ProCoNA: Protein Co-expression Network Analysis

David L Gibbs

May 3, 2016

1 De Novo Peptide Networks

ProCoNA (protein co-expression network analysis) is an R package aimed at constructing and analyzing peptide co-expression networks **?**. These networks are constructed using data derived from mass spectroscopy experiments (primarily accurate mass and time tag based LC-MS). This package streamlines the process of building and testing networks using an S4 object to bundle relevant network information for downstream analysis. The package is built around calls to WGCNA functions (weighted gene co-expression network analysis), which are fast and robust. ProCoNA adds a suite of statistical tests particularly suited to the unique challenges found in proteomics. These tests include permutation testing for module structure and within-protein correlation. Peptide co-expression networks bring novel approaches for biological interpretation, quality control, inference of protein abundance, potentially resolving degenerate peptide-protein mappings, and biomarker signature discovery.

Simulated peptide data is found in the data directory along with a simulated mass tag database. The simulated peptide data was generated using the OpenMS MSSimulator with a random selection of mouse proteins. The peptide data has peptide IDs as column names and a row for each sample. The simulated masstag database is a table mapping each peptide ID to a peptide sequence and potential parent proteins. The simulated phenotypes data frame has five phenotypes labeled A to E.

The first thing we want to do is to remove any peptides with excessive missing values. We do that by taking peptides that are present in greater than 80% of the samples.

```
> options(keep.source = TRUE, width = 70, stringsAsFactors=FALSE, digits=2)
> library(ProCoNA)

======================================================================
*
*  Package WGCNA 1.51 loaded.
*
```

```
*    Important note: It appears that your system supports multi-threading,
*    but it is not enabled within WGCNA in R.
*    To allow multi-threading within WGCNA with all available cores, use
*
*          allowWGCNAThreads()
*
*    within R. Use disableWGCNAThreads() to disable threading if necessary.
*    Alternatively, set the following environment variable on your system:
*
*          ALLOW_WGCNA_THREADS=<number_of_processors>
*
*    for example
*
*          ALLOW_WGCNA_THREADS=8
*
*    To set the environment variable in linux bash shell, type
*
*           export ALLOW_WGCNA_THREADS=8
*
*     before running R. Other operating systems or shells will
*     have a similar command to achieve the same aim.
*
========================================================================
```

```
> data(ProCoNA_Data)
> peptideData <- subsetPeptideData(peptideData, percentageNAsAllowed=0.2)
> dim(peptideData)
```

```
[1]  60 474
```

At this point we have a matrix of peptide data; peptides in columns and samples in rows. The ProCoNA network is based upon a correlation network that is scaled with an appropriate power to make the distribution of correlations approximately scale free. The WGCNA function pickSoftThreshold works very well for this.

```
> beta <- pickSoftThreshold(peptideData, networkType="signed", RsquaredCut=0.8)
```

```
   Power SFT.R.sq slope truncated.R.sq mean.k. median.k. max.k.
1      1  0.18000  8.04           0.99  256.00   257.000  281.0
2      2  0.03970  1.58           0.99  143.00   143.000  173.0
3      3  0.00075  0.14           0.99   82.30    82.200  110.0
4      4  0.04620 -0.79           0.99   48.80    48.500   72.1
5      5  0.08320 -0.83           0.98   29.80    29.100   49.0
6      6  0.20500 -1.05           0.93   18.70    18.000   34.4
7      7  0.32100 -1.18           0.86   12.10    11.400   24.8
8      8  0.44300 -1.26           0.84    7.97     7.340   18.3
```

```
9      9  0.56400 -1.29          0.89    5.39     4.780   13.8
10    10  0.72600 -1.38          0.96    3.72     3.170   10.7
11    12  0.80700 -1.54          0.91    1.87     1.440    6.8
12    14  0.88800 -1.60          0.99    1.00     0.693    4.5
13    16  0.87200 -1.65          0.91    0.56     0.346    3.1
14    18  0.83100 -1.67          0.85    0.33     0.180    2.1
15    20  0.93200 -1.60          0.98    0.20     0.095    1.5
```

```
> beta$powerEstimate

[1] 12
```

The first power with a scale-free model fit of $R^2 \geq 0.8$ is $\beta = 12$. So we'll use that in our network. At this point, the network can be built using a number of different parameters. Only some of the most important parameters will be mentioned here. Please see the documentation for the full list.

