NEArender

Ashwini Jeggari & Andrey Alexeyenko

21 March 2018

Contents

1 Overview

Package **NEArender** is created to provide a faster, less biased, and more convenient procedure for enrichment analysis and rendering the original data into a pathway space. The pathway scores are calculated with a fast algorithm of the network enrichment analysis (NEA). **NEArender** possesses both core and ancillary functionality for NEA.

1.1 General workflow

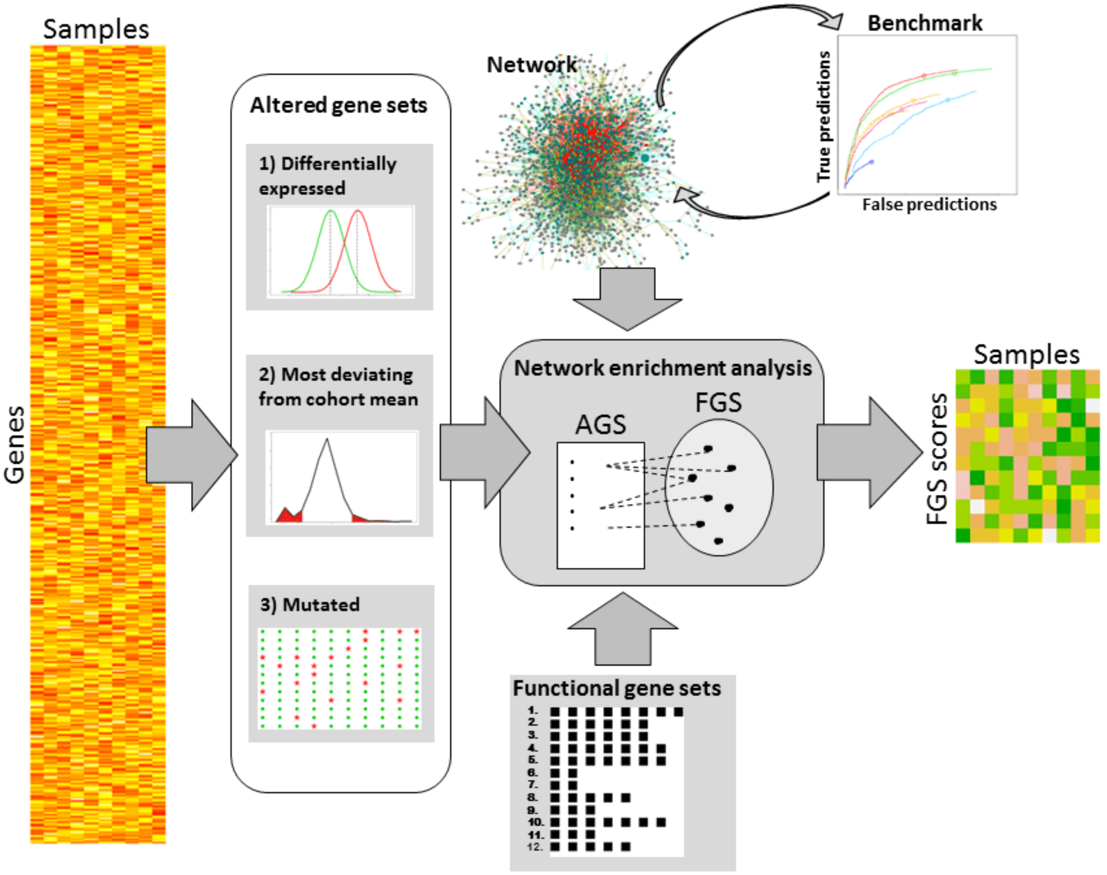

Figure 1: Implementation of NEArender

2 Brief description of NEA and its functions

2.1 Inputs

The three essential input components are:

1. **Altered gene set (AGS)**: a user-defined gene set that represents an experimental result, a patient-specific sample, a biological hypothesis etc.

2. **Functional gene set (FGS)**: a known or a hypothetical set of genes with a common, well defined biological or molecular function: a pathway, an ontology term, a biological process etc.
3. **Global network of functional coupling (NET)**: a graph where edges (functional links) represent arbitrarily defined (e.g. experimentally verified, literature derived, or computationally inferred) relations between gene nodes. In practice, the graph is fully defined with the list of gene pairs representing edges. Edge weights and directions are ignored.

Note: The term "gene" is used here as a logical proxy for a range of functional components of a biological network, such as genomic regions that encode proteins/microRNAs/enhancers/protein molecules. This imposes the practical limitations:

1. Identifiers used in the AGS, FGS, and NET inputs should belong to the same name space (most desirably gene symbols).

2. The parametric algorithm of enrichment evaluation used in this package would be unbiased only in scale-free networks, i.e. where node connectivity values follow the power law distribution (Albert-László Barabási et al., 2004). This is not the case in networks that are artificially constructed from e.g. ChIP-seq based collections of transcription factor binding events (Bovolenta LA et al., 2012) or from computationally predicted microRNA-transcript targeting data (Griffiths-Jones S et al., 2006). If such network or a network component should be included, we recommend employing software that involves network permutation tests, i.e. network randomization (Alexeyenko A et al.,2012, McCormack et al., 2013). In order to see how much a certain network deviates from the scale-free pattern, one can use function connectivity

2.2 Analysis and its outputs

The central function of this package is `nea.render()`. It performs the network enrichment analysis as described first in (Alexeyenko et al.,2012), and output contains a number of relevant statistics:

- chi-squared score,
- p-value,
- q-value (false discovery rate, or p-value adjusted for multiple testing) (Storey and Tibshirani, 2013), and
- z-score, which is, however not a direct product of the enrichment analysis. Instead, it is calculated downstream in order to facilitate the use of NEA values in linear modeling. Since methods of the latter are parametric and expect Gaussian values, the normally distributed (under true null) z-score fits this scenario well.

The output also contains auxiliary values:

- number of network edges that exist between any nodes of AGS and FGS (but does not include those within AGS or within FGS), and
- respective number of edges expected by chance, calculated with the binomial formula.

The most computationally intense part of `nea.render()` is counting the actual network edges in each AGS-FGS pair. At this step, `nea.render()` can employ parallel jobs, which is enabled with R package parallel by using parameter **Parallelize**.

If the user wants to execute the conventional *binomial GSEA*, then function `gsea.render()` can be used. It accepts the same input as `nea.render()` (except parameter `NET`), and produces output from Fisher's exact test arranged similarly to that of `nea.render()`:

- odds ratio estimate,
- p-value,
- q-value, and
- number of genes shared by AGS and FGS.

