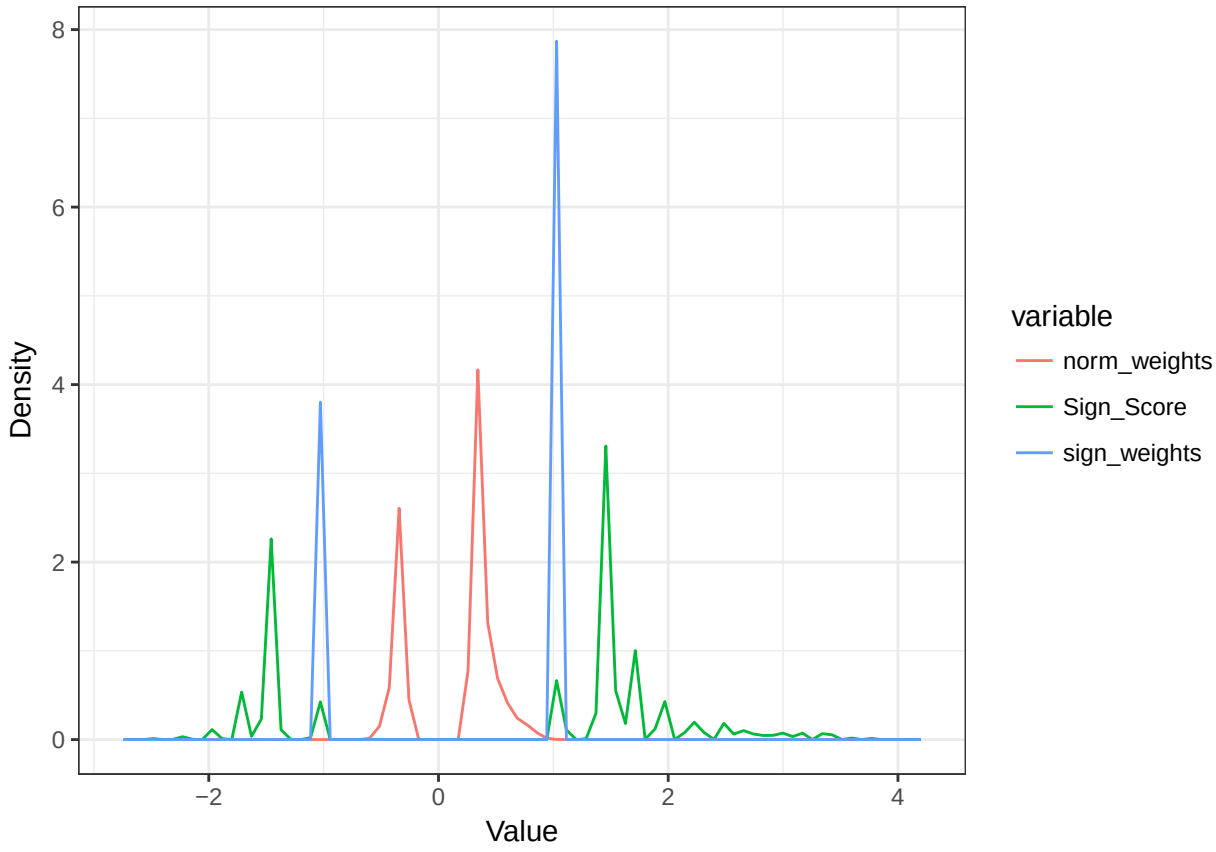
```
ggplot() + geom_boxplot(data = melt_sign_norm, aes(x = variable, y = value, color = variable)) + # ggti
labs(x = "Type of weights", y = "Values") + theme_bw()
```




```
ggsave("type_weights_boxplot.pdf", plot = last_plot(), device = "pdf", dpi = 150, path = "Figures/")
```

```
## Saving 6.5 x 4.5 in image
```

```
# Density

ggplot() + geom_freqpoly(data = melt_sign_norm, bins = 80, aes(value, y = ..density.., color = variable
    labs(x = "Value", y = "Density") + # ggtitle('Densities of Sign Scores, normalized Sign Scores and
theme_bw()
```

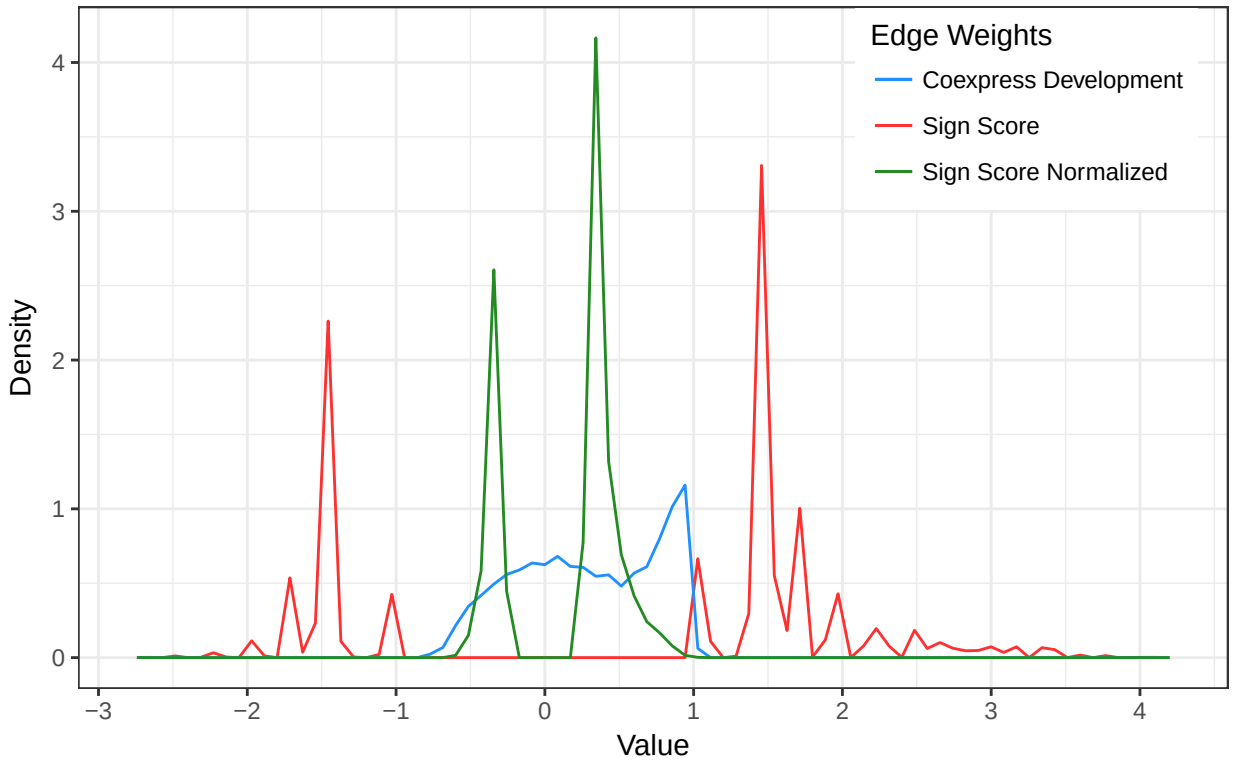



```
ggsave("type_weights_densities.pdf", plot = last_plot(), device = "pdf", dpi = 150, path = "Figures/")
```

```
## Saving 6.5 x 4.5 in image
```

```
ggplot() + geom_freqpoly(data = dros_values, bins = 80, aes(Sign_Score, y = ..density.., color = "Sign S
    geom_freqpoly(data = dros_values, bins = 80, aes(Coexpress_Development, y = ..density.., color = "Co
    geom_freqpoly(data = dros_values, bins = 80, aes(norm_weights, y = ..density.., color = "Sign Score
    labs(x = "Value", y = "Density") + scale_x_continuous(breaks = seq(-3, 5, 1)) + scale_y_continuous(b
    4, 1)) + coord_fixed(ratio = 1) + scale_colour_manual(values = c(`Sign Score` = "firebrick1", `Coexp
    `Sign Score Normalized` = "forestgreen"), name = "Edge Weights") + # ggtitle('Densities of Sign Sco
theme_bw() + theme(legend.position = c(0.825, 0.85))
```

```
## Warning: Removed 94 rows containing non-finite values (stat_bin).
```




```
ggsave("different_weights_densities.pdf", plot = last_plot(), device = "pdf", dpi = 150, path = "Figures
```

```
## Saving 6.5 x 4.5 in image
```

```
## Warning: Removed 94 rows containing non-finite values (stat_bin).
```

Chapter 2

Network centrality-leathality analysis

2.1 Biological gene essentiality

Before we see the network it is important to enrich the data with gene essentiality in order to analyse them altogether.

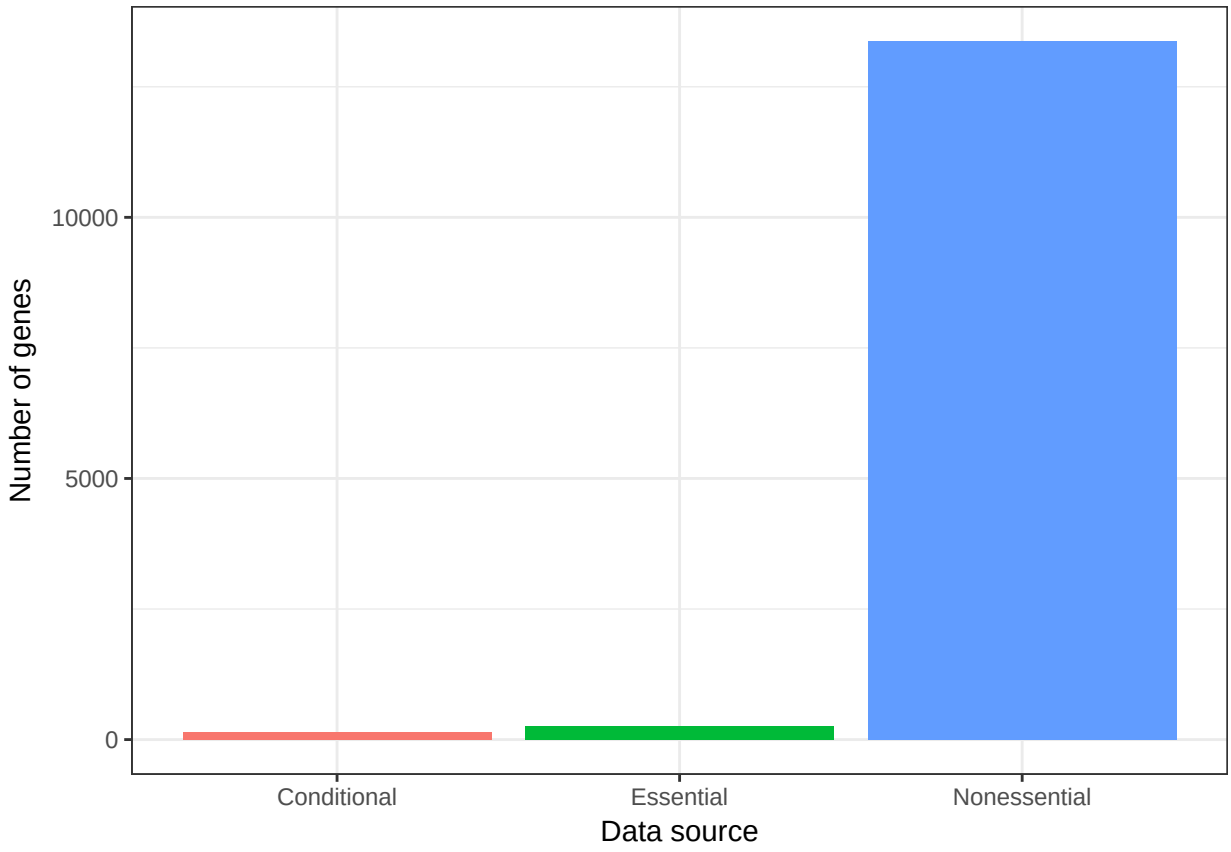
> **Essential** *genes are the genes that when removed from the organism it cannot survive or is infertile.*

The authors of the article **Predicting Essential Genes and Proteins Based on Machine Learning and Network Topological Features/ A Comprehensive Review** write:

> *currently, the available essential genes and protein databases are DEG (Zhang and Lin, 2009), CEG (Ye et al., 2013), OGEE (Chen et al., 2012), and EGGS. These data have enabled researchers to explore the features of essential genes and proteins and, through this exploration, reveal which features are associated with essentiality and, finally, develop computational methods proposed to identify essential genes and proteins.*

We used OGEE for the genes of *Drosophila melanogaster*. We downloaded a file with all the known essential genes and their IDs.

```
##          locus       symbols           datasets       datasetIDs
##  FBgn0000008:    1   Mode:logical   Min.   :1.000   347    :13344
##  FBgn0000014:    1   NA's:13781     1st Qu.:1.000   347,363:  326
##  FBgn0000015:    1                  Median :1.000   363,347:  111
##  FBgn0000017:    1                  Mean   :1.032
##  FBgn0000018:    1                  3rd Qu.:1.000
##  FBgn0000022:    1                  Max.   :2.000
##  (Other)    :13775
##  essentiality.status  essentiality.consensus
##  E    :  263         Conditional :  141
##  E,E  :    4         Essential   :  267
##  E,NE :   37         Nonessential:13373
##  NE   :13081
##  NE,E :  104
##  NE,NE:  292
##
```



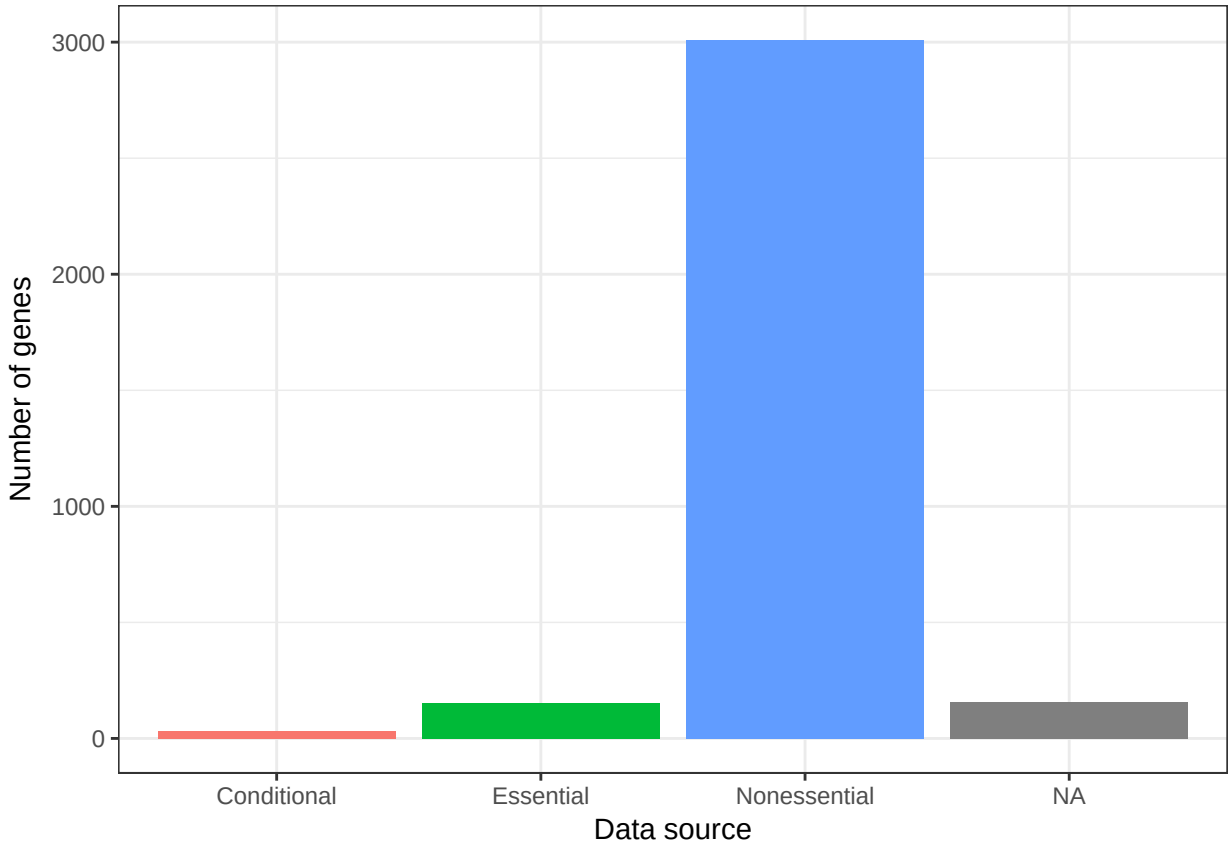

```
## Saving 6.5 x 4.5 in image
```

These are the types of Essentiality Consensus. These data were generated from 2 screens, so Conditional means that one screen identified a gene as *Essential* but the other screen as *Nonessential*.

Next we want to enrich the genes with the Essentiality Consensus.

```
## Enrich every ID in the edge list
dros_values$ID1_consensus <- dros_essential[match(dros_values$ID1, dros_essential$locus), "essentiality
dros_values$ID2_consensus <- dros_essential[match(dros_values$ID2, dros_essential$locus), "essentiality

# Drospphila network genes with IDs
dros_network_genes <- as.data.frame(unique(c(dros_values$ID1, dros_values$ID2)))

dros_genes <- as.data.frame(unique(dros_values[, 1]))
colnames(dros_network_genes)[1] <- "locus"

dros_network_genes <- merge(dros_network_genes, dros_essential, by.x = 1, by.y = 1, all.x = T)
dros_network_genes$color <- with(dros_network_genes, ifelse(essentiality.consensus == "Essential", paste
    ifelse(essentiality.consensus == "Conditional", paste0("orange"), paste0("green"))))
# summary(dros_network_genes$essentiality.consensus)

write.table(dros_network_genes, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_ne
    sep = ",", col.names = TRUE, row.names = FALSE)

## Bar plot of Gene esssentiality consesus of OGEE for Drosophila
ggplot() + geom_bar(data = dros_network_genes, aes(essentiality.consensus, y = ..count.., fill = essenti
    # ggtitle('Drosophila gene network essentiality from different sources')+
```

```
labs(x = "Data source", y = "Number of genes") + guides(fill = FALSE) + theme_bw()
```




```
ggsave("essentiality_consensus_bar_plot.pdf", plot = last_plot(), device = "pdf", dpi = 150, path = "Fig
```

```
## Saving 6.5 x 4.5 in image
```

There are 156 genes that don't match because there is not information about their Essentiality Consensus.

2.2 Network degree distribution

From the edge list, using the package *igraph* we can create the network with edge attribute the **Sign Score**. The network is **directed**. The authors mention the methods that allowed them to extract the interactions directionality.

```
dros_net <- dros_values[, -c(2, 4)]  # edgelist, remove the names of the proteins!
colnames(dros_net)[4] <- "weights"

write.table(dros_net, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_net.csv",
    sep = ",", col.names = TRUE, row.names = FALSE)

dros_vertex_attr <- dros_network_genes[, c(1, 6, 7)]
g <- graph_from_data_frame(dros_net, directed = T, vertices = dros_vertex_attr)
summary(g)
```

```
## IGRAPH DN-- 3352 6094 --
## + attr: name (v/c), essentiality.consensus (v/c), color (v/c),
```

```
## | Source_Type (e/c), weights (e/n), Coexpress_Development (e/n),
## | sign_score_sign (e/c), coexpress_sign (e/c), compare (e/c),
## | color (e/c), IDS (e/c), duplicate (e/l), norm_weights (e/n),
## | sign_weights (e/n), abs_weights (e/n), ID1_consensus (e/c),
## | ID2_consensus (e/c)
```

The network isn't connected.

```
igraph::is.connected(g)
```

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

From the igraph object, the network, it is possible to extract the weighted adjacency matrix.

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

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

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

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

```
## [1] 11229810
```

```
## [1] 6094
```

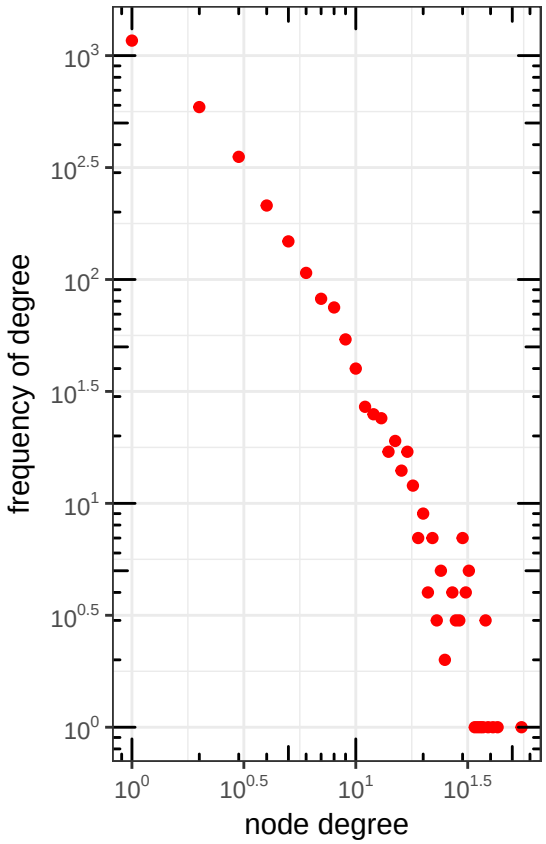
```
## [1] TRUE
```

```
## [1] "Number of non zero elements of the adjacency matrix = 6094"
```

To make further calculations we isolate the giant component.

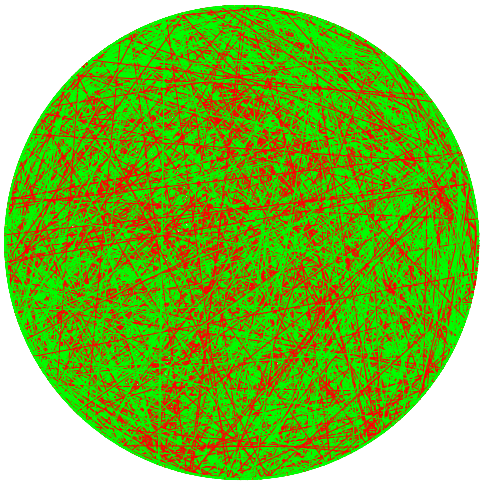
```
decg <- decompose.graph(g, min.vertices = 10)  #upografima tis megalis sinistosas
gcomp <- decg[[1]]
```

One of the first things to examine is the degree distribution of the graph.



Plot of the network.

Drosophila signed PPI network



2.3 Centralities

We calculate degree, betweenness, closeness, local and global transitivity without the weights and with their absolute value and then we compare them in respect to gene essentiality.

```
##### without weights
V(gcomp)$degree <- igraph::degree(gcomp)
V(gcomp)$betweenness <- igraph::betweenness(gcomp, weights = NA, directed = T)
V(gcomp)$closeness_all <- igraph::closeness(gcomp, weights = NA, mode = "all")
V(gcomp)$closeness_in <- igraph::closeness(gcomp, weights = NA, mode = "in")
V(gcomp)$closeness_out <- igraph::closeness(gcomp, weights = NA, mode = "out")
V(gcomp)$transitivity_l <- igraph::transitivity(gcomp, type = "local", weights = NA)
V(gcomp)$transitivity_g <- igraph::transitivity(gcomp, type = "global", weights = NA)

##### with abs weights

V(gcomp)$abs_strength <- igraph::strength(gcomp, weights = E(gcomp)$abs_weights)
V(gcomp)$w_abs_betweenness <- igraph::betweenness(gcomp, weights = E(gcomp)$abs_weights)
V(gcomp)$w_abs_closeness <- igraph::closeness(gcomp, weights = E(gcomp)$abs_weights)
V(gcomp)$w_abs_transitivity_l <- igraph::transitivity(gcomp, type = "local", weights = E(gcomp)$abs_weig
V(gcomp)$w_abs_transitivity_g <- igraph::transitivity(gcomp, type = "global", weights = E(gcomp)$abs_wei


#### with the signed normalised weights

V(gcomp)$strength <- igraph::strength(gcomp, weights = E(gcomp)$norm_weights)
V(gcomp)$w_transitivity_l <- igraph::transitivity(gcomp, type = "local", weights = E(gcomp)$norm_weights
V(gcomp)$w_transitivity_g <- igraph::transitivity(gcomp, type = "global", weights = E(gcomp)$norm_weigh


gcomp_attributes <- sapply(igraph::list.vertex.attributes(gcomp), function(x) igraph::get.vertex.attribu
    x))

gcomp_attributes <- as.data.frame(gcomp_attributes)
gcomp_attributes[, 4] <- as.numeric(as.character(gcomp_attributes[, 4]))
gcomp_attributes[, 5] <- as.numeric(as.character(gcomp_attributes[, 5]))
gcomp_attributes[, 6] <- as.numeric(as.character(gcomp_attributes[, 6]))
gcomp_attributes[, 7] <- as.numeric(as.character(gcomp_attributes[, 7]))
gcomp_attributes[, 8] <- as.numeric(as.character(gcomp_attributes[, 8]))
gcomp_attributes[, 9] <- as.numeric(as.character(gcomp_attributes[, 9]))
gcomp_attributes[, 10] <- as.numeric(as.character(gcomp_attributes[, 10]))
gcomp_attributes[, 11] <- as.numeric(as.character(gcomp_attributes[, 11]))
gcomp_attributes[, 12] <- as.numeric(as.character(gcomp_attributes[, 12]))
gcomp_attributes[, 13] <- as.numeric(as.character(gcomp_attributes[, 13]))
gcomp_attributes[, 14] <- as.numeric(as.character(gcomp_attributes[, 14]))
gcomp_attributes[, 15] <- as.numeric(as.character(gcomp_attributes[, 15]))
gcomp_attributes[, 16] <- as.numeric(as.character(gcomp_attributes[, 16]))
gcomp_attributes[, 17] <- as.numeric(as.character(gcomp_attributes[, 17]))
gcomp_attributes[, 18] <- as.numeric(as.character(gcomp_attributes[, 18]))
gcomp_attributes[, 19] <- as.numeric(as.character(gcomp_attributes[, 19]))
gcomp_attributes[, 20] <- as.numeric(as.character(gcomp_attributes[20]))
```

```
## Warning: NAs introduced by coercion
```

```
gcomp_attributes[, 21] <- with(gcomp_attributes, ifelse(essentiality.consensus == "Nonessential", 0,
    1))
colnames(gcomp_attributes)[21] <- "labels"  # for the evaluation methods
```

2.4 Centralities from Antoniou

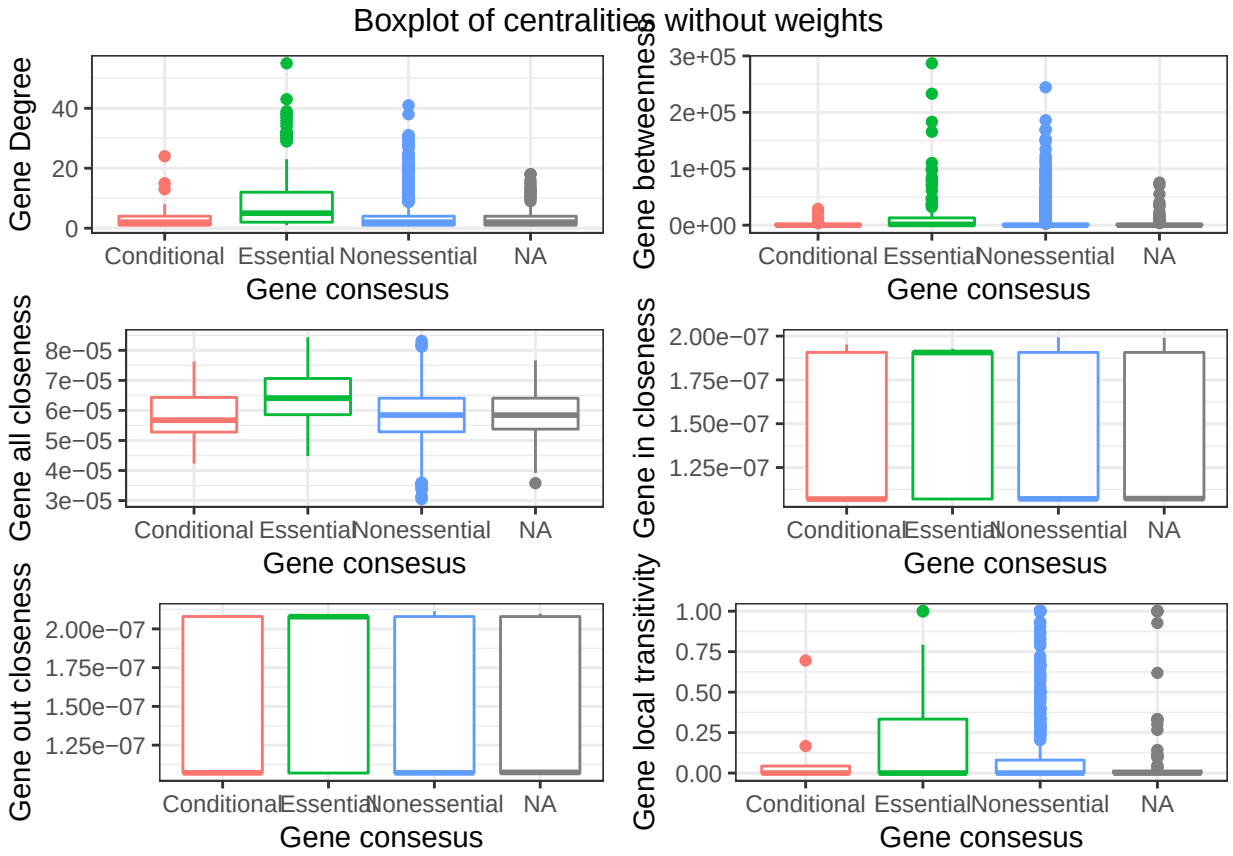
```
drosophila_degreesA <- read_excel("Data/Antoniou_degree_out30.xlsx")
drosophila_degreesA <- as.data.frame(drosophila_degreesA)
dros_essential2 <- dros_essential[, c(1, 6)]

write.table(x = dros_essential2, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_es
    sep = ",", col.names = TRUE, row.names = FALSE)

drosophila_degreesA <- left_join(x = drosophila_degreesA, y = dros_essential2, by = c(label = "locus"))
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factor and character vector, coercing into character vector
```

```
drosophila_degreesA$labels <- with(drosophila_degreesA, ifelse(essentiality.consensus == "Nonessential"
    0, 1))
```

Box plots of centralities.

```
## Warning: Removed 1167 rows containing non-finite values (stat_boxplot).
```

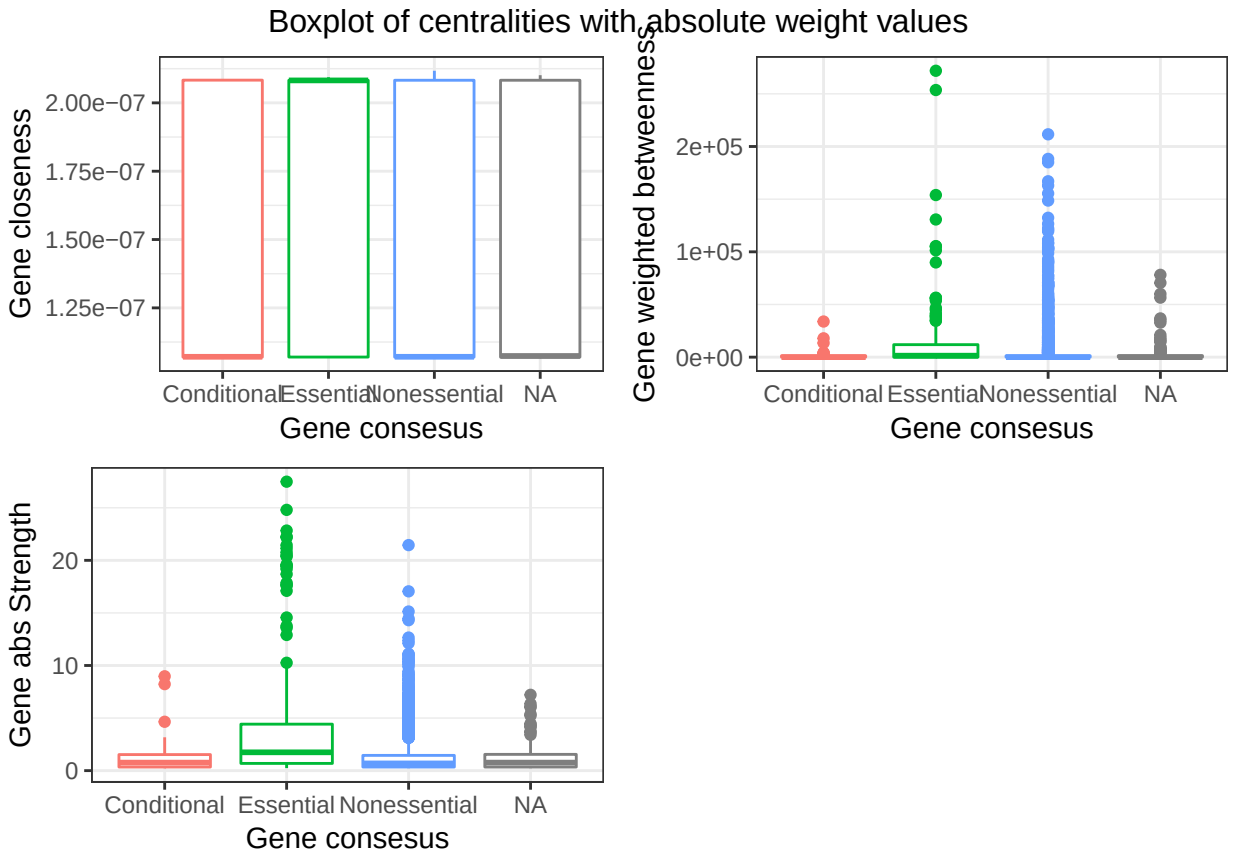



```
## TableGrob (4 x 2) "arrange": 7 grobs
##   z     cells    name                grob
## 1 1 (2-2,1-1) arrange       gtable[layout]
## 2 2 (2-2,2-2) arrange       gtable[layout]
## 3 3 (3-3,1-1) arrange       gtable[layout]
## 4 4 (3-3,2-2) arrange       gtable[layout]
## 5 5 (4-4,1-1) arrange       gtable[layout]
```

```
## 6 6 (4-4,2-2) arrange       gtable[layout]
## 7 7 (1-1,1-2) arrange text[GRID.text.1581]
```

```
## Saving 6.5 x 4.5 in image
```

Box plots with absolute values of normalised weights.

