**A Computational Pipeline for Analysis of Biomedical networks with BioNAR:**

Colin McLean1, Anatoly Sorokin2, J Douglas Armstrong3.4, and Oksana Sorokina4\*

1Health Economics and Data Science, Institute for Genetics and Cancer, University of Edinburgh, UK; colin.d.mclean@ed.ac.uk

2 Biological Systems Unit, Okinawa Institute of Science and Technology, Okinawa, Japan; anatolii.sorokin@oist.jp

3 Computational Biomedicine Institute (IAS-5 / INM-9), Forschungszentrum Jülich, Jülich, Germany

4 School of informatics, University of Edinburgh, UK

\*Correspondent author: oksana.sorokina@ed.ac.uk

**ABSTRACT:**

In a living cell, proteins interact to assemble both transient and constant molecular complexes, which transfer signals/information around internal pathways. Modern proteomic techniques can identify the constituent components of these complexes, but more detailed analysis demands a network approach linking the molecules together and analyzing the emergent architectural properties. The Bioconductor package *BioNAR* combines a selection of existing R protocols for network analysis with newly designed original methodological features to support step-by-step analysis of biological/biomedical networks. Critically, *BioNAR* supports a pipeline approach where many networks and iterative analyses can be performed. Here we present a network analysis pipeline that starts from initiating a network model from a list of components/proteins and their interactions through to identifying its functional components based solely on network topology. We demonstrate *BioNAR* can help users to achieve a number of network analysis goals that are difficult to achieve anywhere else. This includes how users can choose the optimal clustering algorithm from a range of options based on independent annotation enrichment; predict a protein’s influence within and across multiple sub-complexes in the network and estimate the co-occurrence or linkage between meta-data at the network level (e.g., diseases and functions across the network, identifying the clusters whose components are likely to share common function and mechanisms). The package is freely available in Bioconductor release 3.17: [*https://bioconductor.org/packages/release/bioc/html/BioNAR.html*](https://bioconductor.org/packages/release/bioc/html/BioNAR.html)

Basic Protocol 1: Creating and annotating the network

**Support Protocol 1:Installing *BioNAR* from RStudio.**

**Support Protocol 2:Building the sample network from *synaptome.db***

Basic Protocol 2: Network properties and centrality

Basic Protocol 3: Network communities

Basic protocol 4: Choosing the optimal clustering algorithm based on the enrichment with annotation terms.

Basic Protocol 5: Influencing network components and bridgeness

Basic Protocol 6: Co-occurrence of the annotations

.

**KEYWORDS:**

Network analysis, protein-protein interacting network, clustering.

**INTRODUCTION**

The technology available to support biological experiments has made rapid advances in recent years with massive steps forward in both the sensitivity and throughput of methods to analyse biological samples across multiple levels. The most widely known advances are those in the high-throughput sequencing of DNA and RNA but there have been step changes in a wide range of areas including proteomics, metabolomics, and connectomics. Integrating these datasets into network models allows us to elucidate key functional interactions, pathways and complexes which are required for healthy function and that, when perturbed, often lead to disease.

Proteomic data are typically represented via static undirected protein-protein interaction (PPI) networks, where vertices represent the proteins and, edges represent the molecular interactions. From a PPI network one can derive many statistical measures of topology and fundamental properties and use these to gain insight and make predictions about the underlying data (Bánky et al., 2013; Vidal et al., 2011). For example, 'Scale-free' (Barabasi & Albert, 1999) properties and small world paths, found in many biological networks, in combination with various centrality measures, e.g. degree and betweenness, have been widely used to identify ‘hub’ molecules, which often encode disease related proteins (Vidal et al., 2011).

Protein networks obtained from modern proteomic studies are usually large (1000s of proteins) and contain the components of multiple signalling pathways, linked together by other proteins whose role in the network is less well understood. Common approach is to divide PPI networks into communities (or clusters) based on their connecting architecture, which can be achieved with multiple clustering algorithms.



Proteins in a network model are usually annotated with functional or/and disease information. Given the topological structure from clustering, it is often useful to test how these annotations are distributed throughout the network; do they tend to concentrate in specific network communities? Do some annotations share common subnetwork patterns? To study this, disease and/or functional associations can be mapped over PPI vertices. Functional and disease enrichment of clusters can then be calculated using a suitable hypothesis driven test, i.e. the hypergeometric test, and tested against a permutation study (McLean et al., 2016) to reveal the clusters within the network that are significantly enriched for specific annotations. This kind of approach has been extensively applied in our set of case studies and was used to link together neuronal diseases (typically highly polygenic) onto the molecular pathways and clusters in the synapse proteome (Pocklington et al., 2006). In summary, a common and important task in biological network analysis is in testing how well the network topology (interconnectedness) correlates with distribution of specific functional (or dysfunctional/disease) annotations (Fernandez et al., 2009; Klemmer et al., 2009; Zhu et al., 2007).

We designed *BioNAR* to provide a universal topologically based network analysis pipeline deployable on high performance computing environments that enables users to load networks generated and/or annotated using their lab’s own meta-data and perform a step-by-step network analysis with respect to their specific scientific questions, while also making the tool as widely applicable and flexible as possible.

6 protocols presented here form a procedural guide for network analysis using *BioNAR* (Figure 1). Basic Protocol 1 describes how to initiate a network model, import relevant datasets and how to annotate the network with various metadata. This results in a base network model, on which all the other analyses in the pipeline are then based. Support Protocol 1 explains how to install the *BioNAR* package using R integrated development environment (IDE) RStudio. Basic knowledge of R language is assumed.

In Basic Protocol 2 the raw network model is analyzed as a whole and its general properties, e.g., “scale-freeness” and major network centralities are calculated and compared against randomly generated network models of the same size. Basic Protocol 3 describes how to subdivide the network into communities using a selection of nine available algorithms that have been used in a range of published studies. Basic Protocol 4 further compares the results of different clustering algorithms and estimates their effectiveness based on a general assumption that biologically plausible clusters are more likely to show community enrichment with relevant functional annotation terms. Basic Protocol 5 then shows how to estimate the vertex property bridgeness based on the consensus clustering matrix, and how to rank the vertices based on their topological importance within the network. This can be used to identify candidate linkages between functional compartments in the model. Finally, Basic protocol 6 demonstrates how to test the topological overlap of different annotations, which may indicate shared mechanisms that are only visible using the network context.

Given our previous and ongoing work is related to synaptic proteome, we illustrate the package functionality with a postsynaptic network model, derived from the publicly available *synaptome.db* package (version 0.99.12), which is described in Support Protocol 2.

**BASIC PROTOCOL 1**

**Network initiation and annotation**

This protocol describes how a network is initiated in the *BioNAR* package and how annotations are added for further analysis. *BioNAR* implements networks as R data frames; where each row corresponds to a vertex interactor pair and where each vertex has a unique *vertex\_ID.* Therefore, a network can be directly uploaded as a data frame in this format. Alternatively, a network can be imported from most standard graph file formats including gml, or other formats supported by the *igraph* R package. *BioNAR* also allows network import for synaptic protein sets/synaptic compartments/brain regions directly from the *synaptome.db* package (Sorokina et al., 2022), this case is described in Support Protocol 1. For constructing the protein interaction networks described here, we selected NCBI Gene Entrez ID to use as a unique *vertex\_ID* for each node/protein.

Many graph-related algorithms behave poorly when applied to graphs containing disconnected components. Consequently, most biological networks - including PPI networks, are constructed to retain only the network's largest connected component of vertices. We also follow this practice for downstream analysis, but we also store the disconnected components as they may be of importance to the user at a later stage.

Vertices are typically annotated with categorical or continuous metadata. Annotations are usually handled in a three-column data frame format, where the first column contains the *annotation term ID*, the second - *annotation term name*, and the third column – associated *vertex\_ID.* All annotation terms for the same *vertex\_ID* are collected and converted into semicolon-separated lists to store all annotations of the vertex held as a string in the vertex annotation. For example, if a protein is annotated with two different molecular functions A and B, in vertex annotation it will be stored as ‘A; B’.

In proteomic networks, annotations typically include Gene Ontology (GO) annotations and gene-disease association values (GDAs), and the direct import of both these from public databases are implemented in *BioNAR*. However, the package supports any custom annotation, including gene expression values, pathway membership data, and so on. As an example of adding custom data below we added new binary annotation for SynGo ontology (Koopmans et al., 2019). *BioNAR* is also designed to allow the user to assign the results of any vertex calculation as a new vertex attribute. The ability to retain such calculated supports reproducibility as many algorithms used in network analysis have a stochastic component, so multiple invocations of the same analysis can create a distribution of results which can now all be stored for later analysis.

**Necessary Resources:**

1. *Hardware*

A modern compute environment capable of running R > 4.3.0 and Rstudio > 2022.12.0

1. *software*

R > 4.3.0 and Rstudio > 2022.12.0

1. *files*

For the examples you will need the sample network files: PSD\_PPI.txt and/or PSDnetwork.gml. Alternatively, these can be replaced with your own data files in either format.

You also need the additional annotation file that we provide for illustrative purposes: SynGO.txt.

# *Sample File.* PSDnetwork.gml

**Protocol steps with *step annotations***

1. Install RStudio and *BioNAR* package as described in Support Protocol 1.

Open RStudio and launch the library *BioNAR*, as well as a few more frequently used libraries assuming they have not been installed by default. Navigate (setwd) to the directory where you have stored the network and annotation files (in our case it is called “Sample network”).

library(BioNAR)

library(synaptome.db)

library(ggplot2)

library(pander)

library(ggrepel)

library(randomcoloR)

setwd("~/Documents/Sample network")

1. Load your network.

Networks can be loaded in three ways. We provide examples of each of the three different formats using the same test network below (2a-2c):

1. Import a network from a text table:

t <- read.table("PSD\_PPI.txt", sep = "\t", header = T, stringsAsFactors = F)

head(t)

gg <- buildNetwork(t)

summary(gg)

1. Import a network from a graph/gml file:

g <- igraph::read.graph("PSDnetwork.gml",format="gml") #graph from gml

summary(g)

The command summary(gg) and summary(g) will show the same graph composition from the text import steps 1.2.a or gml import in step 1.2.b respectively: 2297 vertices and 9406 edges, with Human Entrez ID used as the node name for all three graphs.