The network is built by first computing a correlation matrix, pairwise using peptides. The correlations can be taken as the absolute value ("unsigned") or the direction of correlation can be preserved ("signed"). The correlations themselves can be computed with Pearson's or with a Robust Bi-weight correlation. Using this correlation matrix as a basis, the rest of the network is computed starting with the topological overlap matrix (TOM). A dendrogram is computed using the distance matrix (1-TOM) which is then clustered using the dynamicTreecut algorithm resulting in module assignments (dynamicColors) for each peptide. Each module has an associated module eigenvector (ME) that acts as an overall summary for the module. The MEs can be correlated with sample phenotypes to search for biological relevance. Next, modules that are considered to be highly similar are merged producing a new set of module assignments (mergedColors), and module eigenvectors (mergedMEs).

After the network is complete, a permutation test can be performed indicating the significance of each module. The test compares the mean topological overlap within a given module to the topological overlap of random peptides (with same size as the test module). The number of permutation can be controlled with the toPermTestPermutes parameter. An example of calling the network building function is shown below.

```
> peptideNetwork <- buildProconaNetwork(networkName="my network",
+                                       pepdat=peptideData,
+                                       networkType="signed",
+                                       pow=beta$powerEstmate,
+                                       pearson=FALSE,
+                                       toPermTestPermutes=1000)

   Power SFT.R.sq slope truncated.R.sq mean.k. median.k. max.k.
1      1   0.0624  3.81          0.990  255.00   256.000  278.0
2      2   0.0094  0.74          0.988  142.00   142.000  168.0
3      3   0.0017  0.20          0.988   81.20    81.100  106.0
```

```
4      4   0.0093 -0.34          0.985   47.80    47.400   68.4
5      5   0.0339 -0.49          0.937   29.00    28.300   45.8
6      6   0.1900 -0.96          0.943   18.00    17.300   32.0
7      7   0.3000 -1.08          0.936   11.50    10.900   23.1
8      8   0.4650 -1.25          0.942    7.56     6.900   17.1
9      9   0.5940 -1.32          0.901    5.07     4.500   12.9
10    10   0.6550 -1.51          0.821    3.47     2.970   10.0
11    11   0.7290 -1.64          0.839    2.42     1.990    7.8
12    12   0.7160 -1.72          0.797    1.72     1.340    6.2
13    13   0.7540 -1.73          0.806    1.24     0.900    5.0
14    14   0.8750 -1.61          0.925    0.90     0.624    4.0
15    15   0.8940 -1.61          0.930    0.67     0.428    3.3
16    16   0.2190 -3.06          0.025    0.50     0.298    2.7
17    17   0.8220 -1.68          0.842    0.38     0.211    2.2
18    18   0.8460 -1.63          0.858    0.29     0.149    1.8
19    19   0.8540 -1.59          0.861    0.22     0.107    1.5
20    20   0.8670 -1.59          0.866    0.17     0.077    1.2
..connectivity..
..matrix multiplication..
..normalization..
..done.
 ..cutHeight not given, setting it to 0.999  ===>  99% of the (truncated) height range in de
 ..done.

  0   1   2   3   4
  6 139 131 108  90
 mergeCloseModules: Merging modules whose distance is less than 0.1
   .. will look for grey label ME0
   multiSetMEs: Calculating module MEs.
     Working on set 1 ...
     moduleEigengenes : Working on ME for module 1
     moduleEigengenes : Working on ME for module 2
     moduleEigengenes : Working on ME for module 3
     moduleEigengenes : Working on ME for module 4
    Changing original colors:
        1 to  1
        2 to  2
        3 to  3
        4 to  4
   Calculating new MEs...
   multiSetMEs: Calculating module MEs.
     Working on set 1 ...
     moduleEigengenes : Working on ME for module 0
     moduleEigengenes : Working on ME for module 1
     moduleEigengenes : Working on ME for module 2
     moduleEigengenes : Working on ME for module 3
```

```
     moduleEigengenes : Working on ME for module 4
```

```
> peptideNetwork
```

```
Network Name:  my network
  Number of samples :  60
  Number of peptides:  474
  Number of modules :  5
  Power used        :  14
  Network Type      :  signed
  ProCoNA version   :  0.99.2
```

