# coexnet: An R package to build CO-EXpression NETworks from Microarray Data

*Juan David Henao*

*2018-04-30*

## Contents

#Abstract

The network analysis of biological data has increased in recent years, due to the capacity of this approach to analyze and represent complex information in a simple way, information that nowadays is growing and that covers different levels of biological resolution (protein-protein interaction, signaling interaction networks, gene regulatory network, among others). Currently, one of the most used and informative representations of biological data are co-expression networks. In this approach a network is created based on data obtained from experimental expression measures, taking into account the existence of particular patterns or relations between expression profiles among different genes, proteins or RNA fragments involved in a specific phenotype. Hereby, we introduce *coexnet*, a new R package for the creation of undirected co-expression networks from microarray data, obtained from GEO Datasets database. This package contains all the necessary functions that pipe the analysis process from the download of microarray datasets, going through the normalization and filtering of genes and experiments, to the creation of the co-expression network using state of the art correlation measures and statistical analysis. The package *coexnet* includes some new functions that allow connecting and using functions from other CRAN and Bioconductor packages for the analysis of genomic data.

#Workflow

##getInfo

All microarray raw data associated to the same study are stored in a *CEL* file, this type of file contains the *GSM* files, each one corresponding to one sample of a gene expression study. The user can also obtain each *GSM* file individually, but it is preferable to obtain all the samples at once as they come in the *CEL* file, in order to avoid the work of joining each *GSM*. Additionally having all *GSM* files at once will allow to perform a simultaneous data analysis (in a future normalization process, for example). Furthermore, all the microarray chips are documented in the *GEO Datasets* database. Each of them is identified with the letters *GPL* adding a unique number. The information in the *GPL* file is then linked to the information of each probeset in the microarray chip, including the gene, function, type and other information. This information is very useful to enrich the analysis of expression data.

This function will create, in your current path, a folder with the *GSE* (unique number) name where the GSM downloaded files will be stored. It also will create the *GPL* (unique number) *.soft* file that contains the microarray chip information.

```
library(coexnet)
```

```
## Setting options('download.file.method.GEOquery'='auto')
```

```
## Setting options('GEOquery.inmemory.gpl'=FALSE)
```

```
# Downloading the microarray raw data from GSE8216 study
# The accession number of the microarray chip related with this study is GPL2025

getInfo(GSE = "GSE8216", GPL = "GPL2025",directory = ".")
```

```
# Shows the actual path file with the folder, its GSE number and the .soft file

dir()
```

**Take into account**

In some cases the information in the *GPL* file is partial, so take this into account if you are willing to run future analysis over the same data, so it is recomended not to store the files in a temporal folder, given that in many cases you will need the raw data to re-process the expression values using, for example, different methods.

##getAffy

The *AffyBatch object* is used to process and analyse microarray expression data. The *AffyBatch object* stores information about the date in which each one of the samples were scanned, as well as the information related with the phenotype, the raw expression values to each probe in the microarray chip and the kind of library to read the expression data among others.

You can use the *AffyBatch object* in many different packages mainly in the affy package, additionally you can modify the *AffyBatch object* if you consider it necessary.

This function searches in your current or designated path file the folder with the *GSE* accession number and reads the *filelist.txt* file that contains the name of each *GSM* sample, in order to recognize them and join them in an unified *AffyBatch object*.

```
# Reading some GSM samples from GSE4773 study, the folder with the
# GSM files are called GSE1234.

affy <- getAffy(GSE = "GSE1234",directory = system.file("extdata",package = "coexnet"))
affy
```

**Take into account**

In some cases the *AffyBatch object* doesn't have all the necessary information, and a warning message appears when you visualize the variable containing the *AffyBatch object*. Nevertheless, you can manually edit the AffyBatch afterwards to complete all the requiered information. If you try to process the AffyBatch in some of the packages that use this kind object with missing information, you will get an error message.

```
# The variable affy doesn't have the CDF (Chip Definition File) information.
# You can include this information modifying the AffyBatch object afterwards.

affy@cdfName <- "HG-U133_Plus_2"
```

##geneSymbol

In most cases, the idea behind creating a co-expression network is to visualize the relationships among diferent genes, proteins, specific DNA or RNA fragments or any other kind of molecular entities, that are identified by a specific ID. For this reason, it is very useful to keep the information of the corresponding ID corresponding to each one of the probesets in the microarray. This kind of information will be used when you need to switch from a matrix of probeset-samples to one of genes (or another ID)-samples before the construction of the co-expression network.