1. Import network from *synaptome.db* and is described in Support Protocol 1
2. Annotate the network with GeneNames.

In this step, the raw graph from (1.2) will be annotated by importing annotation data from human genome-wide gene annotation database *org.Hs.eg.db*. For each node in the graph, Entrez Gene Name is assigned using the annotateGeneNames function.

g<-annotateGeneNames(g)

Sometimes gene IDs become obsolete and gene names are not assigned. We can check for this:

any(is.na(V(g)$GeneName))

If result is TRUE, we can find such genes and assign its GeneName manually. Some functions further down require unique GeneNames to function correctly.

idx<-which(is.na(V(g)$GeneName))

idx

V(g)$GeneName[idx]<-'AKAP2'

1. Annotate the network with metadata.

In this step, the raw graph from (1.2) will be annotated by importing annotation data from relevant external files and/or databases. For each node in the graph the following annotation is added: associated human diseases are extracted directly from the Human Disease Ontology (HDO) using the annotateTopOntoOVG function and Gene Ontology terms (BP, MF and CC) are extracted using the annotateGOont function.

afile<-system.file("extdata", "flatfile\_human\_gene2HDO.csv", package = "BioNAR")

dis <- read.table(afile,sep="\t",skip=1,header=FALSE,strip.white=TRUE,quote="")

g <- annotateTopOntoOVG(g, dis)

g <- annotateGOont(g)

summary(g)

By calling summary(g)you will see that nodes now have attributes: id (v/n), name (v/c), GeneName (v/c), TopOnto\_OVG (v/c), TopOnto\_OVG\_HDO\_ID (v/c), GO\_MF\_ID (v/c), GO\_MF (v/c), GO\_BP\_ID (v/c), GO\_BP (v/c), GO\_CC\_ID (v/c), GO\_CC (v/c), where id corresponds to internal graph ID, name – Human Entrez ID, TopOnto\_OVG – disease name according Human Disease Ontology (HDO), TopOnto\_OVG\_HDO\_ID – disease ID according HDO, GO\_MF\_ID – Gene Ontology Molecular Function ID, GO\_MF - Gene Ontology Molecular Function name, GO\_BP\_ID - Gene Ontology Biological Process ID, GO\_BP- Gene Ontology Biological Process name, GO\_CC\_ID - Gene Ontology Cell Compartment ID, GO\_CC - Gene Ontology Cell Compartment name.

To annotate a graph with your own metadata, you need to provide it in the form of a data frame (see example “SynGO.txt”), where the first column should contain vertex IDs matching name IDs in the network (Human Entrez ID) and the second column the respective attribute values. When using the command annotateVertex(g), you will need to provide the name of your new attribute as parameter name (“syngo” in our case) and the name of your annotation table (“sg”) as parameter values.

sg <- read.table("SynGO.txt", sep = "\t", header = T, stringsAsFactors = F)

g <- annotateVertex(g, name = "syngo", values = sg, idatt = "name")

If you call summary(g) you will now see an additional new attribute “syngo” in addition to those listed above.

**SUPPORT PROTOCOL 1:Installing *BioNAR* from RStudio.**

Support Protocol 1 describes how install *BioNAR* from Bioconductor.

**Necessary Resources:**

1. *Hardware*

A modern compute environment capable of running R > 4.3.0 and Rstudio > 2022.12.0.

1. *software*

R > 4.3.0 and Rstudio > 2022.12.0

1. *files*

none

**Protocol steps with *step annotations***

1. Download R from[***https://www.r-project.org/***](https://www.r-project.org/)*,* open the installer, and follow the installation prompts.
2. Download Rstudio Desktop from [***https://www.rstudio.com***](https://www.rstudio.com/), open the installer, and follow the installation prompts.
3. Open RStudio and install the Bioconductor package *BioNAR* using following command.

if (!require("BiocManager", quietly = TRUE))

install.packages("BiocManager")

BiocManager::install(version = "3.17")

BiocManager::install("BioNAR")

library(BioNAR)

This will check for and then install the Bioconductor package manager “*BiocManager*” if it is not already installed. It will then install *BioNAR* from Bioconductor.

During installation process BiocManager could identify that some packages are obsolete, then it will ask:

**Old packages: 'x.y.z', …**

Update all/some/none? [a/s/n]:

If it is your first package installed, we suggest you type “n”, if you’ve already have working R environment it seems wise to type “a”.

1. Install additional utility packages:

BiocManager::install(c("pander",'randomcoloR','*plotly'*))

1. Check that library is installed correctly by using the command.

library(BioNAR)

**SUPPORT PROTOCOL 2:Building the sample network from *synaptome.db***

*BioNAR* includes a direct plugin to the Synaptic Proteome database, which contains 64 independent proteomic studies of mammalian synapses, recently curated into a single dataset describing a landscape of ∼8000 proteins (Sorokina et al., 2021). *BioNAR* does this through the Bioconductor package *synaptome.db*, allows users to obtain the respective gene information, e.g., subcellular localization, molecular interactions, brain region, gene ontology, disease association, and to construct customised protein–protein interaction network models for gene sets and entire subcellular compartments (Sorokina et al., 2022).

Support Protocol 2 describes how to extract a custom protein-protein interaction (PPI) network for postsynaptic compartment based on a gene set comprising proteins that were found in three or more independent postsynaptic studies.

**Necessary Resources:**

1. *Hardware*

A modern compute environment capable of running R > 4.3.0 and Rstudio > 2022.12.0.

1. *software*

R > 4.3.0 and Rstudio > 2022.12.0.

1. *files*

none

**Protocol steps with *step annotations***

1. Open RStudio and install the Bioconductor package *synaptome.db* using following command.

if (!require("BiocManager", quietly = TRUE))

install.packages("BiocManager")

BiocManager::install(version = "3.17")

BiocManager::install("synaptome.db")

library(synaptome.db)

This will check for and then install the Bioconductor package manager “*BiocManager*” if it is not already installed. It will then install *synaptome.db* from Bioconductor. During installation process BiocManager could identify that some packages have not most recent version, then it will ask:

**Old packages: 'x.y.z', …**

Update all/some/none? [a/s/n]:

If it is your first package installed, we suggest you type “n”, if you’ve already have working R environment it seems wise to type “a”.

1. Extract the compartment ID for postsynaptic compartment.

You need to navigate to the directory using setwd where you will store all the results, in our case it is “Sample network”. The command match will return the database ID for required compartment, which is “1” in our case (for postsynapse compartment).

setwd("~/Documents/Sample network/")

*cid<-match(*"*Postsynaptic*"*, getCompartments()$Name)*

*cid*

1. Get all the gene IDs for specified compartment.

*t<-getAllGenes4Compartment(cid)*

*dim(t)*

1. Select a subset of genes that were found in three or more studies.

In our example, the gene list for the postsynaptic compartment is large (5568 genes), and we want to reduce it to the most confident set of genes, found in at least three (arbitrary decision) different studies, which results in 2643 genes. To get this we specify “3” for findGeneByCompartmentPaperCnt command.

gp <- findGeneByCompartmentPaperCnt(3)

postgp <- gp[gp$Localisation == "Postsynaptic",]

dim(postgp)

head(postgp)

1. Build the PPI for selected set of genes.

library(BioNAR)

ggb<-buildFromSynaptomeGeneTable(postgp, type = "limited")

write\_graph(ggb, file = "PSDnetwork.gml", format = "gml")

Protein-protein interactions will be returned for a “*limited*” type of the network, which includes only genes presented in the list. Alternatively, it is possible to create a network of “*induced*” type, when an expanded network including all possible interactors of the listed genes will be obtained from the database. The *BioNAR* command buildFromSynaptomeGeneTable creates a “*limited*” type network and returns its Largest Connected Component. The “*induced*” type could be created with *synaptome.db* command getIGraphFromPPI and this functionality is out of scope of this paper (Sorokina et al., 2022). The resulting graph contains the one connected component consisting of 2297 vertices and 9406 edges, which we saved in a gml format to use for further analysis.

**BASIC PROTOCOL 2**

**Network properties and centrality.**

To assess any network model for biological plausibility, we often want to test a network's degree distribution for evidence of scale-free structure and compare it against an equivalent randomised network model. The assumption here is that a network based on noisy data will have more random architecture whereas biological networks typically tend towards a more scale-free structure. For this we use the R *PoweRlaw* package (version 0.50.0) (Gillespie, 2015), which deploys a goodness-of-fit approach to estimate the lower bound and the scaling parameter of the discrete power-law distribution for the optimal description of the graph degree distribution.

Although degree is the most commonly used vertex metric there are other centrality metrics that can be estimated from the graph structure. *BioNAR* directly supports the calculation of the following network vertex centrality measures. Some are implemented via *igraph* (Gabor & Tamas, 2006): degree (*DEG*), betweenness (*BET*), clustering coefficient (*CC*), page rank (*PR*) (see *igraph* manual for details). Other measures appear only in BioNAR: semilocal centrality (*SL*), mean shortest path (*mnSP*) and standard deviation of the shortest path (*sdSP*).

Calculated vertex centrality values can be added as vertex attributes (*calcCentrality*) or returned as an R matrix (*getCentralityMatrix*), depending on user preference. Any other numerical characteristics, calculated for vertices and represented in a matrix form, can also be stored as a vertex attribute (*applyMatrixToGraph*).

To enable comparison of an observed network’s vertex centrality values to those of an equivalently sized randomised graph, we enabled three randomisation models including G(n,p) Erdos-Renyi model (Erdos & Renyi, 1959) (‘*gnp*’, illustrated in example below), Barabasi-Albert preferential attachment model (Barabási & Albert, 1999) (‘*pa*’), and the derivation of a new randomised graph from a given graph by iteratively and randomly adding/removing (‘*cgnp*’) or rewiring edges (‘*rw*’).