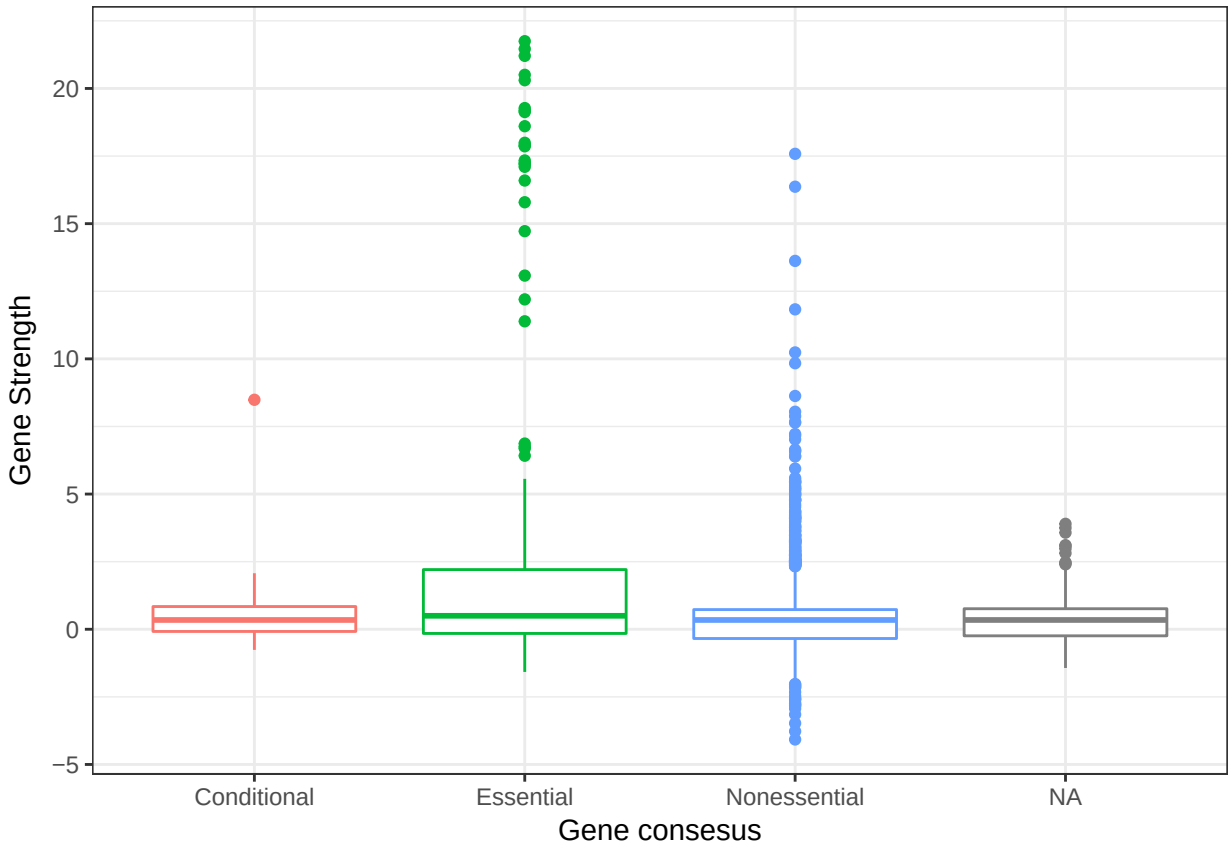



```
## TableGrob (3 x 2) "arrange": 4 grobs
##   z     cells    name                grob
## 1 1 (2-2,1-1) arrange       gtable[layout]
## 2 2 (2-2,2-2) arrange       gtable[layout]
## 3 3 (3-3,1-1) arrange       gtable[layout]
## 4 4 (1-1,1-2) arrange text[GRID.text.1817]
```

```
## Saving 6.5 x 4.5 in image
```

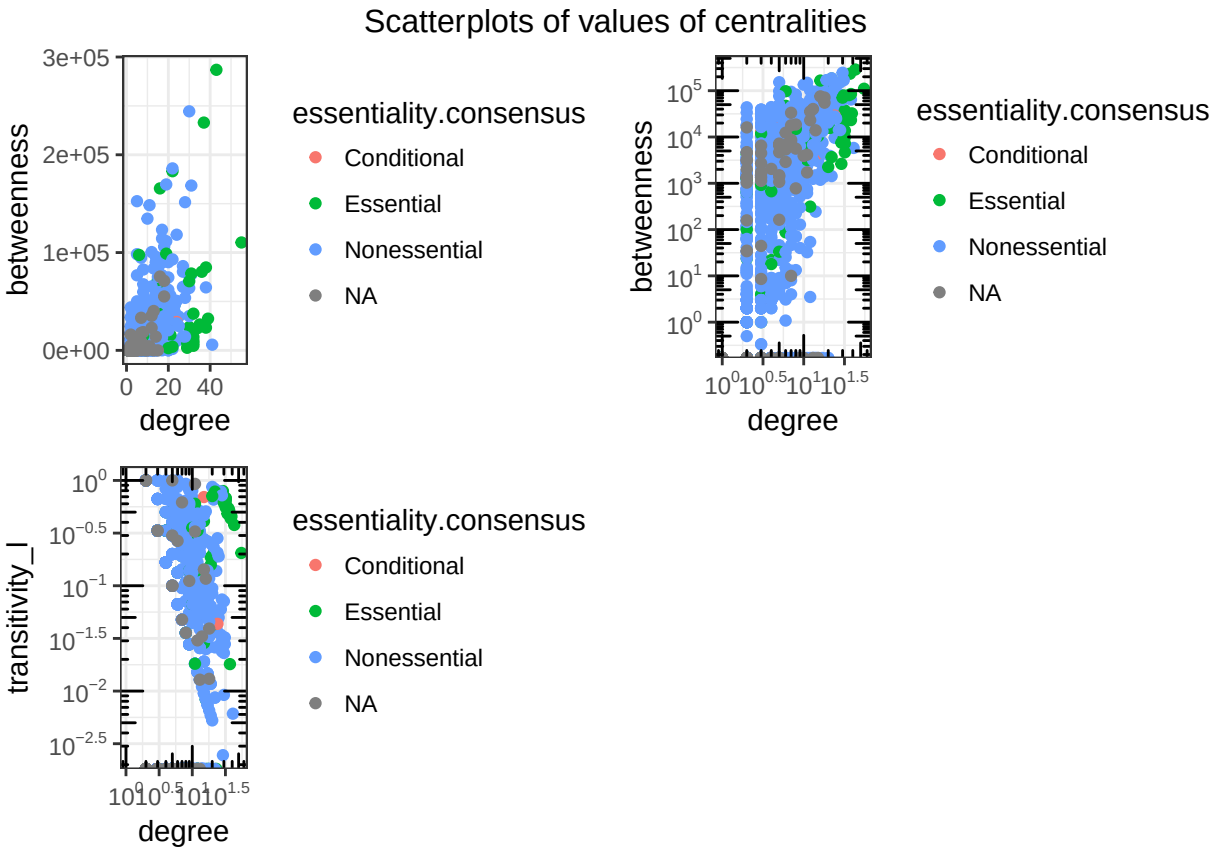
```
## Warning in rm(grid_centralies_abs_drosoplila_network, boxplot_strength, :
## object 'boxplot_betweenness_abs' not found
```

With original weights.




```
## Saving 6.5 x 4.5 in image
```

Scatter plot between gene betweenness and degree in normal - normal and log - log scales. And degree and local transitivity in normal - normal scale.




```
## TableGrob (3 x 2) "arrange": 4 grobs
##   z     cells    name                grob
## 1 1 (2-2,1-1) arrange       gtable[layout]
## 2 2 (2-2,2-2) arrange       gtable[layout]
## 3 3 (3-3,1-1) arrange       gtable[layout]
## 4 4 (1-1,1-2) arrange text[GRID.text.2191]
```

```
## Saving 6.5 x 4.5 in image
```

2.5 Decision trees

We test the application of decision trees on the centrality indices to predict essentiality. It has been reported that integrated methods are more accurate than any single ones.

Here we apply 3 different algorithms:

1. The C5.0 algorithm from the C50 package
2. the rpart package algorithm
3. the C4.5 algorithm from the RWeka package

From these the C5.0 is considered the most reliable and accepted method for decision trees construction.

```
# jdk-8u121-macosx-x64.dmg in command line :sudo R CMD javareconf
# install.packages('rJava',type='source') install.packages('RWeka')
# http://justrocketscience.com/post/install-rweka-mac after it is still broken follow the
# instructions http://charlotte-ngs.github.io/2016/01/MacOsXrJavaProblem.html in RStudio change the
# folder to jdk1.8.0_121.jdk because now there is a new version available!!  continue the
# instructions
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.3.2
```

```
library("rpart.plot")
```

```
## Warning: package 'rpart.plot' was built under R version 3.3.2
```

```
# library(rJava)
library(RWeka)
library(C50)

####
drosophila_degreesA2 <- drosophila_degreesA[, -c(56, 57)]

gcomp_attributes_tree1 <- left_join(gcomp_attributes, drosophila_degreesA2, by = c(name = "label")) %>%
    dplyr::select(-c(1, 3, 4, 5, 6, 11, 12, 16, 17, 19, 20, 21)) %>% na.omit(.) %>% mutate(essentiality
    "Essential", "Essential", "Nonessential")))
```

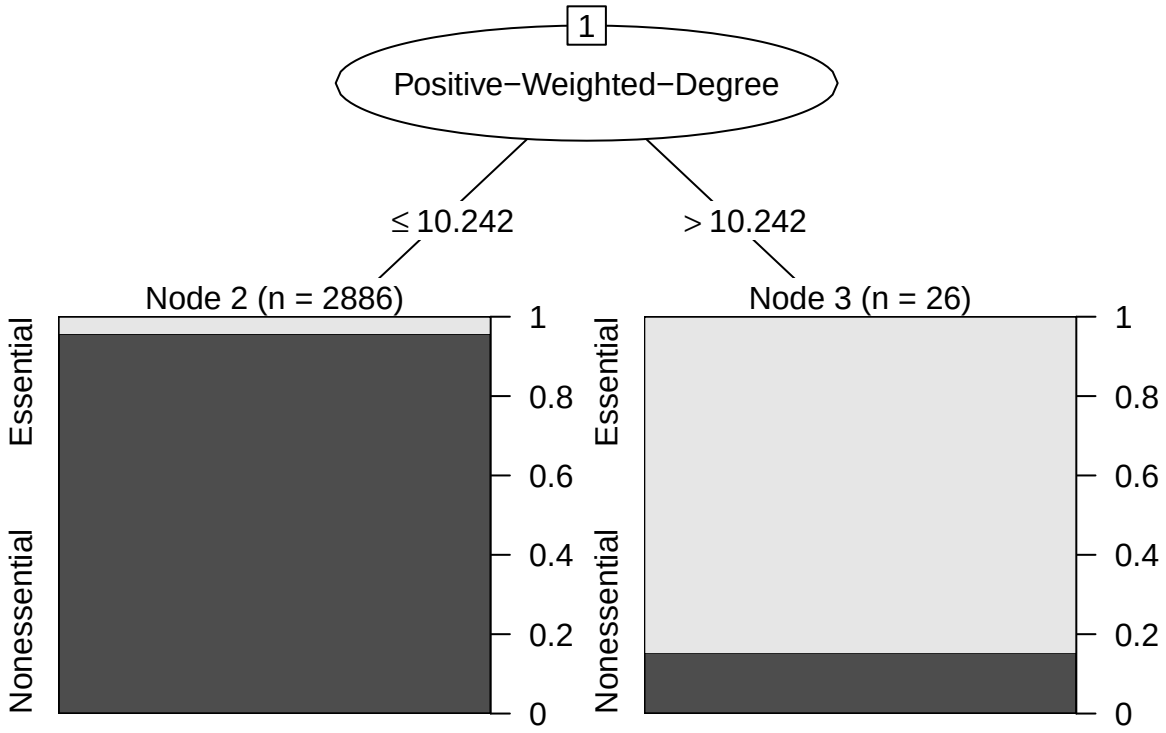
```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## character vector and factor, coercing into character vector
```

```
# Decision tree C50

# fit model
fit <- C5.0(essentiality.consensus ~ ., data = gcomp_attributes_tree1)
# summarize the fit
print(fit)
```

```
##
## Call:
## C5.0.formula(formula = essentiality.consensus ~ ., data
##  = gcomp_attributes_tree1)
##
## Classification Tree
## Number of samples: 2912
## Number of predictors: 62
##
## Tree size: 2
##
## Non-standard options: attempt to group attributes
```

```
# make predictions
predictions <- predict(fit, gcomp_attributes_tree1)

predictions_probs <- predict(fit, gcomp_attributes_tree1, type = "prob")

# summarize accuracy
predictions_table_C50 <- as.data.frame(table(predictions, gcomp_attributes_tree1$essentiality.consensus

# Consolidate objects
gcomp_attributes_tree <- cbind(gcomp_attributes_tree1, predictions_probs)

gcomp_attributes_tree <- gcomp_attributes_tree %>% mutate(labels = as.factor(ifelse(essentiality.consens
    "Essential", 1, 0)))
```

```
plot(fit)
```




```
pdf(file = "Figures/Decision_tree_C5_0.pdf")
plot(fit, trial = 1)
```

```
## Warning in plot.C5.0(fit, trial = 1): Only 1 trials are in the model.
## Setting 'trial' to 0 (the plot code is zero-based).
```

```
dev.off()
```

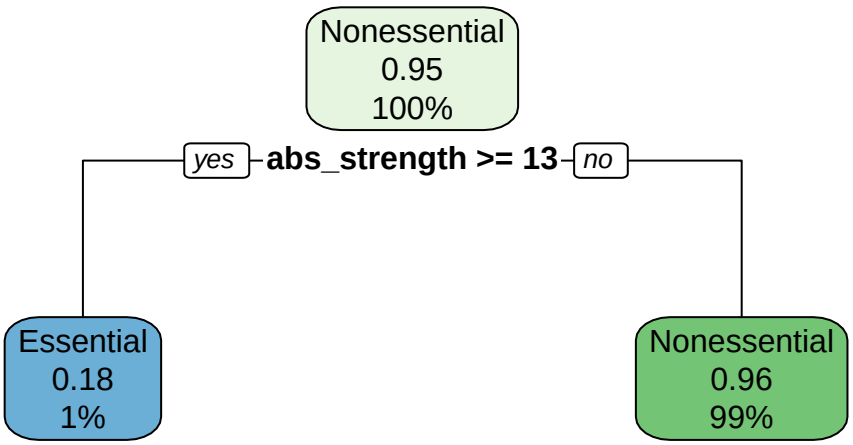
```
## pdf
##   2
```

From the rpart package we get the very similar results. There only two branches in the tree.

```
# Decision tree rpart

fit_rpart <- rpart(essentiality.consensus ~ ., data = gcomp_attributes_tree1, method = "class")


pdf(file = "Figures/Decision_tree_rpart.pdf")
rpart.plot(fit_rpart)
dev.off()
```

```
## pdf
##   2
```

```
rpart.plot(fit_rpart)
```



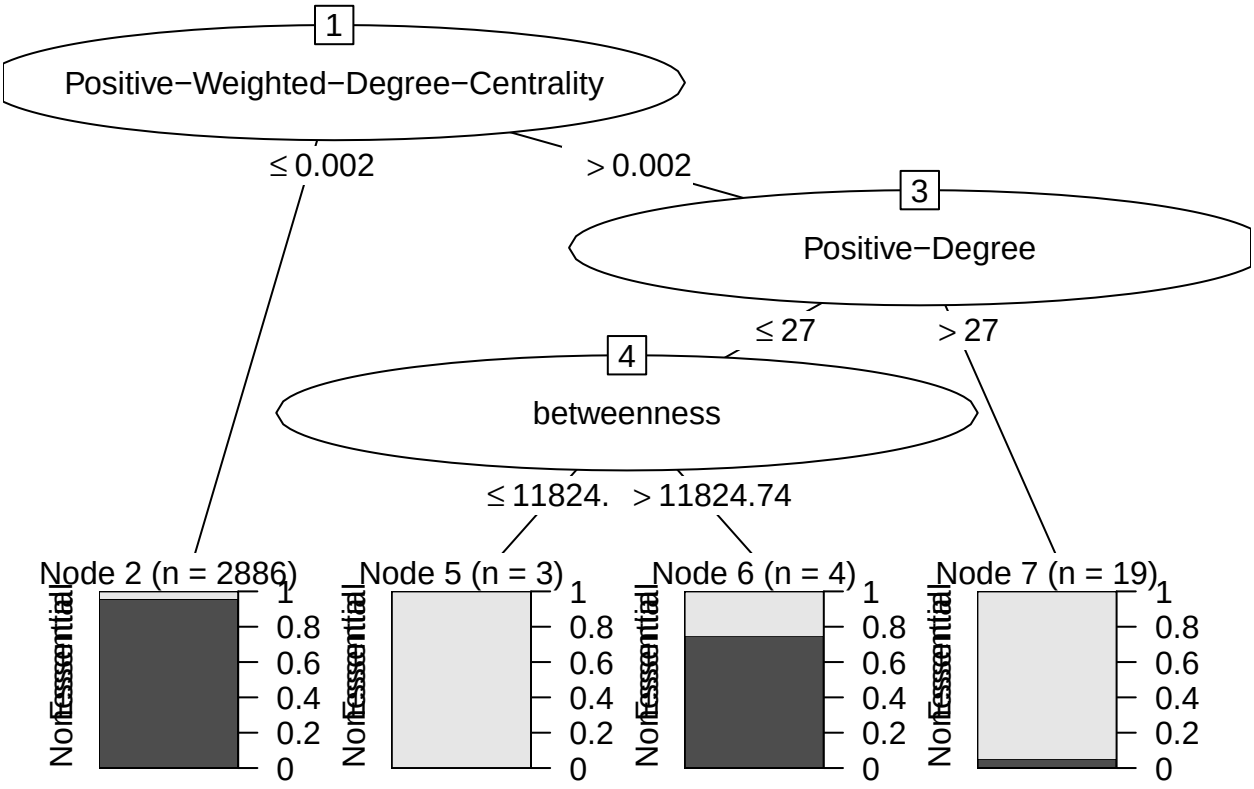

```
# make predictions
predictions <- predict(fit_rpart, type = "class")

predictions_table_rpart <- as.data.frame(table(predictions, gcomp_attributes_tree1$essentiality.consensu

predictions_probs_rpart <- as.data.frame(predict(fit_rpart, gcomp_attributes_tree1, type = "prob"))
colnames(predictions_probs_rpart) <- c("Essential_rpart", "Nonessential_part")

gcomp_attributes_tree <- cbind(gcomp_attributes_tree, predictions_probs_rpart)
```

The J48 function of the RWeka package a more detailed tree.

```
# Decision tree J48 function RWeka
tree_rweka <- J48(essentiality.consensus ~ ., data = gcomp_attributes_tree1)

predictions <- predict(tree_rweka, type = "class")

# probabilities
tree_rweka_pred <- as.data.frame(predict(tree_rweka, type = "prob"))
colnames(tree_rweka_pred) <- c("Essential_rweka", "Nonessential_rweka")

gcomp_attributes_tree <- cbind(gcomp_attributes_tree, tree_rweka_pred)

# confusion matrix
table_tree_rweka <- as.data.frame(table(predictions, gcomp_attributes_tree1$essentiality.consensus))

# plotting
plot(tree_rweka)
```

Table 2.1: Confusion matrix of the 3 different algorithms used for the decision trees

Predicted_condition	C5.0	rpart	C4.5_RWeka
True positives	22	23	21
False negatives	124	123	125
False positives	4	5	1
True negatives	2762	2761	2765




```
pdf(file = "Figures/Decision_tree_RWeka.pdf")
plot(tree_rweka)
dev.off()
```

```
## pdf
##   2
```

The confusion matrix of the results of the 3 algorithms.

2.6 Method comparison

The *Drosophila melanogaster* network used herein contains 154 essential genes. In order evaluate each centrality index that we used as a predictor of gene essentiality we isolate the highest 200 values from each centrality. Then we label each case if it was correctly predicted as essential or not.

```
# Get data ready

essentiality_number <- drosophila_degreesA %>% filter(!essentiality.consensus == "Nonessential") %>%
    nrow()
```

```
# Degree
degree_roc <- gcomp_attributes[, c(6, 21)]
colnames(degree_roc)[1] <- "predictions"
degree_roc <- head(arrange(degree_roc, desc(predictions)), n = essentiality_number)  # top 200.
degree_roc <- degree_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Betweenness
between_roc <- gcomp_attributes[, c(7, 21)]
colnames(between_roc)[1] <- "predictions"
between_roc <- head(arrange(between_roc, desc(predictions)), n = essentiality_number)  # top 200.
between_roc <- between_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Abs weight Betweenness
w_between_roc <- gcomp_attributes[, c(14, 21)]
colnames(w_between_roc)[1] <- "predictions"
w_between_roc <- head(arrange(w_between_roc, desc(predictions)), n = essentiality_number)  # top 200.
w_between_roc <- w_between_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Closeness
close_roc <- gcomp_attributes[, c(8, 21)]
colnames(close_roc)[1] <- "predictions"
close_roc <- head(arrange(close_roc, desc(predictions)), n = essentiality_number)  # top 200.
close_roc <- close_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Information Degree Centrality
information_degree_centr_roc <- drosophila_degreesA[, c(37, 57)]
colnames(information_degree_centr_roc)[1] <- "predictions"
information_degree_centr_roc <- head(arrange(information_degree_centr_roc, desc(predictions)), n = esse
information_degree_centr_roc <- information_degree_centr_roc %>% filter(!labels == "NA")

# Information Degree
information_degree_roc <- drosophila_degreesA[, c(10, 57)]
colnames(information_degree_roc)[1] <- "predictions"
information_degree_roc <- head(arrange(information_degree_roc, desc(predictions)), n = essentiality_numb
information_degree_roc <- information_degree_roc %>% filter(!labels == "NA")

# Decision tree C50
tree_C50_roc <- gcomp_attributes_tree[, c(64, 66)]
colnames(tree_C50_roc)[1] <- "predictions"
tree_C50_roc <- head(arrange(tree_C50_roc, desc(predictions)), n = essentiality_number)  # top 200.
tree_C50_roc <- tree_C50_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Decision tree rpart
tree_rpart_roc <- gcomp_attributes_tree[, c(67, 66)]
colnames(tree_rpart_roc)[1] <- "predictions"
tree_rpart_roc <- head(arrange(tree_rpart_roc, desc(predictions)), n = essentiality_number)  # top 200.
tree_rpart_roc <- tree_rpart_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA

# Decision tree C4.5
tree_C45_roc <- gcomp_attributes_tree[, c(69, 66)]
colnames(tree_C45_roc)[1] <- "predictions"
tree_C45_roc <- head(arrange(tree_C45_roc, desc(predictions)), n = essentiality_number)  # top 200.
tree_C45_roc <- tree_C45_roc %>% filter(!labels == "NA")  #remove rows if essentiality consesus==NA
```

2.6.1 Precision Recall Curve

We will use standard methods for the evaluation of the predicted power of centralities. The Precision Recall curve is a comprehensive visualization for the trade-offs of precision as we increase recall. Precision is defined as:

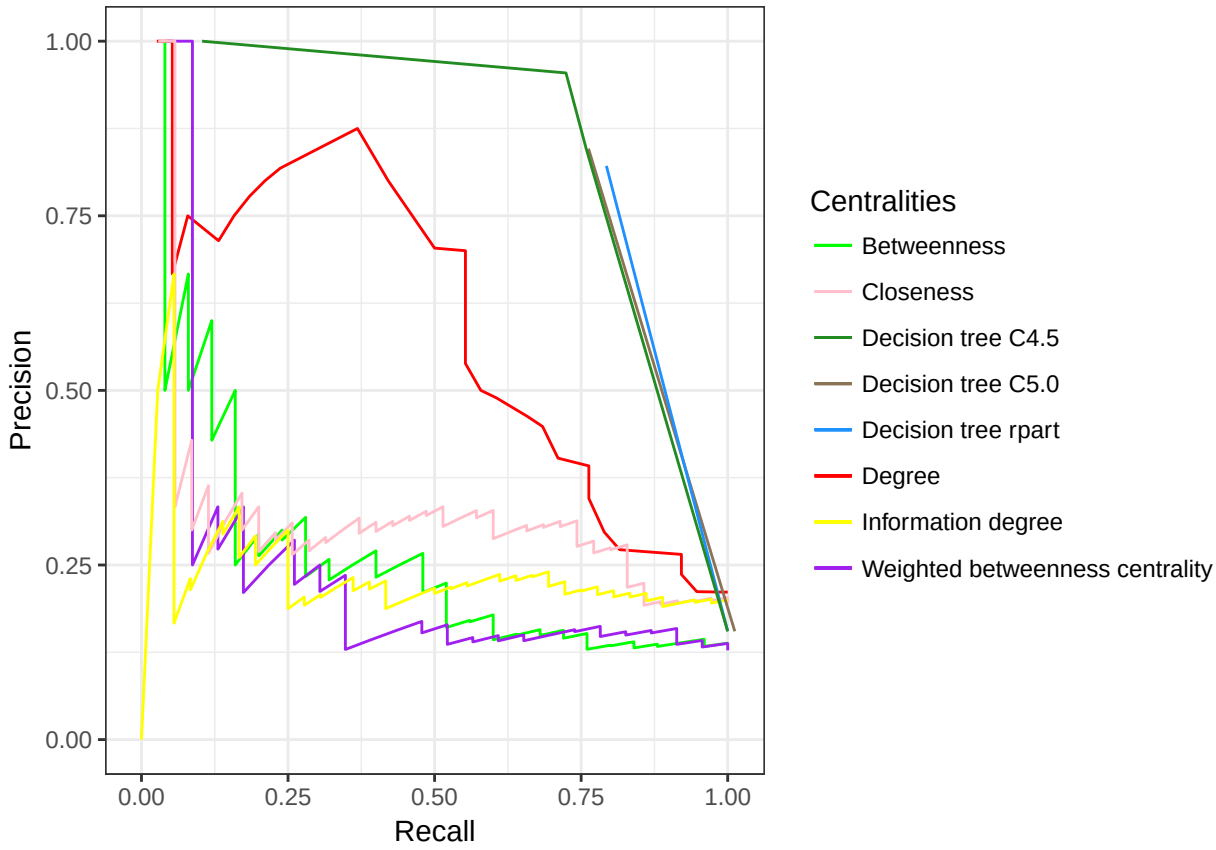
$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

It represents the fraction of the predicted cases that were correct. On the contrary recall is the fraction of predicted cases in respect to all correct cases. Recall is defined as:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

```
library(ROCR)

# Precision Recall Degree
pred_deg <- prediction(degree_roc$predictions, degree_roc$labels)
perf_deg <- performance(pred_deg, "prec", "rec")
df_roc_deg <- cbind(perf_deg@x.values[[1]], perf_deg@y.values[[1]])
df_roc_deg <- as.data.frame(df_roc_deg)

# Betweenness
pred_bet <- prediction(between_roc$predictions, between_roc$labels)
perf_bet <- performance(pred_bet, "prec", "rec")
df_roc_bet <- cbind(perf_bet@x.values[[1]], perf_bet@y.values[[1]])
df_roc_bet <- as.data.frame(df_roc_bet)

# Absolute weights betweenness
w_pred_bet <- prediction(w_between_roc$predictions, w_between_roc$labels)
w_perf_bet <- performance(w_pred_bet, "prec", "rec")
w_df_roc_bet <- cbind(w_perf_bet@x.values[[1]], w_perf_bet@y.values[[1]])
w_df_roc_bet <- as.data.frame(w_df_roc_bet)

# Closeness
pred_close <- prediction(close_roc$predictions, close_roc$labels)
perf_close <- performance(pred_close, "prec", "rec")
df_roc_close <- cbind(perf_close@x.values[[1]], perf_close@y.values[[1]])
df_roc_close <- as.data.frame(df_roc_close)

# Information degree centrality
pred_inf_cent <- prediction(information_degree_centr_roc$predictions, information_degree_centr_roc$label
perf_inf_cent <- performance(pred_inf_cent, "prec", "rec")
df_roc_inf_cent <- cbind(perf_inf_cent@x.values[[1]], perf_inf_cent@y.values[[1]])
df_roc_inf_cent <- as.data.frame(df_roc_inf_cent)

# Information degree
pred_inf_d <- prediction(information_degree_roc$predictions, information_degree_roc$labels)
perf_inf_d <- performance(pred_inf_d, "prec", "rec")
df_roc_inf_d <- cbind(perf_inf_d@x.values[[1]], perf_inf_d@y.values[[1]])
df_roc_inf_d <- as.data.frame(df_roc_inf_d)

# Decision tree C50
```

```
pred_treeC50_d <- prediction(tree_C50_roc$predictions, tree_C50_roc$labels)
perf_treeC50_d <- performance(pred_treeC50_d, "prec", "rec")
df_roc_treeC50_d <- cbind(perf_treeC50_d@x.values[[1]], perf_treeC50_d@y.values[[1]])
df_roc_treeC50_d <- as.data.frame(df_roc_treeC50_d)

# Decision tree rpart
pred_tree_rpart_d <- prediction(tree_rpart_roc$predictions, tree_rpart_roc$labels)
perf_tree_rpart_d <- performance(pred_tree_rpart_d, "prec", "rec")
df_roc_tree_rpart_d <- cbind(perf_tree_rpart_d@x.values[[1]], perf_tree_rpart_d@y.values[[1]])
df_roc_tree_rpart_d <- as.data.frame(df_roc_tree_rpart_d)

# Decision tree C50
pred_treeC45_d <- prediction(tree_C45_roc$predictions, tree_C45_roc$labels)
perf_treeC45_d <- performance(pred_treeC45_d, "prec", "rec")
df_roc_treeC45_d <- cbind(perf_treeC45_d@x.values[[1]], perf_treeC45_d@y.values[[1]])
df_roc_treeC45_d <- as.data.frame(df_roc_treeC45_d)


# Precision Recall Curve
ggplot() + geom_line(data = df_roc_bet, aes(x = V1, y = V2, color = "Betweenness")) + geom_line(data = d
    aes(x = V1, y = V2, color = "Degree")) + geom_line(data = w_df_roc_bet, aes(x = V1, y = V2, color =
    geom_line(data = df_roc_close, aes(x = V1, y = V2, color = "Closeness")) + # geom_line(data = df_ro
# centrality'),position=position_jitter(w=0.02, h=0))+
geom_line(data = df_roc_inf_d, aes(x = V1, y = V2, color = "Information degree")) + geom_line(data = df
    aes(x = V1, y = V2, color = "Decision tree C5.0"), position = position_jitter(w = 0.02, h = 0)) +
    geom_line(data = df_roc_tree_rpart_d, aes(x = V1, y = V2, color = "Decision tree rpart")) + geom_lin
    aes(x = V1, y = V2, color = "Decision tree C4.5")) + scale_colour_manual(values = c(Degree = "red",
    Betweenness = "green1", `Weighted betweenness centrality` = "purple", Closeness = "pink", `Informati
    `Information degree` = "yellow", `Decision tree C5.0` = "burlywood4", `Decision tree rpart` = "dodg
    `Decision tree C4.5` = "forestgreen"), name = "Centralities") + # ggtitle('Precision Recall Curve')
labs(x = "Recall", y = "Precision") + theme_bw()
```




```
ggsave("Centralities_Precision_Recall_Curve.pdf", plot = last_plot(), device = "pdf", dpi = 150, path =
```

2.6.2 ROC Curve

```
# ROC curve

# Degree
ROC_degree <- performance(pred_deg, "tpr", "fpr")
ROC_degree_d <- cbind(ROC_degree@x.values[[1]], ROC_degree@y.values[[1]])
ROC_degree_d <- as.data.frame(ROC_degree_d)

# Betweenness
ROC_bet <- performance(pred_bet, "tpr", "fpr")
ROC_bet_d <- cbind(ROC_bet@x.values[[1]], ROC_bet@y.values[[1]])
ROC_bet_d <- as.data.frame(ROC_bet_d)

# Abs weights betweenneess
ROC_w_bet <- performance(w_pred_bet, "tpr", "fpr")
ROC_w_bet_d <- cbind(ROC_w_bet@x.values[[1]], ROC_w_bet@y.values[[1]])
ROC_w_bet_d <- as.data.frame(ROC_w_bet_d)

# Closeness
ROC_close <- performance(pred_close, "tpr", "fpr")
ROC_close_d <- cbind(ROC_close@x.values[[1]], ROC_close@y.values[[1]])
ROC_close_d <- as.data.frame(ROC_close_d)
```