The *.soft* file, downloaded from *GEO Datasets* database using the *GPL* identifier has the information to create a table with the relationship between a probeset and one "molecular ID", in this table one ID can be related to two or more probesets, the process of mapping the expression value to a gene ID is called *summarization* (see below).

This function searches, in the current or the designated path file, the *.soft* file and creates a data frame, where the first column contains each of the probeset names and the second one contains the corresponding

ID (gene symbol, protein name or another identifier). This matching information is needed to summarize the gene expression data. This step is only needed if microarray chip data is used.

```
# Create the table with the match between probesets and IDs.

gene_table <- geneSymbol(GPL = "GPL2025",directory = system.file("extdata",package = "coexnet"))

head(gene_table)
```

```
##             probe ID
## 1  AFFX-BioB-3_at
## 2  AFFX-BioB-5_at
## 3  AFFX-BioB-M_at
## 4  AFFX-BioC-3_at
## 5  AFFX-BioC-5_at
## 6 AFFX-BioDn-3_at
```

**Take into account**

In some cases, the *.soft* file dosen't have all the IDs that are related to each one of the microarray probesets, you can ignore this probeset under the assumption that another probeset could have the same ID and this second one has the respective annotation. On the other hand, one ID can have more than two names, this function creates an ID with all the related names separated by "-". This happens when a sequence of nucleotides of a specific probeset matches the sequence of two or more genes. For example, in the microarray chip *GPL570*, there exists a probeset whose sequence match with the genes *CPZ* and *GPR78*, so the final ID will be *CPZ-GPR78*.

```
# The created table have NA and empty IDs information.
# We can delete this unuseful information.

# Deletion of IDs with NA information

gene_na <- na.omit(gene_table)

# Deletion of empty IDs

final_table <- gene_na[gene_na$ID != "",]

head(final_table)
```

```
##                   probe           ID
## 118    Os.10.1.S1_s_at Os03g0669200
## 119   Os.10003.1.S1_at Os01g0235100
## 120 Os.10007.2.S1_a_at Os06g0256200
## 121   Os.10017.1.S1_at Os01g0556400
## 122 Os.10038.1.S1_s_at Os07g0153000
## 123   Os.10055.1.S1_at Os01g0231500
```

##exprMat

The raw expression data in a microarray experiment must be processed, in order to transform the original data into an ideal way as to be analyzed and thus obtain high confidence results. The first step is the normalization of the data, that consists in a background correction of the raw data followed by a normalization. The second one is the mapping of probeset to gene or any other ID to represent the molecular entity to analyze. Additionally, there exists the posibility to make a second kind of background correction based on the batch of samples scaned in a separate way due to the large number of samples and the limitation in the size of the particular microarray chip used, this correction is known as *Batch Effect Correction*.

Different methods exist in order to normalize raw expression data from microrray experiments, each one of this methodologies consider a particular way to generate a background correction, the process of normalization and the mapping from probes to probesets. The difference between these methods consists on the underliying mathematic assumptions used and the range of the normalized results, in some cases the expression data have a wider range than anothers. In the same way, the process to transform the probeset-samples matrix in a gene(or another ID)-sample matrix considers diferent methodologies, including the obtantion of the average of the expression values of each of the probesets corresponding to the same gene or protein, the selection of the maximum or minimum value, among others.

This function offers the possibility to choose among two different methods to normalize the raw expression values, including the process of background correction and the mapping from probes to probeset. The first one is *rma* (Robust Multi-Array Average), this method performs a background correction and normalization in separate calculations (Irizarry, *et al.* 2003). The second one is *vsn* (Variance Stabilizing Normalization), this method, contrary to *rma*, generates the background correction and the normalization in the same equation (Huber, *et al.* 2002). This function also offers the option to perform *Batch Effect Correction* identifying the samples belonging to the same batch using the scan date into the *AffyBatch object*.