For proteomic networks with matching multi-condition gene expression data, the scale-free structure has also been demonstrated by using the expression data to perform a perturbation analysis on the network to measure network entropy (Teschendorff et al., 2015). Originally, this kind of analysis was designed for comparing a control with a perturbed network (e.g., wild-type vs mutant, untreated vs treated), where vertices with low entropy rate appear to be the most important players in disease propagation.

However, for the assessment of scale-free structure we do not actually require gene expression data as it based solely on the network topology. In *BioNAR*, following the procedure described in (Teschendorff et al., 2015), all vertexes are artificially assigned a uniform weight then sequentially perturbed with the global entropy rate (*SR*) after each protein’s perturbation being calculated and plotted against the log of the protein’s degree (see Appendix for more detail). In case of scale-free or approximate scale-free topologies, we see a clear bi-modal response between over-weighted vertices and their degree and an opposing bi-phasic response in under-weighted vertices and their degrees.

Protocol 2 starts by estimating and fitting the observed graph degree distribution to a pawer law and comparing this against a random network. At the next step we estimate the graph entropy, again comparing it to the random graph. Finally, we estimate the range of centrality metrics for our graph and a randomly generated graphs, in our example we use the Erdos-Renyi model.

**Protocol steps with *step annotations***

1. Check the network degree distribution.

pFit <- fitDegree(as.vector(igraph::degree(graph=g)), Nsim=1000,

plot=TRUE,WIDTH=2480, HEIGHT=2480)

pwr <- slot(pFit,'alpha')

This command will take the graph created in Protocol 1, fit it to a power law distribution and return a figure of log-log plot of the CDF of postsynaptic PPI network degree distribution (Figure 2, A). During fitting code performs a bootstrapping hypothesis test to determine whether a power law distribution is plausible. The number of bootstrapping rounds is defined by *Nsim* parameter, which is set to 100 by default. Setting *Nsim* parameter to large values makes calculation slower, for example with *Nsim=1000* it takes couple of minutes to get a fit. By using slot function, we extract scaling coefficient α to compare with the random graph results.

1. Compare network degree distribution with a set of random graphs.

The following commands will generate a sample of 5000 random G(n,p) Erdos-Renyi graphs and estimate α coefficients of their best fits to the power law distribution. To save compute time we reduce the number of bootstrapping tests to 10 in this example, however it still takes about hour and a half to complete the task, replace 5000 with 500 to save time. The histogram of obtained values is plotted on the Figure 2, B, where the vertical line on the left indicates the value (2.63) for the actual graph.

lrgnp<-list()  
alphaGNP<-c()  
for(i in 1:5000){

rgnp<-BioNAR:::getGNP(g)

pFit <- fitDegree( as.vector(igraph::degree(graph=rgnp)),

Nsim=10, plot=FALSE,threads=5

p <- slot(pFit,'alpha')

lrgnp[[i]]<-rgnp

alphaGNP[i]<-p

}

qplot(alphaGNP)+geom\_vline(xintercept = pwr)

1. Calculate graph entropy.

As we do not have actual expression data for our graph we can use simulated data, see more detail in the Appendix. Command *getEntropyRate* estimates the global entropy rate for the whole network, while *getEntropy* estimates entropy value for each network vertex. Command *plotEntropy* visualises the results as a plot (Figure 3 A).

ent <- getEntropyRate(g)

ent

SRprime <- getEntropy(g, maxSr = ent$maxSr)

head(SRprime)

plotEntropy(SRprime, subTIT = "Entropy", SRo = ent$SRo, maxSr = ent$maxSr)

1. Calculate the graph centrality measures.

g <- calcCentrality(g)

The command *calcCentrality* assigns centrality measures as following node attributes: *DEG* - degree, *BET* - betweenness, *CC* - clustering coefficient, *SL* - semilocal centrality, *mnSP* - mean shortest path, *PR* - page rank, *sdSP* - standard deviation of the shortest path. Rather than saving these centrality values on the graph, e.g., to provide different names for the vertex centrality attributes, they are be obtained in a matrix form:

mc <- getCentralityMatrix(g)

The matrix will contain eight columns for each of the measures above with the rows corresponding to each node’s database ID (2297 nodes in total, synaptome.db version 0.99.12).

1. Calculate the graph centrality measures for a randomised graph.

ggrm <- getRandomGraphCentrality(g, type = c("cgnp"))

We need to select the type of randomisation, “*cgnp*” in our case, which corresponds to a sampling algorithm, which will randomly perturb the graph adjacency matrix and shuffle its vertices in such a way that the correlation between new and old adjacency matrix is 75%.

**BASIC PROTOCOL 3**

**Identifying the network community structure with clustering**

This protocol splits the network into communities using a non-exhaustive set of commonly used clustering algorithms for molecular networks. These include Modularity-Maximisation based algorithms, such as the popular agglomerative 'Fast-Greedy Community' algorithm (*fc*) (Clauset et al., 2004), process driven agglomerative random walk algorithm 'Walktrap' (*wt*) (Pons & Latapy, 2006), and coupled Potts/Simulated Annealing algorithm 'SpinGlass' (*sg*) (Reichardt & Bornholdt, 2006; Traag & Bruggeman, 2009), the divisive spectral based 'Leading-Eigenvector' (*lec*) (Newman, 2006) and fine-tuning (*Spectral*) (McLean et al., 2016) algorithms, and the hierarchical agglomerative 'Louvain' algorithm (*louvain*) (Blondel et al., 2008). We also included a non-modularity information-theory based algorithm ‘InfoMAP' (*infomap*) (Rosvall et al., 2009; Rosvall & Bergstrom, 2008). All algorithm implementations, apart from Spectral, are imported from R’s *igraph* package (Gabor & Tamas, 2006). The Spectral algorithm (McLean et al., 2016) was written in C++ and wrapped in R within a satellite CRAN package *rSpectral*, (*https://cran.r-project.org/web/packages/rSpectral/index.html*), linked to *BioNAR* (see Supplementary Methods). Default parameters used in the fc, lec, sg, wt and louvain algorithms were chosen to maximise the measure Modularity (Newman & Girvan, 2004); infomap seeks the optimal community structure in the data by maximising the objective function called the Minimum Description Length (Rissanen, 1978; Grwald et al., 2005).

The protocol first obtains community structures for the network used in previous protocol. The user can select some or all the available clustering algorithms to run simultaneously. This community structure(s) can then be visualised. The protocol can also iterate by re-clustering the largest communities after each cycle (threshold for communities to be re-clustered can be defined). Finally, we extract a summarising table that compares across all the clustering algorithms used in the analysis.

**Protocol steps with *step annotations***

1. Use the network *g* obtained with previous protocols.
2. Cluster the network.

We provide two functions: `calcClustering` and `calcAllClustering` that use calcMembership to calculate community memberships and store them within the graph vertices attributes named after the algorithm. The difference between them is that `calcAllClustering` allows the user to calculate memberships for all clustering algorithms simultaneously (this can be slow especially on larger graphs, e.g., up to 30 min for our full example network), and store them as graph vertices attributes, while `calcClustering` command allows users to select a specific algorithm.

g <- calcAllClustering(g)

summary(g)

The resulting graph summary in the example will contain the graph memberships for all nine clustering algorithms: ‘Fast-Greedy Community’ algorithm (fc), ‘Leading-Eigenvector’ (lec), ‘Walktrap’ (wt), ‘SpinGlass’ (sg), ‘Spectral’ (spectral), ‘Louvain’ (louvain) and ‘InfoMAP’ (infomap). SpinGlass algorithm is implemented as three instances (sgG1, sgG2, and sgG5) each based on a selected gamma value (1, 2, and 5 respectively) to specify the desired cluster size.

The user also has the option to select one or more specific clustering algorithm to run over their network, since running all clustering algorithms over the large network might be time consuming. For instance, to cluster the network *g* using only the ‘louvain’ algorithm, the command would look like:

gl <- calcClustering(g, alg = “louvain”)

summary(gl)

1. Compare the output of clustering algorithms.

To compare across different clustering algorithms on a network, a summary matrix is created, consisting of: the maximum Modularity obtained (mod), the number of detected communities (C), the number of singlet communities (Cn1), the number of communities with size >= 100 (Cn100), the fraction of edges lying between communities (mu), the size of the smallest community (Min. C) and the largest community (Max. C), the average (Mean C), median (Median C), first quartile (1st Qu. C), and third quartile (3rd Qu. C) of community size (Table 1).

m <- clusteringSummary(g)

View(m)

1. Visualise the community structure results.

The *BioNAR* package provides functionality to visualise a network’s community structure with an in-built cluster-driven layout algorithm, which is suitable for networks up to tens of thousands of vertices and millions of edges. This layout splits the network into clusters, lays out each cluster individually, and then combines individual layouts with the *igraph* function merge\_coords, so that each distinct community is shown independently and painted in a unique colour. In our example case we show the community structure for the results obtained by “louvain” clustering algorithm,

which resulted in 15 communities (see Table 1). For visualisation purposes we need to extract the membership in a form of a data frame. Palette is provided using distinctColorPalette command from the package *randomcoloR*, which defined an individual colour for each community; layoutByCluster command calculates the layout. Additionally, you can define edge color, size of the node and position of the legend (Figure 4A).

library(randomcoloR)

mem.df<-data.frame(names=V(g)$name,membership=as.numeric(V(g)$louvain))

palette <- distinctColorPalette(max(as.numeric(mem.df$membership)))

lay<-layoutByCluster(g,mem.df,layout = layout\_nicely)

plot(g,vertex.size=3,layout=lay, vertex.label=NA, vertex.color=palette[as.numeric(mem.df$membership)],edge.color='grey95')

legend('topright',legend=names(table(mem.df$membership)), col=palette,pch=19,ncol = 2)

1. Recluster the obtained community structure (optional)

A common phenomenon when applying Modularity based clustering algorithms over networks of a large size, is to end with a small number of larger, or ‘super’, communities which masks any substructures within these super communities. In this situation we provide the user the facility to re-cluster these large/super communities in a hierarchical manner, applying the same, or potentially a different, clustering algorithm at each iteration (recluster). We also need to specify the threshold for cluster size, which will be retained during reclustering procedure, which is 10 in our case, and we use the same algorithm again.

remem<-calcReclusterMatrix(g,mem.df,alg = "louvain",10)

head(remem)

This will return a table where each vertex name (1st column) will be assigned a cluster membership in the original graph (2nd column) and another cluster membership in the reclustered graph (3rd column). In the example case of Louvain clustering results, the original clustered graph had 15 communities, and the reclustered graph had 127.