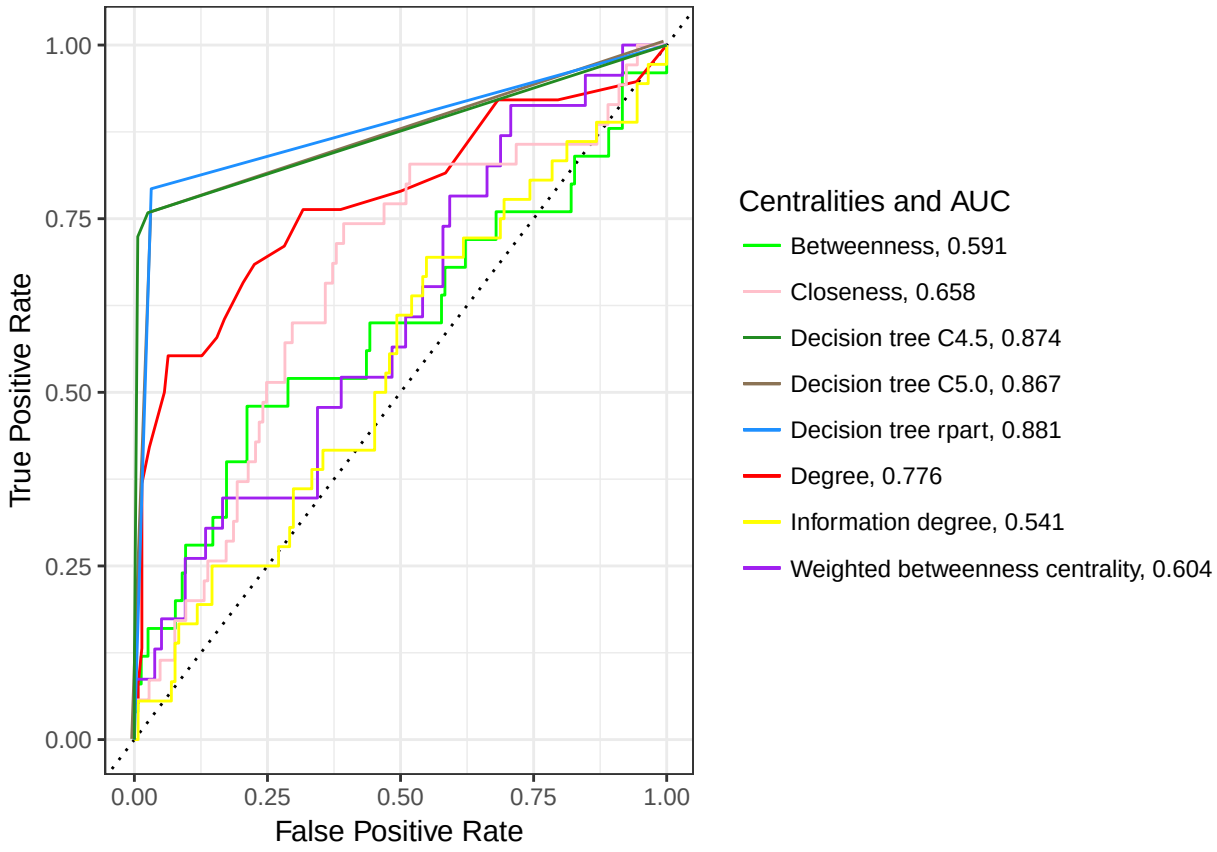
Table 2.2: AUC score of the different centralities as essentiality predictors

Centralities	AUC
Degree	0.776
Betweenneess	0.591
Weighted betweenness centrality	0.604
Closeness	0.658
Information degree	0.541
Decision tree C5.0	0.867
Decision tree rpart	0.881
Decision tree C4.5	0.874

```
# Information degree centrality
ROC_inf_cent <- performance(pred_inf_cent, "tpr", "fpr")
ROC_inf_cent_d <- cbind(ROC_inf_cent@x.values[[1]], ROC_inf_cent@y.values[[1]])
ROC_inf_cent_d <- as.data.frame(ROC_inf_cent_d)

# Information degree
ROC_inf_d <- performance(pred_inf_d, "tpr", "fpr")
ROC_inf_d_d <- cbind(ROC_inf_d@x.values[[1]], ROC_inf_d@y.values[[1]])
ROC_inf_d_d <- as.data.frame(ROC_inf_d_d)

# Decision tree C5.0
ROC_treeC50_d <- performance(pred_treeC50_d, "tpr", "fpr")
ROC_treeC50_d_d <- cbind(ROC_treeC50_d@x.values[[1]], ROC_treeC50_d@y.values[[1]])
ROC_treeC50_d_d <- as.data.frame(ROC_treeC50_d_d)

# Decision tree rpart
ROC_tree_rpart_d <- performance(pred_tree_rpart_d, "tpr", "fpr")
ROC_tree_rpart_d_d <- cbind(ROC_tree_rpart_d@x.values[[1]], ROC_tree_rpart_d@y.values[[1]])
ROC_tree_rpart_d_d <- as.data.frame(ROC_tree_rpart_d_d)

# Decision tree C4.5
ROC_treeC45_d <- performance(pred_treeC45_d, "tpr", "fpr")
ROC_treeC45_d_d <- cbind(ROC_treeC45_d@x.values[[1]], ROC_treeC45_d@y.values[[1]])
ROC_treeC45_d_d <- as.data.frame(ROC_treeC45_d_d)
```

2.6.3 AUC

Afterwards we have to calculate a specific metric like the F-measure and AUC.



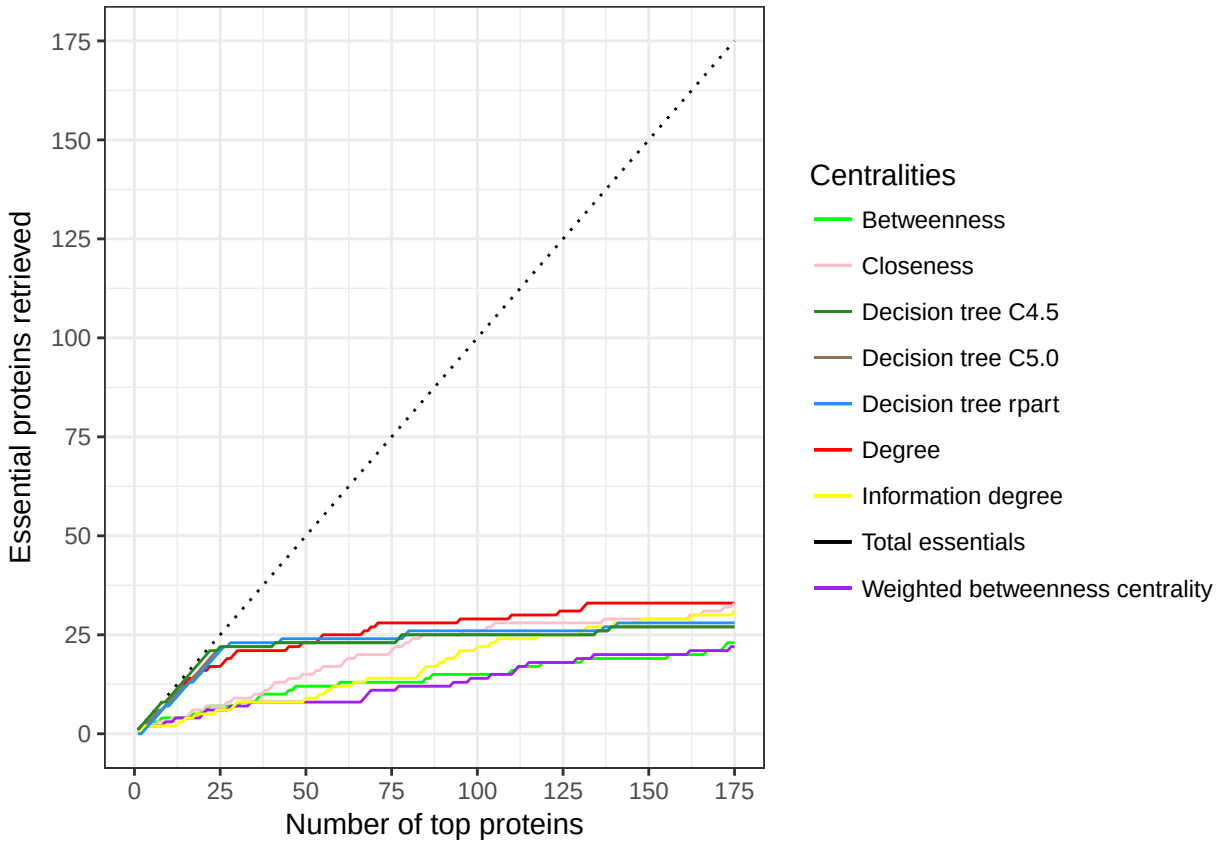

```
## Saving 6.5 x 4.5 in image
```

2.6.4 Top values of centralities

Another visual method to evaluate the centralities is to plot the number of top values of centrality against the number of essential proteins within. This is a simple and intuitive illustration of the raw predictive power of the centralities.

```
# top proteins based on centrality versus the predicted essential proteins
max_top <- 200  # has to be higher than the total essential genes

sorted_attributes_degree <- with(gcomp_attributes, gcomp_attributes[order(degree, decreasing = T), ])
sorted_attributes_betweenness <- with(gcomp_attributes, gcomp_attributes[order(betweenness, decreasing =
    ])
sorted_attributes_closeness <- with(gcomp_attributes, gcomp_attributes[order(closeness_all, decreasing =
    ])
sorted_attributes_w_abs_betweenness <- with(gcomp_attributes, gcomp_attributes[order(w_abs_betweenness,
    decreasing = T), ])
sorted_attributes_information_cent <- with(drosophila_degreesA, drosophila_degreesA[order(drosophila_deg
    37], decreasing = T), ])
sorted_attributes_information_degree <- with(drosophila_degreesA, drosophila_degreesA[order(drosophila_
    10], decreasing = T), ])

sorted_attributes_decision_tree <- with(gcomp_attributes_tree, gcomp_attributes_tree[order(gcomp_attribu
    64], decreasing = T), ])
```

```
sorted_attributes_decision_tree_rpart <- with(gcomp_attributes_tree, gcomp_attributes_tree[order(gcomp_a
    67], decreasing = T), ])

sorted_attributes_decision_tree_C45 <- with(gcomp_attributes_tree, gcomp_attributes_tree[order(gcomp_at
    69], decreasing = T), ])

top_essential <- as.data.frame(matrix(ncol = 1, nrow = max_top))
n_essentials <- length(which(!gcomp_attributes$essentiality.consensus == "Nonessential"))
top_essential[, 1] <- c(1:n_essentials, rep(n_essentials, max_top - n_essentials))

for (i in 1:max_top) {

    head_sort_degree <- head(sorted_attributes_degree, n = i)
    head_sort_bet <- head(sorted_attributes_betweenness, n = i)
    head_sort_close <- head(sorted_attributes_closeness, n = i)
    head_sort_w_abs_betweenness <- head(sorted_attributes_w_abs_betweenness, n = i)
    head_sorted_information_cent <- head(sorted_attributes_information_cent, n = i)
    head_sorted_information_degree <- head(sorted_attributes_information_degree, n = i)
    head_sorted_attributes_decision_tree <- head(sorted_attributes_decision_tree, n = i)
    head_sorted_attributes_decision_tree_rpart <- head(sorted_attributes_decision_tree_rpart, n = i)
    head_sorted_attributes_decision_tree_C45 <- head(sorted_attributes_decision_tree_C45, n = i)

    top_essential[i, 2] <- i
    top_essential[i, 3] <- length(which(head_sort_degree$essentiality.consensus == "Essential"))
    top_essential[i, 4] <- length(which(head_sort_bet$essentiality.consensus == "Essential"))
    top_essential[i, 5] <- length(which(head_sort_close$essentiality.consensus == "Essential"))
    top_essential[i, 6] <- length(which(head_sort_w_abs_betweenness$essentiality.consensus == "Essentia
    top_essential[i, 7] <- length(which(head_sorted_information_cent$essentiality.consensus == "Essenti
    top_essential[i, 8] <- length(which(head_sorted_information_degree$essentiality.consensus == "Essen
    top_essential[i, 9] <- length(which(head_sorted_attributes_decision_tree$essentiality.consensus ==
        "Essential"))
    top_essential[i, 10] <- length(which(head_sorted_attributes_decision_tree_rpart$essentiality.consens
        "Essential"))
    top_essential[i, 11] <- length(which(head_sorted_attributes_decision_tree_C45$essentiality.consensus
        "Essential"))


}

colnames(top_essential) <- c("all_essentials", "top_number", "essentials_degree", "essentials_betweennes
    "essentials_closeness", "abs_weights_betweenness", "information_degree_centrality", "information_deg
    "decision_tree_C50", "decision_tree_rpart", "decision_tree_C45")
```



Let's plot them.

```
## Saving 6.5 x 4.5 in image
```

Chapter 3

Network Decompositions

3.1 Subnetworks based on essentiality

We can extract different sub networks based on the essentiality consensus of the nodes. We explored the following instances:

1. essential nodes and their neighbors
2. only essential nodes
3. only links between essential nodes

3.1.1 Essential Nodes and Their Neighbors

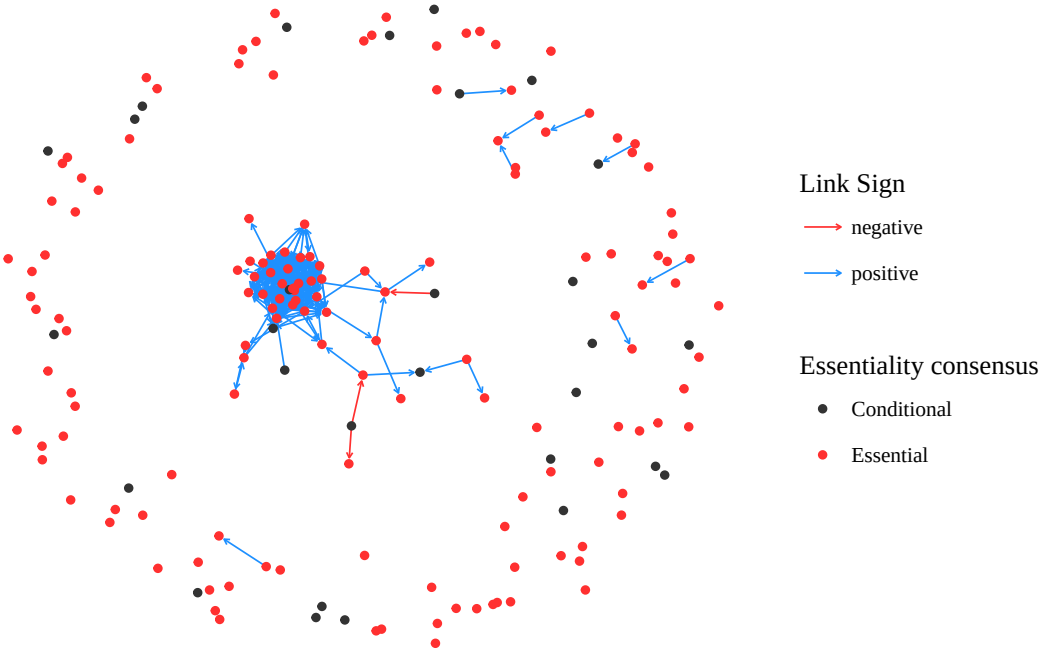
```
# Only essential nodes and their neighbors
essential_genes_net <- subset(dros_net, ID1_consensus == "Essential" | ID1_consensus == "Conditional" |
    ID2_consensus == "Essential" | ID2_consensus == "Conditional")
g_essential_non <- graph_from_data_frame(essential_genes_net, directed = T)

g_essential_non_giant <- decompose.graph(g_essential_non, min.vertices = 10)

g_essential_non_giant <- g_essential_non_giant[[1]]

# layg1 <- layout_nicely(g_essential_non_giant) plot(g_essential_non_giant, layout=layg1,
# vertex.shape='circle', vertex.size=0.295, vertex.color=V(g_essential_non)$color,
# vertex.frame.color=V(g_essential_non)$color, vertex.label=NA, #vertex.label.cex =
# vcount(g)*0.00004, #vertex.label.color ='black', edge.color=E(g_essential_non)$color,
# edge.width=0.53, edge.arrow.size = 0.31, edge.arrow.width = 0.5, #edge.label=round(x =
# E(g)$weight,digits = 3), #edge.label.cex = 0.3, #edge.label.color = E(g)$color, margin = 0, main =
# '', sub = '') title(main = paste('Only essential nodes and their neighbors'), cex.main= 1.5,
# cex.sub = 1.2,outer = F)
```

```
ggraph_gcomp <- ggraph(gcomp, layout = "kk") + geom_edge_link(aes(colour = sign_score_sign)) + geom_nod
    theme_graph()
```

3.1.2 Essential Nodes Links

```
# only essential nodes
g_essential <- induced.subgraph(gcomp, which(V(gcomp)$essentiality.consensus == "Essential" | V(gcomp)$
    "Conditional"))

# layg2 <- layout_nicely(g_essential) plot(g_essential, layout=layg2, vertex.shape='circle',
# vertex.size=0.295, vertex.color=V(g_essential)$color, vertex.frame.color=V(g_essential)$color,
# vertex.label=NA, #vertex.label.cex = vcount(g)*0.00004, #vertex.label.color ='black',
# edge.color=E(g_essential)$color, edge.width=0.53, edge.arrow.size = 0.31, edge.arrow.width = 0.5,
# #edge.label=round(x = E(g)$weight,digits = 3), #edge.label.cex = 0.3, #edge.label.color =
# E(g)$color, margin = 0, main = '', sub = '') title(main = paste('Only essential nodes and their
# neighbors'), cex.main= 1.5, cex.sub = 1.2,outer = F)
```

```
# extrafont
ggraph(g_essential, layout = "kk") + geom_edge_link(edge_width = 0.3, arrow = arrow(length = unit(0.8,
    "mm")), end_cap = circle(0.8, "mm"), aes(colour = sign_score_sign)) + geom_node_point(size = 1, aes
    scale_edge_color_manual(values = c(positive = "dodgerblue", negative = "firebrick1"), name = "Link S
    scale_color_manual(values = c(Essential = "firebrick1", Conditional = "gray20"), name = "Essentiali
    # ggtitle('Subnetwork of the essential proteins of Drosophila')+
theme_graph() + theme(text = element_text(size = 10, family = "Times")) + theme(plot.title = element_tex
    vjust = 1, family = "Times"))
```



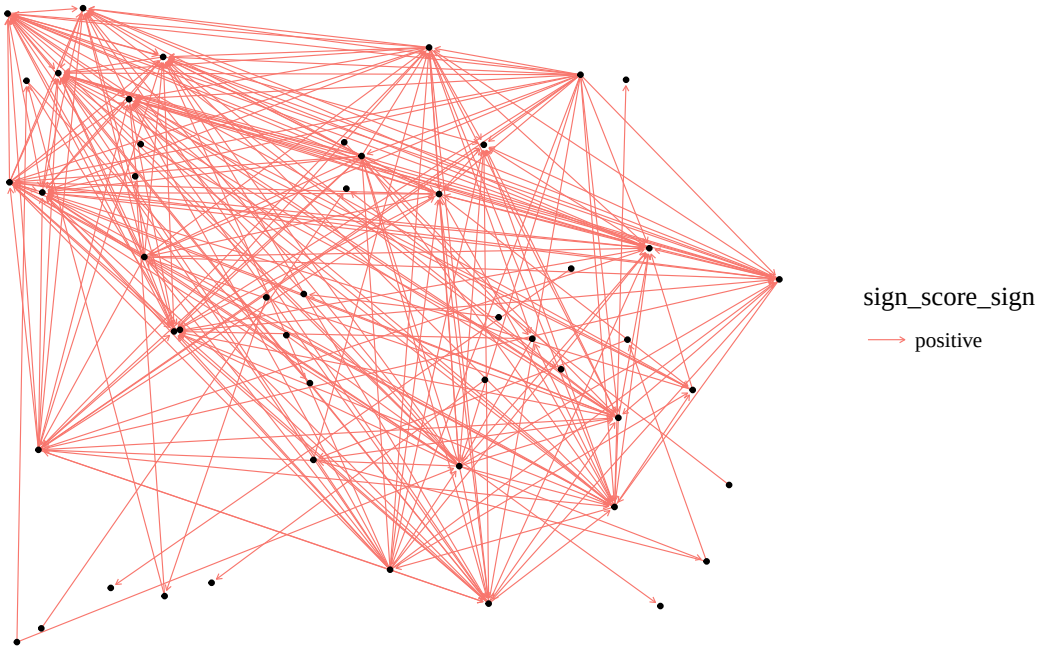

```
ggsave("Subnetwork_Essential_Proteins.pdf", plot = last_plot(), device = "pdf", dpi = 150, path = "Figu
# ggraph_g_essential + facet_nodes(~essentiality.consensus)
```

3.1.3 Essential to Essential

```
# Only links between essential nodes
ONLYessential_genes_net <- subset(dros_net, ID1_consensus == "Essential" & ID2_consensus == "Essential"
```

```
write.table(ONLYessential_genes_net, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/ONL
    sep = ",", col.names = T, row.names = F, quote = F)

g_ONLY_essential <- graph_from_data_frame(ONLYessential_genes_net, directed = T)

# layg3 <- layout_nicely(g_ONLY_essential) plot(g_ONLY_essential, layout=layg3,
# vertex.shape='circle', vertex.size=0.295, vertex.color=V(g_ONLY_essential)$color,
# vertex.frame.color=V(g_ONLY_essential)$color, vertex.label=NA, #vertex.label.cex =
# vcount(g)*0.00004, #vertex.label.color ='black', edge.color=E(g_ONLY_essential)$color,
# edge.width=0.53, edge.arrow.size = 0.31, edge.arrow.width = 0.5, #edge.label=round(x =
# E(g)$weight,digits = 3), #edge.label.cex = 0.3, #edge.label.color = E(g)$color, margin = 0, main =
# '', sub = '') title(main = paste('Only links between essential nodes'), cex.main= 1.5, cex.sub =
# 1.2,outer = F)
```

```
ggraph(g_ONLY_essential, layout = "randomly") + geom_edge_link(edge_width = 0.2, arrow = arrow(length =
    "mm")), end_cap = circle(0.8, "mm"), aes(colour = sign_score_sign)) + geom_node_point(size = 0.5) +
    # ggtitle('Links between essential proteins of Drosophila')+
theme_graph() + theme(text = element_text(size = 10, family = "Times")) + theme(plot.title = element_tex
    vjust = 1, family = "Times"))
```




```
ggsave("Subnetwork_Essential_Proteins_Links.pdf", plot = last_plot(), device = "pdf", dpi = 150, path =
```

```
## Saving 6.5 x 4.5 in image
```

The essential nodes form a sort of clique with positive links. The clique has been mentioned before

> In addition, as reported by (Zotenko et al. 2008), essential proteins tend to form highly connected clusters rather than function independently (Li et al. 2012).

Also look for autocatalytic sets in the complete network and compare with the essential genes subgraph.

> An ACS is a subgraph, each of whose nodes has at least one incoming link from a node belonging to the same subgraph (S. A. Kauffman 1993). In the organized state, all the species except possibly the one being picked for replacement are part of the ACS (Nghe et al. 2015). The ACS consists of a core and a periphery. The core comprises the set of nodes (along with their mutual

> links) from which there is a directed path to every other node in the ACS (Mehrotra, Soni, and Jain 2009).

This result maybe has an evolutionary trait. Something like growth and regulation, cooperation and regulation.

In (Sole 2011):

> Cooperation has been suggested as an essential step toward the emergence of complex, self-organized chemical systems (Eigen 1971)

3.2 Network Frobenius Decomposition of essential cluster

In order to study the inner structure of the essential cliques we will examine the matrices. Are they irreducible?

```
library(popdemo)
is_connected(g_ONLY_essential)
```

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

```
g_ONLY_essential_giant <- decompose.graph(g_ONLY_essential, min.vertices = 10)  #upografima tis megalis

g_ONLY_essential_giant <- g_ONLY_essential_giant[[1]]
essential_adjacency <- get.adjacency(graph = g_ONLY_essential_giant, names = T)
essential_adjacency <- as.matrix(essential_adjacency)

write.table(V(g_ONLY_essential_giant)$name, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tab
    sep = ",", col.names = FALSE, row.names = FALSE)

is.matrix_irreducible(essential_adjacency)
```

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

```
is.matrix_ergodic(essential_adjacency)
```

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

```
reducible_blocks <- blockmatrix(essential_adjacency)  # changes the order of the columns/rows with the

det_re <- det(reducible_blocks$blockmatrix)

svd_matrix <- svd(reducible_blocks$blockmatrix)
```

The definition of irreducibility from (Gradshteyn and Ryzhik 2007):

Definition missing...

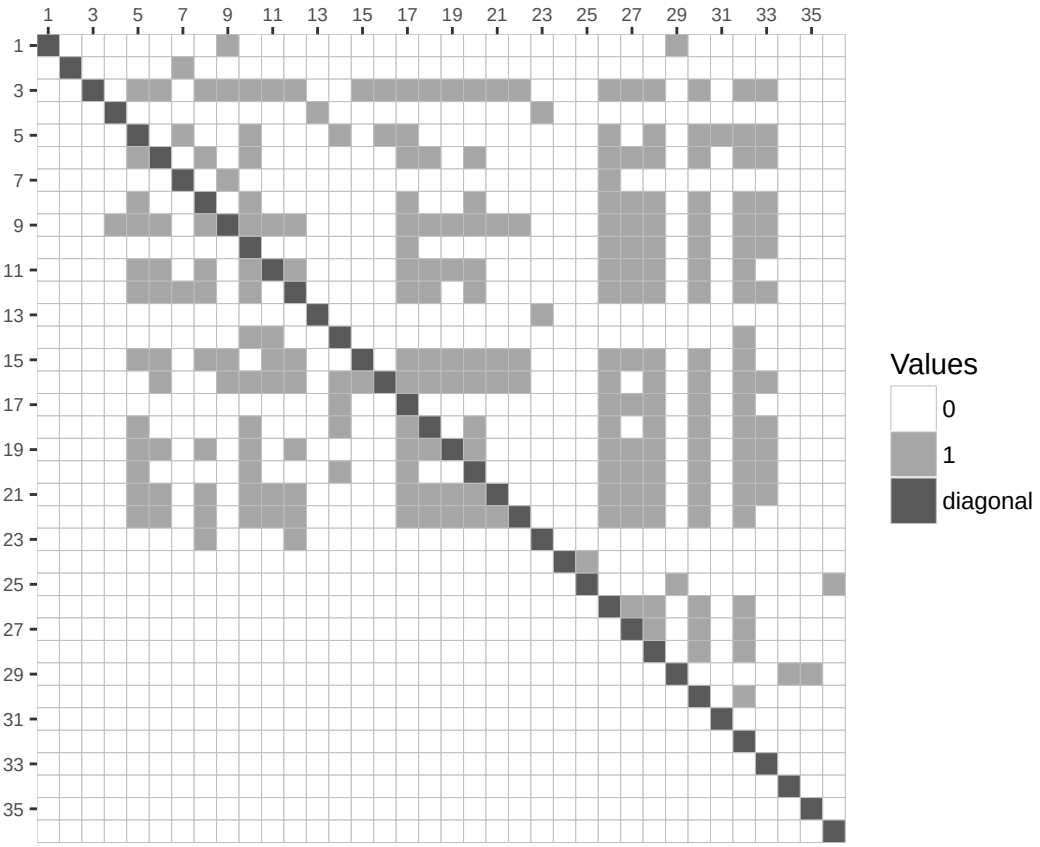
Our matrix is reducible and non-ergodic. But how to find the disjoint sets?

Authors of (Stott et al. 2010) propose that matrices should be defined as irreducible, reducible–ergodic or reducible–non-ergodic and that this information be used to guide analysis.

The reducible matrix of the essential proteins subnetwork.

```
#heatmap.2(reducible_blocks$blockmatrix,dendrogram='none',Rowv=F,Colv = F,trace='none') #Simple and fas

reducible_essentials <- melt(reducible_blocks$blockmatrix) # Var1 is the rows of the blockmatrix and Va

#library(ggthemes)
```

```
x2 <- reducible_essentials
x2$value <-  with(x2, ifelse(Var1==Var2,"diagonal",value)) # insert diagonal information in order to vi

ggplot() + #factor in order to assign a color to the value
  geom_tile(data = x2 , aes(x=Var2, y=Var1, fill=factor(value)),color="gray", size=0.1)+
  #ggtitle("Essential proteins reducible matrix")+
  labs(x=NULL, y=NULL)+
  scale_fill_manual("Values",values = c("0"="white","1"="gray65","diagonal"="gray35"))+
  scale_x_continuous(position = "top", breaks = seq(1,36,2),expand=c(0,0))+
  scale_y_continuous(trans = "reverse", breaks = seq(1,36,2),expand=c(0,0))+
  coord_fixed()+
  coord_equal()+
  theme(axis.text=element_text(size=7,hjust = 0.5))+
  theme(plot.title = element_text(hjust = 0.5))
```




```
ggsave("Essential_proteins_reducible_matrix.pdf", plot = last_plot(), device = "pdf", dpi = 150,path =
```

```
## Saving 6.5 x 4.5 in image
```

The order of columns and rows has been changed with the following: 26, 19, 25, 1, 3, 4, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 20, 21, 22, 24, 27, 29, 30, 2, 23, 28, 5, 13, 18, 10, 31, 32, 33, 34, 35, 36.

Spectral radius of the matrix and eigenvectors.

```
# perron frobenius decomposition
library(sparsevar)

ss <- frobNorm(essential_adjacency)
essential_matrix_spectral_radius <- spectralRadius(essential_adjacency)  # max eigenvalue.
```

3.2.1 Strongly connected components

Two vertices belong to the same strong component if and only if the are reachable from each other via directed paths. In this implementation we used the Tarjan's algorithm, explained in (Knuth 1994), for finding the strong components from the R package RBGL (RBGL: R interface to boost graph library).

If a graph is irreducible then it is strongly connected and vice versa.
Theorem 3.1 (Irreducible and strongly connected)**.** *An n x n complex matrix A is irreducible if and only if its directed graph G(A) is strongly connected.*

3.2.2 Strongly connected component of the subnetwork

```
# Strongly connected components

# http://math.stackexchange.com/questions/350994/prove-that-a-strongly-connected-digraph-has-an-irreduc

library(RBGL)

essential_nodes_graph_NEL <- as_graphnel(g_ONLY_essential_giant)

strong_comp_essential_nodes <- strongComp(essential_nodes_graph_NEL)

strong_comp_essential_nodes_sizes <- as.data.frame(sapply(X = strong_comp_essential_nodes, FUN = length
# 8 has 20 nodes
```

The subnetwork of the edges between the essential proteins has only one strongly connected component with 20 proteins. These are FBgn0010217, FBgn0028694, FBgn0028686, FBgn0015283, FBgn0028687, FBgn0015282, FBgn0028695, FBgn0026380, FBgn0028685, FBgn0011211, FBgn0028693, FBgn0010590, FBgn0002787, FBgn0029134, FBgn0028688, FBgn0028684, FBgn0028692, FBgn0023174, FBgn0016697, FBgn0020235.

```
strongly_conn_essential_proteins <- strong_comp_essential_nodes[[8]]

write.table(strong_comp_essential_nodes[[8]], file = "../Thesis_Essentiality_Drosophila_Signed_Network/
    sep = ",", col.names = FALSE, row.names = FALSE)

graph_strongly_conn_essential_proteins <- induced_subgraph(g_ONLY_essential_giant, vids = strongly_conn

adjacency_graph_strongly_conn_essential_proteins <- get.adjacency(graph = graph_strongly_conn_essential
    names = T)
adjacency_graph_strongly_conn_essential_proteins <- as.matrix(adjacency_graph_strongly_conn_essential_p

is.matrix_irreducible(adjacency_graph_strongly_conn_essential_proteins)
```

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

```
is.matrix_ergodic(adjacency_graph_strongly_conn_essential_proteins)
```

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

The strongly connected component is irreducible as the theorem states.