Additionally, this method considers two ways to calculate the values in the process to map from probesets to gene/ID. The first one is selecting the representative probeset to each of the genes, proteins or another kind of ID. To do that, it calculates the average of each of the probesets associated with the same gene/ID, and the probeset with the highest value in the average is selected. The second one is to obtain the median of each of the samples to the probesets associated with the same gene/ID, getting only one expression value for sample as the transformation of the normalized data.

```
# Loading gata

if (require(affydata)) {
  data(Dilution)
}

# Loading table with probeset and gene/ID information

data("info")

# Calculating the expression matrix with rma

rma <- exprMat(affy = Dilution,genes = info,NormalizeMethod = "rma",
SummaryMethod = "median",BatchCorrect = FALSE)
head(rma)
```

**Take into account**

Consider that *rma* is a method in which the amplitude of the results are narrower than in *vsn*, take into account this situation, in order to select the method to normalize the raw expresion values. In some cases the *vsn* method takes into account every probe in the normalize process, so it could take time to process. In some cases if you made a *Batch Effect Correction*, you will want to compare the results normalizing the raw expression data without the *Batch Effect Correction*.

##cofVar

In some cases, the co-expression network is built from two or more microarrays studies, in this sense, it is necessary to define wich one of these studies accounts for more source of background noise and will probably have a negative impact on the results. One way to determine the most harmful studies is from a variation analysis. By this approach the study holding more variation among the normalized expression values can be considered as the source of future background noise and then it is necessary not to consider the use of this studies in the construction of the co-expression network.

The variation amongst the normalized expression values can be determined by the *coefficient of variation*

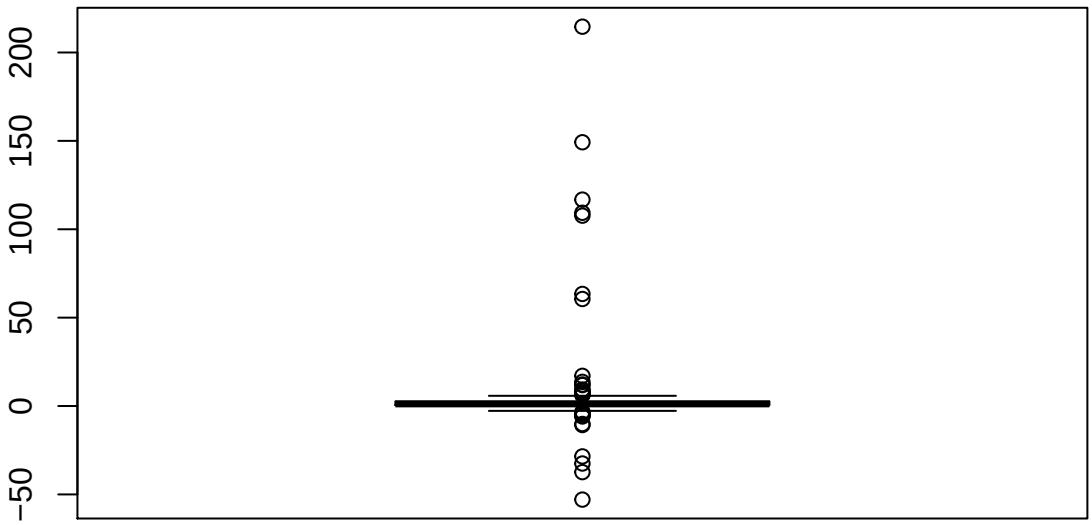
of each one of the genes in each one of the studies and thus generate a boxplot from these results. So, in a graphical way it is possible to define the studies that will generate background noise by visual inspection of the atypical information. On the other hand, it is also possible to define the number of atypical data and determine the more variant studies using the boxplot and the number of atipic data defining a threshold value, for example you can determine the studies with more of 10% of atypical data wont be used in the construction of the co-expression network.