3 Included Datasets

The package contains the following data sets:

- two versions of NET, a smaller:
  - net.kegg (Kanehisa M et al., 2002) and a bigger one:
  - net.merged (Merid SK et al., 2014)
- an example collection of FGSs
  - can.sig.go (2406 distinct genes in 34 KEGG pathways (Kanehisa M et al., 2002) and GO terms (Ashburner M et al., 2000))
- three input datasets for creating AGSs:
  - somatic point mutations tcga.gbm (International Cancer Genome Consortium, 2010)
  - two subsets of FANTOM5 transcriptomics data
    - ∗ fantom5.43samples
    - ∗ fant.carc (FANTOM Consortium and the RIKEN PMI and CLST (DGT), 2014).

Apart from directly using these **.Rdata** files included in the package (such as `data(net.kegg)`), we also describe below, functions of **NEArender** for importing text files and then preparing from them AGS, FGS and NET inputs in the R space.

Properly formatted example text files can be downloaded from here

4 Functions usage and examples

In order to run `nea.render()` and `gsea.render()`, the user should prepare the input components with functions described below.

4.1 Preparing AGS file

Since the AGS is the most dynamic and user-specific part of the input, the functionality for AGS compilation and processing is most developed. As mentioned above, AGSs can be prepared in two alternative ways:

1) From pre-processed R lists and matrices or
2) By providing text file inputs.

4.1.1 Alternative (1):

The function `samples2ags()` creates AGSs from an R matrix where each column corresponds to an individual sample or an experimental condition and each row corresponds to an individual gene/protein (i.e. a potential node in the network – while IDs that are not found as network nodes would be ignored). An R list of AGSs can be prepared with `samples2ags()` by one out of five available algorithms (set with parameter 'method'): "significant", "top", "toppos", "topnorm", "toprandom" . Depending on the used algorithm, the number of genes per AGS would be either data-driven (when all significant ones are included) or user-defined (when N top ranking ones are included regardless of statistical significance). See help to the function (`?samples2ags`) for more details

4.1.1.1 Example of alternative (1):

Here we considered FANTOM5 - 43 carincinoma cell samples where the expression values indicated normalized tags per million (TPM) from CAGE-RNA sequencing (FANTOM Consortium and the RIKEN PMI and CLST (DGT), 2014). AGS lists are then obtained by *"topnorm"* method by which we perform data reduction and obtain sample-specific lists of altered genes (these can partially overlap with each other).

```
# input <-
# fread('http://research.scilifelab.se/andrej_alexeyenko/downloads/test_data/FANTOM5.43samples.txt',
# sep='\t',header=T,stringsAsFactors=FALSE,data.table = F)
# Converting genenames as rownames rownames(input)
# <-input[,1] input <- as.matrix(input[,c(2:ncol(input))])

data("fantom5.43samples")
input <- fantom5.43samples
dim(input)
```

```
## [1] 16619    43
```

```
ags.list1 <- samples2ags(input, Ntop = 20, method = "topnorm")
```

As we see, the matrix (i.e input) consists of 16619 gene rows and 43 sample columns. Next, we apply *"topnorm"* (`method="topnorm"`) to each sample column to obtain an R list `ags.list1` where each sample-specific list element contains one sample-specific AGS: a set of top 20 genes, each of which was selected for being most deviating from its mean across the sample cohort (see the 2nd section of "Altered gene sets" in Figure 1). Note that here 20 genes per AGS were picked regardless of their formal significance. For comparison, by setting parameter `method="significant"` we could select for each AGS all genes that pass a one-sided z-test that pass a specified p-value threshold, adjusted for multiple testing. In this case, AGSs typically would contain gene sets of variable size. Next, a special function `mutations2ags()` also allows direct creation of AGSs from an R matrix that contains full sets of mutated genes for each sample:

```
data("tcga.gbm", package = "NEArender")
ags.list3 <- mutations2ags(tcga.gbm, col.mask = "[-.]01$")
```

we optionally used the parameter ***`col.mask`*** *in order to select only tumor samples by TCGA barcodes – hence the parameter is TCGA-specific.*

4.1.2 Alternative (2):

In case of importing a text file with ready, pre-created AGSs, one should use `import.gs()`. The function returns a list of as many elements as there were distinct AGS labels in the file (i.e. typically multiple AGSs are imported from a single file).

4.1.2.1 Example of alternative (2):

The file **cluster2_Downregulated_ags.txt** contains 274 genes which represents a sample-specific AGSs, i.e. lists of altered genes that we have compiled with a certain procedure before. Now we are just importing the structure to be used as a list **ags.list** within R.

```
ags.list2 <- import.gs("http://research.scilifelab.se/andrej_alexeyenko/downloads/test_data/cluster2_Do
    Lowercase = 1, col.gene = 1, col.set = 3, gs.type = "ags")
```

Based on the format of the above example files, users can create their own TAB-delimited text files to be used as ***ags.list****. The column positions in the files can be arbitrarily changed using parameters* ***col.gene*** *and* ***col.set****. One can view the data formats of current examples listed in NEArender package simply by (*`head(can.sig.go)`* or *`head(fantom5.43samples)`*)*

4.2 Preparing FGS file

Since FGSs usually pre-exist rather than are created from user's data, they are imported from text files. For this reason their format and the import procedure using the function `import.gs()` are identical to the alternative (2) above. However, there is a special option unique to NEA and not available in GSEA, where single genes can be treated as FGS. A full list of such FGSs can be automatically created from all network nodes of NET with `as_genes_fgs()`, so that each FGS item in the output list contains just one gene. Alternatively, users can create more specific single- or multi-gene FGS collections of their own and then import them with `import.gs()`. In such files, one typically uses the gene/protein IDs as the FGS labels.

The users can upload their own FGS text files. Identically to AGS, an FGS file should be a tab-delimited text file containing gene/protein/nodeIDs and their FGS labels (such as pathway, ontology terms, or being user-defined).

4.2.0.2 Examples :

Here we used file `can.sig.go` as a small collection of functional gene sets (FGS). It contains 34 GO terms and KEGG pathway that represent signaling processes related to many diseases, with a special focus on cancer. It can be used for a primary, exploratory analysis of the package functionality (note that since the AGS and FGS formats are identical, it could also be submitted as an AGS collection).

```
fgs.list <- import.gs("http://research.scilifelab.se/andrej_alexeyenko/downloads/test_data/can.sig.go.t
    Lowercase = 1, col.gene = 2, col.set = 3, gs.type = "fgs")
```

4.2.0.2.1 Uploading text file :

Note that in the package the dataset `can.sig.go` has been saved as Rdata so that it can be called directly via `data(can.sig.go)`.

Uploading Rdata Object:

```
data(can.sig.go)
fgs.list <- import.gs(can.sig.go)
```

4.3 Preparing NET file

The network files can be imported via `import.net()`. It requires two columns in a TAB-delimited file so that each line contains two node IDs connected by the given edge.