1. Visualise the reclustered community structure.

lay<-layoutByRecluster(g,remem,layout\_nicely)

plot(g,vertex.size=3,layout=lay,

vertex.label=NA,

vertex.color=palette[as.numeric(mem.df$membership)],

edge.color='grey95')

legend('topright',legend=names(table(mem.df$membership)),

col=palette,pch=19,ncol = 2)

The reclustered graph is shown at Figure 4 B.

**BASIC PROTOCOL 4**

**Identifying the optimal clustering algorithm**

The algorithms considered in Basic Protocol 3 are based on a range of different mathematical approaches and will give different results for the same network (which we can clearly see in Table 1). However, for further analysis, presentation or publication, one usually needs to select one clustered network model based on a single algorithm. In most cases, there is no ground truth, and the choice of ‘correct’ or ‘best’ clustering algorithm is subjective. While accepting that there is no ‘correct’ or ‘best’ clustering approach we can at least assess which algorithms identify ‘useful’ clusters. To do this we rely on an assumption that proteins/molecules that are well known to cooperate in the same complex/molecular pathways should be more likely to be found together in the same cluster. Thus, we can use our functional annotations (e.g. GO) to help identify clustering methods that fit that assumption.

To achieve this, we will estimate the number of enriched communities for each algorithm and for each annotation term, considering the network’s size, size of each cluster, the number of annotated genes in the network and in the clusters (see Appendix for background details that underpin the method).

The protocol starts by estimating the overrepresentation of annotation terms in each community discovered by each clustering algorithm. Over-representation analysis (ORA) is a common approach to identify annotation terms that are significantly over- or under-represented in a given set of vertices compared to a random distribution. In biological networks Gene Ontology terms and Pathway names are amongst the most frequently used. ORA differs from Gene Set Enrichment Analysis (GSEA) as the latter use numerical values associated with genes, such as expression value, while the former relies on presence/absence data utilising null hypothesis tests, such as the hypergeometric test. To keep the package as general purpose as possible, we used the Bioconductor package *fgsea* to implement ORA functionality on top of arbitrary string vertex annotation and vertex grouping, obtained by clustering (but not limited by it). We represent the results of ORA as an R data frame, with rows representing the combination of annotation term and cluster and columns – the enrichment characteristics, including size of the cluster (Cn), number of annotated vertices in the graph (Fn), number of annotated vertices in the cluster (Mu), odds ratio (OR) and its 95% Confidence interval (CIl and CIu), Fold enrichment (Fe and FC). We also provide p-value, adjusted p-value, size of overlap and the list of vertices from the cluster that contribute to the annotation term. Using the odds ratio allows us to distinguish functionally enriched communities relative to functionally depleted communities.

If we were to rank algorithms simply according to the percentage of functionally enriched communities, it would not tell us anything about the size of community the enrichment originates from, i.e., if enrichment occurs within extremely large or small communities. Therefore, we plot the fraction of functionally enriched communities greater or equal to the log of its Fold-enrichment (Fe) value, measured at 100 intervals taken from 0 to the maximum Fe value (the maximum found from all algorithms studied). Since Fe values take into consideration the size of communities, at this step we can exclude enrichment from communities at the extreme sizes.  In general, functional enrichment of non-extreme communities is observed by those clustering algorithms following a sigmoid distribution (see Figure 5, A). Thus, we test how well each distribution fits to a generalised sigmoid function using the two-sample Kolmogorov-Smirnov (KS) test to the goodness of fit of each distribution to our set of five idealised sigmoid curves (see Appendix for technical detail).

To reproduce clustering results analysed in this step we recommend using the example network, built and stored in the external file “PSD\_annot\_cls.gml” in *BioNAR*. The network in this file is the same as in previous protocols but contains the clustering results already assigned to each node.

**Protocol steps with *step annotations***

1. Load the network from *BioNAR*.

file <- system.file("extdata", "PSD\_annot\_cls.gml", package = "BioNAR")  
gg <- igraph::read.graph(file,format="gml") *#graph from gml*  
summary(gg)

1. Estimate enrichment for the one clustering algorithm and one annotation.

If we want to take a look at overrepresentation for specific algorithm, we need to select the algorithm, and use the function clusterORA to perform overrepresentation analysis for the results obtained with specified algorithm. We also need to specify the annotation set that we will use for assessing the clustering results. Here we use GO Molecular function annotation (‘GOMFID’), but it could be Biological Process or any other annotation, stored as a node attribute in our graph.

ora<- clusterORA(gg, 'louvain', name = 'GOMFID',

vid = "name", alpha = 1, col = COLLAPSE)

1. Estimate enrichment for all selected clustering algorithms against the annotation set.

To compare all the clustering results, we need to select all algorithms, which we want to compare. Then, using function clusterORA we perform the overrepresentation analysis on the results obtained with specified algorithms. As before, we specify the annotation: GOMFID. Finally, we need to calculate FeMax, which corresponds to the maximum Fe value, and FcMax, which corresponds to maximum of Fc value. These values will be needed for further analysis to specify the required range of Fe and Fc values.

algs<-c('lec', 'wt', 'fc', 'infomap', 'louvain',

'sgG1', 'sgG2', 'sgG5', 'spectral')

ora<-lapply(algs, function(alg){clusterORA(gg, alg, name = 'GOMFID',

vid = "name",alpha = 1, col = COLLAPSE)})

names(ora)<-algs

FeMax<-log2(max(sapply(ora,function(d){max(d$Fe)})))

FcMax<-log2(max(sapply(ora,function(d){max(d$Fc)})))

1. Analyze unadjusted p-values for enrichment and print summary table (optional).

The command summaryStats will combine the results of enrichment obtained in the step 1, which will produce a list of tables: ‘SUM’, ‘SUM2’, ‘SUM3’, ‘SUM4’ and ‘CAN’, where ‘SUM’ table contains main summary that can be used for detailed analysis of algorithm/term pairs, which is outside the scope of this protocol; ‘SUM2’, ‘SUM3’ and ‘SUM4’ contain the auxiliary values for further fitting to sigmoid curves.

The key table for the enrichment analysis is ‘CAN’ which records the annotation-term-to-cluster-association data for each clustering algorithm. This shows the user which clusters an enriched annotated term is associated with. It can be seen from the example below that table ‘CAN’ consists of 4 columns, where ‘ALG’ corresponds to the name of the algorithm, ‘Fn’ – to the respective GO term ID, ‘C’- number of the cluster, and ‘Mu’ – number of the genes in the cluster annotated with this term (cardinality of the pair) (see Table 2 for the first six rows of ‘CAN’ table.

statsR1 <- summaryStats(ora, 0.1, usePadj=FALSE, FeMAX=FeMax, FcMAX=FcMax).

names(statsR1)

View(head(statsR1$CAN))

1. Plot the fraction of enriched communities and rank the algorithms.

The command plotRatio creates a rank table for the algorithms and four plots: p1-4, all showing the distribution of Fraction of enriched communities against Fe (Fold-enrichment). p1 and p3 show the distribution plotted against Log2(Fe), while p2 and p4 use Log(Log2(Fe)). p1 and p2 highlight the top three distributions, while p3 and p4 (shown on Figure 5, B) plot all algorithms each with a different colour. From the Table 3 and Figure 5, B it can be seen that Louvain algorithm gives the most useful distribution, followed by sgG2 and fc (for the example given).

plots<-plotRatio(x=statsR1, desc="p.values",LEGtextSize=0.75, LEGlineSize=2)

View(plots$ranktable) print(plots$p3)

1. Fit sigmoid function.

This command fits a sigmoid distribution (described in Appendix) to the fraction of enriched communities against Fe values for each clustering algorithm (Figure 6). The function creates a list called fitres, which contains the fitting results. To visualize the fitting results the user needs to print gridplot, which is the part of fitres.

fitres<-fitSigmoid(statsR1)

print(fitres[['0']]$gridplot)

To view the fitting results, you will need to select fitInfo. They will be presented in a form of table with columns: ‘alg’, corresponding to algorithm’s name, ‘isConv’ – if the fit has converged, ‘finTol’ – to final fitting error, ‘stopMessage’- message describing the reason to stop fitting. You can also assess the results of two-sample Kolmogorov-Smirnov (KS) test by selecting ks below. Here the columns will correspond to the p-values from Kolmogorov-Smirnov test of correspondence between Fe distribution and sigmoid function rates.

View(fitres[['0']]$fitInfo,caption = 'Summary of the fitting results')

View(fitres[['0']]$ks,caption = 'Summary of the Kolmogorov-Smirnov test')

1. Test the robustness of obtained fitting results.

To test the robustness of fitting, we can add some noise to results obtained in previous step. We added noise to each data point by randomly sampling from a Gaussian distribution with mean zero, and standard deviation of [0.01, 0.05, 0.1, 0.5]. For this we must specify the level of noise (‘0.05’ here) whilst executing the in the commands in step 4. The resulting fitting results with noise and values can be assessed in similar way (Figure 7). It could be seen that fc, Louvain and sgG1 algorithms give the more robust results.

print(fitres[['0.05']]$gridplot)

View(fitres[['0.05']]$fitInfo,align = 'lllrrrl',landscape = TRUE,

main = 'Summary of the fitting results')

View(fitres[['0.05']]$ks,align = 'lllrrrl',landscape = TRUE,

main = 'Summary of the Kolmogorov-Smirnov test')

*.*

We should note here that users should be cautious not use the same datasets (directly or indirectly) for generating the clustered network and testing it. In our example here we used Gene Ontology terms which are not linked to the methods or datasets used in either the clustering methods or the construction of the network architecture including selection of nodes and edges.