This function takes the normalized ID-sample matrix and calculates the median and the *coefficient of variation* for each one of the IDs, this process must be applied in a study-by-study basis. Additionally, this function allows to calculate the mean and the *coefficient of variation* to cases and control samples separately, using a vector of 0s and 1s to identify the cases and control samples. This vector can be defined in the description of each sample in the GEO Datasets database and it is necessary in the process of identification of the genes (or another ID as proteins) that are differentially expressed (see bellow).

```
# Simulated expression data

n <- 200
m <- 20

# The vector with treatment samples and control samples

t <- c(rep(0,10),rep(1,10))

# Calculating the expression values normalized

mat <- as.matrix(rexp(n, rate = 1))
norm <- t(apply(mat, 1, function(nm) rnorm(m, mean=nm, sd=1)))

# Calculating the coefficient of variation to case samples

case <- cofVar(expData = norm,complete = FALSE,treatment = t,type = "case")
head(case)
```

```
##             1      1.1         1.2         1.3       1.4        1.5
## 1  2.56757073 1.729936  0.81478990  0.45555730 1.7448195  0.6960645
## 2  0.05879636 1.969489 -0.50586764  0.37888245 0.5476221 -0.4165011
## 3  0.77115332 1.235371  0.02112802 -0.79382873 0.2911337 -0.2541970
## 4  5.17411457 5.648968  5.89905312  5.41057260 6.3828924  3.0967189
## 5  1.15392557 1.888980  2.21155543  2.00418970 1.4874575  1.3339982
## 6 -0.06379032 1.818576 -0.48442168 -0.04802429 1.3244801  1.3132551
##           1.6         1.7       1.8        1.9      mean        cv
## 1 1.268736814  1.52082145 -0.761329  3.1645949 1.3201562 0.8402507
## 2 0.005808028  0.72782642  1.776354 -0.2030624 0.4339348 1.9732416
## 3 1.263365142 -0.91724921  0.326770 -0.8091219 0.1134524 7.1793950
## 4 5.516506556  6.26781107  5.575256  6.0089203 5.4980813 0.1682918
## 5 0.200515224  2.40150873  1.452931  0.7195728 1.4854634 0.4582233
## 6 1.154698429  0.03937074  1.925166 -0.6988571 0.6280452 1.5575674
```

```
# Creating the boxplot to coefficient of variation results

boxplot(case$cv)
```




```
# Extracting the number of atipic data

length(boxplot.stats(case$cv)$out)
```

```
## [1] 38
```

**Take into account**

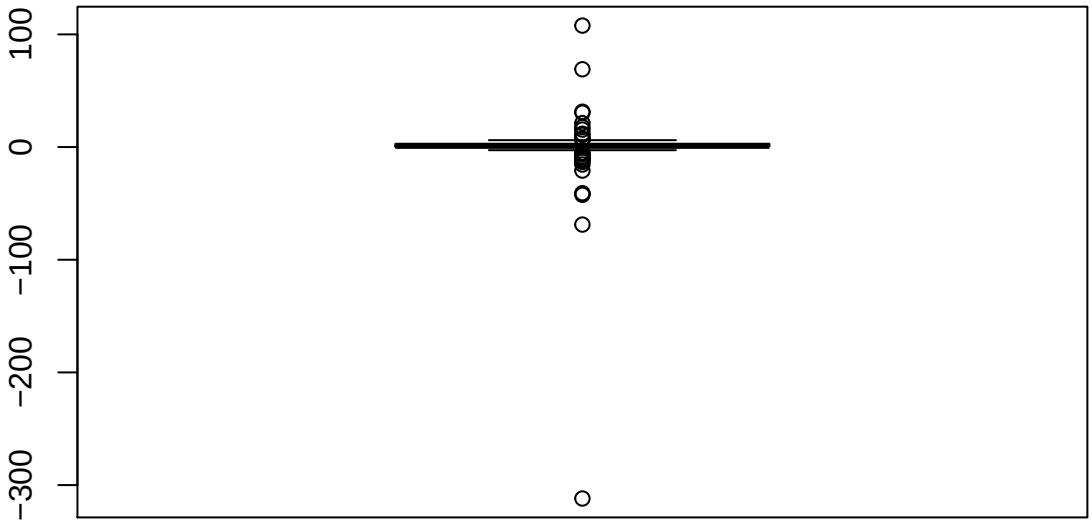
The decision of discarding a microarray study from our analysis, based on the result of the coefficient of variation analysis, depends on the data and the criteria of the researcher to filter the studies (the selection of a threshold value), there is no a Gold Standard to discard a study, so it is advisable to calculate the *coefficient of variation* of all samples at the same time to compare and to determine which one shows more variation.

```
# Calculating the coefficient of variation to whole matrix

complete <- cofVar(norm)
head(complete)
```

```
##             V1          V2         V3        V4         V5        V6
## 1  1.203314822  1.78618456  2.1269521 0.7235420 1.29538734  1.144328
## 2  1.380979515 -0.03400279  0.7933288 1.1512881 1.06946955  1.478422
## 3  0.170476373 -1.34198137 -0.5691631 0.2052357 0.07398204 -0.758203
## 4  6.865851804  4.76418204  2.7004399 4.0464555 4.84335055  4.926209
## 5  4.172486109  1.94125185  1.6367046 1.5312019 2.24818755  3.155441
## 6 -0.002848971  0.94452716 -0.5439651 1.0763211 0.90076820 -1.313343
##           V7         V8         V9        V10         V11      V12
## 1  1.4480887  1.7733880  0.1200740 -0.8188624  2.56757073 1.729936
## 2  0.2156304 -0.1219805 -0.4077094  0.5578137  0.05879636 1.969489
## 3 -0.6881020  1.4074458 -0.1452236  1.0258533  0.77115332 1.235371
## 4  6.4911231  3.3716925  4.6608200  2.8102709  5.17411457 5.648968
## 5  1.3070244  0.6455713  2.6373563  1.5652344  1.15392557 1.888980
## 6  1.2788222  2.2934467  1.2893841  2.2695882 -0.06379032 1.818576
##           V13         V14       V15        V16         V17         V18
## 1  0.81478990  0.45555730 1.7448195  0.6960645 1.268736814  1.52082145
## 2 -0.50586764  0.37888245 0.5476221 -0.4165011 0.005808028  0.72782642
## 3  0.02112802 -0.79382873 0.2911337 -0.2541970 1.263365142 -0.91724921
## 4  5.89905312  5.41057260 6.3828924  3.0967189 5.516506556  6.26781107
## 5  2.21155543  2.00418970 1.4874575  1.3339982 0.200515224  2.40150873
## 6 -0.48442168 -0.04802429 1.3244801  1.3132551 1.154698429  0.03937074
```

```
##         V19        V20       mean         cv
## 1 -0.761329  3.1645949 1.20019795  0.8175466
## 2  1.776354 -0.2030624 0.52112935  1.4454926
## 3  0.326770 -0.8091219 0.02574223 31.3707358
## 4  5.575256  6.0089203 5.02306041  0.2490075
## 5  1.452931  0.7195728 1.78475466  0.5023374
## 6  1.925166 -0.6988571 0.72365765  1.4414820
```

```
# Creating the boxplot to coefficient of variation results

boxplot(complete$cv)
```




```
# Extracting the number of atipic data

length(boxplot.stats(complete$cv)$out)
```

```
## [1] 34
```