4.3.0.3 Examples :

`net.kegg` is a network obtained by downloading the KEGG pathways as separate KGML files, extracting from the latter all gene-gene links, and then merging the links into one global network, which erased borders between the pathways. Like `can.sig.go`, in NEA render `net.kegg` can be directly used as `data(net.kegg)` or as text file:

```
data(net.kegg)
net <- import.net(net.kegg)
```

4.3.0.3.1 Uploading Rdata Object:

```
## [1] "Network of 42491 edges between 4064 nodes..."
```

```r
print(paste(names(net$links)[10], net$links[[10]], sep = ": "))
```

```
## [1] "abcc8: cacna1a" "abcc8: cacna1b" "abcc8: cacna1c" "abcc8: cacna1d"
## [5] "abcc8: cacna1e" "abcc8: cacna1g"
```

```r


net <- import.net("http://research.scilifelab.se/andrej_alexeyenko/downloads/test_data/net.kegg.txt")
```

4.3.0.3.2 Uploading text file :

```
## [1] "Network of 42491 edges between 4064 nodes..."
```

We can expect that within-pathway connectivity would be higher within original pathways than between them, i.e. overall in the network. This can be seen at the respective plot (Funcoup 3.0 network) in Figure 15.

We can see that the connectivity pattern differs from e.g. networks derived in pathway-ignorant way from multi-facetted data integration STRING9 and merged. `Net.merged` is the network previously used by (Merid SK et al., 2014). Briefly, this is largely (>90%) a FunCoup based network, i.e a network from Bayesian integration of multiple literature and high-throughput data sources. Then this basic network was merged with KEGG pathways, CORUM protein complexes, and PhosphoSite kinase-substrate links.

```r
net.merged <- "http://research.scilifelab.se/andrej_alexeyenko/downloads/test_data/merged6_and_wir1_HC2
net <- import.net(net.merged)
```

```
## [1] "Network of 971577 edges between 19027 nodes..."
```

4.4 Other Miscellaneous functions

Function `save_gs_list()` helps to save a collection of AGSs (such as ags.list) as a text file. The latter, can serve an example of the file format or be submitted to the web site.

```r
data(net.kegg)
net <- import.net(net.kegg)
```

```
## [1] "Network of 42491 edges between 4064 nodes..."
```

```r
fgs.genes <- as_genes_fgs(net)
# save_gs_list(fgs.genes, File =
# '~/single_gene_ags.groups.tsv');
```

5 NEA-analysis

From the above described functions, we created inputs for AGS (FANTOM5.43samples.txt), FGS (can.sig.go) and NET (net.kegg) and can now demonstrate using the main function `nea.render`.

```
n1 <- nea.render(AGS = ags.list1, FGS = fgs.list, NET = net)
```

```
## [1] "Preparing input datasets:"
## [1] "Network: 4064 genes/proteins."
## [1] "FGS: 1293 genes in 34 groups."
## [1] "AGS: 854 genes in 43 groups..."
## [1] "Calculating N links expected by chance..."
## [1] "Rendering integer IDs..."
##    user  system elapsed
##   1.228   0.000   1.229
## [1] "Counting actual links..."
##    user  system elapsed
##   0.089   0.000   0.089
## [1] "Calculating statistics..."
## [1] "Done."
```

```
hist(n1$chi, breaks = 100)
```

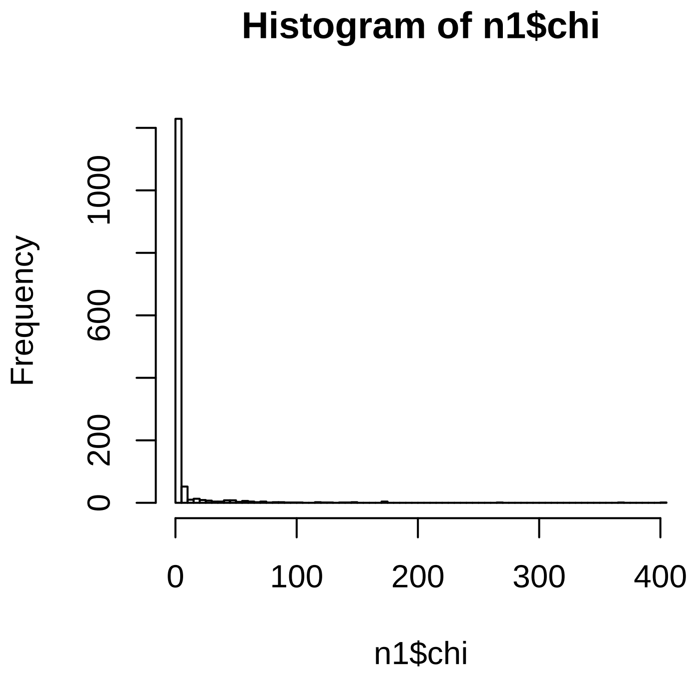

Figure 2: n1$chi - chi-square estimate

```
hist(n1$z, breaks = 100)
```

```
hist(n1$p, breaks = 100)
```

```
hist(n1$q, breaks = 100)
```

Note that the values of chi, p, and q are rank-invariant, i.e. unambiguously related to each other. In other words, they differ only in terms of scale and density distributions. Consequtively, ranking enrichment scores in the AGS-FGS pairs can be done by either of these values with the same result.