The second Frobenious theorem. Since the max abs eigenvalue is 1 we cannot decompose the graph.

```
ee <- eigen(adjacency_graph_strongly_conn_essential_proteins)

ee_ss <- as.data.frame(ee$values)
colnames(ee_ss) <- "values"
ee_ss$modulus <- Mod(ee_ss$values)
ee_ss$real <- Re(ee_ss$values)

ee_ss$mod_duplicates <- duplicated(ee_ss$modulus) | duplicated(ee_ss$modulus, fromLast = TRUE)
ee_ss$real_duplicates <- duplicated(ee_ss$real) | duplicated(ee_ss$real, fromLast = TRUE)

ee_ss$duplicates <- duplicated(ee_ss$values, fromlast = F) | duplicated(ee_ss$values, fromlast = T)
```

Create the network of components.

```
# # From graph object to edgelist format.
Func_essential_component_strong_components <- strong_comp_essential_nodes

essential_edgelist_names <- as.data.frame(get.edgelist(g_ONLY_essential_giant))
essential_edge_attributes <- as.data.frame(igraph::get.edge.attribute(graph = g_ONLY_essential_giant,
    index = E(g_ONLY_essential_giant)))

essential_edgelist_ok <- cbind(essential_edgelist_names, essential_edge_attributes)

# strong connected components to dataframe best solution for list to dataframe !!!

essential_strongly_connected_components_dataframe <- data.frame(Component = rep(names(Func_essential_co
    lapply(Func_essential_component_strong_components, length)), Vertex = unlist(Func_essential_componer

# Sizes of strongly connected components
essential_strongly_connected_components_dataframe <- essential_strongly_connected_components_dataframe %
    group_by(Component) %>% mutate(Size = n()) %>% dplyr::select(Vertex, Component, Size)

essential_strongly_connected_components_dataframe_size <- essential_strongly_connected_components_dataf
    group_by(Component, Size) %>% distinct()

# Summarised components : Number of components with x vertices
strongly_connected_components_dataframe_SUMMARIZED <- essential_strongly_connected_components_dataframe
    group_by(Size, Component) %>% distinct() %>% group_by(Size) %>% summarise(`Number of components` = n

# Merge edgelist with the strongly connected components IDs
essential_edgelist_components <- left_join(x = essential_edgelist_ok, y = essential_strongly_connected_
    by = c(V1 = "Vertex"))
```

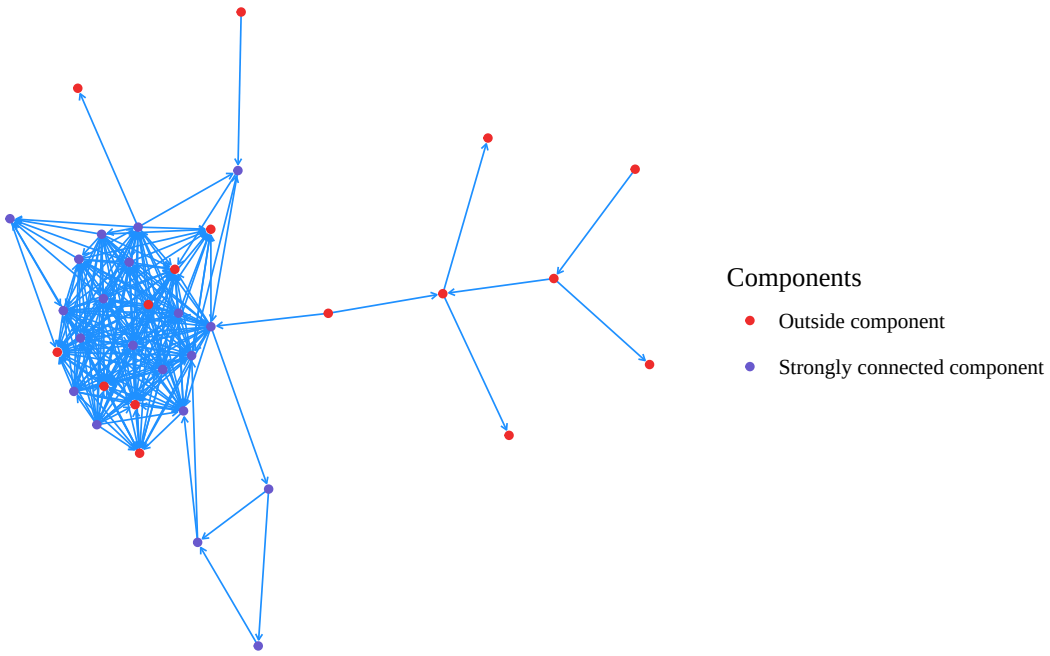
```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
essential_edgelist_components <- left_join(x = essential_edgelist_components, y = essential_strongly_co
    by = c(V2 = "Vertex"))
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
# Summarise edges between components
essential_edgelist_components_SUMMARIZED <- essential_edgelist_components %>% group_by(Component.x, Siz
    Component.y, Size.y) %>% summarise(count = n()) %>% dplyr::select(Component.x, Component.y, Size.x,
    Size.y, count)
```

```
write.table(x = essential_edgelist_components_SUMMARIZED, file = "../Thesis_Essentiality_Drosophila_Sig
    sep = ",", col.names = TRUE, row.names = FALSE)

g_essential_components <- graph_from_data_frame(essential_edgelist_components_SUMMARIZED, directed = T,
    vertices = essential_strongly_connected_components_dataframe_size)

g_essential_components <- igraph::simplify(g_essential_components, remove.loops = T, remove.multiple = 

g_ONLY_essential_giant <- graph_from_data_frame(essential_edgelist_components, directed = T, vertices =

V(g_ONLY_essential_giant)$In_Degree <- igraph::degree(g_ONLY_essential_giant, mode = "in")
V(g_ONLY_essential_giant)$Color <- ifelse(V(g_ONLY_essential_giant)$Component != 8, "Outside component"
    "Strongly connected component")

V(g_essential_components)$In_Degree <- igraph::degree(g_essential_components, mode = "in")
V(g_essential_components)$Out_Degree <- igraph::degree(g_essential_components, mode = "out")
V(g_essential_components)$color <- with(essential_strongly_connected_components_dataframe_size, ifelse(S
    1, "firebrick2", "slateblue3"))
```

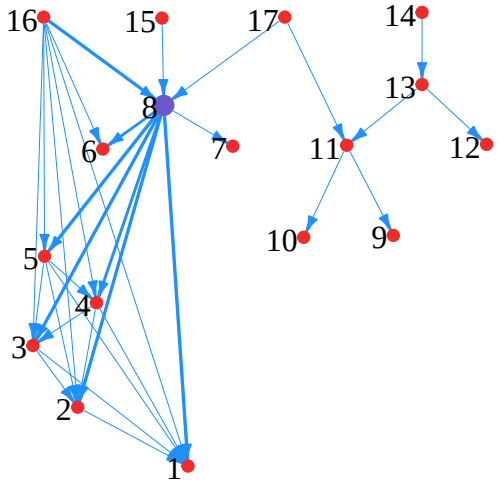



```
## Saving 6.5 x 4.5 in image
```

```
## pdf
##   2
```

Table 3.1: Counts of strongly connected components in the whole Drosophila network with the same number of vertices

Vetrices in the components	Number of components
1	2445
3	1
9	1
601	1



3.3 Strongly connected components of the whole network

Strongly connected components in the whole drosophila network.

```
# all drosophila network
giant_component_strong_components <- as_graphnel(gcomp)

giant_component_strong_components <- strongComp(giant_component_strong_components)

dd_strong <- as.data.frame(sapply(X = giant_component_strong_components, FUN = length))
colnames(dd_strong) <- "vertices_in_components"

dd_strong_counts <- dd_strong %>% group_by(vertices_in_components) %>% summarise(count_components = n()

write.table(giant_component_strong_components[[882]], file = "../Thesis_Essentiality_Drosophila_Signed_
    sep = ",", col.names = FALSE, row.names = FALSE)
```

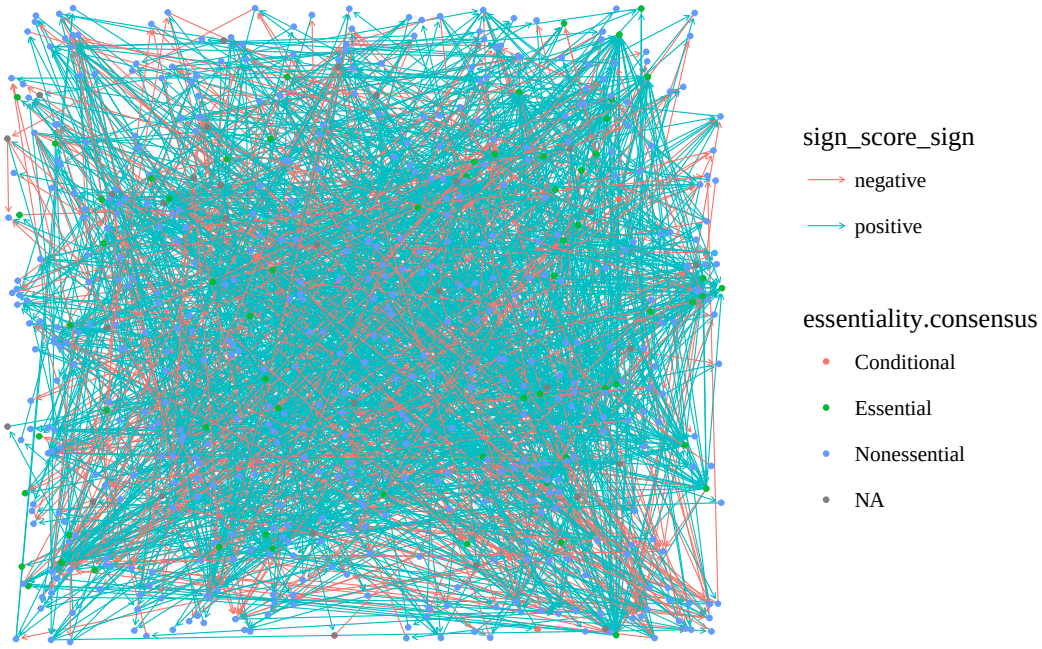
There is one big strongly connected component in drosophila's network.

The subnetwork of the strongly connected component.

```
subnet_stronly_con <- induced_subgraph(gcomp, vids = giant_component_strong_components[[882]])


# layg4 <- layout_nicely(subnet_stronly_con) plot(subnet_stronly_con, layout=layg4,
# vertex.shape='circle', vertex.size=0.295, vertex.color=V(subnet_stronly_con)$color,
# vertex.frame.color=V(subnet_stronly_con)$color, vertex.label=NA, #vertex.label.cex =
```

```
# vcount(g)*0.00004, #vertex.label.color ='black', edge.color=E(subnet_stronly_con)$color,
# edge.width=0.53, edge.arrow.size = 0.31, edge.arrow.width = 0.5, #edge.label=round(x =
# E(g)$weight,digits = 3), #edge.label.cex = 0.3, #edge.label.color = E(g)$color, margin = 0, main =
# '', sub = '') title(main = paste('The largest strongly connected component of Drosophila signed
# network'), cex.main= 1.5, cex.sub = 1.2,outer = F)
```

```
ggraph(subnet_stronly_con, layout = "randomly") + geom_edge_link(edge_width = 0.2, arrow = arrow(length
    "mm")), end_cap = circle(0.8, "mm"), aes(colour = sign_score_sign)) + geom_node_point(size = 0.5,
    aes(color = essentiality.consensus)) + # ggtitle('The irreducible component of the drosophila signe
theme_graph() + theme(text = element_text(size = 10, family = "Times")) + theme(plot.title = element_te
    vjust = 1, family = "Times"))
```




```
ggsave("Subnetwork_Irreducible_Component_All_Drosophila_Network.pdf", plot = last_plot(), device = "pdf
    dpi = 150, path = "Figures/")
```

The second Frobenius theorem. For maximum eigenvalues. There is one maximum eigenvalue as before. So this analysis is done. No further decomposition is applicable according to the theorem.

```
adjacency_subnet_stronly_con <- get.adjacency(graph = subnet_stronly_con, names = T)
adjacency_subnet_stronly_con <- as.matrix(adjacency_subnet_stronly_con)

eigen_adjacency_subnet_stronly_con <- eigen(adjacency_subnet_stronly_con)

eigen_adjacency_subnet_stronly_con_ss <- as.data.frame(eigen_adjacency_subnet_stronly_con$values)
colnames(eigen_adjacency_subnet_stronly_con_ss) <- "values"
eigen_adjacency_subnet_stronly_con_ss$modulus <- Mod(eigen_adjacency_subnet_stronly_con_ss$values)

eigen_adjacency_subnet_stronly_con_ss$mod_duplicates <- duplicated(eigen_adjacency_subnet_stronly_con_ss
    duplicated(eigen_adjacency_subnet_stronly_con_ss$modulus, fromLast = TRUE)

eigen_adjacency_subnet_stronly_con_ss$duplicates <- duplicated(eigen_adjacency_subnet_stronly_con_ss$val
    fromlast = F) | duplicated(eigen_adjacency_subnet_stronly_con_ss$values, fromlast = T)
```

Attributes of protein in the strongly connected component.

```
# vetrices
protein_attr_stronlgt_con_comp <- sapply(igraph::list.vertex.attributes(subnet_stronly_con), function(x)
    x))

protein_attr_stronlgt_con_comp <- as.data.frame(protein_attr_stronlgt_con_comp)

protein_attr_stronlgt_con_comp_essential <- protein_attr_stronlgt_con_comp %>% group_by(essentiality.co
    summarise(count = n())

# edges

edges_attr_strongly_con_comp <- edge_attr(subnet_stronly_con)
edges_attr_strongly_con_comp <- as.data.frame(edges_attr_strongly_con_comp, sep = "\t")

edges_attr_strongly_con_comp_signs <- edges_attr_strongly_con_comp %>% group_by(sign_weights) %>% summa
```

3.3.1 Network decomposition with strongly connected components

```
# all drosophila network

# requires igraph, RGBL, dplyr input a network, igraph object Func_giant_component_strong_components
# <- as_graphnel(gcomp) # slow function # The components Tarjan's algorithm
# Func_giant_component_strong_components <- RBGL::strongComp(Func_giant_component_strong_components)
# # From graph object to edgelist format.
Func_giant_component_strong_components <- giant_component_strong_components

gcomp_edgelist_names <- as.data.frame(get.edgelist(gcomp))
gcomp_edge_attributes <- as.data.frame(igraph::get.edge.attribute(graph = gcomp, index = E(gcomp)))

gcomp_edgelist_ok <- cbind(gcomp_edgelist_names, gcomp_edge_attributes)

# strong connected components to dataframe best solution for list to dataframe !!!

strongly_connected_components_dataframe <- data.frame(Component = rep(names(Func_giant_component_strong
    lapply(Func_giant_component_strong_components, length)), Vertex = unlist(Func_giant_component_strong

# Sizes of strongly connected components
strongly_connected_components_dataframe <- strongly_connected_components_dataframe %>% group_by(Compone
    mutate(Size = n())

strongly_connected_components_ONLYdataframe <- strongly_connected_components_dataframe %>% group_by(Com
    Size) %>% dplyr::distinct()

# Summarised components : Number of components with x vertices
strongly_connected_components_dataframe_SUMMARIZED <- strongly_connected_components_dataframe %>% group
    Component) %>% distinct() %>% group_by(Size) %>% summarise(`Number of components` = n())

# Merge edgelist with the strongly connected components IDs
gcomp_edgelist_components <- left_join(x = gcomp_edgelist_ok, y = strongly_connected_components_datafra
    by = c(V1 = "Vertex"))
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
```

```
## factors with different levels, coercing to character vector
```

```
gcomp_edgelist_components <- left_join(x = gcomp_edgelist_components, y = strongly_connected_components_
    by = c(V2 = "Vertex"))
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
# Summarise edges between components
gcomp_edgelist_components_SUMMARIZED <- gcomp_edgelist_components %>% group_by(Component.x, Size.x, Comp
    Size.y) %>% summarise(count = n())

write.table(x = gcomp_edgelist_components_SUMMARIZED, file = "../Thesis_Essentiality_Drosophila_Signed_I
    sep = ",", col.names = TRUE, row.names = FALSE)
```

3.3.2 Cycles and Motifs

```
library(sna)

cycle_test <- kcycle.census(dat = adjacency_graph_strongly_conn_essential_proteins, maxlen = 20, mode =
    cycle.comembership = "bylength", tabulate.by.vertex = FALSE)

mmm <- count_motifs(graph_strongly_conn_essential_proteins, size = 3)
```

3.3.3 The Perron - Frobenius decomposition

The blocks after the permutation with the blockmatrix function of the popdemo package weren't irreducible. So there is something wrong..

```
# Are blocks across the diagonal irreducible?

ddd <- reducible_blocks$blockmatrix[c(5:13), c(5:13)]
is.matrix_irreducible(ddd)
```

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

```
dddd <- blockmatrix(ddd)
sss <- dddd$blockmatrix[c(1:7), c(1:7)]
is.matrix_irreducible(sss)
```

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

```
####
frob_IDS <- as.data.frame(x = rownames(essential_adjacency))
colnames(frob_IDS)[1] <- "IDS"
frob_IDS$original_index <- seq(1, nrow(frob_IDS), by = 1)
frob_IDS$permuted_index <- reducible_blocks$order

frob_IDS$frob_decomposition <- with(frob_IDS, ifelse(permuted_index < 24, 1, ifelse(permuted_index ==
    24, 2, 3)))  # the blocks are : 1-23, 24, 25-36

reducible_essentials_edgelist <- left_join(x = reducible_essentials, y = frob_IDS, by = c(Var1 = "permu

reducible_essentials_edgelist <- left_join(x = reducible_essentials_edgelist, y = frob_IDS, by = c(Var2
```

Table 3.2: Counts of edges in and between frobenius blocks

From Frobenius block	To Frobenius block	Edges count
1	1	135
1	3	93
2	3	1
3	3	14

```
reducible_essentials_edgelist2 <- reducible_essentials_edgelist %>% filter(value > 0)

frob_block_edgelist <- reducible_essentials_edgelist2 %>% group_by(frob_decomposition.x, frob_decomposi
    summarise(count = n())  # count the links in and between the blocks!!
```

3.3.4 Functional annotation with Panther

Functional information of the proteins inside the cluster. Is there any biological explaination for the positive edges and the structure of the cluster as represented from the Frobenius decomposition.

```
# from the panther database.

panther_annotation <- read.delim("Data/pantherEssentialProtein_cluster.txt", header = F, sep = "\t")

frob_IDS <- left_join(frob_IDS, panther_annotation, by = c(IDS = "V2"))

# go term finder
```

3.3.5 Decomposition of essentials and neighbours

For the essential proteins and their neighbours.

```
adjacency_g_essential_non_giant <- get.adjacency(graph = g_essential_non_giant, names = T)

adjacency_g_essential_non_giant <- as.matrix(adjacency_g_essential_non_giant)

ddd <- melt(adjacency_g_essential_non_giant)

dddcast <- dcast(ddd, Var1 ~ Var2, value.var = "value")
dddcast <- tibble::column_to_rownames(dddcast, var = "Var1")

############################

essential_neighbors <- dros_net %>% filter(ID1_consensus == "Essential" | ID1_consensus == "Conditional
    ID2_consensus == "Essential" | ID2_consensus == "Conditional") %>% mutate(adjacency_value = 1)
sss <- essential_neighbors[, c(1, 2)]
essential_neighbors_graph <- graph_from_data_frame(sss, directed = T)

is_connected(essential_neighbors_graph)
essential_neighbors_graph_deg <- decompose.graph(essential_neighbors_graph, min.vertices = 10)

essential_neighbors_graph_giant <- essential_neighbors_graph_deg[[1]]
```

```
essential_neighbors_adja_igraph <- as_adjacency_matrix(essential_neighbors_graph_giant, names = T, spars
    attr = NULL)

essential_neighbors_adja_igraph <- as.matrix(essential_neighbors_adja_igraph)

# adjacency_g_essential_non_giant <- get.adjacency(graph = g_essential_non_giant,names = T)

sum(colSums(essential_neighbors_adja_igraph))
is.matrix_irreducible(essential_neighbors_adja_igraph)
is.matrix_ergodic(essential_neighbors_adja_igraph)
eee <- blockmatrix(essential_neighbors_adja_igraph)

##################################

g_negative <- subgraph.edges(gcomp, eids = which(E(gcomp)$weights > 0))

g_negative_deg <- decompose.graph(g_negative, min.vertices = 10)

g_negative_deg <- g_negative_deg[[1]]

g_negative_deg_adja <- as_adjacency_matrix(g_negative_deg, names = T, sparse = F, attr = NULL)

g_negative_deg_adja <- as.matrix(g_negative_deg_adja)
is.matrix_irreducible(g_negative_deg_adja)

###############################
essential_neighbors_adjacency <- dcast(essential_neighbors, ID1 ~ ID2, value.var = "adjacency_value")
essential_neighbors_adjacency <- tibble::column_to_rownames(essential_neighbors_adjacency, var = "ID1")

essential_neighbors_adjacency[is.na(essential_neighbors_adjacency)] <- 0

sum(colSums(essential_neighbors_adjacency)) == nrow(essential_neighbors)  # check if the number of edge

# is.matrix_irreducible(as.matrix(essential_neighbors_adjacency))

ss <- ddd %>% filter(value < 0)  # no negative values..

sum(is.na(adjacency_g_essential_non_giant))  # no NA

# Erdos renyi network is irreducible!
ssss <- erdos.renyi.game(1520, p.or.m = 8e-04, directed = T)  # without the direction
es <- decompose.graph(ssss, min.vertices = 10)
esd <- es[[1]]
sss <- as.matrix(get.adjacency(esd, names = T))
# is.matrix_irreducible(sss)

#
deee <- decompose.graph(g_essential, min.vertices = 10)
deee_giant <- deee[[1]]
deee_giant_m <- as.matrix(get.adjacency(deee_giant, names = T))

# is.matrix_irreducible(adjacency_g_essential_non_giant)
```

```
# is.matrix_ergodic(adjacency_g_essential_non_giant)

# essential_neighbours_reducible_blocks <- blockmatrix(adjacency_g_essential_non_giant)
```

Probably the problem is that the matrix is very sparse. There are several computational challenges associated with empirical network analysis and network simulations. Networks are typically very sparse, high- dimensional structures.

3.4 Network link analysis

Link clustering Overlapping communities

```
library(linkcomm)
```

Chapter 4

Protein complexes

In protein networks finding communities with modularity based methods doesn't answer any important question. Proteins rarely function on their own, usually they form complexes with other proteins. For the protein interaction networks is crucial to determine the protein complexes which are the actual functional machines. Protein complexes vary in sizes and they share their building blocks, so they overlap in the protein interaction network.

The experimental appoach is to pull down the complexes and identify them with mass spectrometry. The computational approach either uses homology to identify known complexies from one organism to the other or de novo identification with clustering algorithms in PPI networks.

For yeast Databases: *Up-to-date catalogues of yeast protein complexes* Experiment: *Global landscape of protein complexes in the yeast Saccharomyces* cerevisiae

For drosophila Compleat database: *Protein Complex–Based Analysis Framework for High-Throughput Data Sets* Experiment: *A Protein Complex Network of Drosophila melanogaster*

The prediction of protein complexes from PPI networks is an important task for biologists. Many different algorithms exist today. The authors of the article (Ou-Yang, Dai, and Zhang 2015) implement signed networks for the complex prediction and more specificaly they used the drosophila signed PPI network. They mention:

> In this section, we compare SGNMF with seven state-of-the-art approaches that detect >protein complexes from PPI networks, namely ClusterONE [19], MCL [15], MINE [18], >MCODE [16], SPICi [17], EC-BNMF [22] and NMF [30]. The performance of all the >compared methods are evaluated in terms of three metrics (Acc, FRAC and MMR) with >respect to four gold standards (COMPLEAT, CYC2008, MIPS and SGD).

CYC2008

4.1 COMPLEAT database

From the COMPLEAT database and online tool we downloaded all the protein complexes of drosophila. The complexes are both from literature and predicted with a variety of tools. The methology is explained in the paper as well as in the online portal (A Vinayagam et al. 2013).

```
dros_complexes <- read.delim(file = "Data/drosophila_complexes_compleat0.txt", header = F, sep = "\t")
# summary(dros_complexes)
nrow(with(dros_complexes, dros_complexes[V3 == "Literature", ]))
```

```
## [1] 2045
```

```
nrow(with(dros_complexes, dros_complexes[V3 == "Predicted", ]))
```

```
## [1] 3282
```

```
# homo sapiens
compleat_homo <- read.delim(file = "Data/compleat1_homo.txt", header = F, sep = "\t")
compleat_homo <- compleat_homo[!(is.na(compleat_homo$V2)), ]  # remove empty rows
# yeast
compleat_yeast <- read.delim(file = "Data/compleat2_yeast.txt", header = F, sep = "\t")
compleat_yeast <- compleat_yeast[!(is.na(compleat_yeast$V2)), ]  # remove empty rows
```

There are 5786 distinct proteins and 5326 complexes of drosophila in the data set.

```
# All complexes
dros_complexes_bi <- dros_complexes[, c(1, 12)]
splitted_complexes <- strsplit(x = as.character(dros_complexes_bi$V12), " ")
Protein_Complex_bipartite <- data.frame(Complex = rep.int(dros_complexes_bi$V1, sapply(splitted_complexe
    length)), Protein = unlist(splitted_complexes))  # create data frame with 2 columns (Complexes,Prot

length(unique(Protein_Complex_bipartite$Protein))
```

```
## [1] 5786
```

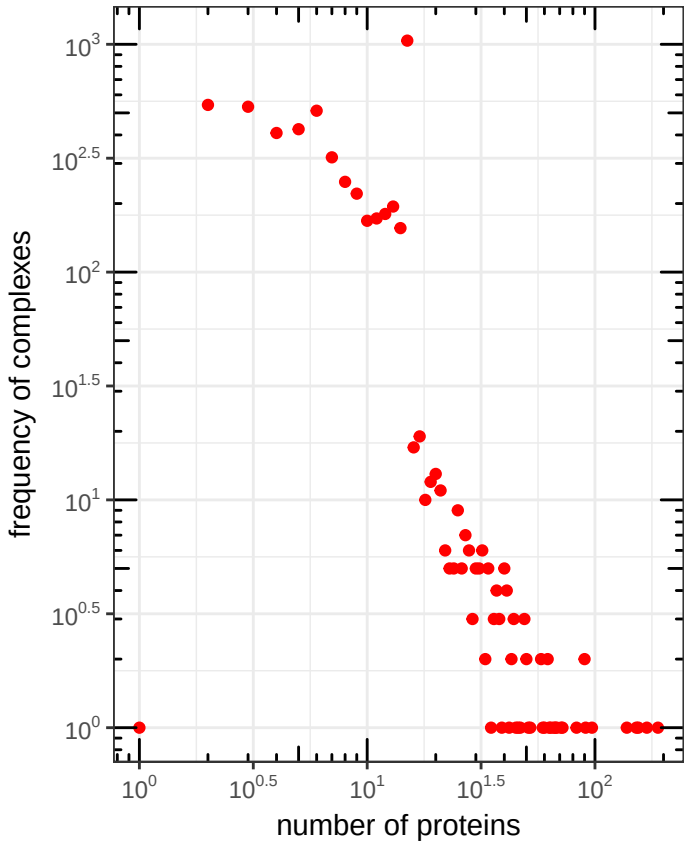
```
length(unique(Protein_Complex_bipartite$Complex))
```

```
## [1] 5326
```

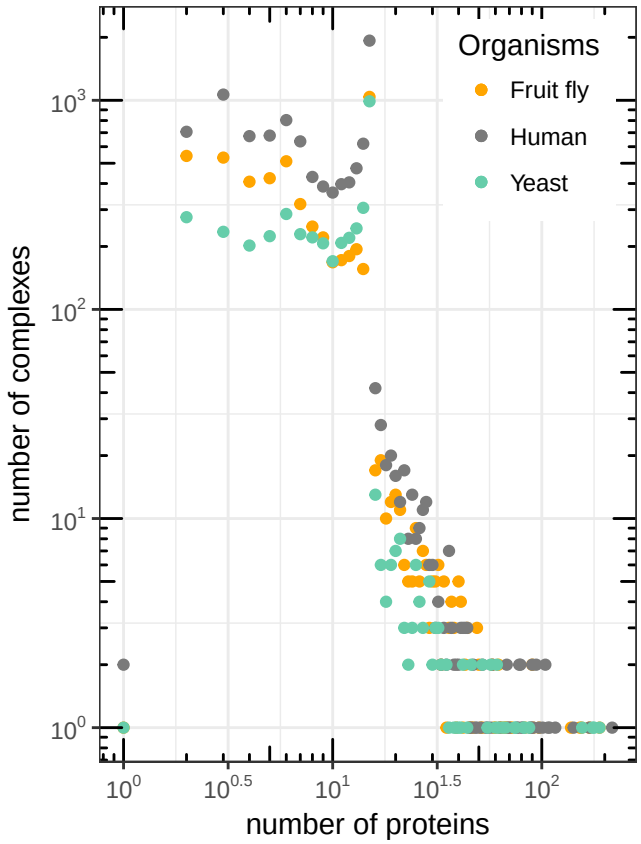
4.2 Proteins and Complexes Distributions

4.2.1 How many proteins each complex contain?

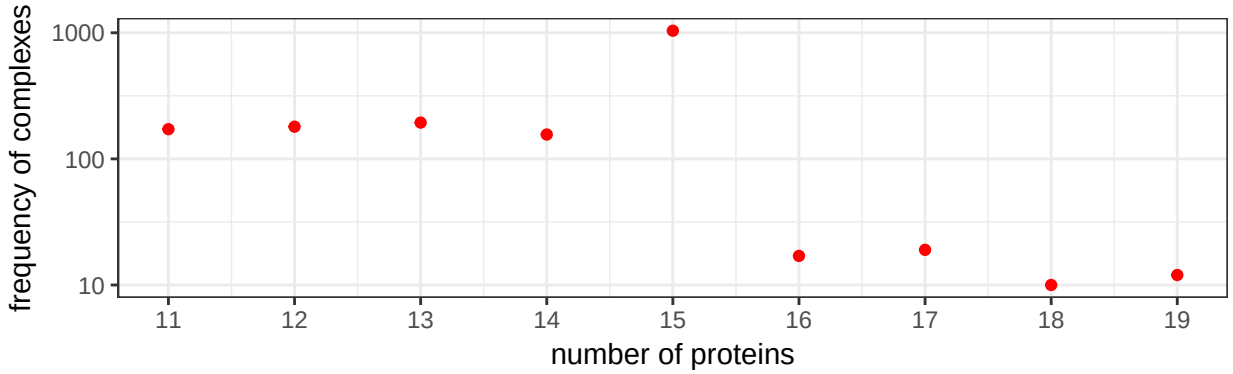
The distribution of the proteins that the complexes consist of will elucidate the sizes and abundance of protein complexes.




```
## Saving 6.5 x 4.5 in image
```

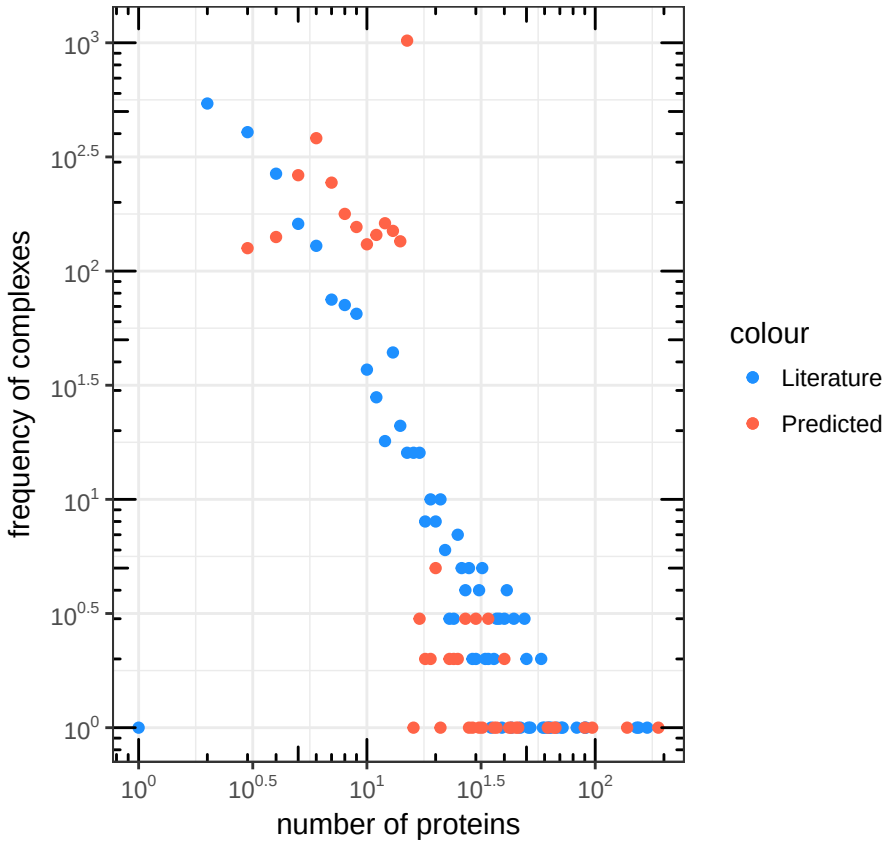


There is a gap in the distribution between 15 and 16 number of proteins in the complexes. Below we plot this part only in log-lin scale. It looks like a phase transition but most probably has something to do with the data and the way they were produced.