##difExprs

When expression data of gene, proteins, or another kind of ID are used to build a co-expression network, in most cases it is convenient to asses differences in the expression value of each of them. A differential expression analysis will allow us to identify the genes/IDs that are over-expressed or under-expressed with respect the whole data set and thus to establish the possible molecular components that are associated directly or indirectly with the onset and/or development of a certain phenotype and use them to create the co-expression network.

There are several methodologies to identify differentially expressed genes/IDs, some methods are more predictive than others, so depending of the method, it is possible obtain genes/IDs that clearly differentiate from others in their expression values or it is also possible to obtain genes/IDs whose expression value are slightly different from others, but that given the criteria of the method used, they are considered as differentialy expressed. In both cases it is possible to have genes/IDs that are identified as differentialy expressed by error and it is necessary to consider a measure of this error. One common measure of error is the *False Discover Rate* or *FDR*, this metric describes the probability of one gene/ID being selected as differentially expressed by error.

This function considers two ways to calculate the differentially expressed genes/IDs. It is possible to carry out a predictive methodology that obtains the majority of genes or IDs considered as differentially expressed, in this case the *sam* method is used. This method basically uses a difference of means to calculate the genes/IDs that are over-expressed or under-expressed using a permutation process to test the results and prove that the

genes/IDs selected where not randomly selecte. Through these permutations the *FDR* value is calculated (Tusher, *et al.* 2001). This function can also use the *acde* method to calculate and obtain the genes/IDs that are differentially expressed in a less predictive way. This method consists, essentially, on the application of the main components to characterize the genes differentially expressed by calculating the *FDR* using multiple hypothesis tests according to Benjamini and Hochberg (1995) (Acosta & López-Kleine, 2015).

```
# Creating a matrix with 200 genes and 20 samples

n <- 200
m <- 20

# The vector with treatment samples and control samples

t <- c(rep(0,10),rep(1,10))

# Calculating the expression values normalized

mat <- as.matrix(rexp(n, rate = 1))
norm <- t(apply(mat, 1, function(nm) rnorm(m, mean=nm, sd=1)))

# Running the function using the two approaches

sam <- difExprs(expData = norm,treatment = t,fdr = 0.2,DifferentialMethod = "sam")
head(sam)
```

**Take into account**

This function identifies the genes/IDs differentially expressed taking into account the expected *FDR*, so independently of the method used (*sam* or *acde*), the number of genes/IDs identified as differentially expressed will be guided by the *FDR* expected by the user, thus increasing the predictive power in the final results.