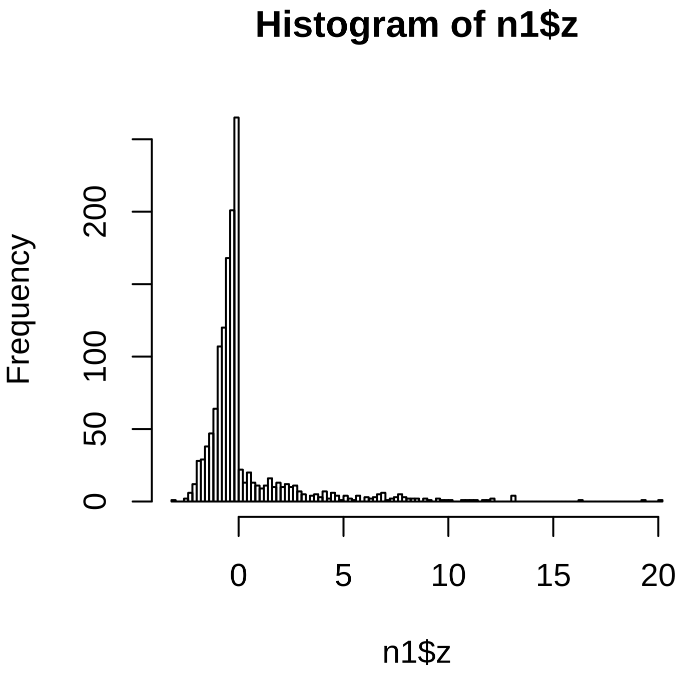

Figure 3: n1$z- zscores

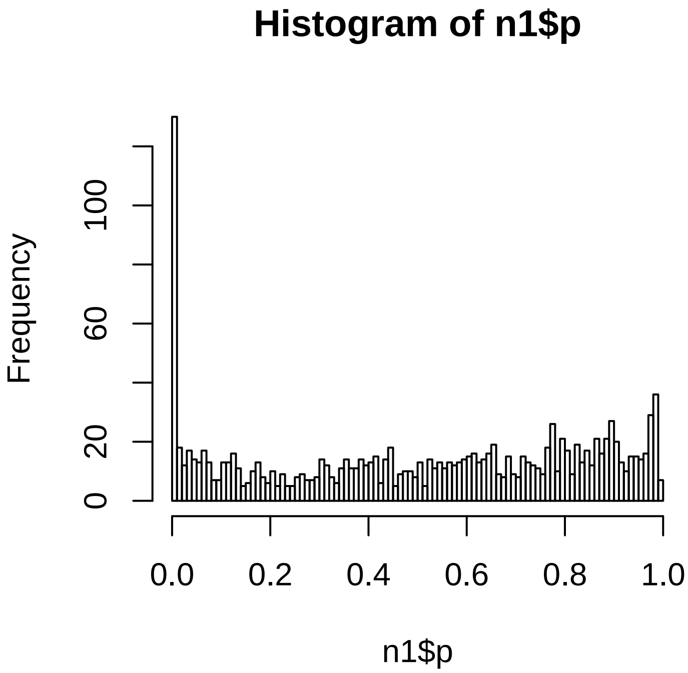

Figure 4: NEA- pvalues

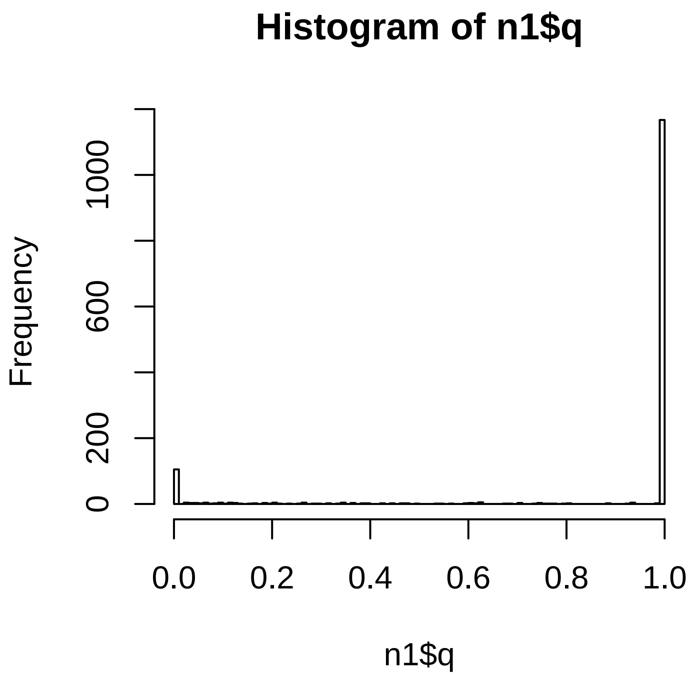

Figure 5: NEA-qvalues

In general purpose, exploratory analyses the statistical significance can be established by **p-** and **q-values**. The latter is generally more correct, but the **q-values** are reliable only with sufficiently many (at least 300-500) AGS-FGS tests done at once. In case of fewer tests, another correction (e.g. Bonferroni) should be applied.

However, using the NEA output for more complex downstream analyses, such as phenotype modeling, is more challenging. The chi-squared statistic has two drawbacks from this perspective:

1) It is defined on the non-negative domain and hence cannot distinguish between enrichment and depletion.
2) It is not normally distributed and thus cannot be processed by statistical methods that employ least squares estimation or otherwise expect Gaussian input (Pearson linear correlation, ANOVA, PCA, or even survival analysis).

In order to address these problems, we recommend using as input to such modeling **z-scores** that are calculated to AGS-FGS pairs from p-values of the chi-squared statistics. Negative signs are assigned to those AGS-FGS pairs where the expected number of network edges exceeds the actual value (i.e. depletion). Thus, the resulting z-score distribution should appear normal under true null. The latter is generally more correct, but the **q-values** are reliable only with sufficiently many (at least 300-500) AGS-FGS tests done at once. In case of fewer tests, another correction (e.g. Bonferroni) should be applied.

6 Binomial GSEA

```
ags.list2 <- samples2ags(fantom5.43samples, Ntop = 1000, method = "topnorm")
g1 <- gsea.render(AGS = ags.list2, FGS = fgs.list, Lowercase = 1,
    ags.gene.col = 2, ags.group.col = 3, fgs.gene.col = 2, fgs.group.col = 3,
    echo = 1, Ntotal = 20000, Parallelize = 1)
```

```
## [1] "Preparing input datasets:"
## [1] "AGS: 15271 genes in 43 groups."
## [1] "Calculating overlap statistics..."
```

```
hist(log(g1$estimate), breaks = 100)
```

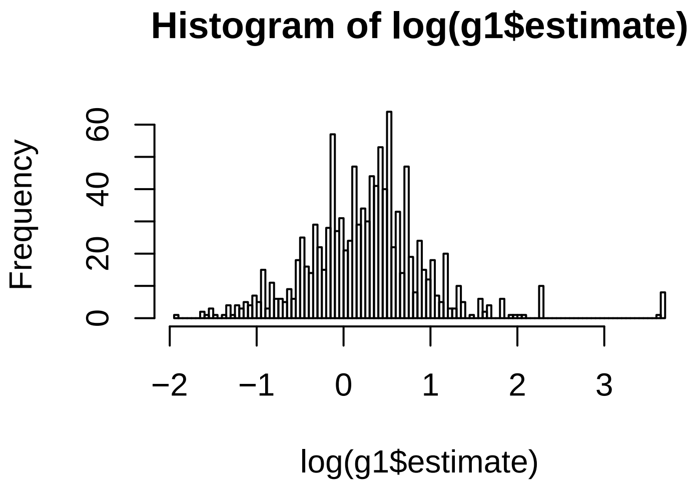

Figure 6: g1$estimate - an estimate of the odds ratio

```
hist(g1$n, breaks = 100)
```

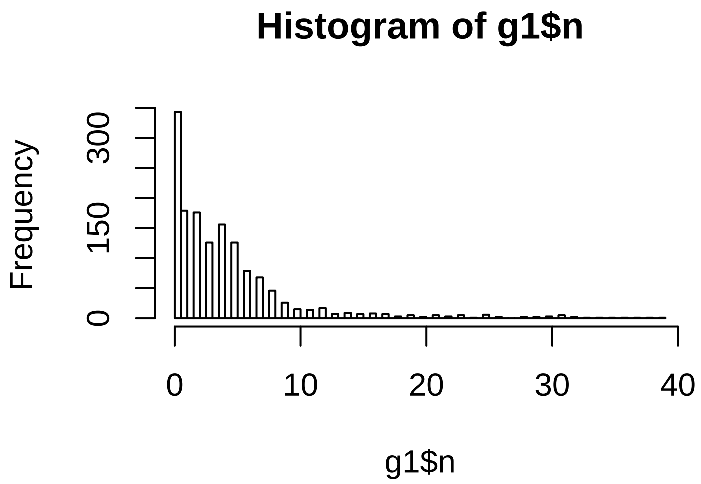

Figure 7: g1$n - number of ags genes that belongs to corresponding fgs

```
hist(g1$p, breaks = 100)
```

```
hist(g1$q, breaks = 100)
```