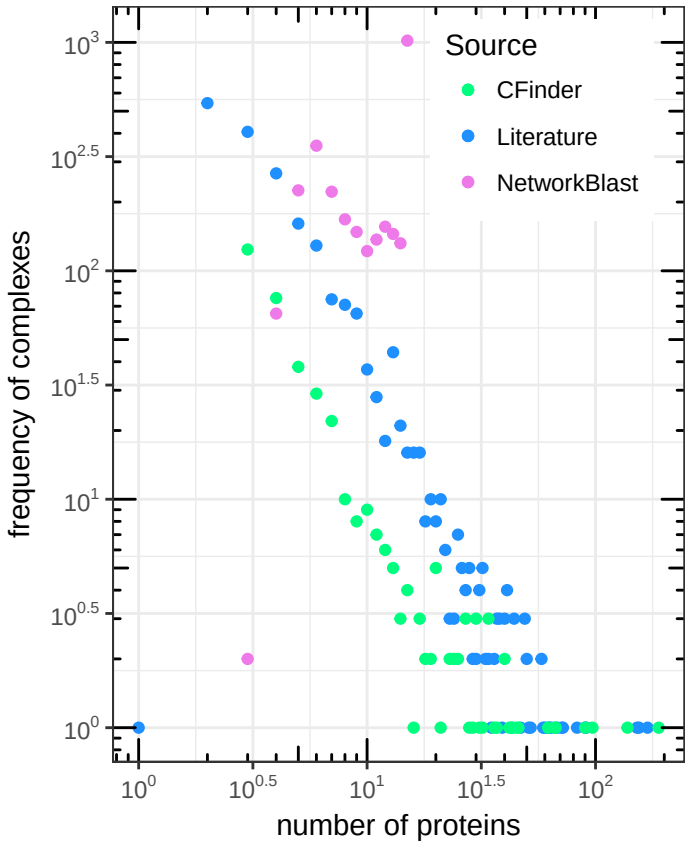



```
## Saving 6.5 x 4.5 in image
```

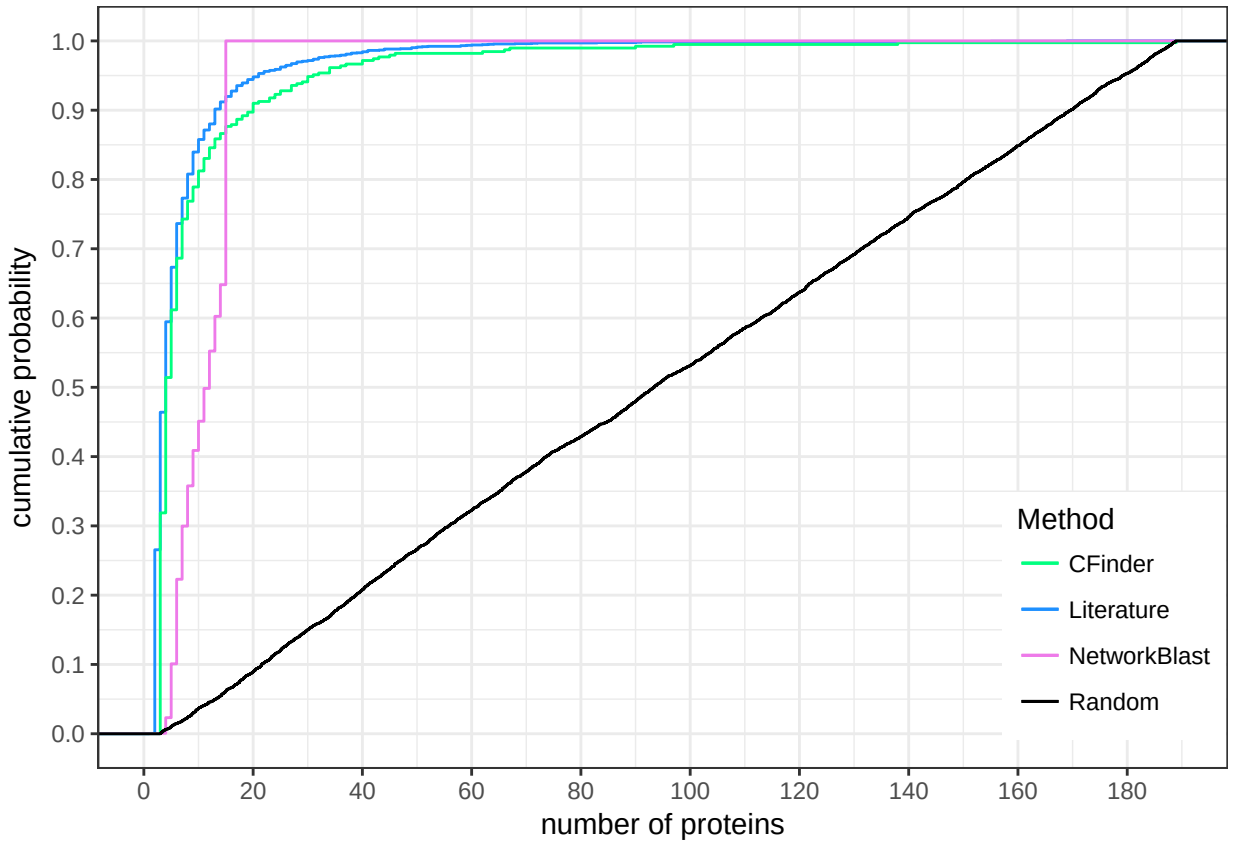
Difference in the distribution between “Literature” and “Predicted”.




```
## Saving 6.5 x 4.5 in image
```




```
compleat_drosophila_predicted2 <- dros_complexes %>% filter(V7 == "NetworkBlast" | V7 == "CFinder")

random_dist <- as.data.frame(x = runif(n = 5000, min = 3, max = 189))
colnames(random_dist) <- "Random"

dros_complexes_lit <- dros_complexes %>% filter(V3 == "Literature")

ggplot() + stat_ecdf(data = dros_complexes_lit, aes(x = V2, colour = "Literature"), geom = "step") +
    stat_ecdf(data = compleat_drosophila_predicted2, aes(x = V2, colour = V7), geom = "step") + stat_ec
    aes(x = Random, colour = "Random"), geom = "step") + scale_colour_manual(values = c(Literature = "d
    CFinder = "springgreen", NetworkBlast = "orchid2", Random = "black"), name = "Method") + scale_x_co
    breaks = seq(0, 200, 20)) + scale_y_continuous(position = "left", breaks = seq(0, 1, 0.1)) + # ggti
labs(x = "number of proteins", y = "cumulative probability") + theme_bw() + theme(legend.position = c(0
    0.2))
```




```
ggsave("DistributionCumulative_Methods_All_Proteins_in_Complexes_Drosoplila_Compleat.pdf", plot = last_p
    device = "pdf", dpi = 150, path = "Figures/")
```

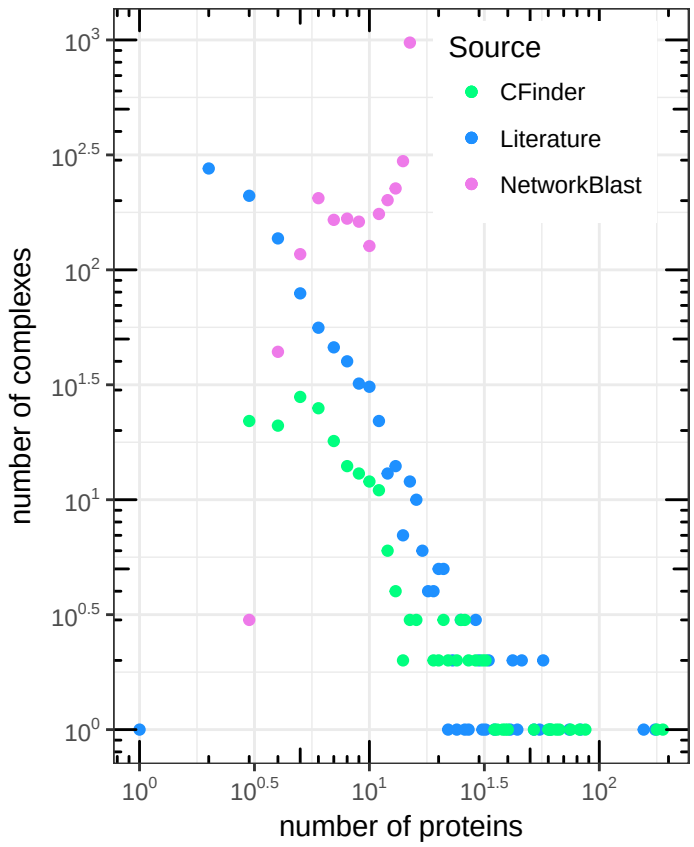
```
## Saving 6.5 x 4.5 in image
```

For yeast.

```
# yeast
compleat_yeast_Literature <- compleat_yeast %>% filter(V3 == "Literature") %>% group_by(V2) %>% summaris
colnames(compleat_yeast_Literature)[1] <- "complex_with_k_proteins"

compleat_yeast_NetworkBlast <- compleat_yeast %>% filter(V7 == "NetworkBlast") %>% group_by(V2) %>% summ
colnames(compleat_yeast_NetworkBlast)[1] <- "complex_with_k_proteins"

compleat_yeast_CFinder <- compleat_yeast %>% filter(V7 == "CFinder") %>% group_by(V2) %>% summarise(n_co
colnames(compleat_yeast_CFinder)[1] <- "complex_with_k_proteins"


ggplot() + geom_point(data = compleat_yeast_Literature, aes(x = complex_with_k_proteins, y = n_complex_w
    color = "Literature")) + geom_point(data = compleat_yeast_CFinder, aes(x = complex_with_k_proteins,
    y = n_complex_with_k_proteins, color = "CFinder")) + geom_point(data = compleat_yeast_NetworkBlast,
    aes(x = complex_with_k_proteins, y = n_complex_with_k_proteins, color = "NetworkBlast")) + # ggtitl
scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = trans_format("log10", math_form
    scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = trans_format("log10", math_
    annotation_logticks(sides = "trbl") + coord_fixed(ratio = 1) + scale_colour_manual(values = c(Litera
    CFinder = "springgreen", NetworkBlast = "orchid2"), name = "Source") + labs(x = "number of proteins"
    y = "number of complexes") + theme_bw() + theme(legend.position = c(0.75, 0.85))
```

Table 4.1: Methods used for the prediction of complexes

Method	Predictions
CFinder	389
NetworkBlast	2893

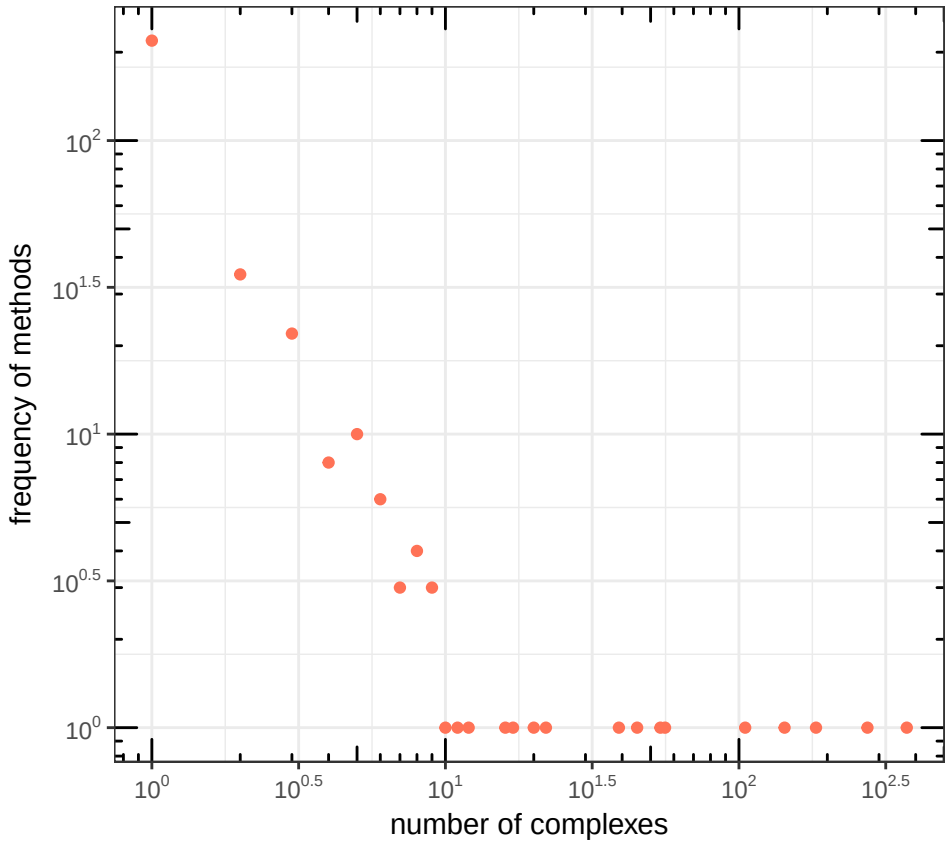



```
ggsave("Distribution_Methods_All_Proteins_in_Complexes_Yeast_Compleat.pdf", width = 10, height = 11.5,
    units = "cm", plot = last_plot(), device = "pdf", dpi = 300, path = "Figures/")
```

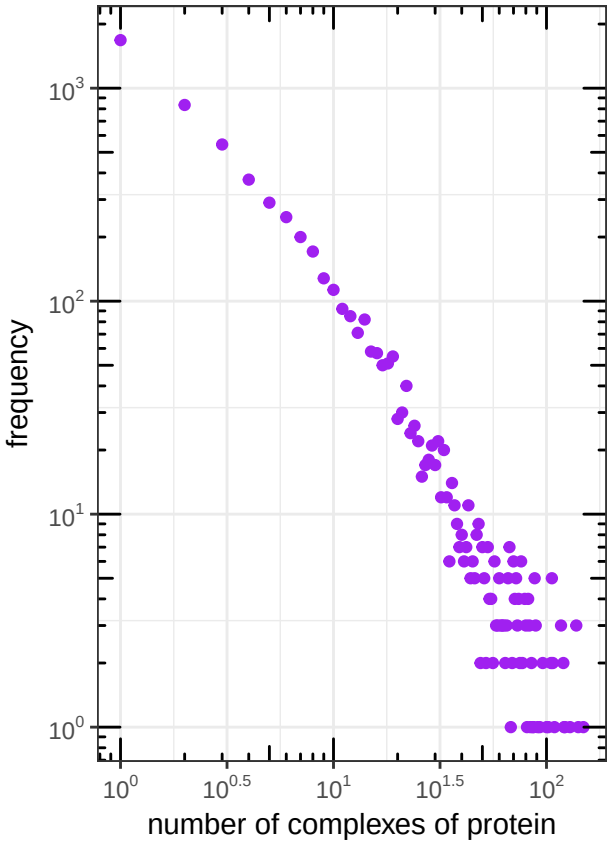
It's obvious that the predicted complexes are responsible for the previous "irregularity" in the distribution. The tools used from (A Vinayagam et al. 2013) for the prediction of the complexes are the CFinder and NetworkBLAST. For all complexes, e.g literature curated and predicted complexes combined, 328 methods were used and for 112 complexes the reference method is missing. NetworkBlast provided more complexes by an order of magnitude compared to the other methods and is also biased towards medium sized complexes.

```
Network_blast <- dros_complexes %>% filter(V7 == "NetworkBlast")
```

Next we take a closer look at the literature methods and the protein complexes they have discovered.


frequency of methods
10^2
$10^{1.5}$
10^1
$10^{0.5}$
10^0
10^0
$10^{0.5}$
10^1
$10^{1.5}$
10^2
$10^{2.5}$
number of complexes

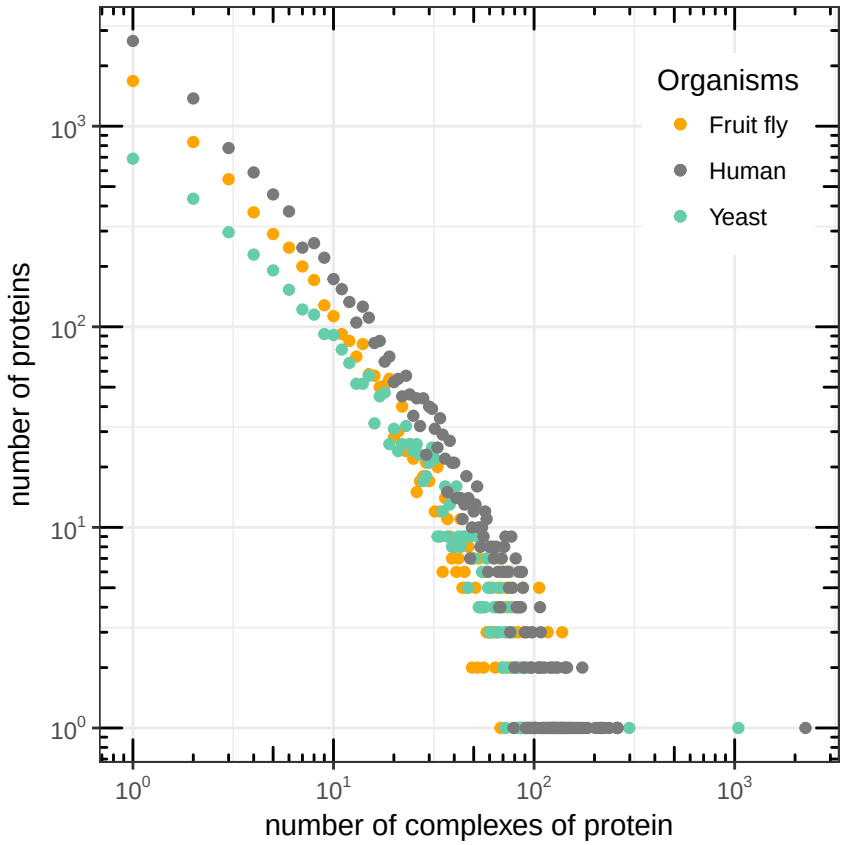
4.2.2 In how many complexes each protein is participating?




```
# drosophila
compleat_drosophila2 <- dros_complexes[, c(1, 3, 7, 12)]
drosophila_splitted_complexes <- strsplit(x = as.character(compleat_drosophila2$V12), " ")

compleat_drosophila_long <- data.frame(Complex = rep.int(compleat_drosophila2$V1, sapply(drosophila_spli
    length)), Method = rep.int(compleat_drosophila2$V3, sapply(drosophila_splitted_complexes, length)),
    Tool = rep.int(compleat_drosophila2$V7, sapply(drosophila_splitted_complexes, length)), Protein = u

# yeast
compleat_yeast2 <- compleat_yeast[, c(1, 3, 7, 12)]
yeast_splitted_complexes <- strsplit(x = as.character(compleat_yeast2$V12), " ")

compleat_yeast_long <- data.frame(Complex = rep.int(compleat_yeast2$V1, sapply(yeast_splitted_complexes
    length)), Method = rep.int(compleat_yeast2$V3, sapply(yeast_splitted_complexes, length)), Tool = re
    sapply(yeast_splitted_complexes, length)), Protein = unlist(yeast_splitted_complexes))

# human

compleat_homo2 <- compleat_homo[, c(1, 5)]
homo_splitted_complexes <- strsplit(x = as.character(compleat_homo2$V5), ";")

compleat_homo_long <- data.frame(Complex = rep.int(compleat_homo2$V1, sapply(homo_splitted_complexes,
    length)), Protein = unlist(homo_splitted_complexes))
```

```
# drosophila
compleat_drosophila_protein_dist <- compleat_drosophila_long %>% group_by(Protein) %>% summarize(k_compl
    group_by(k_complexes) %>% summarise(n_protein_k_complexes = n())
# yeast
compleat_yeast_protein_dist <- compleat_yeast_long %>% group_by(Protein) %>% summarize(k_complexes = n()
    group_by(k_complexes) %>% summarise(n_protein_k_complexes = n())
# homo
compleat_homo_protein_dist <- compleat_homo_long %>% group_by(Protein) %>% summarize(k_complexes = n())
    group_by(k_complexes) %>% summarise(n_protein_k_complexes = n())
```

```
ggplot() + geom_point(data = compleat_drosophila_protein_dist, aes(x = k_complexes, y = n_protein_k_comp
    color = "Fruit fly")) + geom_point(data = compleat_yeast_protein_dist, aes(x = k_complexes, y = n_pr
    color = "Yeast")) + geom_point(data = compleat_homo_protein_dist, aes(x = k_complexes, y = n_protein
    color = "Human")) + scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = trans_
    math_format(10^.x))) + scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = tran
    math_format(10^.x))) + annotation_logticks(sides = "trbl") + coord_fixed(ratio = 1) + scale_colour_m
    Yeast = "mediumaquamarine", Human = "gray48"), name = "Organisms") + # ggtitle('Distribution of Pro
labs(x = "number of complexes of protein", y = "number of proteins") + theme_bw() + theme(legend.positio
    0.805))
```




```
ggsave("Distribution_Complexes_in_Proteins_ALL_Compleat.pdf", width = 10, height = 10, units = "cm",
    plot = last_plot(), device = "pdf", dpi = 150, path = "Figures/")
```

4.2.3 Complexes in the network

Which of these protein complexes are present in our data set?

```
dros_network_genes <- read_csv("../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_network_ge
    col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   locus = col_character(),
##   symbols = col_character(),
##   datasets = col_integer(),
##   datasetIDs = col_number(),
##   essentiality.status = col_character(),
##   essentiality.consensus = col_character(),
##   color = col_character()
## )
```

```
complexes_included <- as.data.frame(Protein_Complex_bipartite[Protein_Complex_bipartite$Protein %in%
    dros_network_genes$locus, ])

length(unique(complexes_included$Complex))
```

```
## [1] 5027
```

```
complex_exist_proteins <- as.data.frame(table(complexes_included$Complex))

complex_exist_proteins_m <- merge(x = complex_exist_proteins, y = dros_complexes, by.x = 1, by.y = 1,
    all.x = T)

# the complete complexes of the dataset.
complete_complexes <- complex_exist_proteins_m[complex_exist_proteins_m$Freq == complex_exist_proteins_
    ]

complete_complexes %>% filter(V3 == "Literature") %>% nrow()
```

```
## [1] 326
```

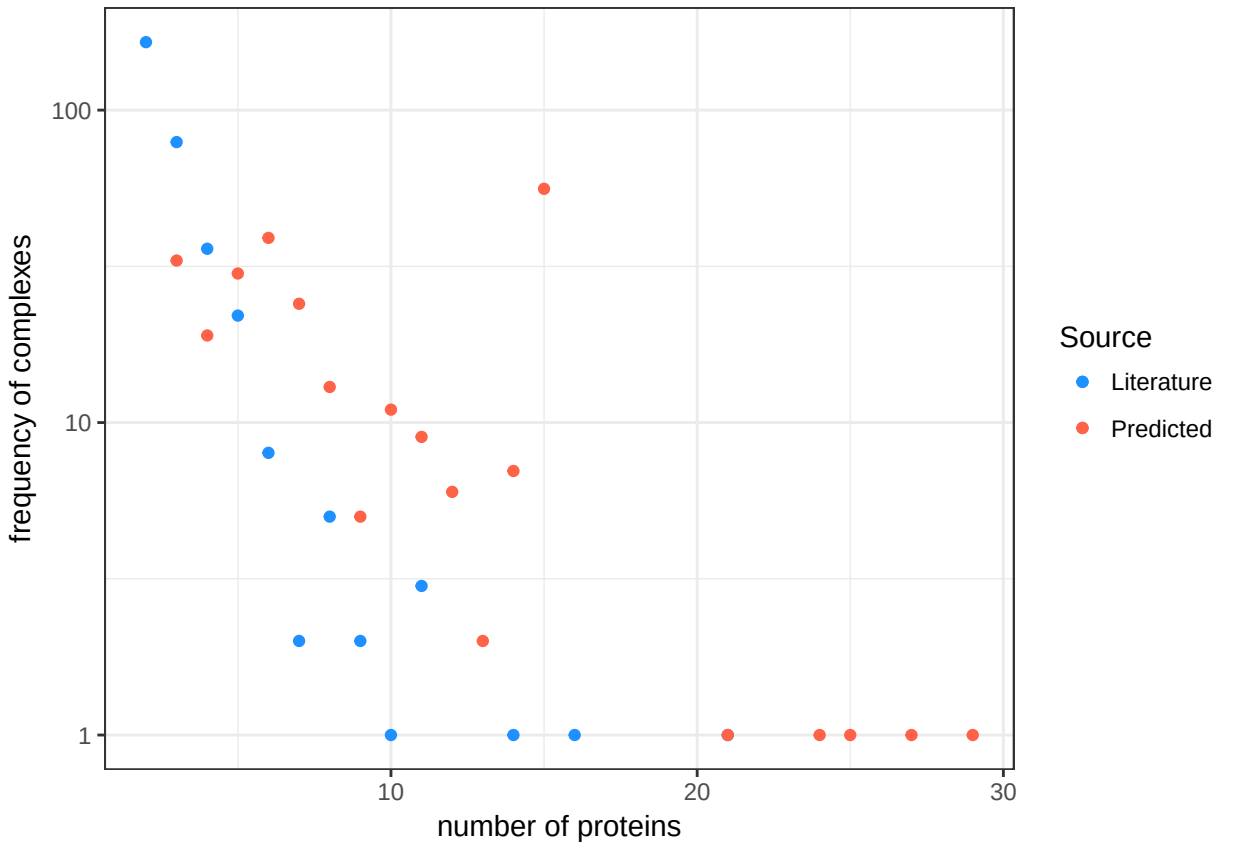
```
complete_complexes %>% filter(V3 == "Predicted") %>% nrow()
```

```
## [1] 259
```

From the protein complex annotation of the signed PPI network of Drosophila we found that only 585 complexes are complete. Meaning that all of their proteins are present (not necessarily interacting with each other) in the data set even though 5027 complexes have 1 or more proteins appearing in the network. The 326 of them are from literature and the 259 are predicted with the tools mentioned before.

```
# predicted distribution of complete complexes
complete_complexes_dist_predicted <- complete_complexes %>% filter(V3 == "Predicted") %>% group_by(Freq
    dplyr::summarise(n_complex_with_k_proteins = n())

# literature distribution of complete complexes

complete_complexes_dist_literature <- complete_complexes %>% filter(V3 == "Literature") %>% group_by(Fr
    dplyr::summarise(n_complex_with_k_proteins = n())

ggplot() + geom_point(data = complete_complexes_dist_literature, aes(x = Freq, y = n_complex_with_k_pro
    color = "Literature")) + geom_point(data = complete_complexes_dist_predicted, aes(x = Freq, y = n_c
    color = "Predicted")) + # ggtitle('Distribution of Proteins in Drosophila's Complete Protein Comple
scale_y_log10() + # scale_x_log10()+ coord_fixed(ratio = 1)+
scale_colour_manual(values = c(Literature = "dodgerblue", Predicted = "tomato1"), name = "Source") +
```

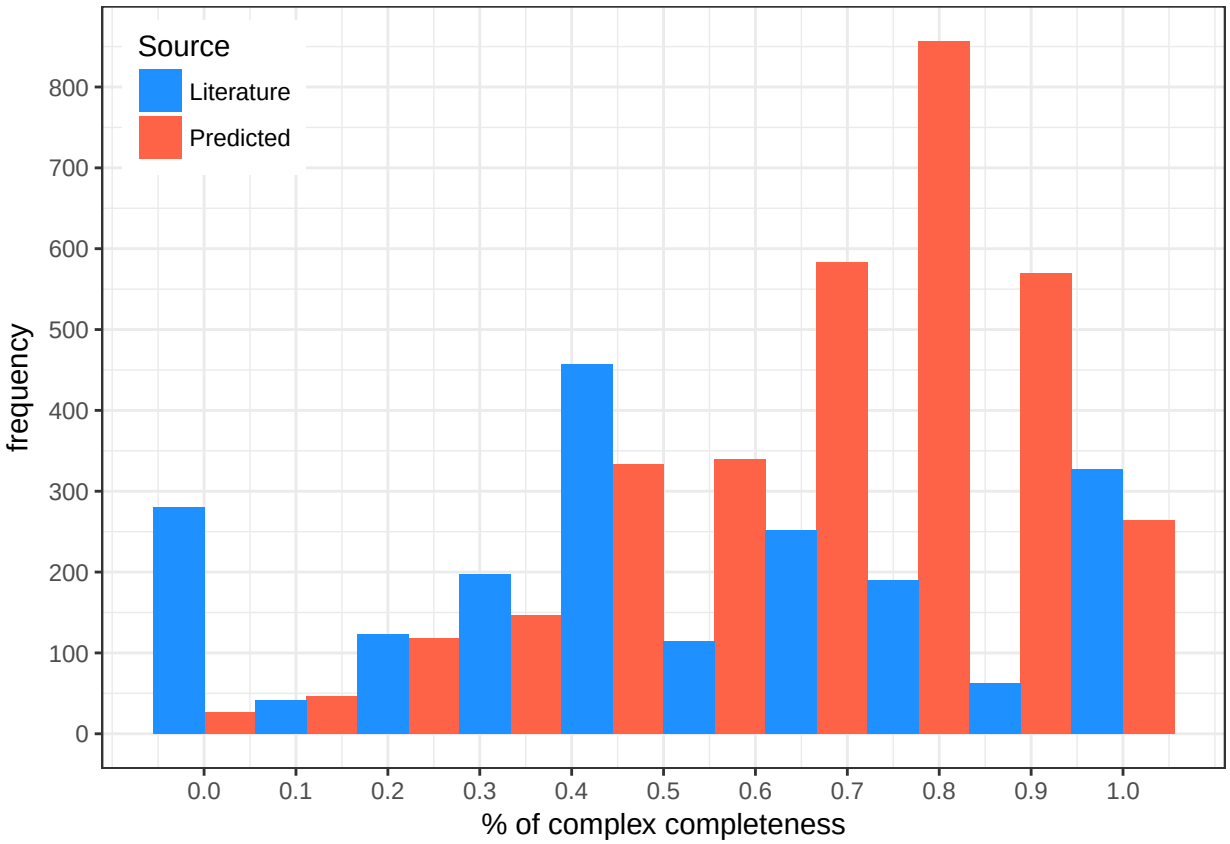
```
    labs(x = "number of proteins", y = "frequency of complexes") + theme_bw()
```




```
ggsave("Distribution_Proteins_in_Complexes_Drosoplila_Complete_in_Network.pdf", width = 7, height = 11,
    units = "cm", plot = last_plot(), device = "pdf", dpi = 150, path = "Figures/")
```

What is the % of the missing proteins?

```
compare_complexes_proteins <- complex_exist_proteins_m[, c(1, 2, 3, 4)]

# % of missing proteins

compare_complexes_proteins$compare_percent <- with(compare_complexes_proteins, compare_complexes_protein

ggplot() + geom_histogram(data = compare_complexes_proteins, aes(x = compare_percent, y = ..count..,
    fill = V3), bins = 10, position = "dodge") + # ggtitle('Distribution of the percentage of the exist
scale_fill_manual(values = c(Literature = "dodgerblue", Predicted = "tomato1"), name = "Source") + scale
    breaks = seq(0, 1, 0.1)) + scale_y_continuous(position = "left", breaks = seq(0, 900, 100)) + labs(
    y = "frequency") + theme_bw() + theme(legend.position = c(0.1, 0.88))
```

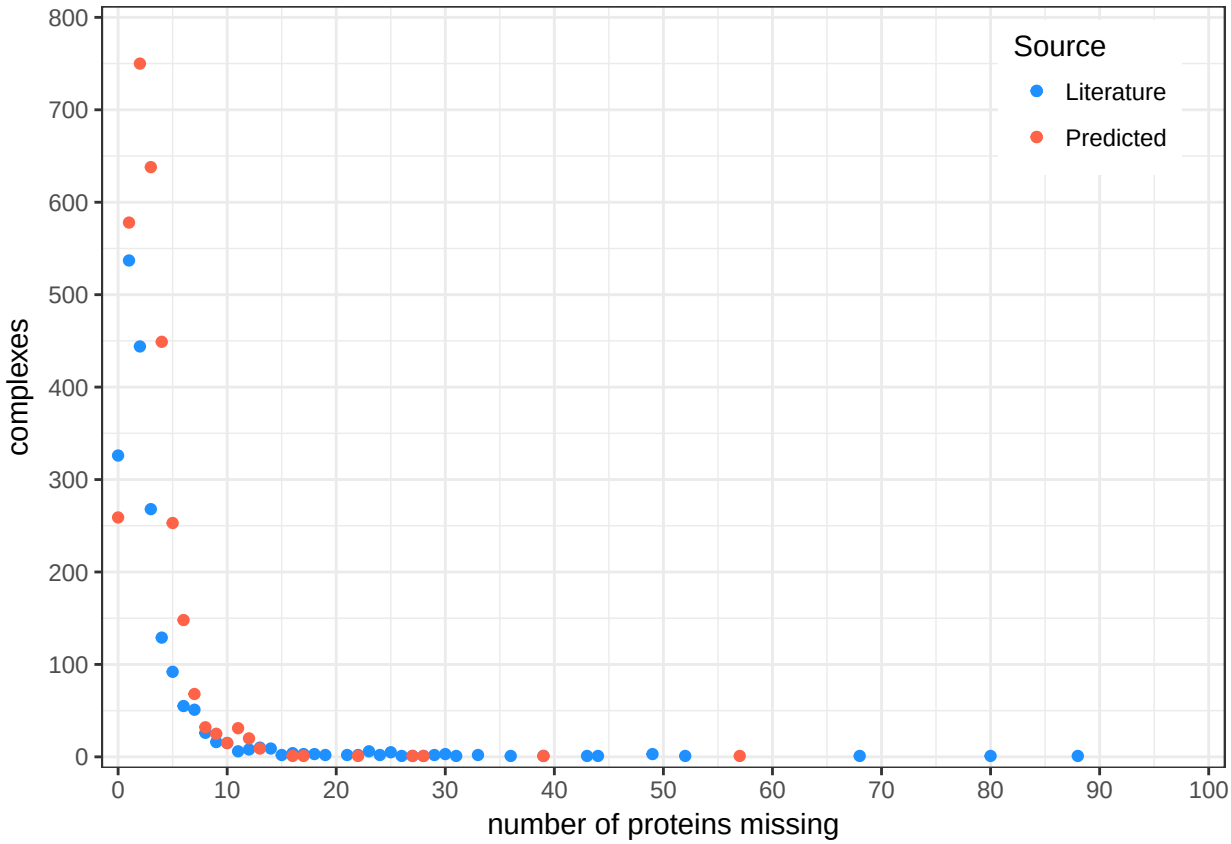



```
ggsave("Histogram_Complex_Completeness_in_network.pdf", plot = last_plot(), device = "pdf", dpi = 150,
    path = "Figures/")
```

```
## Saving 6.5 x 4.5 in image
```