##findThreshold

Once you have the final expression matrix, it is used as basis to obtain the co-expression network. There are two methods widely used to obtain it, both of them are related to the definition of correlation value between all the genes/IDs creating a square matrix. On one hand you can calculate the *Pearson Correlation Coefficient*, this method calculates the correlation between each genes/IDs expression values, as result, the square matrix will have values between zero and one, given that for the future construction of the co-expression network it is necessary to use the absolute value of the results. On the other hand, the *Mutual Information* approach is based on the entropy of the data and in a simmilar manner a square matrix is created, but, in this case, it is necessary to perform an additional transformation of the results in order to obtain a square matrix with values between zero and one.

Obtaining a square matrix with a range of values between zero and one is necessary in order to perform a future transformation of this correlation matrix into an adjacency matrix (see bellow). Additionally, it is also necessary to work on this range of values because a threshold value must be defined in order to establish the final relationships between the genes/IDs. In order to achieve this, a value between zero and one is defined and the values of correlation below a threshold will indicate the no-existance of a real correlation among them, allowing us to finally obtain the relationships between the genes/IDs expressed as co-expression network.

This function computes a threshold value using a novel method based on two *Biological Systems* approaches. First, each possible threshold value, from 0.01 to 0.99 with an increase of 0.01 is examined. Each of this values is then analyzed using the Pearson Correlation Coefficient. Thus the *Clustering Coefficient* is calculated for the created network using the current threshold value under test, this is performed for each value. Thereafter, a new artificial *Clustering Coefficient* is calculated to simulate a random network, created using the same threshold value. Then, the difference between the two *Clustering Coefficient* values is calculated, and the result that meets the criteria of Elo, *et al* (2008), is used for next analysis. Finally, the remaining threshold

values are analyzed using the *Degree Distribution* under normal distribution, using the *Kolmogorov-Smirnov* test, expecting that the resulting p-value rejects the distribution, as a result of the assumption that the biological networks do not have a normal distribution when the *Degree Distribution* is analyzed. Finally, the minimum threshold value that satisfies this two criteria will be selected as the final threshold value for the construction of the co-expression network (Leal, *et al.* 2014).

```
# Loading data

pathfile <- system.file("extdata","expression_example.txt",package = "coexnet")
data <- read.table(pathfile,stringsAsFactors = FALSE)

# Finding threshold value

cor_pearson <- findThreshold(expData = data,method = "correlation")
cor_pearson
```

**Take into account**

*Mutual Information* is used, in most cases, when you need to analyze a huge amount of expression data, for example, when the study was designed to use a lot of samples and you must process all information simultaneously. In most cases, the threshold value is selected by the researcher without any biological assumption. Here, we present this novel methodology to select a threshold value under a network biology assumption.

##createNet

The last step in the construction of a co-expression network is the creation of a data structure that stores the information necessary to create a network graph. Once you had defined a threshold value, the last step is to transform the expression matrix in a adjacency matrix using the correlation or the mutual information method to obtain the values of relationship between the diferent genes or proteins (or another kind of ID).

The process to go from an expression matrix to a network graph consists on two steps. The first one is the building of a correlation matrix, to do that is necessary to apply one of two methods to calculate the relationship among the genes/ID (Pearson correlation or mutual information, see above) (López-Kleine, *et al.* 2013). The second one is the transformation from a expression matrix to adjacency matrix, for which it is necessary to apply a threshold value. Every correlation value inside the matrix that is less than the threshold will be replaced by zero, while all remaining values will be replaced by one. Additionally, the diagonal in the square matrix will be replaced with zero to avoid loops in the co-expression network.

Finally, based on the adjacency matrix, a list is created, where connected gene/IDs are separated by a space. For example if *gene A* and *gene B*, have a value of one in the adjacency matrix, then in the final edge list they will be shown as:

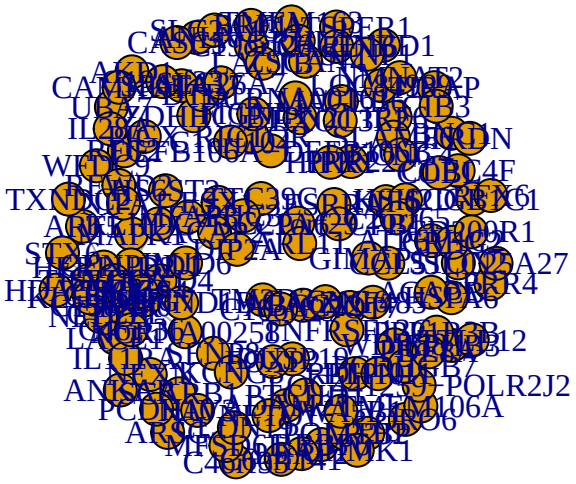
*gene A – gene B*

This way every genes/IDs are connected in the final co-expression network.