**BASIC PROTOCOL 5**

**Identifying the influential proteins with bridgeness.**

Not all proteins act similarly in propagating signals, or information, through the network. It is often assumed that proteins that interact with lots of partners have more significant impact on signal propagation or on disease mechanisms when perturbed. It is also generally found that the majority of proteins interact with just a few neighbours and thus their contribution is generally predicted to be less impactful (Vidal et al., 2011). The importance of a protein in propagating information appears to be dependent on its nearest neighbours, as well as its ability to influence other communities relative to the one it most likely belongs (Nepusz et al., 2008; Nepusz et al., 2012). Bridgeness metric, reflects the probability that a vertex could belong to more than one community at the same time, thus providing a useful measure to rank the vertices based on their topological influence and form linkages between clusters in the network model (Nepusz et al., 2008; Nepusz et al., 2012).

The protocol starts from the testing of the robustness of communities obtained in the “most useful” algorithm, obtained in Protocol 3. For this, the consensus matrix is produced by creating smaller network by randomly keeping a proportion (by default 80%) of the network edges (*type*=1) or vertices (*type*=2) and rerunning the clustering algorithm on largest connected component of that network. This is then repeated to produce a distribution (matrix) of clustered networks (by default set to 500 times). This new matrix is used to estimate the bridgeness, which takes the values between 0 - implying a vertex clearly belongs to a single community, to 1 - implying a vertex forms a ‘global bridge’ across every community in the network with the same probability (Nepusz et al., 2008; Nepusz et al., 2012).

Although useful itself as a measure of a “global” network importance, bridgeness becomes more informative when combined with other vertex centrality measures, e.g., semi-local centrality. Semi-local centrality considers the nearest and next to the nearest vertex neighbours, so reflects the “local” importance of the protein. It also lies between 0 and 1 indicating whether the vertex is likely to have local influence. Plotting bridgeness against semi-local centrality, allows us to categorise both the local and global influence of each vertex within a network given only the network structure (see case study). *BioNAR* also supports the comparison of bridgeness against any vertex centrality measure (or any normalised numeric vertex value) of the user’s choice, e.g., against Page Rank.

The protocol for doing this starts by estimating the bridgeness from the consensus matrix obtained in Protocol 4 and proceeds by plotting the bridgeness against the centrality measure of choice, which is implemented in two ways. In this example we continue working with the network from Protocol 4 and the algorithm, which showed the best enrichment performance, i.e. ‘louvain’.

**Protocol steps with *step annotations***

1. First, calculate the consensus community structure:

conmat <- makeConsensusMatrix(g, N=5,

alg = "Louvain", type = 2,

mask = 10, reclust = FALSE,

Cnmax = 10)

Here, *'alg'* selects the clustering algorithm to be used, *'type'* the sampling scheme (1 sample edges, and 2 sample vertices) used, *'mask'* the percentage of edges or vertices to remove from the graph, and *'reclust'* whether reclustering should be performed on the community set found, *'Cnmin'* minimum cluster size and *'Cnmax'* the maximum cluster size above which reclustering will be performed (if *reClust*=TRUE). The resulting matrix *conmat* has dimensions 2297 x 2297, where each matrix element is assigned the frequency with which a pair of nodes vertices is found in the same cluster.

1. Calculate the robustness of community structure.

Next the robustness of discovered communities will be tested, assigning to each cluster a value in a range from 0, indicating no confidence in the community existing, to 1, indicating absolute confidence in the cluster existing, thus evaluating the "goodness" of a chosen clustering algorithm.

clrob<-getRobustness(g, alg = "louvain", conmat)

pander(clrob)

This will return a table (15x5), where each of the 15 clusters obtain by Louvain algorithm has the following values: *C* (cluster number), *Cn* (cluster size), *Crob* (cluster robustness), and *CrobScaled* (0-1, *Crob* after scaling). *CrobScaled* should be used to indicate cluster robustness. It can be seen in the example that the most robust cluster is Cl9 (*CrobScaled* = 1) and the least robust is Cl15 (*CrobScaled* = 0).

1. Calculate the bridgeness.

br<-getBridgeness(gg,alg = "louvain", conmat)

head(br)

The command *getBridgeness* takes the graph and consensus matrix (conmat) from step 1. as an input and returns the bridgeness results in a form of table with three columns, where the first column contains *ID* (Human Entrez ID), second *GENE.NAME* (Human Gene Name) and the third - *BRIDGENESS.louvain* (bridgeness values obtained in louvain algorithm). Next, assign bridgeness values as vertex attributes:

g<-calcBridgeness(g,alg = alg, conmat)

vertex\_attr\_names(g)

For convenience, bridgeness values will also be stored as vertex attributes.

1. Plot bridgeness against the semilocal centrality (*SL*).

If we want to highlight the proteins of interest on the plot, we can specify their IDs as VIP vertices. In our case we will highlight the synaptic proteins with known function “Protein cluster”, based on a classification, extracted from an unrelated published study (Lips et al., 2012), which is also stored in *BioNAR* as an external data file.

sfile<-system.file("extdata", "SCH\_flatfile.csv", package = "BioNAR")

shan<- read.table(sfile,sep="\t",skip=1,header=FALSE,strip.white=TRUE,quote="")

head(shan)

Table shan consists of three columns, where the first column contains the function number, second the function description, and third the gene ID (Entrez ID), which is associated with this function. Here, we want to select gene IDs with “protein cluster” functions.

table(shan$V2)

shan[shan$V2 =="Protein\_cluster",] -> prCl

dim(prCl)

plotBridgeness(g,alg = "louvain",

VIPs=prCl$V3,

Xatt='SL',

Xlab = "Semilocal Centrality (SL)",

Ylab = "Bridgeness (B)",

bsize = 3,

spsize =7,

MainDivSize = 0.8,

xmin = 0,

xmax = 1,

ymin = 0,

ymax = 1,

baseColor="royalblue2",

SPColor="royalblue2")

*.*

By plotting bridgeness against semi-local centrality (Figure 8) we have categorised the influence each protein found in our network has on the overall network structure:

* Region 1 (top left), proteins having a 'global' rather than 'local' influence in the network (vertices in this region have also been referred to as bottle-neck bridges, connector or kinless hubs (0<SL<0.5; 0.5<B<1).
* Region 2 (top right), proteins having 'global' and 'local' influence (0.5<SL<1, 0.5<B<1).
* Region 3 (bottom left), proteins centred within the community they belong to, but also communicating with a few other specific communities (0<SL<0.5; 0<B<0.5).
* Region 4 (bottom right) proteins with 'local' impact, primarily restricted to just one or two communities (also referred to as local or party hubs, 0.5<SL<1, 0<B<0.5).

To plot the same figure in an interactive manner (to see the names for non-highlighted dots) the function ggplotly from library *plotly* can be called as follows:

library(plotly)

gp<-plotBridgeness(g,alg = "louvain",

VIPs= prCl$V3,

Xatt='SL',

Xlab = "Semilocal Centrality (SL)",

Ylab = "Bridgeness (B)",

bsize = 1,

spsize =2,

MainDivSize = 0.8,

xmin = 0,

xmax = 1,

ymin = 0,

ymax = 1,

baseColor="royalblue2",

SPColor="royalblue2")

ggplotly(gp)

**BASIC PROTOCOL 6**

**Identification of overlapping annotations.**

This protocol describes how to identify which and where, within the network, different annotations tend to overlap (or conversely are distinct and separate). For example, within large-scale molecular networks, disease associated genes are often found closely linked to one another (referred to as disease modules by (Menche et al., 2015) and based on the shortest path in the network linking the disease terms) and the composition of these modules can be compared across different diseases. Disease annotations which tend to overlap in these modules also tend to show significant similarities at the level of gene co-expression patterns, clinical phenotype, and comorbidity. Conversely disease annotations residing in separated network neighbourhoods appear to be more phenotypically distinct.

This phenomenon is not restricted to diseases and can be generalised so that given two annotations distributed across a network, a common query would be to find the points of intersection where the two annotation sets overlap (or segregate). To support such queries, we implemented the algorithm from (Menche et al., 2015), which tests if the observed mean shortest paths between two distinct annotation sets, superimposed on a network, is significant compared to a randomly annotated network.

The following example shows the estimation of disease separation for a manually selected set of diseases: DOID:10652 (Alzheimer's\_disease), DOID:3312 (Bipolar\_disorder), DOID:14330 (Parkinson’s disease), DOID:0060041 (Autistic Spectrum disorder), DOID:1826 (Epilepsy syndrome), DOID:5419 (Schizophrenia), DOID:9255 (Frontotemporal Dementia), DOID:1059 (Intellectual Disability). Command calcDiseasePairs estimates the separation of pairwise annotations across the graph and compares it with a randomly reannotated graph. This can be useful for generating a qualitative overview of the relationships between the annotations.

We do not need the cluster structure for this analysis, so the annotated network from Protocol 2 (g) is sufficient to perform the Protocol 6.

**Protocol steps with *step annotations***

1. Calculate the annotation pairs.

The *BioNAR* command calcDiseasePairs calculates the observed overlap between two annotation sets on a network, and compares this to a single instance of the network with annotations randomly permuted; this is useful for a qualitative estimate of how likely an overlap is simply a random occurrence.