Here we present the distribution of the value of missing proteins in each complex of the 5027 that have at least one protein appearing in the network.

```
# value of missing proteins from complexes.
compare_complexes_proteins$compare_missing <- with(compare_complexes_proteins, compare_complexes_protein
    compare_complexes_proteins$Freq)

compare_complexes_proteins_lit <- compare_complexes_proteins %>% filter(V3 == "Literature") %>% group_by
    summarise(n = n())

compare_complexes_proteins_pred <- compare_complexes_proteins %>% filter(V3 == "Predicted") %>% group_by
    summarise(n = n())


ggplot() + geom_point(data = compare_complexes_proteins_lit, aes(x = compare_missing, y = n, color = "Li
    geom_point(data = compare_complexes_proteins_pred, aes(x = compare_missing, y = n, color = "Predicte
    # ggtitle('Distribution of the percentage of the existing proteins of complexes')+
scale_color_manual(values = c(Literature = "dodgerblue", Predicted = "tomato1"), name = "Source") + scal
    breaks = seq(0, 100, 10), limits = c(0, 100), expand = c(0.015, 0)) + scale_y_continuous(position =
    breaks = seq(0, 800, 100), limits = c(0, 800), expand = c(0.015, 0)) + labs(x = "number of proteins
    y = "complexes") + theme_bw() + theme(legend.position = c(0.88, 0.88))
```




```
ggsave("Histogram_Complex_Missing_Proteins_in_network.pdf", plot = last_plot(), device = "pdf", dpi = 15
    path = "Figures/")
```

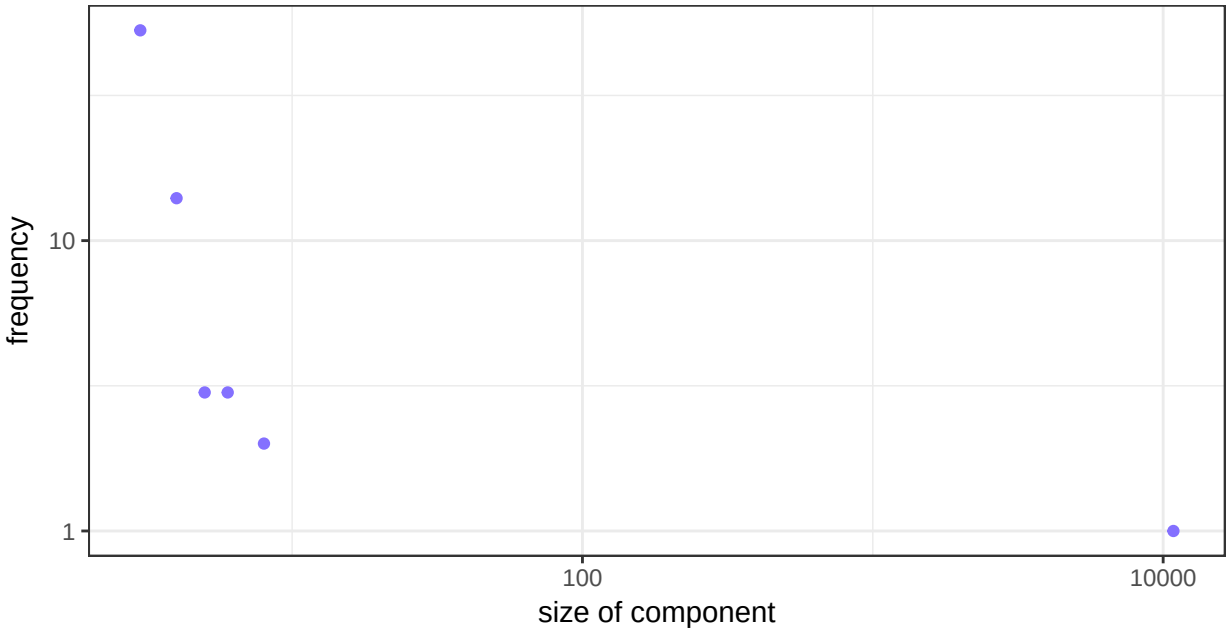
```
## Saving 6.5 x 4.5 in image
```

4.3 Complexes and proteins as bipartite network

4.3.1 Degree distribution

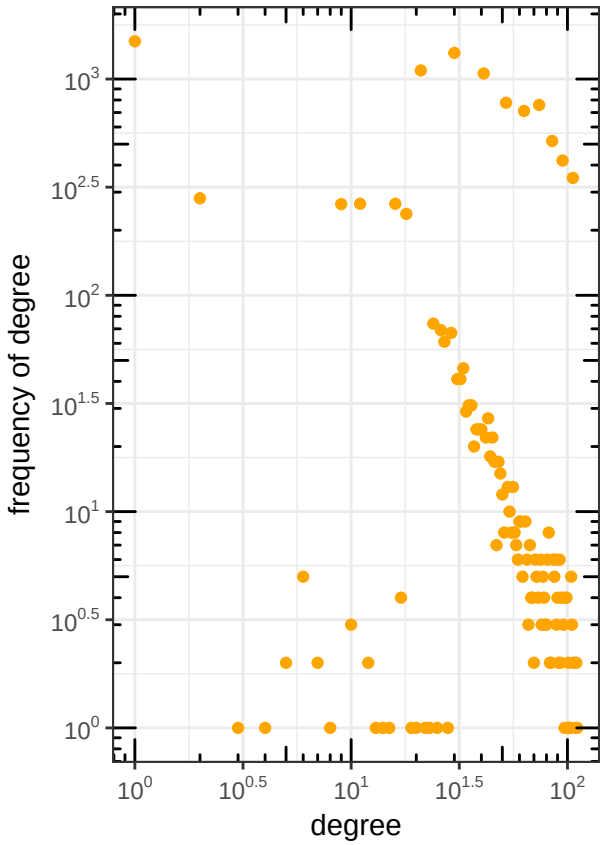
```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

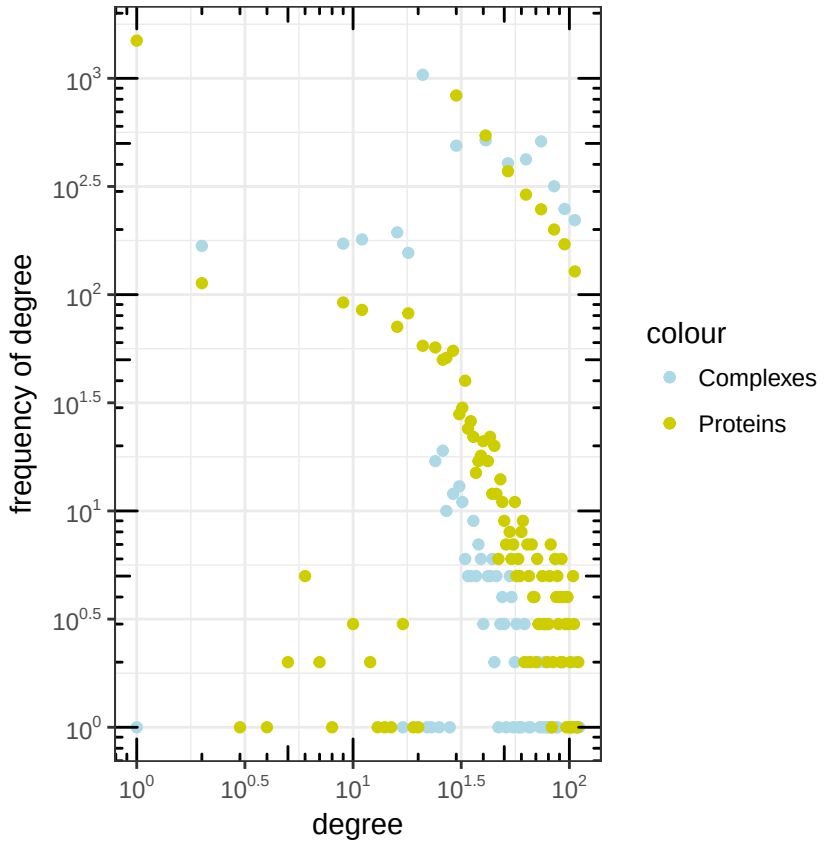
```
## [1] FALSE
```




```
## Saving 6.5 x 4.5 in image
```

Degree distribution of the bipartite network.





4.3.2 Nestedness

```
library(vegan)

is_connected(protein_complexes_net_giant)

adjacency_complex_protein <- as_adjacency_matrix(protein_complexes_net_giant, attr = NULL, names = T)

adjacency_complex_protein <- as.data.frame(as.matrix(adjacency_complex_protein))

proteins <- Protein_Complex_bipartite %>% distinct(Protein)
proteins <- as.vector(proteins$Protein)

complexes <- Protein_Complex_bipartite %>% distinct(Complex)
complexes <- as.vector(complexes$Complex)

# remove the complexes IDS from the rows and protein IDS from the columns.
adjacency_complex_protein <- adjacency_complex_protein[!rownames(adjacency_complex_protein) %in% comple
    !(colnames(adjacency_complex_protein) %in% proteins)]

# adjacency_complex_protein <- as.matrix(adjacency_complex_protein) adjacency_complex_protein <-
# as.data.frame(adjacency_complex_protein)

# this function takes input a data frame.
nestedchecker_complex_protein <- nestedchecker(adjacency_complex_protein)
```

```
ddd <- nestednodf(adjacency_complex_protein)  ## Takes many hours to run!!!
# error!!!!!
nestedtemp_complex_protein <- nestedtemp(adjacency_complex_protein)

# statistical test function : oecosimu
library(bipartite)
nestedness_bi <- nestedness(as.matrix(adjacency_complex_protein))

# nordf
```

4.4 Protein complexes and essentiality

Since the actual functional machines are protein complexes and not individual proteins the author of the article (Ryan et al. 2013) explored the essentiality on the scale of protein complexes. They found that essentiality is a property of the complexes which is transmitted to it's protein components.

With the same data set from OGEE database we will check the essentiality of the complexes.

```
# data from OGEE
dros_essential2 <- read_csv(file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_essenti
    col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   locus = col_character(),
##   essentiality.consensus = col_character()
## )
```

```
protein_Complex_essentiality <- left_join(Protein_Complex_bipartite, dros_essential2, by = c(Protein = "
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## character vector and factor, coercing into character vector
```

```
# Find the essentiality consensus of proteins from the COMPLEAT database
essentiality_consensus_Compleat_proteins <- protein_Complex_essentiality %>% distinct(Protein, essentiali
    group_by(essentiality.consensus) %>% dplyr::summarise(count = n())

colnames(essentiality_consensus_Compleat_proteins) <- c("Essentiality consensus", "Count")
kable(x = essentiality_consensus_Compleat_proteins)
```

Essentiality consensus	Count
Conditional	69
Essential	181
Nonessential	5455
NA	81

```
write.table(x = essentiality_consensus_Compleat_proteins, file = "../Thesis_Essentiality_Drosophila_Sign
    sep = ",", col.names = TRUE, row.names = FALSE)
```

There are 81 proteins from the COMPLEAT data base with unknown essentiality consensus. Next we have to estimate the essentiality fraction of each complex. That is:

$$\text{Essentiality Fraction} = \frac{\text{Number of essential proteins in the complex}}{\text{Number of total proteins}}$$

```
# Find the occurrencies of each essentiality status for the complexes based on the consensus of the
# proteins that each complex has.
complex_essentiality <- protein_Complex_essentiality %>% group_by(Complex, essentiality.consensus) %>%
    dplyr::summarise(n_proteins = n())

# transform from long to wide format in order to make calculations for every complex
complex_essentiality_wide <- spread(complex_essentiality, key = essentiality.consensus, value = n_prote
complex_essentiality_wide[is.na(complex_essentiality_wide)] <- 0  # Make NA as 0

# Make numeric
complex_essentiality_wide[, c(2:5)] <- sapply(complex_essentiality_wide[, c(2:5)], as.numeric)

complex_essentiality_wide$protein_count <- rowSums(complex_essentiality_wide[, 2:5])

# Calculate the essentiality fraction for each complex.
complex_essentiality_wide <- complex_essentiality_wide %>% mutate(essentiality_fraction = Essential/pro
    ungroup()

summary(complex_essentiality_wide$essentiality_fraction)
```

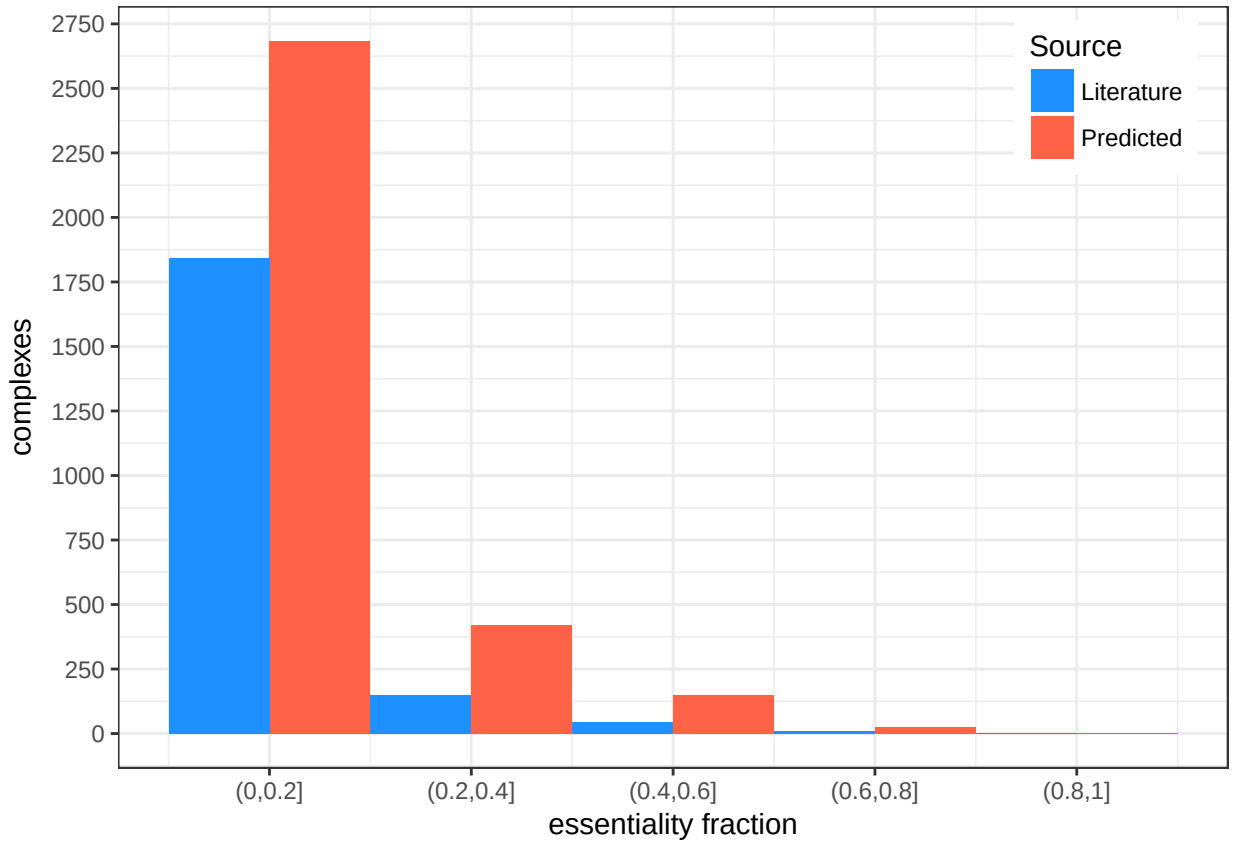
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.05332 0.02254 1.00000
```

```
table(complex_essentiality_wide$essentiality_fraction > 0)
```

```
##
## FALSE  TRUE
##  3991  1335
```

```
# All info of complexes + essentiality
complex_essentiality_wide_source <- complex_essentiality_wide %>% left_join(., dros_complexes, by = c(C

write.table(complex_essentiality_wide_source, file = "../Thesis_Essentiality_Drosophila_Signed_Network/
    sep = ",", col.names = TRUE, row.names = FALSE)
```

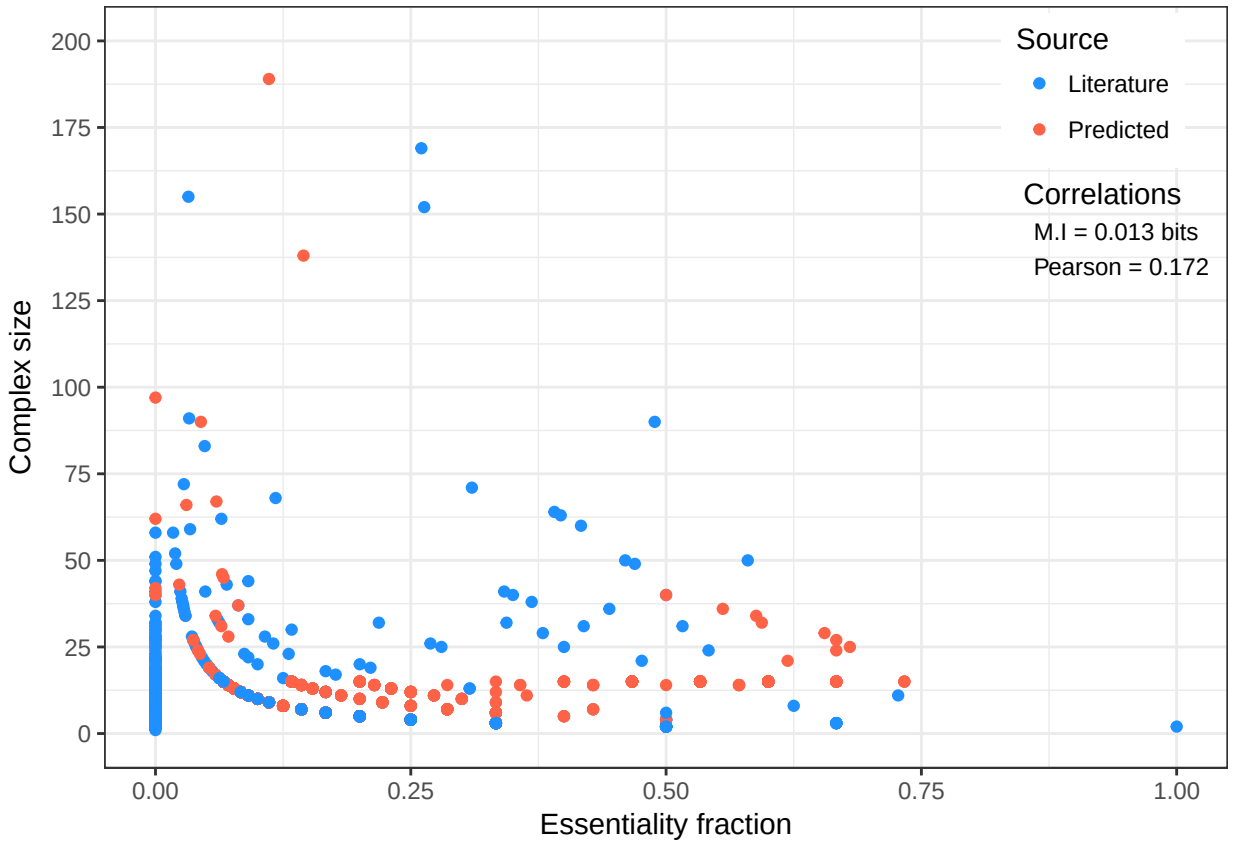
Now we can plot but in order to visualise better we will exclude the complexes with 0 essentiality which are. There are 3991 complexes with no essential components.




```
complex_essentiality_wide %>% filter(essentiality_fraction > 0.85) %>% nrow()
```

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

```
pearson <- cor(complex_essentiality_wide_source$protein_count, complex_essentiality_wide_source$essentia
    method = "pearson")

library(entropy)

mi <- mi.empirical(rbind(complex_essentiality_wide_source$protein_count, complex_essentiality_wide_sour
    unit = "log2")


ggplot() + geom_point(data = complex_essentiality_wide_source, aes(x = essentiality_fraction, y = protei
    color = V3), show.legend = TRUE) + geom_smooth() + annotate("text", size = 4, x = 0.85, y = 156,
    label = "Correlations", hjust = 0) + annotate("text", size = 3.2, x = 0.86, y = 140, label = paste0
    round(mi, 3), " bits", "\n", "Pearson = ", round(pearson, 3)), hjust = 0) + scale_y_continuous(breal
    200, 25), limits = c(0, 200)) + scale_colour_manual(values = c(Literature = "dodgerblue", Predicted
    name = "Source") + labs(x = "Essentiality fraction", y = "Complex size") + theme_bw() + theme(legend
    0.89))
```




```
ggsave("Essentiality_Fraction_and_Size_of_Complexes.pdf", plot = last_plot(), device = "pdf", dpi = 150
    path = "Figures/")
```

```
## Saving 6.5 x 4.5 in image
```

4.4.1 Background bootstrap in essentiality fraction

For a more comprehensive comparison and examination of modularity of the complexes essentiality we performed bootstrap sampling of the essentiality fraction. **All or Nothing** claim holds for the fruit fly? Another approach would be to perform bootstrap on proteins of the complexes instead of the their essentiality fraction.

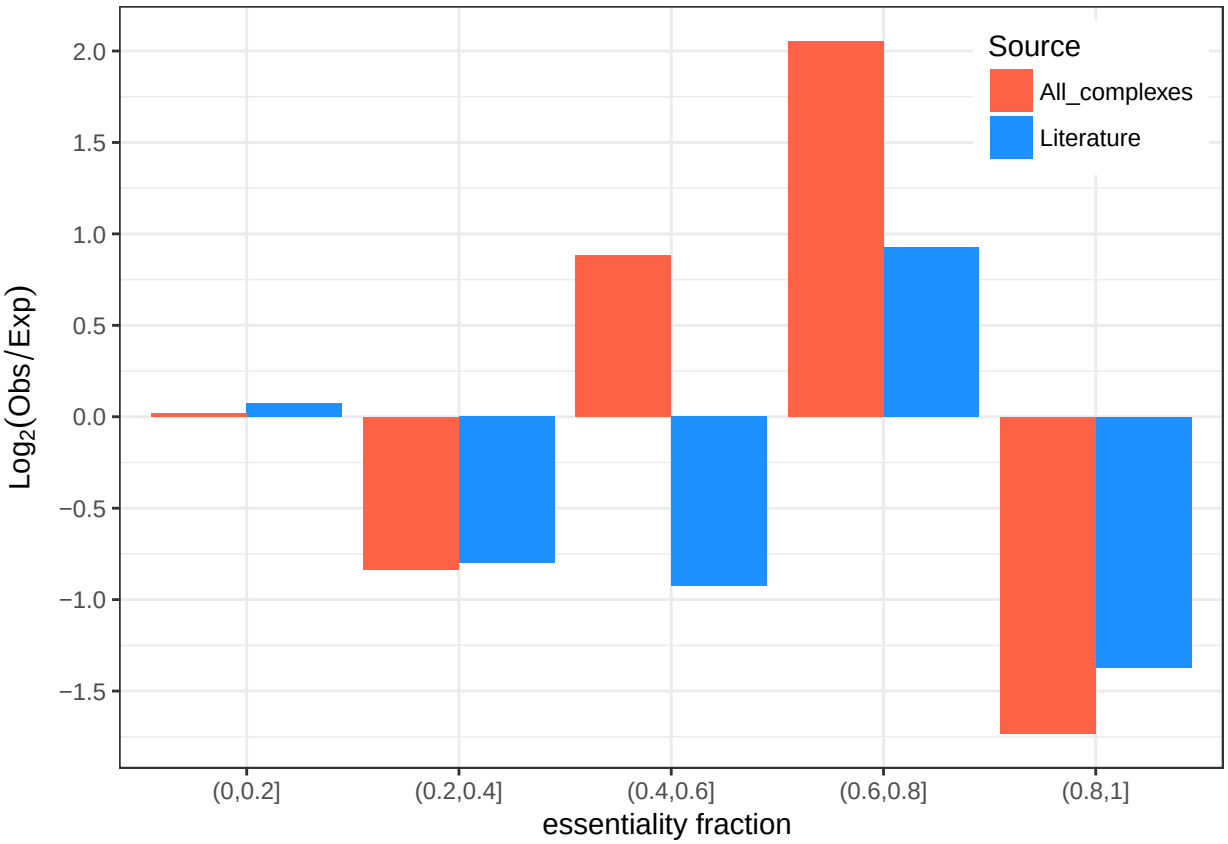
```
###################### Bootstrap ################################

# ~ 8 mins to run

B = 1000  # number of randomizations
n = nrow(protein_Complex_essentiality)

protein_Complex_essentiality$essentiality.consensus <- with(protein_Complex_essentiality, ifelse(essenti
    "Essential", 1, 0))

protein_Complex_essentiality[is.na(protein_Complex_essentiality)] <- 0  # Make NA as 0

bootstrap_essentiality_proteins <- as.data.frame(t(matrix(sample(protein_Complex_essentiality$essentiali
    size = B * n, replace = TRUE), nrow = B, ncol = n)))  # Bootstrap the proteins based on their essen
```

```
# Calculations of the essentiality franction for all the randomizations
start.time <- Sys.time()

Complex_essentiality_bootstrap_essentiality_proteins <- cbind(protein_Complex_essentiality, bootstrap_es
    dplyr::select(., -Protein) %>% group_by(Complex) %>% mutate_all(funs(as.numeric(.))) %>% summarise_a

end.time <- Sys.time()
time.taken <- round(end.time - start.time, 2)
time.taken

# it took Time difference of 19.78 mins !!
write.table(x = Complex_essentiality_bootstrap_essentiality_proteins, file = "../Thesis_Essentiality_Dro
    sep = ",", col.names = TRUE, row.names = FALSE)

# Only complexes from literature

# runs in 3 min!!!

dros_complexes_bipartite_literature <- dros_complexes_bipartite_lit %>% filter(V3 == "Literature") %>%
    left_join(., dros_essential2, by = c(Protein = "locus")) %>% dplyr::select(Complex, Protein, essenti

dros_complexes_bipartite_literature$essentiality.consensus <- with(dros_complexes_bipartite_literature,
    ifelse(essentiality.consensus == "Essential", 1, 0))

dros_complexes_bipartite_literature[is.na(dros_complexes_bipartite_literature)] <- 0  # Make NA as 0

# Bootstrap
B = 1000  # number of randomizations
n = nrow(dros_complexes_bipartite_literature)

bootstrap_essentiality_proteins_literature <- as.data.frame(t(matrix(sample(dros_complexes_bipartite_lit
    size = B * n, replace = TRUE), nrow = B, ncol = n)))  # Bootstrap the proteins based on their essen

# Calculations of the essentiality franction for all the randomizations
start.time <- Sys.time()

Complex_essentiality_bootstrap_essentiality_proteins_literature <- cbind(dros_complexes_bipartite_litera
    bootstrap_essentiality_proteins_literature) %>% dplyr::select(., -Protein) %>% group_by(Complex) %>%
    mutate_all(funs(as.numeric(.))) %>% summarise_all(funs(n = sum(.)/n()))

end.time <- Sys.time()
time.taken <- round(end.time - start.time, 2)
time.taken

write.table(x = Complex_essentiality_bootstrap_essentiality_proteins_literature, file = "../Thesis_Essen
    sep = ",", col.names = TRUE, row.names = FALSE)
```

We will use the log ratio for the comparison of the bootstraped distribution with the real distribution.

$$\text{Log Ratio} = \text{Log}_2(\frac{\text{Observed essentiality fraction}}{\text{Expected essentiality fraction}})$$

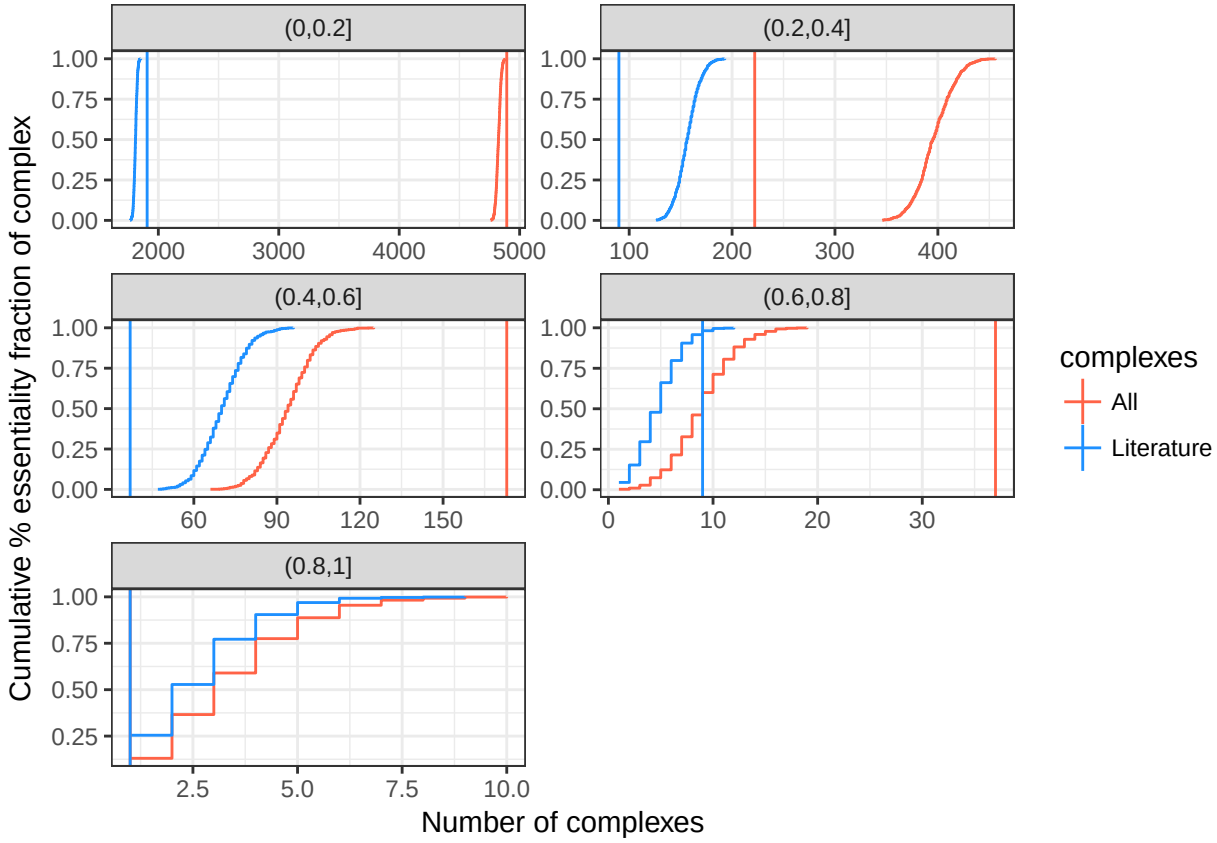
```
# All complexes binning

d_BINS_Complex_essential <- Complex_essentiality_bootstrap_essentiality_proteins %>% dplyr::select(-Comp
    mutate_all(funs(cut(., breaks = c(-1e-04, 0.2, 0.4, 0.6, 0.8, 1)))) %>% gather() %>% group_by(key,
    value) %>% summarise(n = n())

d_BINS_Complex_essential$value <- gsub("-0.0001,0.2", "0,0.2", d_BINS_Complex_essential$value)

d_essential <- d_BINS_Complex_essential %>% filter(key == "essentiality.consensus_n")

d_bootstrap <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n") %>% group_by(valu
    summarise(mean_bootstrap = mean(n))

## Log ratio
d_essential_boot <- left_join(d_essential, d_bootstrap, by = c(value = "value")) %>% mutate(log_ratio =
    base = 2))

d_essential_boot$key <- "All_complexes"
# Literature

d_literature <- Complex_essentiality_bootstrap_essentiality_proteins_literature %>% dplyr::select(-Compl
    mutate_all(funs(cut(., breaks = c(-1e-04, 0.2, 0.4, 0.6, 0.8, 1)))) %>% gather() %>% group_by(key,
    value) %>% summarise(n = n())

d_literature$value <- gsub("-0.0001,0.2", "0,0.2", d_literature$value)

d_essential_literature <- d_literature %>% filter(key == "essentiality.consensus_n")

d_bootstrap_literature <- d_literature %>% filter(key != "essentiality.consensus_n") %>% group_by(value)
    summarise(mean_bootstrap = mean(n))

## Log ratio
d_essential_boot_literature <- left_join(d_essential_literature, d_bootstrap_literature, by = c(value =
    mutate(log_ratio = log(n/mean_bootstrap, base = 2))

d_essential_boot_literature$key <- "Literature"

# Combine

d_bind <- rbind(d_essential_boot, d_essential_boot_literature)

write.table(x = d_bind, file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/Complex_essentia
    sep = ",", col.names = TRUE, row.names = FALSE)
```