This function takes the expression matrix and creates the correlation matrix using *Pearson Correlation Coefficient* or *Mutual Informtion*. After that, it creates the adjacency matrix using the threshold value given by the user and finally creates the network from the adjacency matrix as an igraph object to be analyzed using the igraph R package or any other tool that recognizes this type of object.

```
# Loading data

pathfile <- system.file("extdata","expression_example.txt",package = "coexnet")
data <- read.table(pathfile,stringsAsFactors = FALSE)

# Building the network
```

```
cor_pearson <- createNet(expData = data,threshold = 0.7,method = "correlation")
plot(cor_pearson)
```



**Take into account**

In the process of construction of the adjacency matrix, sometimes a gene/ID can not be related to another one and in the process passing from a matrix to an edge list, this gene/ID will be deleted. Additionally, the network in the igraph object can be exported as an edge list using the igraph package to be visualized in another tool such as *Cytoscape* or *Gephi*.

##ppiNet

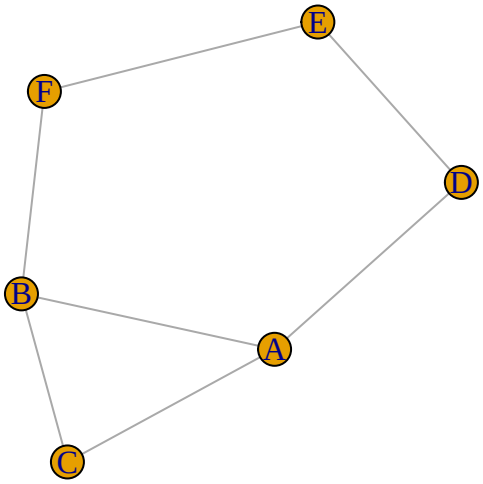
In many cases, it is necessary to consider additional information to have a more robust analysis. Protein-protein interaction (PPI) is the most used additional information to support the relationships detected by the co-expression network. One way to relate the protein-protein interaction information with the co-expression data is by building a PPI prediction network. To create this kind of networks, it is useful to start with the list of genes obtained from the differential expression analysis results (see above). From this list of genes, the network will be created based on different pieces of evidence like experimental data and the distances of the genes inside in the genome, among others.

This function, creates a PPI network from a vector of genes IDs or another type of commonly used identifier to be recognized by the STRING database. STRING contains information on protein-protein interaction from many species and supports the relationships between proteins with different types of pieces of evidence like experimental data, co-occurrence of the proteins among related species, text mining, among others. Inside STRING database, each relationship between proteins is supported using the information in KEGG database, a widely used and curated database of information on metabolic pathways. Finally, this function returns the PPI network as an igraph object to be analyzed in the same way as a co-expression network.

```
# Creating a vector with identifiers

ID <- c("FN1","HAMP","ILK","MIF","NME1","PROCR","RAC1","RBBP7",
"TMEM176A","TUBG1","UBC","VKORC1")

# Creating the PPI network

ppi <- ppiNet(molecularIDs = ID,evidence = c("neighborhood","coexpression","experiments"))
plot(ppi)
```

```
# Creating a PPI network from external data

ppi <- ppiNet(file = system.file("extdata","ppi.txt",package = "coexnet"))
```

```
plot(ppi)
```




**Take into account**

It is key that all IDs used are those identifiers used in main bioinformatics databases (such as UNIPROT, GeneBank, ENA or KEGG), in order to be efficiently mapped by STRING. Additionally, this database uses its own IDs to recognize the species of interest, in this case, by default, the function has the ID "9606" which corresponds to the human species, for additional information about species IDs, visit the database website (http://string-db.org/).