To calculate annotation overlap we will need to specify the name of annotation. In our example it is TopOntoOVGHDOID, which contains the HDO IDs for diseases. We will also need to specify the IDs, for which the overlap will be calculated. ‘r’ corresponds to random permutations (other options are: ‘none’ -- no permutation will be applied at all, and ‘binned’ when permutations would try to take into account the node degree, which is slightly longer than random).

p <- calcDiseasePairs(

g,

name = " TopOnto\_OVG\_HDO\_ID",

diseases = c("DOID:10652","DOID:3312", "DOID:14330", "DOID:0060041", "DOID:1826", "DOID:5419", "DOID:9255", "DOID:1059"),

permute = "r"

)

pander(p$disease\_separation)

The command pander will return a snapshot of 8x8 table, where for each of disease pair there is a value, which ranges from negative (indicates potential overlap) to positive (indicates separation).

1. Calculate significance of the annotation overlap.

To calculate the significance of observed overlaps (or separations) the observed annotation pairs on the network the command runPermDisease should be used. This compares the overlap against multiple permutations of the network (where the user can define the number of permutations). Executing this command may take considerable time depending on the number of permutations chosen. It generates a results table containing the overlap of each annotation pair with p-value, p.adjusted by Bonferroni test, and q-value.

We selected 100 permutations in our simple example below, but for better significance there should be 10000 permutations (~ 1 hour of computing time for this size of network and this set of annotations).

r <- runPermDisease(

g,

name = "TopOnto\_OVG\_HDO\_ID",

diseases = c("DOID:10652","DOID:3312"),

Nperm = 100,

alpha = c(0.05, 0.01, 0.001)

)

pander(r$Disease\_overlap\_sig)

The pander command will return a table which contains the number of rows equivalent to annotation pairs and the following columns: HDO.ID.A – HDO ID for the first disease, N.A. – number of genes associated with this disease, HDO.ID.B – HDO I for the second disease, N.B – number of genes associated with second disease. sAB – a value, which characterises the overlap (negative for overlap and positive for separation), Separated – is the pair separated based on sAB, Overlap- is the pair overlapping based on sAB, zScore – Zscore for this pair of annotations (see the formula in REF), p-value -respective p-value for the overlap, Separation/Overlap.than.chance – if the ratio the separation and overlap is bigger than chance, Bonferroni – Bonferroni value for the overlap, p.adjusted – p-value, adjusted for multiple testing, q-value, p-value adjusted by FDR (the resulting table if this step is presented as a Supplementary Table 1). It can be seen that a few pairs of diseases are significantly overlapping (highlighted in yellow), e.g. Alzheimer Disease and Parkinson Disease (p.adj = 6.01E-06, bonf \*\*\*), Schizophrenia and Bipolar Disorder (4.25E-013, bonf \*\*\*), Autistic Spectrum Disorder and Bipolar Disorder (5.51E-08, bonf \*\*\*)

**GUIDELINES FOR UNDERSTANDING RESULTS**

Each of the protocols described here have been illustrated using a walk-through example from the mammalian synaptic proteome (See Figure 1). In brief we extracted a consensus list of proteins from a curated and published database of synaptic proteomics studies. We filtered the database to extract a list of proteins that were found in studies looking at the post-synaptic density in mouse, rat or human and were reported by at least three different published studies. The same database also contains a curated list of direct protein-protein interactions between these proteins extracted from public databases.

Basic Protocol 1 walks the user through the initial generation of a network model containing vertices (proteins) and edges (protein-protein interactions). For your own network you will need both these elements to build the network. One word of caution is to consider how well defined the interactions are: Are the interactions based on direct molecule to molecule data? Many interaction databases comprise a mix of direct and indirect interactions where indirect interactions mean that there may be undefined intermediate partners involved in the interaction between two species. These are factors that need to be considered depending on the application area. Alternatively, a model can be assembled using any other network package and saved in gml format for importing into *BioNAR*. The advantage of using *BioNAR* in this step is that it can be scripted for multiple instances of networks from larger datasets.

Basic Protocol 2 then provides a comprehensive suite of methods to analyse the network structure. The first of three sub procedures calculate a set of commonly used core network parameters which are generally used to assess whether the network looks real or could be contaminated with random noise (biological network models do typically contain at least some random noise). *BioNAR* offers two methods to measure how scale-free a network is, operating under the assumption that real biological networks tend towards a scale free rather than a random architecture. One is based on comparing the observed network against a distribution of generated random networks of the same size and the other based on measuring entropy within the network.

Basic Protocol 3 then applies a set of frequently used clustering algorithms to the raw network generated in Basic Protocol 1. *BioNAR* enables the use of a hierarchical approach to the network where large (defined by user) clusters can be treated as networks in their own right and clustered again to dissect substructures within. Finally, the clustered network can be visualised graphically within *BioNAR* (figure 4), or alternatively a clustered network can be exported to be visualised in a package of choice.

Basic Protocol 4 considers the tricky problem of which clustering algorithm is ‘best’. There are few, if any, reliable ground truths and clustering algorithms all behave differently as networks change in size and/or density of connections. Rather than call any algorithm “better” we prefer the term “useful”. The basic concept here is that we usually do know something about the networks we are working on. In the case of protein interaction networks there are well known classic biochemical pathways and the molecules within these pathways tend to share annotation terms in GO and they generally directly interact with one another. Therefore, it is not unreasonable to expect these terms to enrich within specific clusters and indeed we do see this in clustered networks. *BioNAR* provides a suite of tools to compare across these networks and plot statistics that enable the user to assess how useful different clustering algorithms might be in terms of producing clusters which reproduce the small elements of prior knowledge we do have. Of course, any assessment based on optimising enrichment will depend on the quality of the annotations used and how relevant they are to the biology. In all cases it is vital to report exactly how a clustering algorithm was selected and share the raw network (e.g. from Basic Protocol 1) to allow others to try alternative algorithms.

Most clustering methods are based on a fundamental assumption that the node exists in exactly one cluster within the network. However, molecules can move, they exist in in multiple copies which permit different instances to be involved in different places at the same time. Further the methods used to identify molecules, or their interactions often lack full molecular resolution and identify a parent/canonical molecule rather than differentiate between all the expressed variants in a cell or tissue sample. As a partial move towards a more probabilistic network model, we incorporated bridgeness to *BioNAR* in Basic Protocol 5. Through sampling the network and re-clustering multiple times we can estimate how likely each molecule is to belong to each specific cluster. While many vertices can be assigned fairly easily and are robust to perturbation, we find others that are much more susceptible to perturbation and may reflect molecules that are pleotropic in their influence and interact with multiple pathways or molecular complexes. When we combine this measure with a local influencing metric, we can start to tease out molecules that are core to specific clusters vs molecules that have broad influence over the entire network.

Finally, we included Basic Protocol 6 to allow the users to examine relationships between annotations in the content of the network they have. This does not use the clustering in protocols 3-5 but rather builds off the initial annotated raw network in Basic Protocol 1. The results from this can provide fundamental insights into why annotation terms (such as genes associated with disease) often correlate despite there being no obvious similarity at the sequence level. When we overlay this onto the molecular interaction network we can reveal where in the network these interactions occur and identify other molecules that are intimately associated with the correlation that might underpin a shared mechanism of disease. Put more simply the molecules with shared function might be different at the sequence level but they tend to be close within the network. Conceptually this is the reverse of protocol 4.

**COMMENTARY**

**Background Information:**

Communication, or information transfer, between the components of a network is a common feature across many biological scales. Therefore, it is not entirely surprising that methods designed to analyse information flow in other networks have proven to be well suited to the analysis of biological networks. The analysis of social networks has proven to be a rich hunting ground for useful methods where many have translated well for the analysis of molecular interaction networks (Li et al., 2011; Ma & Zeng, 2003; Wunderlich & Mirny, 2006).

Many individual software tools have been developed to address the basic steps required for the type of analyses described above. For example, the *igraph* package in R supports building a network, estimating popular centrality measures and several types of clustering (Gabor & Tamas, 2006). Cytoscape, (Shannon et al., 2003) and its plugins, support graphical representation and reconstruction of molecular networks and provide a number of approaches to extract interactions from a variety of sources, such as the STRING interaction database (von Mering et al., 2003), and perform clustering and estimation of the main centrality measures. Various other tools exist for functional enrichment analysis based on the GO, KEGG (Kanehisa & Goto, 2000) and Reactome (Gillespie et al., 2022) ontologies such as the widely used DAVID package (Sherman et al., 2022), GOrilla (Eden et al., 2009), BiNGO (Maere et al., 2005), and GSEA (Subramanian et al., 2005), to name a few. The Bioconductor (Huber et al., 2015) package *ClusterProfiler* allows estimation of the annotation overrepresentation for the clusters within a network (Wu et al., 2021) and the package *fgsea* provides fast enrichment analysis against an arbitrary set of annotations (Korotkevich et al., 2021).

*BioNAR* combines the features above and added new ones, reflecting the research the authors and their collaborators have been involved with in recent years. The methods presented here will work with any other similar type of graphical data. The protocols were described here for undirected graphs but some of the methods will work fine on directed graphs and incorporation of the graph directionality into package code is ongoing effort. The vertices are unweighted, although some methods will work on weighted graphs, but the current package has not been tested on them. Finally, *BioNAR* does not currently delve into probabilistic graphical models. All three of these are avenues for future development and as *BioNAR* is entirely open-source there are no restrictions on end-user customisation or extension.

**Critical Parameters:**

Users of these protocols should ensure that they are using a recent version of R and *BioNAR*. At the time of this writing, R (4.2.2) and BioNAR (1.2.0) are used.

**Troubleshooting:**

Some potential errors and respective solutions are presented in Table 4.