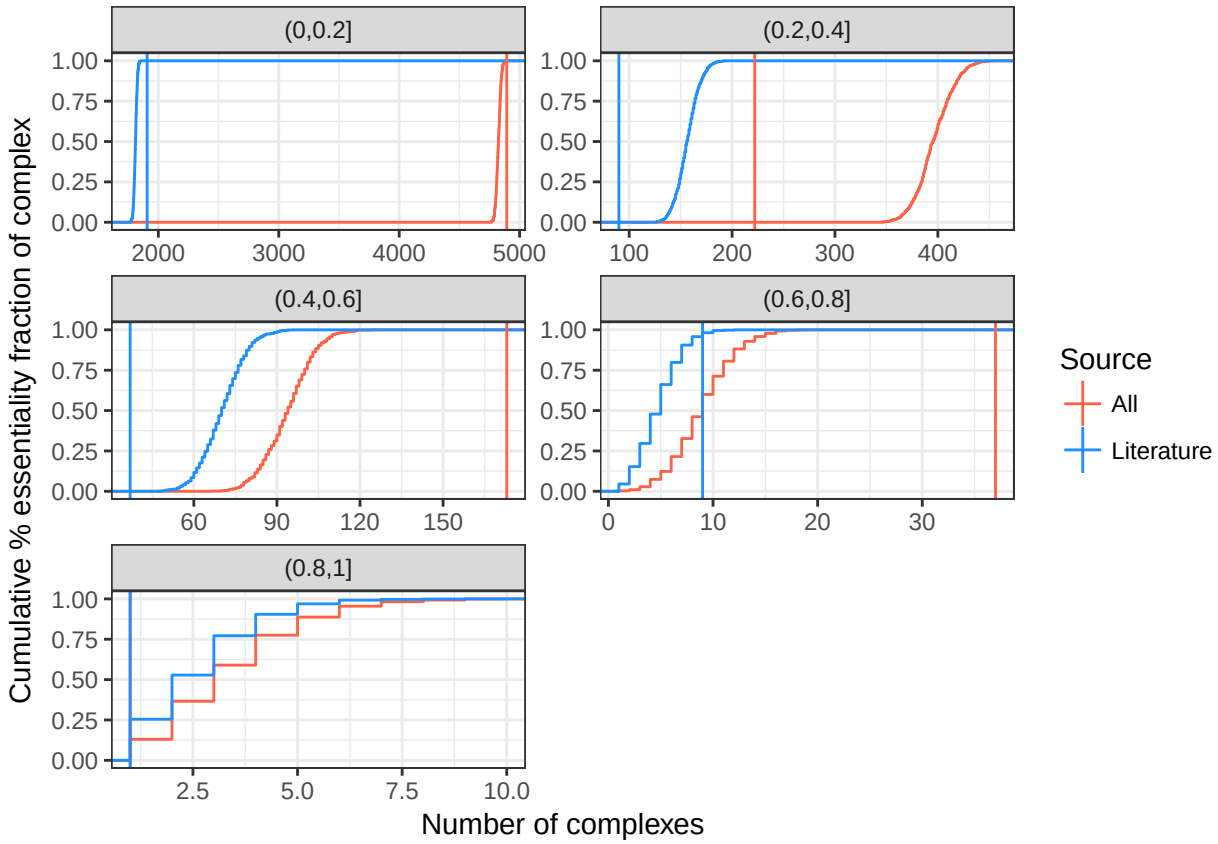


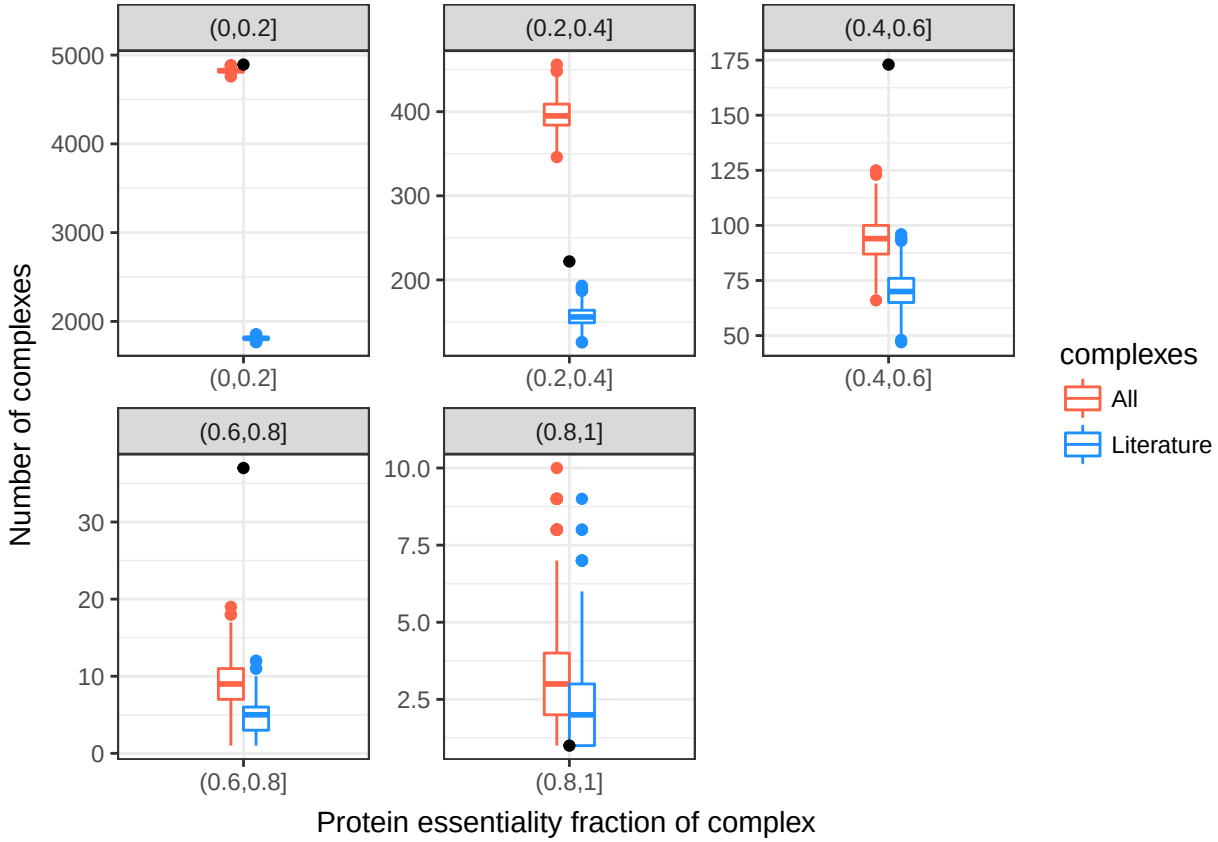
Significance test for each bin. We have a distribution for each bin so it is possible to calculate the p-value for each one in respect to the odserved value.

```
# Cumulative distribution estimation for each bin of the bootstraped values

d_essential_test <- d_BINS_Complex_essential %>% filter(key == "essentiality.consensus_n") %>% mutate(co

d_bootstrap_cumulative <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n") %>%
    group_by(value, n) %>% summarise(count = n()) %>% mutate(count_percent = count/sum(n), cumsum = cums
    cumsum_percent = cumsum/max(cumsum)) %>% mutate(complexes = "All")

d_bootstrap_boxplot <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n") %>% ungrou
    mutate(complexes = "All")

# Literature

d_essential_test_literature <- d_literature %>% filter(key == "essentiality.consensus_n") %>% mutate(cor

d_bootstrap_cumulative_literature <- d_literature %>% filter(key != "essentiality.consensus_n") %>% grou
    n) %>% summarise(count = n()) %>% mutate(count_percent = count/sum(n), cumsum = cumsum(count), cumsu
    mutate(complexes = "Literature")

d_bootstrap_boxplot_literature <- d_literature %>% filter(key != "essentiality.consensus_n") %>% ungroup
    mutate(complexes = "Literature")

# Merge together
```

```
d_essential_merged <- rbind(d_essential_test, d_essential_test_literature)
d_bootstrap_cumulative_merged <- rbind(d_bootstrap_cumulative, d_bootstrap_cumulative_literature)
d_bootstrap_boxplot_merged <- rbind(d_bootstrap_boxplot, d_bootstrap_boxplot_literature)

# P values
d_bootstrap_test_1 <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n" & value ==
    "(-0.0001,0.2]") %>% ungroup()

d_bootstrap_test_2 <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n" & value ==
    "(0.2,0.4]") %>% ungroup()

d_bootstrap_test_3 <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n" & value ==
    "(0.6,0.8]") %>% ungroup()

d_bootstrap_test_4_lit <- d_bootstrap_boxplot_literature %>% filter(key != "essentiality.consensus_n" &
    value == "(0.6,0.8]") %>% ungroup()

d_bootstrap_test_4 <- d_BINS_Complex_essential %>% filter(key != "essentiality.consensus_n" & value ==
    "(0.8,1]") %>% ungroup()

pval1 <- round(sum(d_bootstrap_test_1$n >= 4893)/length(d_bootstrap_test_1$n), 10)

pval2 <- sum(d_bootstrap_test_2$n >= 222)/length(d_bootstrap_test_2$n)

pval4_lit <- sum(d_bootstrap_test_4_lit$n >= 9)/length(d_bootstrap_test_4_lit$n)
```

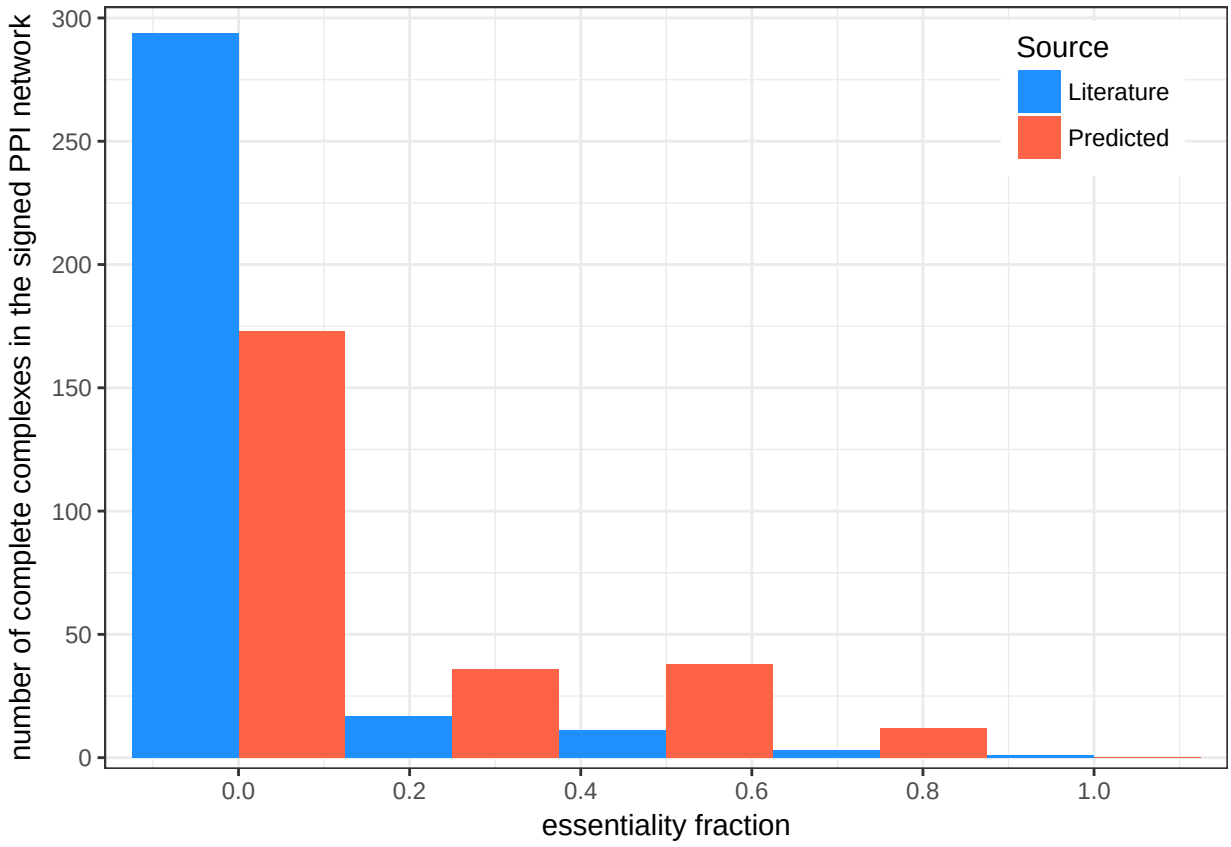



(0,0.2]
(0.2,0.4]
(0.4,0.6]
(0.6,0.8]
(0.8,1]
Cumulative % essentiality fraction of complex
Number of complexes
Source
All
Literature


(0,0.2]
(0.2,0.4]
(0.4,0.6]
(0.6,0.8]
(0.8,1]
Number of complexes
Protein essentiality fraction of complex
complexes
All
Literature

4.5 Protein complexes in the signed PPI network of drosophila

In this section we will focus on the complexes that are fully represented by the protein in the PPI network of drosophila.




```
## Saving 6.5 x 4.5 in image
```

```
dros_complete_complexes_included %>% filter(essentiality_fraction > 0.5) %>% nrow()
```

```
## [1] 49
```

Forty nine protein complexes, from the 585 complexes that are complete in the signed network of drosophila, consist of 50% or more essential proteins.

4.6 Network contraction with protein complexes

To contract a network with complexes from the protein - protein interaction network it is necessery to determine the which complexes to use. The rule we applied in this instance is to use only the complexes that all of their proteins are present in the network. This resulted in 585 complexes. Others can use a different threshold, like to use complexes that have >80% of their proteins present. Or take a completely different approach, like using GO annotation in the original network for the selection of complexes.

```
# complexes their containing proteins that are fully represented in the drosophila network
dros_net_complete_complex_protein <- as.data.frame(Protein_Complex_bipartite[Protein_Complex_bipartite$
    complete_complexes$Var1, ])

dros_net_complete_complex_protein %>% distinct(Protein) %>% nrow()
```

```
## [1] 1063
```

There are 1063 proteins that make the 585 complexes. About the 1/3 nodes from the original network.

```
# keep only the interactions between proteins that both exist in complete complexes.
dros_net <- read_csv(file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/dros_net.csv", col
```

```
## Parsed with column specification:
## cols(
##   ID1 = col_character(),
##   ID2 = col_character(),
##   Source_Type = col_character(),
##   weights = col_double(),
##   Coexpress_Development = col_double(),
##   sign_score_sign = col_character(),
##   coexpress_sign = col_character(),
##   compare = col_character(),
##   color = col_character(),
##   IDS = col_character(),
##   duplicate = col_logical(),
##   norm_weights = col_double(),
##   sign_weights = col_integer(),
##   abs_weights = col_double(),
##   ID1_consensus = col_character(),
##   ID2_consensus = col_character()
## )
```

```
dros_net_protein_in_complex <- as.data.frame(dros_net[dros_net$ID1 %in% dros_net_complete_complex_protei
    dros_net$ID2 %in% dros_net_complete_complex_protein$Protein, ])
```

There are also 2123 edges between these 1063 proteins. So again the 1/3 of the original network.

```
# Input
edgelist <- dros_net_protein_in_complex

complex_protein_bipartite <- dros_net_complete_complex_protein

# Processing
complexes_edgelist <- c()


for (i in 1:nrow(edgelist)) {

    protein_out <- edgelist[i, 1]
    protein_in <- edgelist[i, 2]
    edge_weight <- edgelist[i, 4]
    edge_IDs <- edgelist[i, 10]

    complexes_protein_out <- complex_protein_bipartite %>% filter(Protein == protein_out)  # Which comp

    complexes_protein_in <- complex_protein_bipartite %>% filter(Protein == protein_in)
    # Which complexes have this protein, the in edge.

    # Make all the possible combinations of the complexes but keeping them in different columns so the
    # oriantation of the original link remains.
    complexes_edgelist[[i]] <- expand.grid(ID1 = complexes_protein_out$Complex, ID2 = complexes_protein
```

```
        weight = edge_weight, original_edge = edge_IDs)

}

# It isn't that the for-loop is slow (which many R users mistakingly believe), it is the incremental
# collection of results into a data frame is slow and that is one of the steps the lapply method is
# avoiding.

# Output

complexes_edgelist <- do.call(rbind, complexes_edgelist)
```

Now that we created the edgelist but it is important to eliminate the redundant edges. These are:

1. Duplicated edges
2. Self loops

But we want to keep as much information as possible so we will treat positive and negative edges independently. More specifically, from all the redundant edges with the same direction, we will keep 2, one positive and one negative. The weight of the positive edge will be the normalized weight from all the positive edges. We' ll repeat that for the negative edges. Finally we will normalize all the weights with maximun weight, absolute value, in order to have all the edge weights in the [-1,1].

```
complexes_edgelist_processed <- complexes_edgelist %>% filter(!ID1 == ID2)

positive_e <- complexes_edgelist_processed %>% filter(weight >= 0) %>% group_by(ID1, ID2) %>% summarise

negative_e <- complexes_edgelist_processed %>% filter(weight <= 0) %>% group_by(ID1, ID2) %>% summarise

complexes_edgelist_final <- rbind(positive_e, negative_e)

complexes_edgelist_final <- complexes_edgelist_final %>% mutate(complex_edge = paste0(ID1, ",", ID2)) %
    mutate(norm_weights = mean_weights/max(abs(complexes_edgelist_final$mean_weights)))

# determine which are duplicates because there are negative and positive edges
complexes_edgelist_final <- complexes_edgelist_final %>% mutate(duplicates = duplicated(complex_edge) |
    duplicated(complex_edge, fromLast = T))

qqq <- complexes_edgelist_final %>% group_by(ID1, ID2) %>% summarise(count = n()) %>% group_by(count) %
    summarise(count_count = n())


### Counting links

posi <- complexes_edgelist_final %>% filter(norm_weights > 0) %>% nrow()

only_posi <- complexes_edgelist_final %>% filter(duplicates == "FALSE" & norm_weights > 0) %>% nrow()

nega <- complexes_edgelist_final %>% filter(norm_weights < 0) %>% nrow()

only_nega <- complexes_edgelist_final %>% filter(duplicates == "FALSE" & norm_weights < 0) %>% group_by
    ID2) %>% distinct() %>% nrow()


total_both <- complexes_edgelist_final %>% filter(duplicates == "TRUE") %>% nrow()
```

Table 4.2: Signed edges of complexes network

Type	Unique edges	Both signs	Total
Positive edges	11872	2397	14269
Negative edges	3684	2397	6081
Total	15556	4794	20350

```
# which edges appear with negative AND positive weights
unique_both <- complexes_edgelist_final %>% filter(duplicates == "TRUE") %>% group_by(complex_edge) %>%
    distinct() %>% nrow()
```

4.7 Protein complexes network

```
# complexes_edgelist_final
```

Chapter 5

Gene ontology and Functional enrichment analysis

Have to do gene annotation and after that, biological process annotation for the functional categories in GO. The latter has to be accompanied with a statistical test.

In cytoscape app Bingo is possible to make gene annotation from GO. I followed the instructions from the tutorial and managed to annotate the drosophila genes. It performs an hypergeometric test for the selection and the output is a graph (and a table) with the GO terms and their parent-child directed connections accompanied with p-value.

```
## this chunk doesn't run because: eval=FALSE export data for the Cytoscape app Bingo.

flybase_annotation <- read.delim(file = "Data/gene_association.fb.gz.txt", header = F, sep = "\t")  #

dros_net_FLYBASE <- transform(dros_net, ID1 = sprintf("FLYBASE:%s", ID1))  # needs the database name be
dros_net_FLYBASE <- transform(dros_net, ID2 = sprintf("FLYBASE:%s", ID2))
ONLYessential_genes_net2 <- read_csv(file = "../Thesis_Essentiality_Drosophila_Signed_Network/Tables/ONL
    col_names = TRUE)

colnames(ONLYessential_genes_net2)[colnames(ONLYessential_genes_net2) == "ID1"] <- "source"
colnames(ONLYessential_genes_net2)[colnames(ONLYessential_genes_net2) == "ID2"] <- "target"
ONLYessential_genes_net2 <- transform(ONLYessential_genes_net2, source = sprintf("FLYBASE:%s", source))
ONLYessential_genes_net2 <- transform(ONLYessential_genes_net2, target = sprintf("FLYBASE:%s", target))



library(mygene)
ssss <- tail(dros_net$ID1)
# qqqq <- getGenes(ssss,fields = 'all',return.as = 'DataFrame')

# Export network to Cytoscape to use BINGO for gene enrichment.
dros_net_cytoscape <- dros_net[, c(1, 2, 12, 15, 16)]
dros_net_cytoscape <- transform(dros_net_cytoscape, ID1 = sprintf("FLYBASE:%s", ID1))
dros_net_cytoscape <- transform(dros_net_cytoscape, ID2 = sprintf("FLYBASE:%s", ID2))

names(dros_net_cytoscape) <- c("source", "target", "norm_weights", "source_essentiality", "target_essen
write.table(dros_net_cytoscape, file = "dros_net_cytoscape.txt", sep = "\t", col.names = T, row.names =
    quote = F)
```

Table 5.1: Number of annotations for each ontology from GO

Ontology	Number of annotations
BP	2858
CC	2655
MF	2721
None	317

Table 5.2: Number of ontologies which the genes were annotated to

Number of ontologies	Number of genes
0	317
1	214
2	443
3	2378

5.1 Gene ontology annotation

The Bioconductor has packages in R that do Gene Ontology annotation in genes of many organisms with different types of database IDs.

```
library(AnnotationDbi)
library(org.Dm.eg.db)
library(GO.db)

columnsDB <- columns(org.Dm.eg.db)
keysDB <- head(keys(org.Dm.eg.db, keytype = "FLYBASE"))
dros_IDs_Flybase <- as.vector(dros_network_genes$locus)
dros_network_genes_ANNOTATION <- AnnotationDbi::select(org.Dm.eg.db, keys = dros_IDs_Flybase, columns =
    "GENENAME", "PATH", "ONTOLOGY"), keytype = "FLYBASE")

# What is evidence? Why there are many GO ids to single Fly ID?
```

From the 3352 proteins of the network the 317 are not included in Gene Ontology. For some genes there are multiple annotations because there exist three ontologies, Biological Process, Molecular Function and Cellular Component, and also 15 different evidence methods.

5.2 Identifier conversion with Bioconductor

In the bioconductor package org.Dm.eg.db there are multiple functions, one for each corresponding identifier. In this particular instance we will convert ids from Flybase to Entrez gene.

```
## Find different IDs of Drosophila

x <- org.Dm.egFLYBASE
# Get the entrez gene identifiers that are mapped to a Flybase ID
mapped_genes <- mappedkeys(x)

flybase_to_entrez <- as.list(x[mapped_genes])
sum(sapply(X = flybase_to_entrez, FUN = length)) == length(flybase_to_entrez)  # it's true so the list
```

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

```
flybase_to_entrez_df <- as.data.frame(do.call(rbind, flybase_to_entrez))
flybase_to_entrez_df <- tibble::rownames_to_column(df = flybase_to_entrez_df, var = "Entrez_ID")
```

Finally we have a dataframe with 2 columns with the matching identifiers.

Next we will subset the genes from the network with the entrez gene identifiers because these are recognisable from the topGO package for the functional enrichment analysis.

```
all_drosophila_genes_DATABASE <- as.character(flybase_to_entrez_df$Entrez_ID)
all_genes <- as.character(dros_network_genes$locus)  # Gene Universe
all_net_strong_comp <- as.vector(as.matrix(read_csv(file = "../Thesis_Essentiality_Drosophila_Signed_Ne
    col_names = F)))
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character()
## )
```

```
intresting_genes_Essential_strongly_con <- as.vector(as.matrix(read_csv(file = "../Thesis_Essentiality_
    col_names = F)))  # essential strongly connected
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character()
## )
```

```
intresting_genes_Essential <- as.vector(as.matrix(read_csv(file = "../Thesis_Essentiality_Drosophila_Si
    col_names = F)))
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character()
## )
```

```
dros_net_flybase_to_entrez_df <- flybase_to_entrez_df %>% filter(V1 %in% all_genes)  # a lot are missin
all_strong_dros_net_flybase_to_entrez_df <- flybase_to_entrez_df %>% filter(V1 %in% all_net_strong_comp
essential_dros_net_flybase_to_entrez_df <- flybase_to_entrez_df %>% filter(V1 %in% intresting_genes_Ess

# vectors
universe_dros_net_entrez <- as.factor(dros_net_flybase_to_entrez_df$Entrez_ID)
essential_dros_net_entrez <- as.factor(essential_dros_net_flybase_to_entrez_df$Entrez_ID)
strong_dros_net_entrez <- as.factor(all_strong_dros_net_flybase_to_entrez_df$Entrez_ID)
```

There are some identifiers that did not match. These are 96 genes. All the essential 20 genes that form a strongly connected component are matched. For the big strongly connected component 4 genes weren't matched.

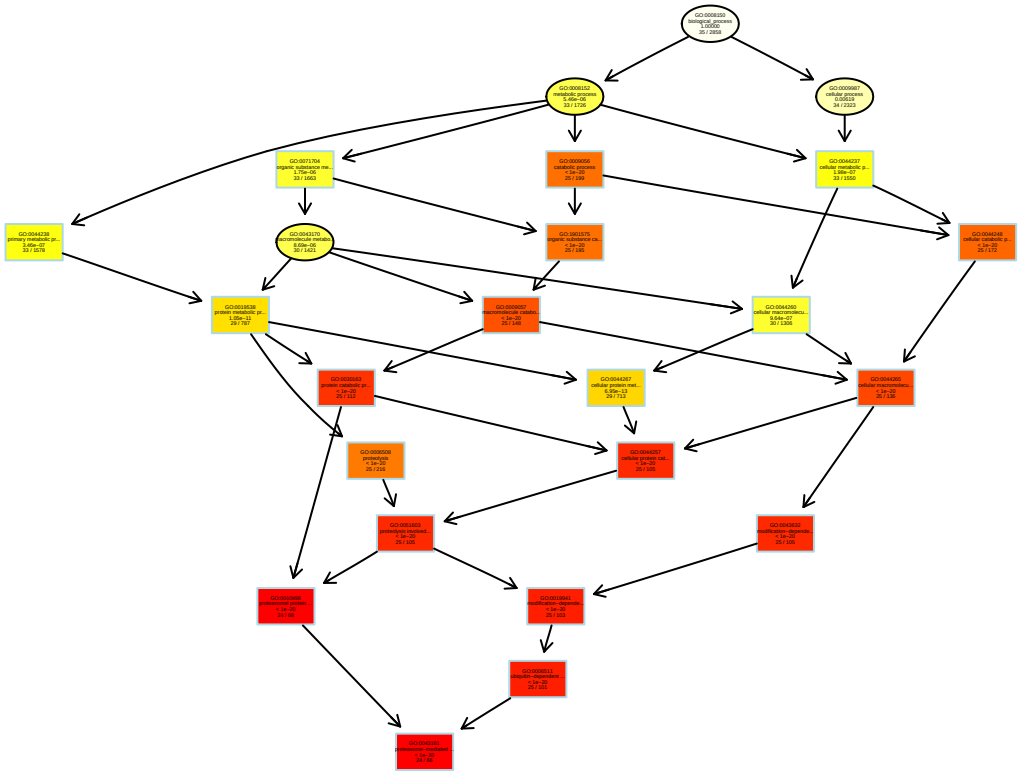
5.3 Singular enrichment analysis

In order to examine which gene ontology terms are overrepresented in the genes of our network we have to do a statistic test. Most commonly used tests is the Fisher's exact test, the chi squere test and the binomial test. We will use the Fisher' s exact test for the Biological Process component of Gene Ontology.

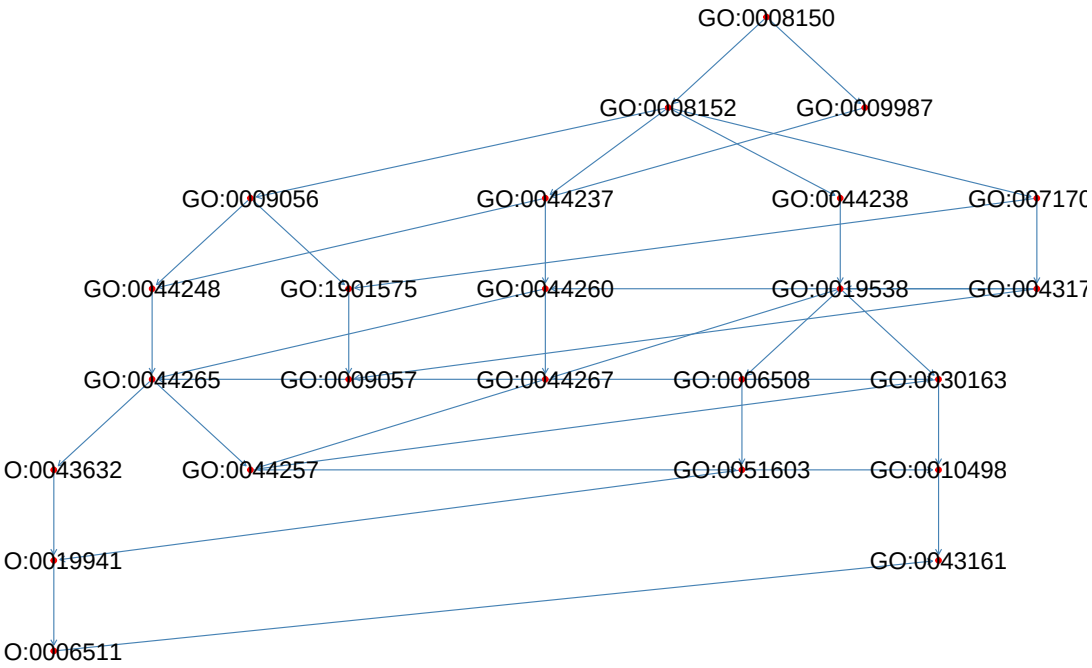
All these test share the same assumption for the null hypothesis, that the probabilities for the selection of each gene are equal (Rivals et al. 2007). But it turns out that thay are not because the structure of gene

ontology bipartite network of genes and gene terms has a heavy tail distribution and hence these tests are biased to high degree terms (Glass and Girvan 2014).