The peptideNetwork is a S4 object of class "proconaNet". The object has a number of slots to store the various parts of the network: the correlation matrix, TOM, dendrogram, and module assignments as well as the permutation test results and parameters used in building the network. You can see the entire list by:

```
> getSlots("proconaNet")
```

```
proconaVersion    networkName        samples            adj
   "character"    "character"    "character"       "matrix"
           TOM       peptides        pepTree  dynamicColors
      "matrix"    "character"       "hclust"      "numeric"
           MEs      mergedMEs   mergedColors     colorOrder
  "data.frame"   "data.frame"      "numeric"    "character"
         power    networkType       permtest
     "numeric"    "character"       "matrix"
```

We can get a summary of the network:

```
> printNet(peptideNetwork)
```

```
Network Name:  my network
  Number of samples :  60
  Number of peptides:  474
  Number of modules :  5
  Power used        :  14
  Network Type      :  signed
  ProCoNA version   :  0.99.2
```

To access the various slots, we use accessor functions.

```
> networkName(peptideNetwork)
```

```
[1] "my network"
```

```
> samples(peptideNetwork)[1:5]
```

```
[1] "1" "2" "3" "4" "5"
```

```
> peptides(peptideNetwork)[1:5]
```

```
[1] "33699225"  "7393735"   "191136635" "7786631"   "83228277"
```

```
> mergedColors(peptideNetwork)[1:5]
```

```
3 4 3   5
1 2 1 0 4
```

The topological significance of network modules is shown by:

```
> peptideNetwork <- toPermTest(peptideNetwork, 100)
> permtest(peptideNetwork)
```

```
     module moduleSize TOMmean PermMean p-value
[1,]      1        139  0.0097   0.0025    0.01
[2,]      2        131  0.0074   0.0026    0.01
[3,]      4         90  0.0077   0.0026    0.01
[4,]      3        108  0.0104   0.0026    0.01
```

We can get a sense of what the network "looks" like by plotting the dendrogram.

```
> plotNet(peptideNetwork)
```

To learn more about individual modules we can subset peptides by module, and compute the mean topological overlap.

```
> module1 <- which(mergedColors(peptideNetwork) == 1)
> module1_TOM <- TOM(peptideNetwork)[module1, module1]
> mean(utri(module1_TOM))
```

```
[1] 0.0097
```

Ultimately, we want to know if this network contains modules related to a biological or clinical phenotype. To do that, we look at the correlation of the module eigenvectors with a quantitative phenotype. The function correlationWithPhenotypesHeatMap returns a data frame containing correlations and p-values between modules and phenotypes and also plots a heatmap showing the strength of correlation for each pair (as -log(p-value)). If a particular module eigenvector has a significant correlation to some given phenotype, then we may have found something interesting.

The function moduleMemberCorrelations instead returns a matrix of Pearson's correlations but for peptides and both modules and phenotypes. This matrix shows how each peptide is connected to both the module eigenvector and the various phenotypes. Peptides can be sorted within the module by their

centrality (connection to module eigenvector), imposing a sort of importance measure.

To extract peptide information and peptide correlations by module, enabling one to sort peptides by connection to the module eigenvector or highest correlation to a given phenotype, the moduleData function below is used, provided below.

```
> phenotypeCor <- correlationWithPhenotypesHeatMap(net=peptideNetwork,
+                                                  phenotypes=phenotypes[,1:5],
+                                                  modules=1:5,
+                                                  plotName="my plot",
+                                                  title="snazzy heatmap",
+                                                  textSize=0.7)
> pepcor <- moduleMemberCorrelations(pnet=peptideNetwork,
+                                    pepdat=peptideData,
+                                    phenotypes=phenotypes)
> ##########################################################################
> # quick function to write out the tables for specific modules.
> moduleData <- function(pepnet, pepcors, module, pepinfo, fileprefix) {
+   moduleX <- peptides(pepnet)[which(mergedColors(pepnet)==module)]
+   moduleInfo <- pepinfo[which(pepinfo$Mass_Tag_ID %in% moduleX),]
+   moduleCors <- pepcors[which(pepcors$Module==module),]
+   corname <- paste(fileprefix, "_correlations.csv", sep="")
+   write.table(moduleCors, file=corname, sep=",", row.names=F)
+   infoname <- paste(fileprefix, "_peptide_info.csv", sep="")
+   write.table(moduleInfo, file=infoname, sep=",", row.names=F)
+ }
> ##########################################################################
>
> # WRITE OUT A TABLE WITH THE BELOW FUNCTION CALL#
> # moduleData(peptideNetwork, pepcor, 1, masstagdb, "Module_1")
```