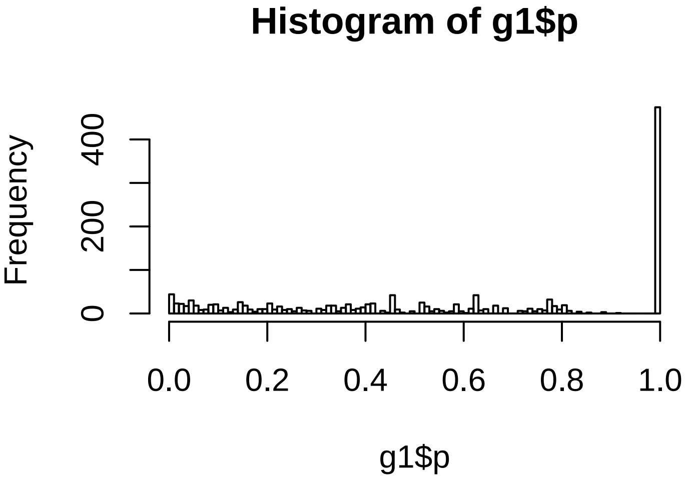

Figure 8: g1$p - the p-value of the fishers.exact test

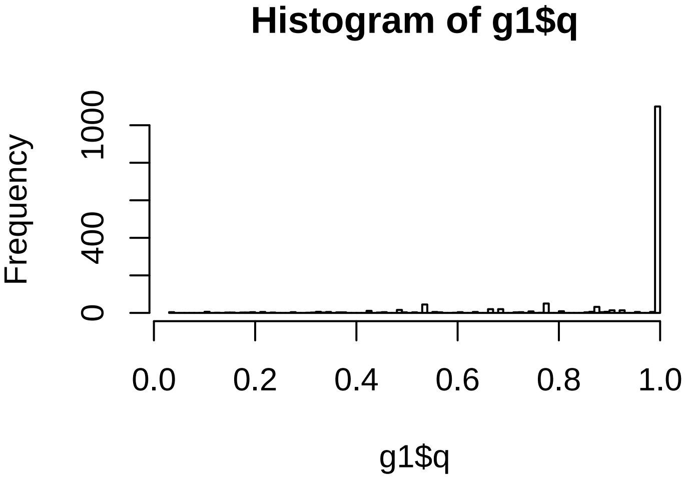

Figure 9: g1$q - Adjusted p-values by “BH-method”

7 Benchmarking and ROC curves

The package also contains functions `connectivity()`,`topology2nd()`, `benchmark()` and `roc()` which enable evaluating and benchmarking alternative NETs using either standard or custom FGSs, as described in (Merid SK et al., 2014). Briefly, the `benchmark()` consists of many test cases as there are FGS members in total (multiple occurrences of the same genes in different FGS are treated separately). The procedure tests each such gene for being an FGS member. The true positive or false negative result is assigned if the gene receives an NEA score above or below a certain threshold, respectively. In parallel, randomly picked genes with matching node degree are tested against the same FGS in order to estimate specificity via the false positive versus true negative ratio. The counts of alternative test outcomes TP, TN, FP, and FN at variable NEA thresholds are used for plotting ROC curves. The function can work in the `parallel` mode (Parallelize>1), similarly to `nea.render()` above:

```
b0 <- benchmark(NET = net, GS = fgs.list, echo = 1, graph = TRUE,
    na.replace = 0, mask = ".", minN = 0, coff.z = 1.965, coff.fdr = 0.1,
    Parallelize = 1)
```

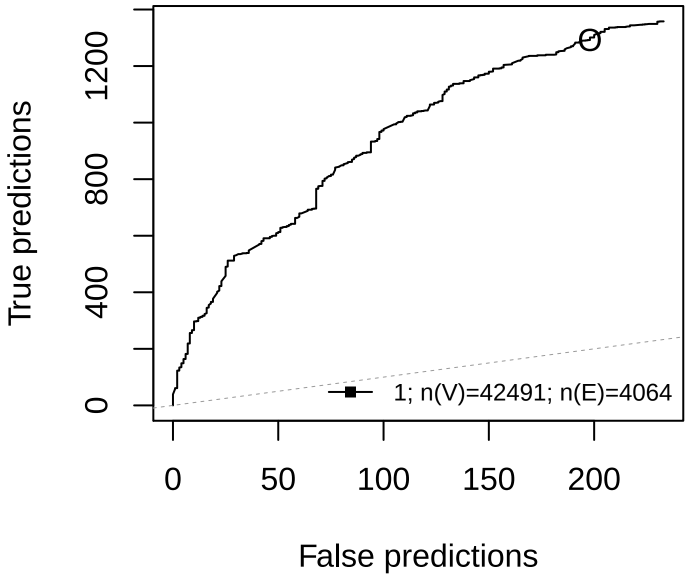

Figure 10: ROC curve evaluating KEGG network (net.kegg)“

Note: The following lines of code takes longer time to calculate the actual links. Please make use of ***`parallel`*** *to increase the number of CPUs cores.*

```
b1 <- NULL
for (mask in c("kegg_", "go_")) {
    b1[[mask]] <- NULL
    ref_list <- list(net.kegg = net.kegg, net.merged = net.merged)
    for (file.net in c("net.kegg", "net.merged")) {
        # a series of networks can be put here: c('net.kegg1',
        # 'net.kegg2', 'net.kegg3') in ref_list
        net <- import.net(ref_list[[file.net]], col.1 = 1, col.2 = 2,
            Lowercase = 1, echo = 1)
        b1[[mask]][[file.net]] <- benchmark(NET = net, GS = fgs.list,
            echo = 1, graph = FALSE, na.replace = 0, mask = mask,
            minN = 0, Parallelize = 3)
        roc(b1[["kegg_"]], coff.z = 2.57, main = "kegg_")
        roc(b1[["go_"]], coff.z = 2.57, main = "go_")
    }
}
```