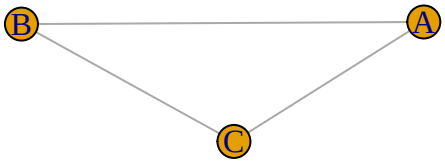
##CCP

The Common Connection Pattern (CCP), is a new methodological proposal to identify molecular components linked together and common in several biological networks. The principal assumption behind Common Connection Pattern is that the networks to be compared must have the same molecular information from, i.e., must compare one layer of molecular abstraction at the same time, for example, co-expression layer, protein-protein layer, the gene regulation layer, among others.

In general, the comparison of biological networks is made to determine common elements or biomarkers among several related phenotypes. In this case, the Common Connection Patterns aims to identify common molecular elements between these phenotypes that are associated also with each other in a specific way. For this, the intersection between biological networks is calculated whose result can have two kinds of elements. On the one hand, the shared nodes without any connection with other nodes in the intersection network. On the other hand, nodes connected to one or more nodes in the intersection network. Each connected component in the intersection will be considered as a Common Connection Pattern.

This function obtains the Common Connection Patterns making two steps. In the first one, it generates the intersection of the networks given in the input as graph objects. In the second one, it identifies the solitary nodes in the intersection network and then they are removed to leave the connected components only. Finally, the function returns all the Common Connection Patterns in one graph object.

```
# Loading data

data("net1")
data("net2")

# Obtaining Common Connection Patterns

ccp <- CCP(net1,net2)
plot(ccp)
```


**Take into account**

Although the idea is to compare biological networks from the same nature and created to study related phenotypes, the function can have results, when it compares any kind network. Additionally, is possible not find Common Connection Patterns when two or more networks are compared, but, is possible to find shared solitary nodes between them.

##sharedComponents

Solitary nodes obtained when two or more biological networks are compared also have relevant information associated because these nodes are molecular components implicated in more than two phenotypes. This is especially true if the filter used to include genes was that of the differential expression. The reason behind obtaining solitary nodes may be that there is not enough information to relate them with another component in the network or that the node has a relationship in another layer of the network (protein-protein interaction layer, genetic regulation layer, SNP layer, etc).

Thus, obtaining these solitary nodes can enrich the comparisons made between generated networks from the same type of molecular information and also seeks to find common elements among related phenotypes such as the Common Connection Patterns (see above). Additionally, it is possible to find none Common Connection Patterns between networks but only common elements.

This function obtains the shared components between two or more biological networks in two steps. During the first one, it gets the intersection from the networks in the input as igraph objects. During the second one, it extracts the solitary nodes present in the resulting intersected network . Finally, the result is a vector with the names of each solitary node. On the other hand, the nodes connected in the intersection network won't be taken into account because they are considered as being part of some Common Connection Pattern (see above).

```
# Loading data

data("net1")
data("net2")

# Obtain shared components

share <- sharedComponents(net1,net2)
share
```

```
## [1] "P" "X" "O" "Y"
```

**Take into account**

The assumption behind the solitary nodes in the intersection network as molecular elements associated

with related phenotypes is that all the networks were created using the same molecular information (gene co-expression, protein-protein interaction, TF site or any other) although this function is also able to find shared components between any biological network types.

#References


1. Acosta, J. P., & López-Kleine, L. (2015). Identification of Differentially Expressed Genes with Artificial Components–the acde Package.
2. Elo, L. L., Järvenpää, H., Orešič, M., Lahesmaa, R., & Aittokallio, T. (2007). Systematic construction of gene coexpression networks with applications to human T helper cell differentiation process. Bioinformatics, 23(16), 2096-2103.
3. Huber, W., Von Heydebreck, A., Sultmann, H., Poustka, A., & Vingron, M. (2002). Variance stabilization applied to microarray data calibration and to the quantification of differential expression. Bioinformatics, 18(suppl 1), S96-S104.
4. Irizarry, R. A., Hobbs, B., Collin, F., Beazer Barclay, Y. D., Antonellis, K. J., Scherf, U., & Speed, T. P. (2003). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. Biostatistics, 4(2), 249-264.
5. Leal, L. G., Lopez, C., & Lopez-Kleine, L. (2014). Construction and comparison of gene co-expression networks shows complex plant immune responses. PeerJ, 2, e610.
6. López-Kleine, L., Leal, L., & López, C. (2013). Biostatistical approaches for the reconstruction of gene co-expression networks based on transcriptomic data. Briefings in functional genomics, elt003.
7. Tusher, V. G., Tibshirani, R., & Chu, G. (2001). Significance analysis of microarrays applied to the ionizing radiation response. Proceedings of the National Academy of Sciences, 98(9), 5116-5121.