**Table 4.**Sources and Solutions to Potential Errors

|  |  |  |
| --- | --- | --- |
| **Problem** | **Possible Cause** | **Solution** |
| The are no results created in the environment by the command. | There is typo or mistake in the command, such as wrong argument name or non-existing variable. | Type the following: options("show.error.messages"=TRUE) and repeat the command. R will give you meaningful description of the error. |
| The getEntropy function returns NaN in UP and DOWN columns | The entropy calculation assumes fully connected graph. In the case of disconnected graph entropy value is set to NaN. | Get the largest connected component of your network by gLCC<-findLCC(g) and repeat analysis with gLCC. |
| The getBridgeness function returns NaN in UP and DOWN columns | The bridgeness calculation assumes fully connected graph. In the case of disconnected graph entropy value is set to NA. | Get the largest connected component of your network by gLCC<-findLCC(g) and repeat analysis with gLCC. |
| Warning is issued:  1: In getClustering (gg, alg):  Clustering calculations for algorithm … failed. NULL is returned. | Some clustering algorithms, in particular SpinGlass, assume fully connected graph. In the case of disconnected graph warning is shown, NULL is returned as a clustering result and no membership attribute is added to the graph. | Get the largest connected component of your network by gLCC<-findLCC(g) and repeat analysis with gLCC. |

All functions in BioNAR have built-in help, accessible by typing ?function\_name. In addition, documentation exists at http://www.qiime.org, and the help forum at http://forum.qiime.org is typically quite active and useful.

**CONFLICT OF INTEREST STATEMENT:**

The authors do not have conflicts of interest.

**DATA AVAILABILITY STATEMENT:**

The package is available from Bioconductor release 3.17: https://bioconductor.org/packages/release/bioc/html/BioNAR.html

**ACKNOWLEDGEMENTS:**

This research has received funding from the European Union’s Horizon 2020 Framework Programme for Research and Innovation [No 945539 (Human Brain Project SGA3)], and Cancer Research UK [No CTRQQR-2021\100006 (Cancer Research UK Scotland Centre)].

**LITERATURE CITED:**

Bibliographic listing of the references cited in the text. Current Protocols uses APA style. References should be listed alphabetically by author’s last name. Should you wish to use EndNote or Zotero, please download the corresponding output style on our For Author’s Page (<https://currentprotocols.onlinelibrary.wiley.com/hub/forauthors>).

Barabasi, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509-512. <https://doi.org/10.1126/science.286.5439.509>

Benjamini, Y., & Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics*, *29*(4), 1165-1188, 1124.

Blondel, V., Guillaume, J., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *P10008*.

Bánky, D., Iván, G., & Grolmusz, V. (2013). Equal opportunity for low-degree network nodes: a PageRank-based method for protein target identification in metabolic graphs. *PLoS One*, *8*(1), e54204. <https://doi.org/10.1371/journal.pone.0054204>

Clauset, A., Newman, M. E., & Moore, C. (2004). Finding community structure in very large networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, *70*(6 Pt 2), 066111. <https://doi.org/10.1103/PhysRevE.70.066111>

Eden, E., Navon, R., Steinfeld, I., Lipson, D., & Yakhini, Z. (2009). GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics*, *10*, 48. <https://doi.org/10.1186/1471-2105-10-48>

Erdos, P., & Renyi, A. (1959). On random graphs. *Publicationes Mathematicae*, *6*, 290–297.

Fernandez, E., Collins, M. O., Uren, R. T., Kopanitsa, M. V., Komiyama, N. H., Croning, M. D., . . . Grant, S. G. (2009). Targeted tandem affinity purification of PSD-95 recovers core postsynaptic complexes and schizophrenia susceptibility proteins. *Mol Syst Biol*, *5*, 269. <https://doi.org/10.1038/msb.2009.27>

Gabor, C., & Tamas, N. (2006). The igraph software package for complex network research. *InterJournal*, *Complex Systems*, 1695.

Gillespie, C. S. (2015). Fitting Heavy Tailed Distributions: The poweRlaw Package. *Jaournal of Statistical Software*, *64*.

Gillespie, M., Jassal, B., Stephan, R., Milacic, M., Rothfels, K., Senff-Ribeiro, A., . . . D'Eustachio, P. (2022). The reactome pathway knowledgebase 2022. *Nucleic Acids Res*, *50*(D1), D687-d692. <https://doi.org/10.1093/nar/gkab1028>

Huber, W., Carey, V. J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B. S., . . . Morgan, M. (2015). Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods*, *12*(2), 115-121. <https://doi.org/10.1038/nmeth.3252>

Kanehisa, M., & Goto, S. (2000). KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, *28*(1), 27-30. <https://doi.org/10.1093/nar/28.1.27>

Klemmer, P., Smit, A. B., & Li, K. W. (2009). Proteomics analysis of immuno-precipitated synaptic protein complexes. *J Proteomics*, *72*(1), 82-90. <https://doi.org/10.1016/j.jprot.2008.10.005>

Korotkevich, G., Sukhov, V., Budin, N., Shpak, B., Artyomov, M. N., & Sergushichev, A. (2021). Fast gene set enrichment analysis. *bioRxiv*, 060012. <https://doi.org/10.1101/060012>

Li, M., Wang, J., Chen, X., Wang, H., & Pan, Y. (2011). A local average connectivity-based method for identifying essential proteins from the network level. *Comput Biol Chem*, *35*(3), 143-150. <https://doi.org/10.1016/j.compbiolchem.2011.04.002>

Ma, H., & Zeng, A. P. (2003). Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics*, *19*(2), 270-277. <https://doi.org/10.1093/bioinformatics/19.2.270>

Maere, S., Heymans, K., & Kuiper, M. (2005). BiNGO: a Cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks. *Bioinformatics*, *21*(16), 3448-3449. <https://doi.org/10.1093/bioinformatics/bti551>

McLean, C., He, X., Simpson, I., & DJ, A. (2016). mproved Functional Enrichment Analysis of Biological Networks using Scalable Modularity Based Clustering. *Journal of Proteomics and Bioinformatics*, *9*(1), 9-18. <https://doi.org/https://doi.org/10.4172/jpb.1000383>

Menche, J., Sharma, A., Kitsak, M., Ghiassian, S. D., Vidal, M., Loscalzo, J., & Barabási, A. L. (2015). Disease networks. Uncovering disease-disease relationships through the incomplete interactome. *Science*, *347*(6224), 1257601. <https://doi.org/10.1126/science.1257601>

Nepusz, T., Petróczi, A., Négyessy, L., & Bazsó, F. (2008). Fuzzy communities and the concept of bridgeness in complex networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, *77*(1 Pt 2), 016107. <https://doi.org/10.1103/PhysRevE.77.016107>

Nepusz, T., Yu, H., & Paccanaro, A. (2012). Detecting overlapping protein complexes in protein-protein interaction networks. *Nat Methods*, *9*(5), 471-472. <https://doi.org/10.1038/nmeth.1938>

Newman, M. E. (2006). Modularity and community structure in networks. *Proc Natl Acad Sci U S A*, *103*(23), 8577-8582. <https://doi.org/10.1073/pnas.0601602103>

Pocklington, A. J., Armstrong, J. D., & Grant, S. G. (2006). Organization of brain complexity--synapse proteome form and function. *Brief Funct Genomic Proteomic*, *5*(1), 66-73. <https://doi.org/10.1093/bfgp/ell013>

Pons, P., & Latapy, M. (2006). Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms and Applications*, *10*(2), 191-218.

Reichardt, J., & Bornholdt, S. (2006). Statistical mechanics of community detection. *Phys Rev E Stat Nonlin Soft Matter Phys*, *74*(1 Pt 2), 016110. <https://doi.org/10.1103/PhysRevE.74.016110>

Rosvall, M., Axelsson, D., & Bergstrom, C. T. (2009). The map equation. *Eur. Phys. J. Special Topics*, *178*, 13-23.

Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proc Natl Acad Sci U S A*, *105*(4), 1118-1123. <https://doi.org/10.1073/pnas.0706851105>

Shannon, P., Markiel, A., Ozier, O., Baliga, N. S., Wang, J. T., Ramage, D., . . . Ideker, T. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, *13*(11), 2498-2504. <https://doi.org/10.1101/gr.1239303>

Sherman, B. T., Hao, M., Qiu, J., Jiao, X., Baseler, M. W., Lane, H. C., . . . Chang, W. (2022). DAVID: a web server for functional enrichment analysis and functional annotation of gene lists (2021 update). *Nucleic Acids Res*, *50*(W1), W216-221. <https://doi.org/10.1093/nar/gkac194>

Sorokina, O., Sorokin, A., & Armstrong, J. D. (2022). Synaptome.db: A Bioconductor package for synaptic proteomics data. *Bioinformatics Advances*. <https://doi.org/10.1093/bioadv/vbac086>

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., . . . Mesirov, J. P. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci U S A*, *102*(43), 15545-15550. <https://doi.org/10.1073/pnas.0506580102>

Teschendorff, A. E., Banerji, C. R., Severini, S., Kuehn, R., & Sollich, P. (2015). Increased signaling entropy in cancer requires the scale-free property of protein interaction networks. *Sci Rep*, *5*, 9646. <https://doi.org/10.1038/srep09646>

Traag, V. A., & Bruggeman, J. (2009). Community detection in networks with positive and negative links. *Phys Rev E Stat Nonlin Soft Matter Phys*, *80*(3 Pt 2), 036115. <https://doi.org/10.1103/PhysRevE.80.036115>

Vidal, M., Cusick, M. E., & Barabasi, A. L. (2011). Interactome networks and human disease. *Cell*, *144*(6), 986-998. <https://doi.org/10.1016/j.cell.2011.02.016>

von Mering, C., Huynen, M., Jaeggi, D., Schmidt, S., Bork, P., & Snel, B. (2003). STRING: a database of predicted functional associations between proteins. *Nucleic Acids Res*, *31*(1), 258-261. <https://doi.org/10.1093/nar/gkg034>

Wu, T., Hu, E., Xu, S., Chen, M., Guo, P., Dai, Z., . . . Yu, G. (2021). clusterProfiler 4.0: A universal enrichment tool for interpreting omics data. *Innovation (Camb)*, *2*(3), 100141. <https://doi.org/10.1016/j.xinn.2021.100141>

Wunderlich, Z., & Mirny, L. A. (2006). Using the topology of metabolic networks to predict viability of mutant strains. *Biophys J*, *91*(6), 2304-2311. <https://doi.org/10.1529/biophysj.105.080572>

Zhu, X., Gerstein, M., & Snyder, M. (2007). Getting connected: analysis and principles of biological networks. *Genes Dev*, *21*(9), 1010-1024. <https://doi.org/10.1101/gad.1528707>

**APPENDIX:** (Optional)

If a more detailed discussion of the technique’s theoretical foundations is warranted, please present it in the Appendix separate from the Commentary. Readers without a strong computational background should be able to perform and understand the protocol without relying on the Appendix.