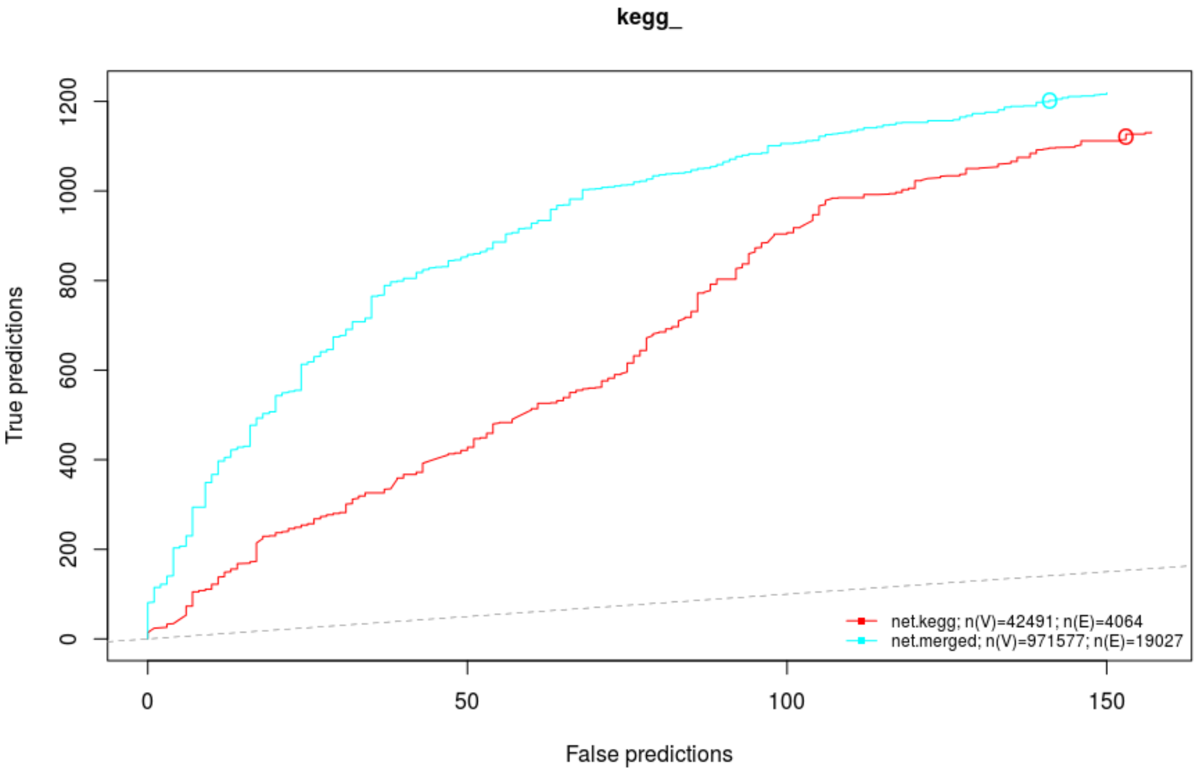

Figure 11: ROC curves evaluating differential performance of net.kegg and net.merged in predicting KEGG terms

The full description of the ROC approach to network benchmarking can be found in (Merid SK et al., 2014). Note that this approach is very different from the trivial measurement of edge overlap between two networks under a variable edge confidence threshold. Instead, we run multiple network enrichment tests in regard of individual genes´ membership in different pathways.

True Predictions (correct pathway membership) are plotted vertically, False Predictions (wrong/unknown membership) are plotted horizontally. As always with ROC curves, one should watch false positive (X-axis) versus false negative (Y-axis) rates. Practically, the best network is the one with the highest ROC curve elevation in the upper left corner of the plot, i.e. the one that, under a suitable siginifcant threshold has led to most correct classification of true pathway members and conveyed least misleading information on false (or yet unknown) membership.

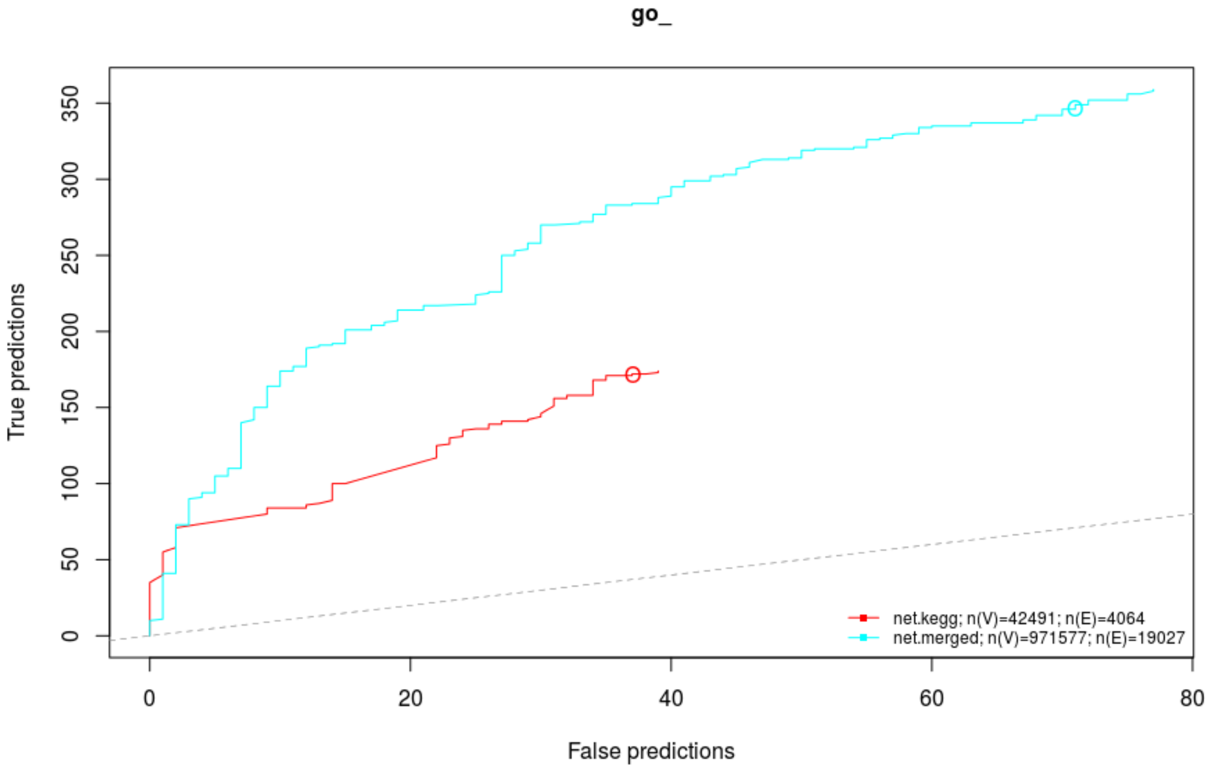

Figure 12: ROC curves evaluating differential performance of net.kegg and net.merged in predicting GO terms

8 Estimating topological properties of used networks

8.1 Scale-free property

Most of the known networks, especially the biological ones are *scale-free*. This means that the distribution of node degrees (also called node connectivity values or simply the numbers of edges for each node) follows the *power law*. Practically, it is expressed as "few nodes have many edges, whereas many nodes have few edges". After a log-log transformation, this dependency appears as a straight line. Knowing this property is especially important because in the current package the binomial test of network enrichment expects scale-freeness in the analyzed network. If a particular network seems too different (though some deviations from the power law occur in almost every real-world network), then we would recommend using other tools, such as those based on full network randomization (McCormack et al., 2013 , Merid et al., 2014).