2 Running ProCoNA with the MSnbase infrastructure

In order to use this package, the data needed to be in a matrix form, with peptides (or proteins) in columns, and samples in rows. However, most MS data is not in this format. What to do!? One solution is to use the MSnbase package in Bioconductor, we can read data, normalize it, summarize it by feature, and export a quantitative expression! Consult the `MSnbase` vignette available with `vignette("MSnbase-demo", package = "MSnbase")` for more details.

```
> library(MSnbase)
> file <- dir(system.file(package = "MSnbase", dir = "extdata"),
+             full.names = TRUE, pattern = "mzXML$")
```

```
> rawdata <- readMSData(file, msLevel = 2, verbose = FALSE)
> experiment <- removePeaks(itraqdata, t = 400, verbose = FALSE)
> experiment <- trimMz(experiment, mzlim = c(112, 120))
> qnt <- quantify(experiment,
+                 method = "trap",
+                 reporters = iTRAQ4,
+                 strict = FALSE,
+                 parallel = FALSE,
+                 verbose = FALSE)
> qnt.quant <- normalise(qnt, "quantiles")
> gb <- fData(qnt)$PeptideSequence
> qnt2 <- combineFeatures(qnt.quant, groupBy = gb, fun = "median")
```

Similarly to the `subsetPeptideData` function above, it is easy to remove data with too many missing values in an `MSnSet` instance with the `filterNA` methods. The `pNA` argument controls the percentage of allowed `NA` values. Below we remove proteins with any `NA` but setting `pNA = 0.2` would have the same effect as the `subsetPeptideData(peptideData, percentageNAsAllowed=0.2)` call above.

```
> sum(is.na(qnt2))

[1] 1

> qnt2 <- filterNA(qnt2, pNA = 0)
> sum(is.na(qnt2))

[1] 0
```

Finally, although one could extract (and transpose[1]) the expression data with `peptideData <- t(exprs(qnt2))`, it is possible to directly proceed with the standard ProCoNA analysis using an `MSnSet` instance:

```
> peptideNetwork <- buildProconaNetwork(networkName="my network",
+                                       pepdat=qnt2,
+                                       networkType="signed",
+                                       pow=beta$powerEstmate,
+                                       pearson=FALSE,
+                                       toPermTestPermutes=1000)
```

Be sure to see `RforProteomics` for more information about proteomics and R/Bioconductor.

[1] Please note that ProCoNA requires input matrices to have samples in rows, which means the assay data matrix must be transposed before use.

References

Gibbs D.L., Baratt A., Baric R.S., Kawaoka Y., Smith R.D., Orwoll E.S., Katze M.G., McWeeney S.K. Protein Co-expression Network Analysis (ProCoNA) Journal of Clinical Bioinformatics 2013, 3:11 doi:10.1186/2043-9113-3-11

Langfelder, P. (2008). WGCNA: an R package for weighted gene co-expression network analysis. In *BMC Bioinformatics,*.

Falcon, S. and Gentleman, R. (2007). Using GOstats to test gene lists for GO term association *Bioinformatics* 23 (2): 257-258

Langfelder P., Zhang, B., Horvath, S. (2008). Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R *Bioinformatics* 24 (5): 719-720

Mason, M.J., Fan, G., Plath, K., Zhou, Q., and Horvath, S. (2009) Signed weighted gene co-expression network analysis of transcriptional regulation in murine embryonic stem cells. *BMC Genomics*. Jul 20;10:327

Gatto L., Lilley K. (2011) MSnbase – an R/Bioconductor package for isobaric tagged mass spectrometry data visualisation, processing and quantitation *Bioinformatics* Jan 15;28(2):288-9.

Gatto L., Christoforou, A. (2013) Using R and Bioconductor for proteomics data analysis. *B*iochim Biophys Acta. 2013 May 18. pii: S1570-9639(13)00186-6. doi: 10.1016/j.bbapap.2013.04.032.