**Estimating entropy effects in the network.**

We test for evidence of structure in each network by performing an entropy-based perturbation analysis in a synthetically weighted network. In this analysis the global entropy rate (SR) of the network is measured after each gene is individually perturbed, either by over-expression (SR\_UP) or under-expression (SR\_DOWN) and plotted against the degree of the perturbed gene. To assign expression values to each vertex in the perturbation analysis we followed (Teschendorff et al., 2015) (implemented in *BioNAR* as a default procedure). Vertex weights are set to initial values of 2 then sequentially perturbed to values of 14 when modelling activity and set to initial values of 16 with perturbed values of – 14 when modelling inactively.

**Comparing of cluster enrichment for identifying the optimal algorithm.**

The hypergeometric distribution was used to calculate the significance of enrichment of each cluster for each annotation (equation (1)):

-(1)

Where is the total number of genes in the network; – the number of genes in the community; – the total number of functional annotated genes in the network, and – the number of functional annotated genes per community.

Enrichment was calculated using equation (2), which gives us the one-sided exact Fisher test, while depletion was calculated using alternative test given in equation (3).

*-(2)*

*-(3)*

P-values calculated where corrected using the Benjamini and Yekutieli (B-Y) (Benjamini & Yekutieli, 2001) procedure and tested against the more stringent Bonferroni correction at the 0.05 (\*), 0.01 (\*\*) and 0.001 (\*\*\*) significance levels.

Given the enrichment results, we can calculate the log of the Odds Ratio (OR) as:

*-(4)*

and it’s upper and lower 95% Confidence Interval:

-(5)

, where terms are the same as those stated above.

Fold-enrichment (Fe) value, measured at 100 intervals taken from 0 to the maximum Fe value (the maximum found from all algorithms studied) is estimated as follows:

*- (6)*

where as in above is the total number of genes in the network; – the number of genes in the community; – the total number of functional annotated genes in the network, and – the number of functional annotated genes per community.

For each algorithm we measure the difference in fraction of enriched communities between log2(Fe) ‘cut off’ values of 0.5 (this translates to interval point 7 out of 100) and 4.8 (which is the interval point 54 out of 100).

So, finally, we test how well each distribution is fitted to a generalised sigmoid function:

-(7)

Where ‘a’ gives the sigmoid’s lower asymptote, ‘b’ the sigmoid’s maximum value, ‘c’ the rate of change of the sigmoid and ‘d’ the x value of the sigmoid’s midpoint. An ideal sigmoid for our case would occur when: a = 0, b = 1, c = -2, and d = 3. Of the four parameters, it is the rate of change of the sigmoid curve which is of interest to us, as lower and upper asymptote are defined by nature of fraction value as zero and one. To test how well each distribution (in Figure 7) fitted to a set of five ‘idealised’ sigmoid function with the other parameters fixed, and ‘c’ set too [-10, -5, -2, -1, -0.5], we use the two-sample Kolmogorov-Smirnov (KS) test to the goodness of fit of each distribution to our set of five idealised sigmoid curves.

**FIGURE LEGENDS:**

*Figure 1. The pipeline for a typical network analysis study is shown with dependencies between the respective Protocols (described in detail in the main text).*

*Figure 2. A. The results of fit of network degree distribution (P(k)), versus its degree (k) for the postsynaptic network. The red line shows the best fit power-law distribution with estimates of the scaling parameter α (2.63) and kmin (13), with a goodness-of-fit of 0.32 after 1000 iterations of the bootstrapping. B. Distribution of alpha values for 5000 random G(n,p) Erdos-Renyi graphs with actual α value (2.63) shown as a vertical line.*

*Figure 3. A. Shown is the global entropy (SR) for postsynaptic network under this study plotted against the log of the protein’s degree. The horizontal dashed line on the plot corresponds to the entropy rate of the unperturbed graph (SRo = 0.698)). We can see that entropy for both up- and down-perturbed low-degree vertices stays close to this value, while the hubs (high-degree vertices) deviate from it significantly, which makes a bi-modal response between gene over-expression and degree and opposing bi-phasic response relative to over/under-expression between global entropy rate and degree, observed only in networks with scale-free or approximate scale-free topology. B Shown the (SR) values for a randomised G(n,p) Erdos-Renyi network, perturbed in a similar way.*

*Figure 4. A. Results of “Louvain” clustering visualised with BioNAR B. Results of reclustering for “Louvain” algorithms visualised with BioNAR. It can be seen that several of the larger clusters from figure 4 A are now split to smaller clusters.*

*Figure 5. A. A schematic representation of possible scenarios of Fe distribution. Blue curve corresponds to the case when most of cluster-term pairs have small Fe values because either large clusters cover large part of the network or a tiny fraction of the vertices are annotated. The red curve corresponds to the algorithm which produce a lot of small clusters, so majority of cluster-term pairs have very large Fe value. To avoid both extreme scenarios, we are trying to find clustering algorithm with Fe distribution close to the black curve. B. Plot (type p3) of fraction of enriched communities against the Fold -enrichment for each of the nine algorithm clustering results obtained for the postsynaptic network, each colour corresponds to the specific algorithm.*

*Figure 6. Results of fit to five generalized sigmoid functions. Each panel A-I looks at a different clustering method. In each panel four grey sigmoid curves correspond to four reference sigmoid curves with rates parameter c, (see Appendix) equals to -10, -5, -1 and -0.5, while the solid black sigmoid defines the ideal/desired behavior with rate equal to -2. The open circles and red line correspond to the actual fit, while the dashed blue line indicates the 95% confidence interval. Panels C and E show that fc and Louvain algorithms give the closes fit to the ideal, with KS values 1E-01 and 9e-02.*

*Figure 7. Results of fit to five generalized sigmoid functions with added noise. It can be seen that the confidence interval is wider for wt (B) than on the figure 7; and unbounded for infomap (D), sgG5 (H) and spectral (I), which indicates that fitting for results of these algorithms is unstable. See also legend to Figure 6.*

*Figure 8. Plot of bridgeness (B) against the semilocal centrality (SL) for Louvain clustering results. Highlighted are the proteins with known function “protein cluster”. It can be seen that from highlighted proteins only a few belong to Region 1 and have the global impact on the network (PICK1, CASK, PASCIN1, MPP3). Rest of them belong to Region 3, and largely influence their own communities.*

**TABLES:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alg | mod | C | Cn1 | Cn100 | mu | Min. C | 1st Qu. C | Median C | Mean C | 3rd Qu. C | Max. C |
| lec | 0.339 | 16 | 3 | 8 | 0.526 | 1 | 5.25 | 102.5 | 143.5625 | 215.75 | 502 |
| wt | 0.2846 | 490 | 359 | 3 | 0.447 | 1 | 1 | 1 | 4.688 | 2 | 773 |
| fc | 0.394 | 20 | 0 | 6 | 0.390 | 2 | 5.25 | 22.5 | 114.85 | 105.25 | 565 |
| infomap | 0.379 | 174 | 0 | 2 | 0.597 | 2 | 6 | 8 | 13.201 | 14 | 125 |
| louvain | 0.425 | 15 | 0 | 11 | 0.479 | 22 | 101 | 121 | 153.133 | 216 | 311 |
| sgG1 | 0.448 | 24 | 0 | 9 | 0.440 | 2 | 5.75 | 11.5 | 95.708 | 231.5 | 316 |
| sgG2 | 0.433 | 42 | 0 | 8 | 0.523 | 3 | 10.75 | 56 | 54.690 | 71.5 | 158 |
| sgG5 | 0.368 | 102 | 0 | 0 | 0.617 | 3 | 14 | 22 | 22.520 | 27 | 78 |
| spectral | 0.370 | 63 | 9 | 4 | 0.595 | 1 | 3 | 28 | 36.460 | 56 | 184 |

*Table 1. Summary for the clustering algorithms used on the protocol. Note that your own summary may look slightly different as some of the algorithms use randomization, so each time will give slightly (but not significantly) different results.*

|  |  |  |  |
| --- | --- | --- | --- |
| ALG | Fn | C | Mu |
| lec | GO:0035615 | 1 | 7 |
| lec | GO:0005484 | 2 | 13 |
| lec | GO:0000149 | 2 | 18 |
| lec | GO:0003730 | 3 | 10 |
| lec | GO:1990841 | 3 | 5 |
| lec | GO:0003678 | 3 | 4 |

*Table 2. Presented first six rows of ‘CAN’ table, were ALG is the algorithm’s name, Fn – enriched GO term ID, C- number of enriched cluster, Mu – number of genes associated with a term.*

| Alg | Fraction of Communities Enriched  log2(FE)>0.5\_log2(FE)<4.8 | Fraction of Communities Enriched  log2(FE)>4.8\_log2(FE)<8.0 |
| --- | --- | --- |
| louvain | 1 | 0 |
| sgG2 | 0.973 | 0.027 |
| fc | 0.96 | 0.038 |
| spectral | 0.95 | 0.049 |
| sgG5 | 0.88 | 0.122 |
| sgG1 | 0.86 | 0.139 |
| lec | 0.81 | 0.190 |
| infomap | 0.72 | 0.279 |
| wt | 0.617857142857143 | 0.382142857142857 |

*Table 3. Shown the result of ranking of the algorithms based on proportion of enriched communities. The table is obtained from View(plots$ranktable) command*