The package contains a utility function `connectivity()` for quick visual inspection of this property. The function receives an input network (either a text file or an R list imported with `import.net()`) and plots on the log-log scale its node degree distribution summarized into few bins. After the log-log transformation we expect the linear fit to be matching the bin top points. The better the fit, the more scale-free is the network.

```
connectivityfile <- system.file("extdata", "Connectivity.FC1_full",
    package = "NEArender")
connect <- connectivity(connectivityfile)
```

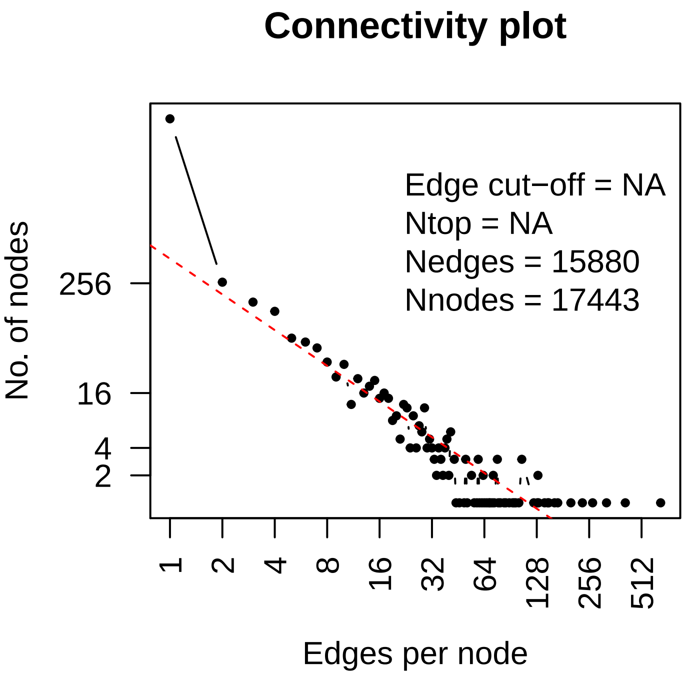

Figure 13: Node degree distribution

For example out of the nine networks presented in Figure 16, we can see that *FunCoup 3.0* and *KINASE2SUBSTRATE* were likely the most and the least scale-free, respectively. Although none of the nine networks followed the power law perfectly (the red lines indicate ideal distributions given network size and connectivity range). The origin and descriptions of these networks can be found at https://www.evinet.org/.

8.2 Second order topology

Similarly to the property of scale-freeness, estimation of network enrichment in the current package might depend on non-randomness of node degree distribution across network edges.

For example, it was shown that highly connected nodes tend to ‘avoid’ each other in a yeast network, i.e. such connect with each other less frequently than it would be expected by chance (Maslov and Sneppen, 2002). We, however, can often see an opposite tendency Figure 15. In order to visualize and evaluate this property, one can employ function `topology2nd` in a fashion similar to `connectivity` described above:

```
topology2nd(NET = connectivityfile, Lowercase = 1, col.1 = 1,
    col.2 = 2)
```

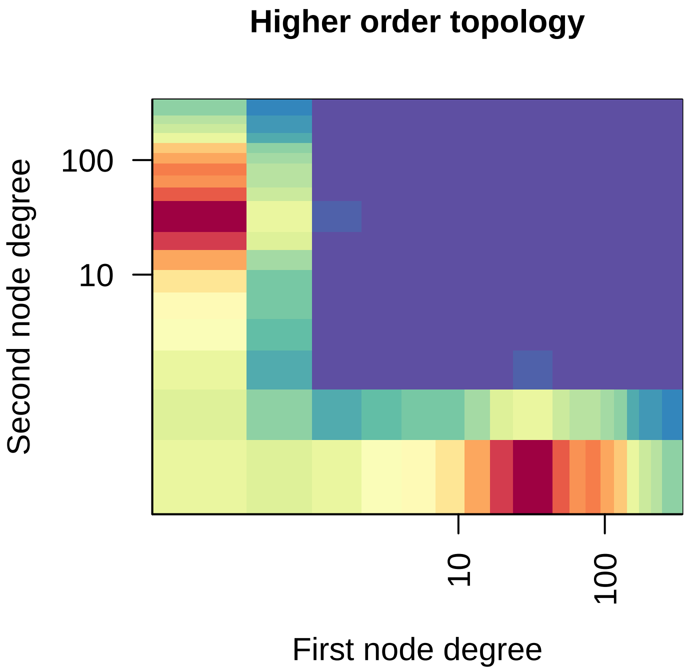

Figure 14: Second order topology

8.2.1 Second order topology observed in nine example biological networks

From plot, networks *FunCoup 3.0* and *TF_and_targets* appear the best and the worst in terms of higher order topological bias. In FunCoup 3.0 only the most connected nodes stand for the bias, which would affect NEA scores of AGS-FGS pairs only if such nodes are unevenly distributed across the gene sets. The much more generalized bias in *TF_and_targets* (or, similarly, in *KINASE2SUBSTRATE*) is likely due to the special nature of such networks: they contain regulators and regulated proteins as two distinct node classes. In case of a strong bias discovered in a certain network, it is important to consider if it should be removed or not from the perspective of the biological study and its purposes (indeed, the answer is not always upfront). If the bias still has to be removed, one can use a special algorithm in the C++ tool by (McCormack et al., 2013).