5.3.1 The essential cluster



```
## $dag
## A graphNEL graph with directed edges
## Number of Nodes = 24
## Number of Edges = 39
##
## $complete.dag
## [1] "A graph with 24 nodes."
```



5.3.1.1 Functional Bipartite Network

After the enrichment of the significantly represented gene ontology terms we can construct a bipartite network with the genes and the GO terms. That way it is possible to decipher functional relationships between GO terms as well as genes.

```
# sdsds <- dros_network_genes_ANNOTATION[dros_network_genes_ANNOTATION$FLYBASE %in%
# intresting_genes_Essential,]

# ################################## Bipartite network
allRes_001_sig_IDS <- allRes_001_sig[, "GO.ID"]  # the significant GO ids.
genesInTerm_essential_strong <- genesInTerm(sampleGOdata, allRes_001_sig_IDS)  # which genes where anno

genesInTerm_essential_strong <- lapply(genesInTerm_essential_strong, paste0, collapse = ",")  # Concarn

sig_GO_toEntrez_Essential <- as.data.frame(do.call(rbind, genesInTerm_essential_strong)) %>% tibble::rou
    var = "GO_ID") %>% mutate(., V1 = strsplit(as.character(V1), ",")) %>% unnest(., V1)

colnames(sig_GO_toEntrez_Essential) <- c("GO_ID", "Entrez_ID")

bipartite_GO_Gene_essential_strong <- sig_GO_toEntrez_Essential %>% subset(., Entrez_ID %in% essential_

# Attributes dataframe
essential_strong_gene_attributes <- left_join(dros_network_genes_ANNOTATION, dros_net_flybase_to_entrez_
    by = c(FLYBASE = "V1")) %>% subset(., Entrez_ID %in% unique(bipartite_GO_Gene_essential_strong$Entre
    dplyr::select(., c(GENENAME, Entrez_ID)) %>% group_by(GENENAME, Entrez_ID) %>% distinct() %>% ungrou

essential_strong_gene_attributes <- essential_strong_gene_attributes[, c(2, 1)] %>% mutate(., Type = "En
colnames(essential_strong_gene_attributes) <- c("ID", "Term", "Type")

allRes_001_sig_ID_Term <- allRes_001_sig[, c(1, 2)] %>% mutate(., Type = "GO_ID")
colnames(allRes_001_sig_ID_Term) <- c("ID", "Term", "Type")
```

```
bipartite_GO_Gene_essential_strong_attributes <- as.data.frame(do.call("rbind", list(allRes_001_sig_ID_
    essential_strong_gene_attributes)))

### Complete the edgelist ###
bipartite_GO_Gene_essential_strong_edgelist_full <- left_join(bipartite_GO_Gene_essential_strong, bipar
    by = c(GO_ID = "ID")) %>% left_join(., bipartite_GO_Gene_essential_strong_attributes, by = c(Entrez_
    left_join(., dros_net_flybase_to_entrez_df, by = c(Entrez_ID = "Entrez_ID"))
colnames(bipartite_GO_Gene_essential_strong_edgelist_full) <- c("GO_ID", "Entrez_ID", "Term_GO_ID", "Ty
    "Term_Entrez_ID", "Type_Entrez_ID", "FLYBASE")

#### Load the network ####
bipartite_GO_Gene_essential_strong_g <- graph_from_data_frame(bipartite_GO_Gene_essential_strong, direc
    vertices = bipartite_GO_Gene_essential_strong_attributes)

bipartite_GO_Gene_essential_strong_g_comp <- decompose.graph(bipartite_GO_Gene_essential_strong_g, min.
bipartite_GO_Gene_essential_strong_g_giant <- bipartite_GO_Gene_essential_strong_g_comp[[1]]

library(bipartite)
```

```
## Loading required package: vegan
```

```
## Loading required package: permute
```

```
##
## Attaching package: 'permute'
```

```
## The following object is masked from 'package:igraph':
##
##     permute
```

```
## Loading required package: lattice
```

```
## This is vegan 2.4-2
```

```
##
## Attaching package: 'vegan'
```

```
## The following object is masked from 'package:igraph':
##
##     diversity
```

```
##  This is bipartite 2.08
##  For latest changes see versionlog in  ?"bipartite-package".
##  For citation see: citation("bipartite").
##  Have a nice time plotting and analysing two-mode networks.
```

```
##
## Attaching package: 'bipartite'
```

```
## The following object is masked from 'package:vegan':
##
##     nullmodel
```

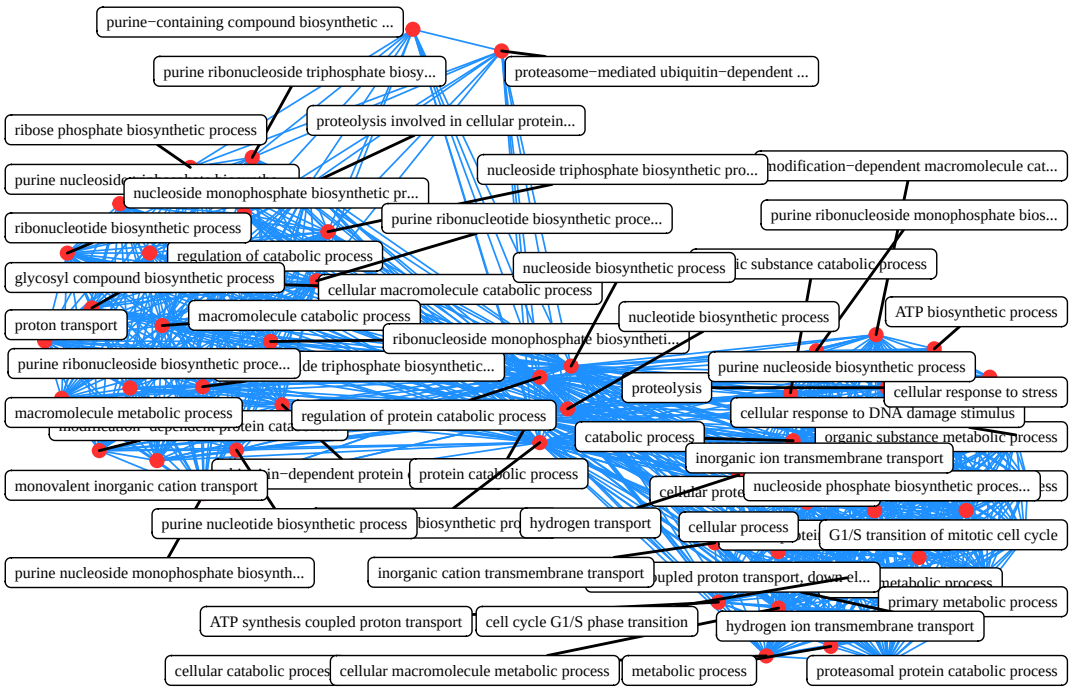
```
## The following object is masked from 'package:igraph':
##
##     strength
```

```
# All sig annotations without correction
adjacency_bipartite_GO_Gene_essential_strong <- bipartite_GO_Gene_essential_strong %>% mutate(value = 1
```

```
    spread(., key = Entrez_ID, value = value, fill = 0) %>% tibble::remove_rownames(.) %>% tibble::colu
    var = "GO_ID") %>% as.matrix()

Functional_net_GO_essential_cluster_adjacency <- as.one.mode(adjacency_bipartite_GO_Gene_essential_stro
    project = "lower")

Functional_net_GO_essential_cluster <- graph_from_adjacency_matrix(Functional_net_GO_essential_cluster_
    mode = "undirected", weighted = TRUE)
V(Functional_net_GO_essential_cluster)$Term <- allRes_001_sig$Term
V(Functional_net_GO_essential_cluster)$Bonferroni <- allRes_001_sig$p_adjust_Bonferroni
```

If we use only the Bonferroni adjastement for the p-value we take 23 terms and the functional network is fully connected. So we didn't used this filter for the functional network.

```
# Bonferroni sig annotations
GO_ids_bonferroni <- allRes_001_sig[allRes_001_sig$p_adjust_Bonferroni < 0.05, ]

adjacency_bipartite_GO_Gene_essential_strong_bonf <- bipartite_GO_Gene_essential_strong %>% filter(.,
    GO_ID %in% GO_ids_bonferroni$GO.ID) %>% mutate(value = 1) %>% spread(., key = Entrez_ID, value = va
    fill = 0) %>% tibble::remove_rownames(.) %>% tibble::column_to_rownames(., var = "GO_ID") %>% as.ma

Functional_net_GO_essential_cluster_adjacency_bonf <- as.one.mode(adjacency_bipartite_GO_Gene_essential
    project = "lower")

Functional_net_GO_essential_cluster <- graph_from_adjacency_matrix(Functional_net_GO_essential_cluster_
    mode = "undirected", weighted = TRUE)
V(Functional_net_GO_essential_cluster)$Term <- allRes_001_sig$Term
V(Functional_net_GO_essential_cluster)$Bonferroni <- allRes_001_sig$p_adjust_Bonferroni
```

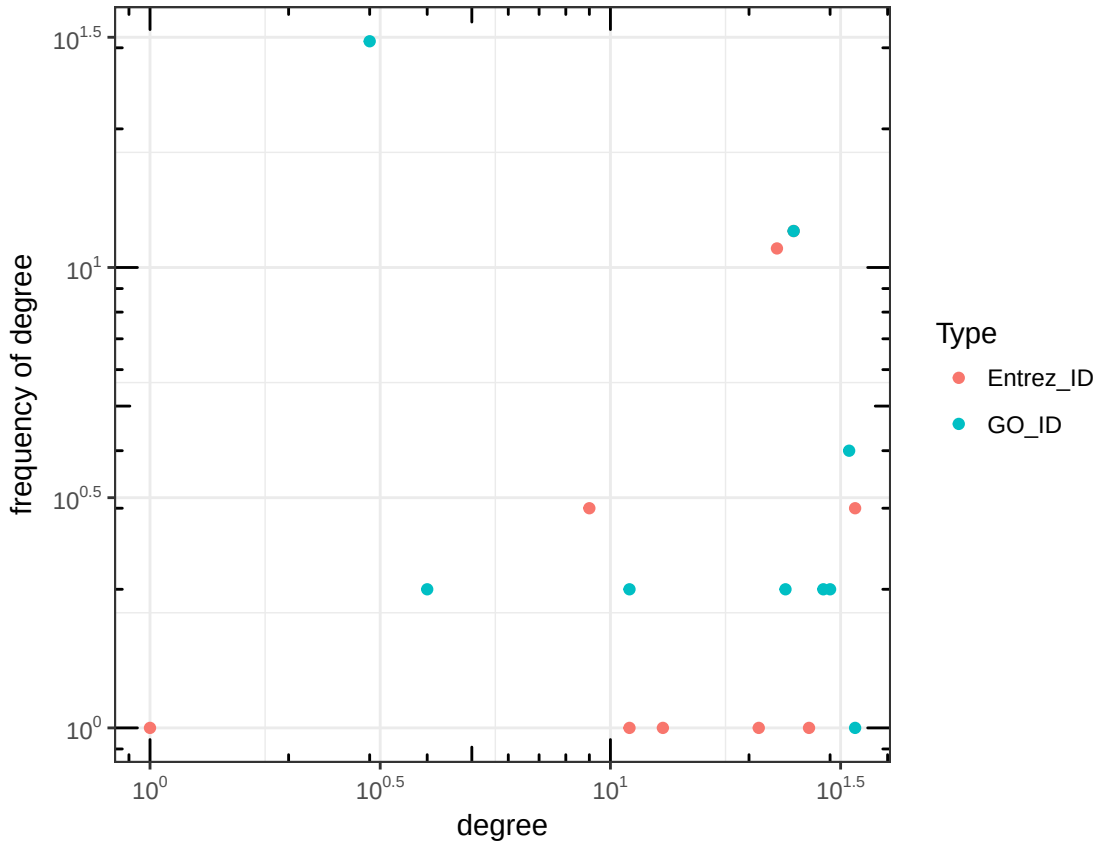


The original gene list had 36 essential genes that their proteins were positively interacting with each other. After the gene set enrichement we are left with 34 genes because the genes 246653, 36571 weren't enriched to a significant GO term.

After a quick observation it's apparent that about 3 functional groups of essential genes are linked together with positive links. These are :

1. proteolysis, protein catabolic processes
2. ATP synthesis, nucleotide synthesis
3. Response to stress

```
degree_bipartite_GO_Gene_essential_strong_g <- as.data.frame(igraph::degree(bipartite_GO_Gene_essential_
    tibble::rownames_to_column(., var = "ID")
colnames(degree_bipartite_GO_Gene_essential_strong_g) <- c("ID", "Degree")

bipartite_GO_Gene_essential_strong_attributes <- left_join(bipartite_GO_Gene_essential_strong_attributes
    degree_bipartite_GO_Gene_essential_strong_g, by = c(ID = "ID"))

bipartite_GO_Gene_essential_strong_attributes_all <- bipartite_GO_Gene_essential_strong_attributes %>%
    group_by(Degree) %>% summarise(count = n())
bipartite_GO_Gene_essential_strong_attributes_group <- bipartite_GO_Gene_essential_strong_attributes %>%
    group_by(Degree, Type) %>% summarise(count = n())

ggplot() + # geom_point(data = bipartite_GO_Gene_essential_strong_attributes_all, aes(x = Degree, y =
# count,color='All' ))+
geom_point(data = bipartite_GO_Gene_essential_strong_attributes_group, aes(x = Degree, y = count, color
    # ggtitle('Degree Distribution GO Terms and Genes network')+
scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = trans_format("log10", math_form
    scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x), labels = trans_format("log10", math_
    annotation_logticks(sides = "trbl") + coord_fixed(ratio = 1) + labs(x = "degree", y = "frequency of
    # scale_colour_manual(values = c('Complexes'='lightblue','Proteins'='yellow3'))+
theme_bw()
```

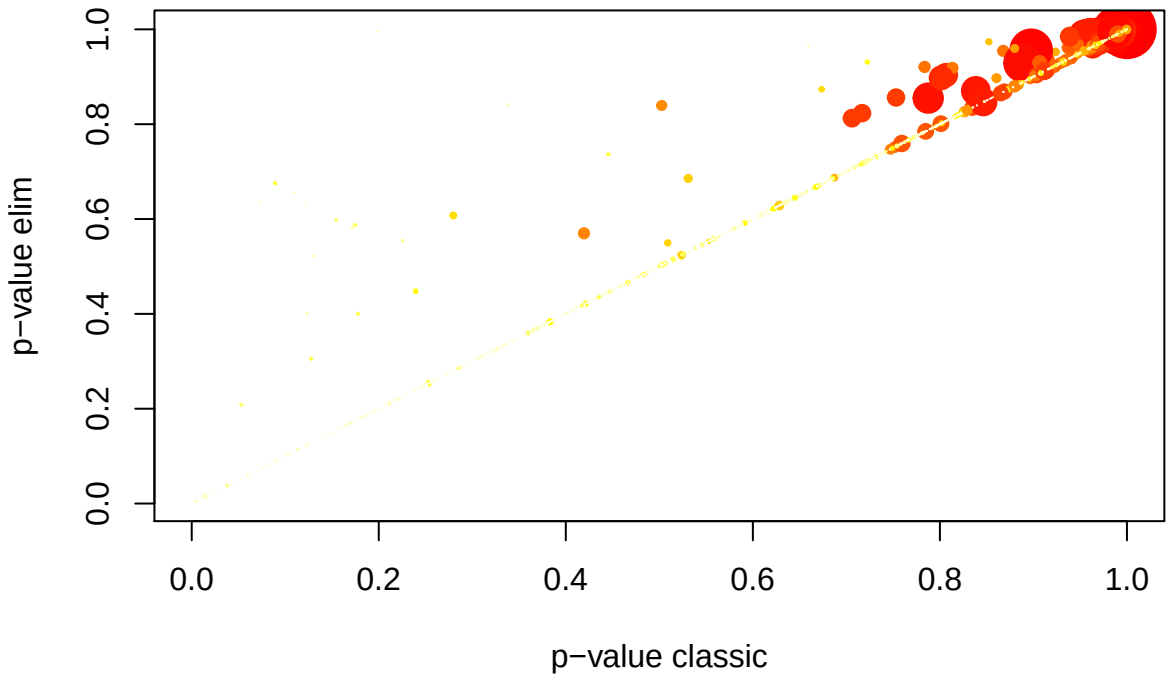


```
ggsave("Bipartite_GO_Term_Genes_Degree_Distribution_.pdf", width = 7, height = 11, units = "cm", plot =
    device = "pdf", dpi = 150, path = "Figures/")
```

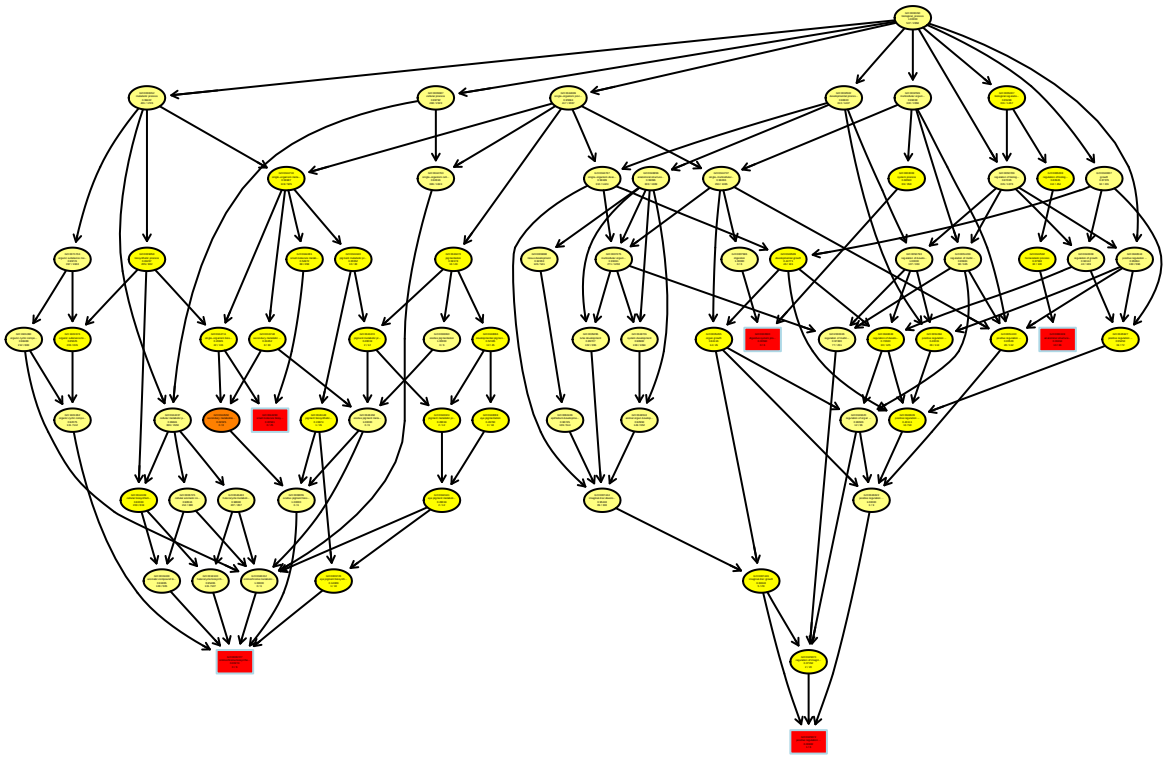
5.3.2 Strongly connected component

```
##
## Building most specific GOs .....
##  ( 2784 GO terms found. )
##
## Build GO DAG topology ..........
##  ( 5110 GO terms and 11590 relations. )
##
## Annotating nodes ...............
##  ( 2858 genes annotated to the GO terms. )
##
##           -- Classic Algorithm --
##
##       the algorithm is scoring 2309 nontrivial nodes
##       parameters:
##           test statistic: fisher
##
##           -- Classic Algorithm --
##
##       the algorithm is scoring 2517 nontrivial nodes
##       parameters:
##           test statistic: ks
##           score order: increasing
##
##           -- Elim Algorithm --
##
##       the algorithm is scoring 2517 nontrivial nodes
##       parameters:
##           test statistic: ks
##           cutOff: 0.01
##           score order: increasing
##
##   Level 18:  1 nodes to be scored    (0 eliminated genes)
##
##   Level 17:  1 nodes to be scored    (0 eliminated genes)
##
##   Level 16:  2 nodes to be scored    (0 eliminated genes)
##
##   Level 15:  17 nodes to be scored   (0 eliminated genes)
##
##   Level 14:  34 nodes to be scored   (0 eliminated genes)
```

```
##
##   Level 13:  65 nodes to be scored   (0 eliminated genes)
##
##   Level 12:  97 nodes to be scored   (0 eliminated genes)
##
##   Level 11:  141 nodes to be scored  (0 eliminated genes)
##
##   Level 10:  212 nodes to be scored  (0 eliminated genes)
##
##   Level 9:   318 nodes to be scored  (6 eliminated genes)
##
##   Level 8:   344 nodes to be scored  (12 eliminated genes)
##
##   Level 7:   375 nodes to be scored  (12 eliminated genes)
##
##   Level 6:   374 nodes to be scored  (39 eliminated genes)
##
##   Level 5:   302 nodes to be scored  (51 eliminated genes)
##
##   Level 4:   163 nodes to be scored  (140 eliminated genes)
##
##   Level 3:   52 nodes to be scored   (140 eliminated genes)
##
##   Level 2:   18 nodes to be scored   (140 eliminated genes)
##
##   Level 1:   1 nodes to be scored    (140 eliminated genes)
```



```
##             Annotated Significant Expected      elim    classic
## GO:0045570         20           2     3.76 0.3715187 0.4149973
```



```
## $dag
## A graphNEL graph with directed edges
## Number of Nodes = 74
## Number of Edges = 144
##
## $complete.dag
## [1] "A graph with 74 nodes."
```

```
## tGO_elim_5_all  --- no of nodes:  74
```

5.3.3 All network

```
##
## Building most specific GOs .....
```

```
##  ( 4100 GO terms found. )
```

```
##
## Build GO DAG topology ..........
```

```
##  ( 6620 GO terms and 14960 relations. )
```

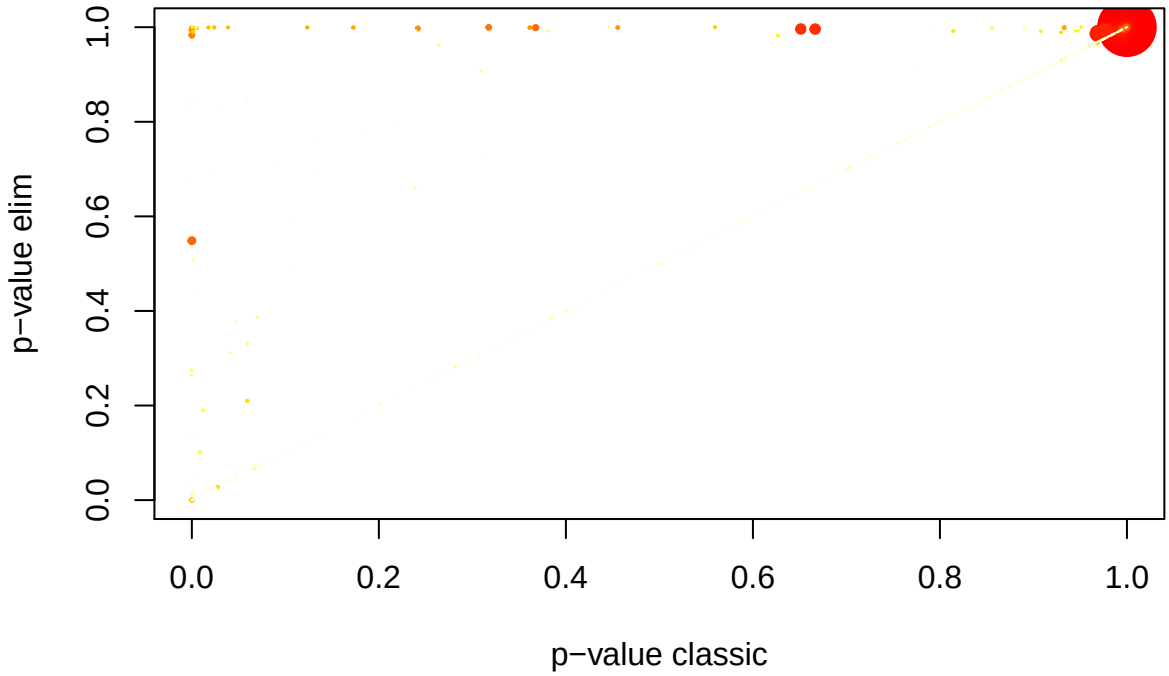
```
##
## Annotating nodes ...............
```

```
##  ( 12699 genes annotated to the GO terms. )
```

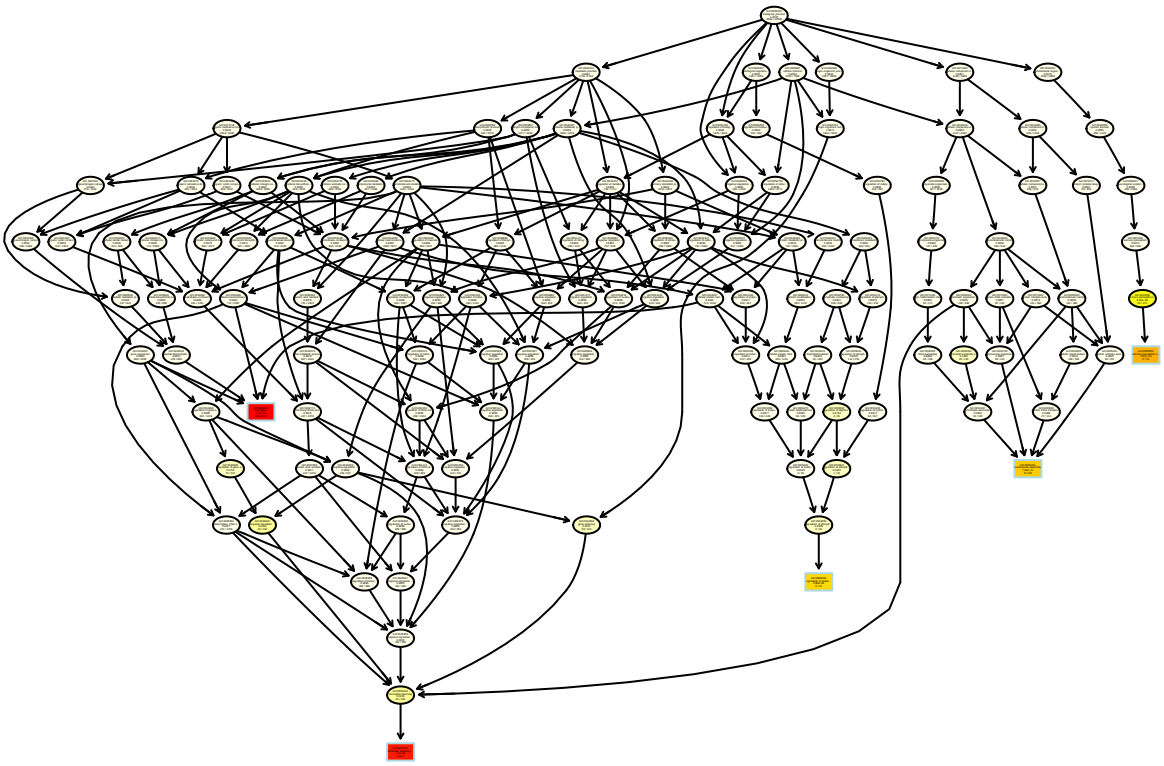
```
##
##           -- Classic Algorithm --
##
```

```
##       the algorithm is scoring 3510 nontrivial nodes
##       parameters:
##           test statistic: fisher
##
##           -- Classic Algorithm --
##
##       the algorithm is scoring 3629 nontrivial nodes
##       parameters:
##           test statistic: ks
##           score order: increasing
##
##           -- Elim Algorithm --
##
##       the algorithm is scoring 3629 nontrivial nodes
##       parameters:
##           test statistic: ks
##           cutOff: 0.01
##           score order: increasing
##
##   Level 18:  1 nodes to be scored    (0 eliminated genes)
##
##   Level 17:  1 nodes to be scored    (0 eliminated genes)
##
##   Level 16:  8 nodes to be scored    (0 eliminated genes)
##
##   Level 15:  31 nodes to be scored   (0 eliminated genes)
##
##   Level 14:  58 nodes to be scored   (0 eliminated genes)
##
##   Level 13:  98 nodes to be scored   (41 eliminated genes)
##
##   Level 12:  151 nodes to be scored  (50 eliminated genes)
##
##   Level 11:  230 nodes to be scored  (50 eliminated genes)
##
##   Level 10:  350 nodes to be scored  (62 eliminated genes)
##
##   Level 9:   498 nodes to be scored  (88 eliminated genes)
##
##   Level 8:   511 nodes to be scored  (214 eliminated genes)
##
##   Level 7:   544 nodes to be scored  (1177 eliminated genes)
##
##   Level 6:   514 nodes to be scored  (1345 eliminated genes)
##
##   Level 5:   369 nodes to be scored  (1544 eliminated genes)
```

```
##
##   Level 4:   192 nodes to be scored  (1950 eliminated genes)


##
##   Level 3:   54 nodes to be scored   (1950 eliminated genes)


##
##   Level 2:   18 nodes to be scored   (1950 eliminated genes)


##
##   Level 1:   1 nodes to be scored    (1950 eliminated genes)
```




```
##            Annotated Significant Expected      elim   classic
## GO:0006338        98          53    22.06 0.9966889 0.9978844
## GO:0006810      1695         518   381.47 0.9950343 0.9989680
## GO:0009628       441         154    99.25 0.9984316 0.9990459
## GO:0009889      1141         480   256.79 0.9998150 0.9998477
## GO:0010556      1087         464   244.64 0.9997606 0.9998001
## GO:0031326      1140         480   256.57 0.9998141 0.9998469
## GO:0032268       438         212    98.58 0.9974048 0.9975236
## GO:0032501      4899        1556  1102.55 0.9978611 0.9980800
## GO:0042221       999         346   224.83 0.9952123 0.9987017
## GO:0051179      2179         720   490.40 0.9997651 1.0000000
## GO:0051234      1767         555   397.68 0.9979787 0.9999951
## GO:0051246       467         228   105.10 0.9976844 0.9977876
## GO:2000112      1084         462   243.96 0.9997573 0.9997972
```



```
## $dag
## A graphNEL graph with directed edges
## Number of Nodes = 127
## Number of Edges = 267
##
## $complete.dag
## [1] "A graph with 127 nodes."
```

```
## tGO_elim_5_all  --- no of nodes:  127
```

This has to be done also for big strongly connected component, the essential genes cluster.

5.4 Functional Analysis of essential cluster

The essential protein cluster.

```
tripartite_net_GO_GENE_COMPLEX <- left_join(bipartite_GO_Gene_essential_strong_edgelist_full, Protein_C
    by = c(FLYBASE = "Protein")) %>% left_join(., dros_complexes, by = c(Complex = "V1"))
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
tripartite_net_GO_GENE_COMPLEX <- tripartite_net_GO_GENE_COMPLEX %>% group_by(., Complex, FLYBASE) %>%
    distinct() %>% group_by(., Complex) %>% mutate(., Complex_protein = n()) %>% group_by(Complex, Compl
    distinct() %>% right_join(., tripartite_net_GO_GENE_COMPLEX, by = c(Complex = "Complex"))

tripartite_net_GO_GENE_COMPLEX <- tripartite_net_GO_GENE_COMPLEX %>% mutate(., Missing_Complex_proteins
    Complex_protein) %>% mutate(., Missing_Complex_fraction = (V2 - Complex_protein)/V2) %>% ungroup()

tripartite_net_GO_GENE_COMPLEX <- tripartite_net_GO_GENE_COMPLEX[, c(1, 9, 4, 3, 10, 21, 22, 2, 12, 5,
```

```
    7, 6, 8, 11, 13, 14, 15, 16, 17, 18, 19, 20)]

colnames(tripartite_net_GO_GENE_COMPLEX)[5] <- "Total_Complex_Proteins"
colnames(tripartite_net_GO_GENE_COMPLEX)[9] <- "Term_Complex"

tripartite_net_GO_GENE_COMPLEX_literature <- tripartite_net_GO_GENE_COMPLEX %>% filter(., V3 == "Litera
setdiff(unique(tripartite_net_GO_GENE_COMPLEX$FLYBASE), unique(tripartite_net_GO_GENE_COMPLEX_literatur
```

```
## [1] "FBgn0031549"
```

```
write.table(x = tripartite_net_GO_GENE_COMPLEX_literature, file = "../Thesis_Essentiality_Drosophila_Si
    sep = ",", col.names = TRUE, row.names = FALSE)
```

The gene FBgn0031549 is included in 2 protein complexes which are predicted by NetworkBlast. This is the reason that we lose that gene when we filter the list for the literature complexes only.

```
bipartite_GO_Gene_essential_strong_g_giant <- as_bipartite(bipartite_GO_Gene_essential_strong_g_giant)
```

References

Glass, Kimberly, and Michelle Girvan. 2014. "Annotation enrichment analysis: an alternative method for evaluating the functional properties of gene sets." *Sci. Rep.* 4: 4191. doi:10.1038/srep04191.

Gradshteyn, I.S., and I.M. Ryzhik. 2007. *Table of Integrals, Series, and Products*. Edited by Alan Jeffrey and Daniel Zwillinger. 7th Editio. San Diego: Academic Press, Inc.

Kauffman, Stuart A. 1993. *The Origins of Order Self-Organization and Selection in Evolution*. New York: Oxford University Press.

Knuth, Donald E. 1994. *GraphBase: A Platform for Combinatorial Computing*. Standford University: ACM Press.

Li, Min, Hanhui Zhang, Jian-xin Wang, and Yi Pan. 2012. "A new essential protein discovery method based on the integration of protein-protein interaction and gene expression data." *BMC Syst. Biol.* 6 (1): 15. doi:10.1186/1752-0509-6-15.

Mehrotra, Ravi, Vikram Soni, and Sanjay Jain. 2009. "Diversity sustains an evolving network." *J. R. Soc. Interface* 6 (38): 793–9. doi:10.1098/rsif.2008.0412.

Nghe, Philippe, Wim Hordijk, Stuart A Kauffman, Sara I Walker, Francis J Schmidt, Harry Kemble, Jessica A M Yeates, and Niles Lehman. 2015. "Prebiotic network evolution: six key parameters." *Mol. Biosyst.* 11 (12). Royal Society of Chemistry: 3206–17. doi:10.1039/c5mb00593k.

Ou-Yang, Le, Dao Qing Dai, and Xiao Fei Zhang. 2015. "Detecting Protein Complexes from Signed Protein-Protein Interaction Networks." *IEEE/ACM Trans. Comput. Biol. Bioinforma.* 12 (6): 1333–44. doi:10.1109/TCBB.2015.2401014.

Rivals, Isabelle, Léon Personnaz, Lieng Taing, and Marie Claude Potier. 2007. "Enrichment or depletion of a GO category within a class of genes: Which test?" *Bioinformatics* 23 (4): 401–7. doi:10.1093/bioinformatics/btl633.

Ryan, Colm J., Nevan J. Krogan, Pádraig Cunningham, and Gerard Cagney. 2013. "All or nothing: Protein complexes flip essentiality between distantly related eukaryotes." *Genome Biol. Evol.* 5 (6): 1049–59. doi:10.1093/gbe/evt074.

Sole, R. V. 2011. *Phase Transitions*. Princeton: Princeton University Press.

Stott, Iain, Stuart Townley, David Carslake, and David J Hodgson. 2010. "On reducibility and ergodicity of population projection matrix models." *Methods Ecol. Evol.* 1 (3): 242–52. doi:10.1111/j.2041-210X.2010.00032.x.

Vinayagam, A, Y Hu, M Kulkarni, C Roesel, R Sopko, S E Mohr, and N Perrimon. 2013. "Protein complex-based analysis framework for high-throughput data sets." *Sci Signal* 6 (264): rs5. doi:10.1126/scisignal.2003629.

Vinayagam, Arunachalam, Jonathan Zirin, Charles Roesel, Yanhui Hu, Bahar Yilmazel, Anastasia A. Samsonova, Ralph A. Neumüller, Stephanie E. Mohr, and Norbert Perrimon. 2014. "Integrating protein-

protein interaction networks with phenotypes reveals signs of interactions." *Nat Methods* 11 (1): 94–99. doi:doi:10.1038/nmeth.2733.

Zotenko, Elena, Julian Mestre, Dianne P. O'Leary, and Teresa M. Przytycka. 2008. "Why do hubs in the yeast protein interaction network tend to be essential: Reexamining the connection between the network topology and essentiality." *PLoS Comput. Biol.* 4 (8). doi:10.1371/journal.pcbi.1000140.