8.2.2 Node degree distributions in nine example biological networks

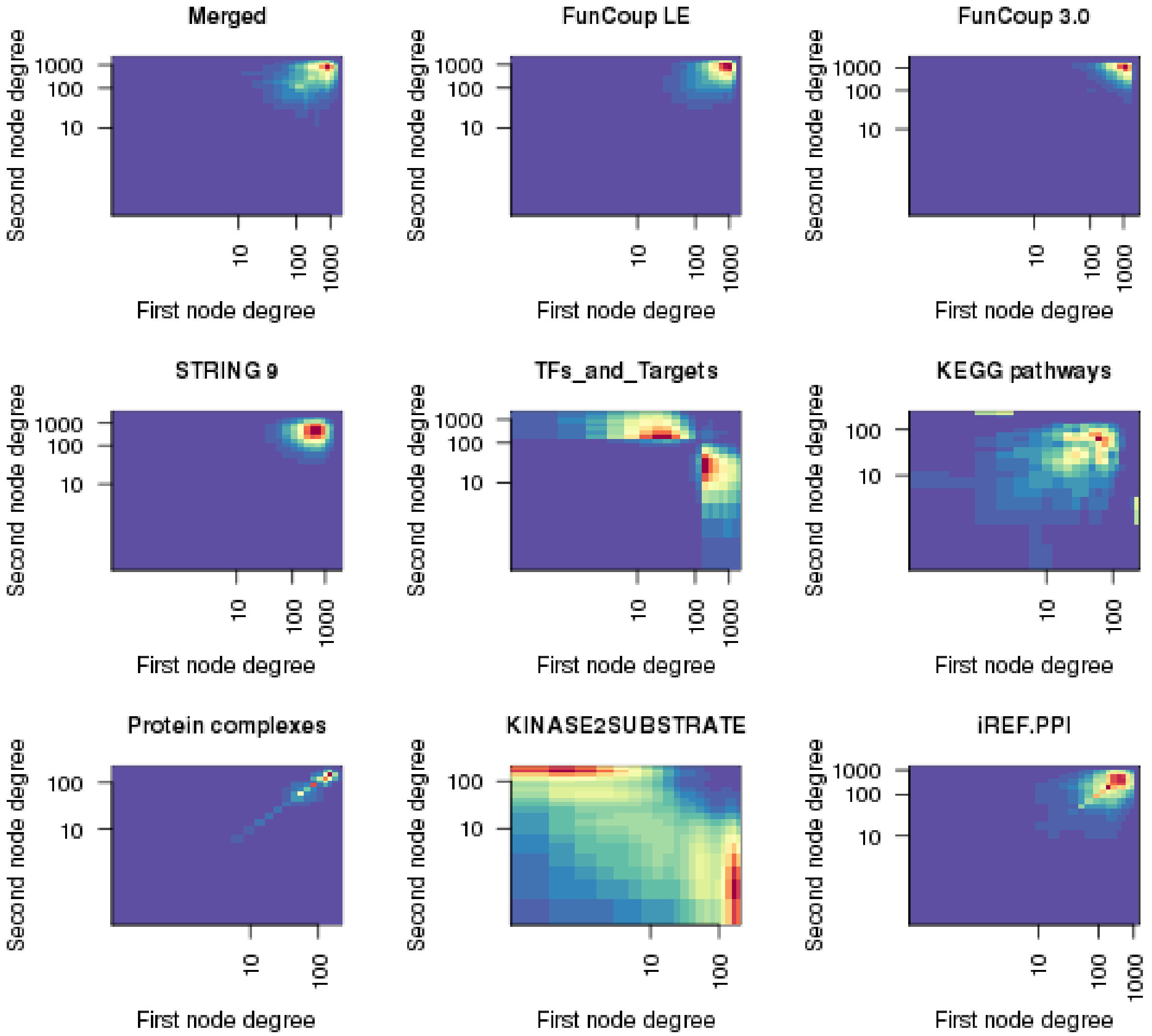

Figure 15: Second order topology observed in nine example biological networks

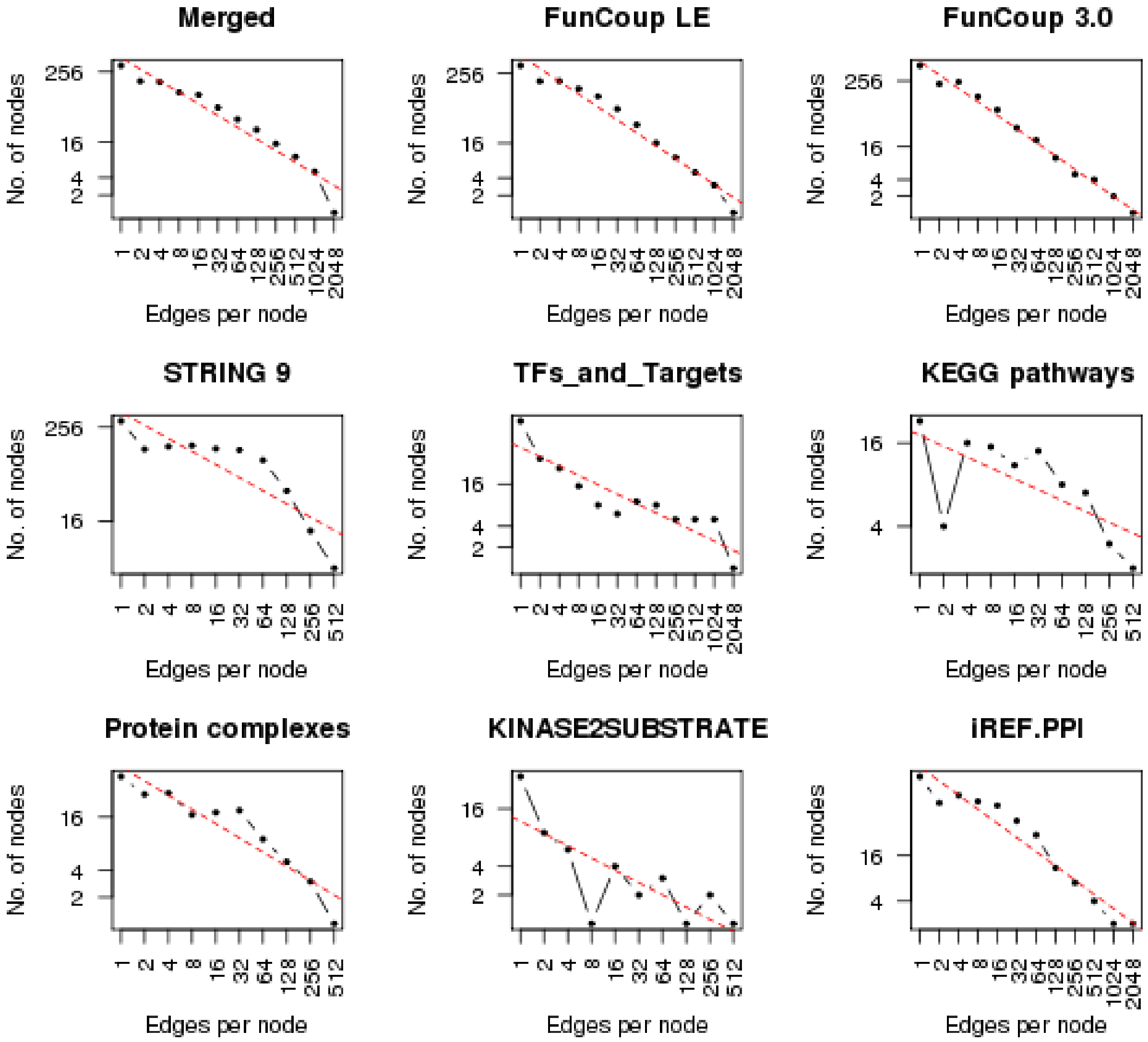

Figure 16: Node degree distributions in nine example biological networks.