pwOmics - Pathway-based Integration of time-series Omics Data using public database knowledge

Astrid Wachter
Medical Statistics, University Medical Center Göttingen, Germany

August 24, 2015

Contents

1 Introduction

Characterization of biological processes can be performed in great detail with the increased generation of omics data on different functional levels of the cell. Especially interpretation of time-series omics data measured in parallel with different platforms is a complex but promising task, needing consideration of time-independent combination of omics data and additionally time-dependent signaling analysis. As each measurement technique shows a certain bias and has natural limitations in identifying full signaling responses [**yeger-lotem_bridging_2009**], such cross-platform analysis is an up-to-date approach in order to connect biological implications on different signaling levels. Using diverse data types is expected to provide a deeper understanding of global biological functions and the underlying complex processes [**Kholodenko2012**]. This is why computational data analysis tools for interpretation of data from proteomics and transcriptomics measurements in parallel are needed.
pwOmics is a tool for pathway-based level-specific data comparison and analysis of single time point or time-series omics data measured in parallel. It provides individual analysis workflows for the different omics data sets (see Figure **??**) and in addition enables consensus analysis of omics data as shown in the workflow overview in Figure **??**.

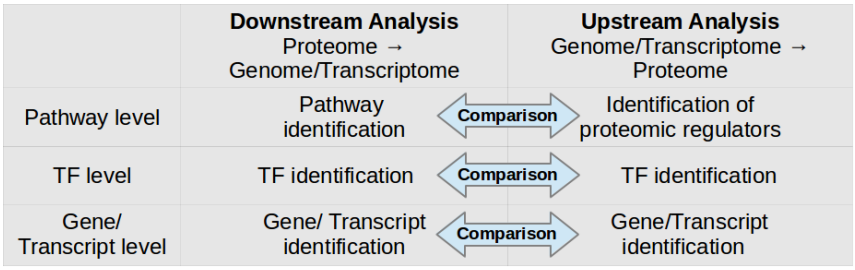

Figure 1: *pwOmics* downstream and upstream analysis.

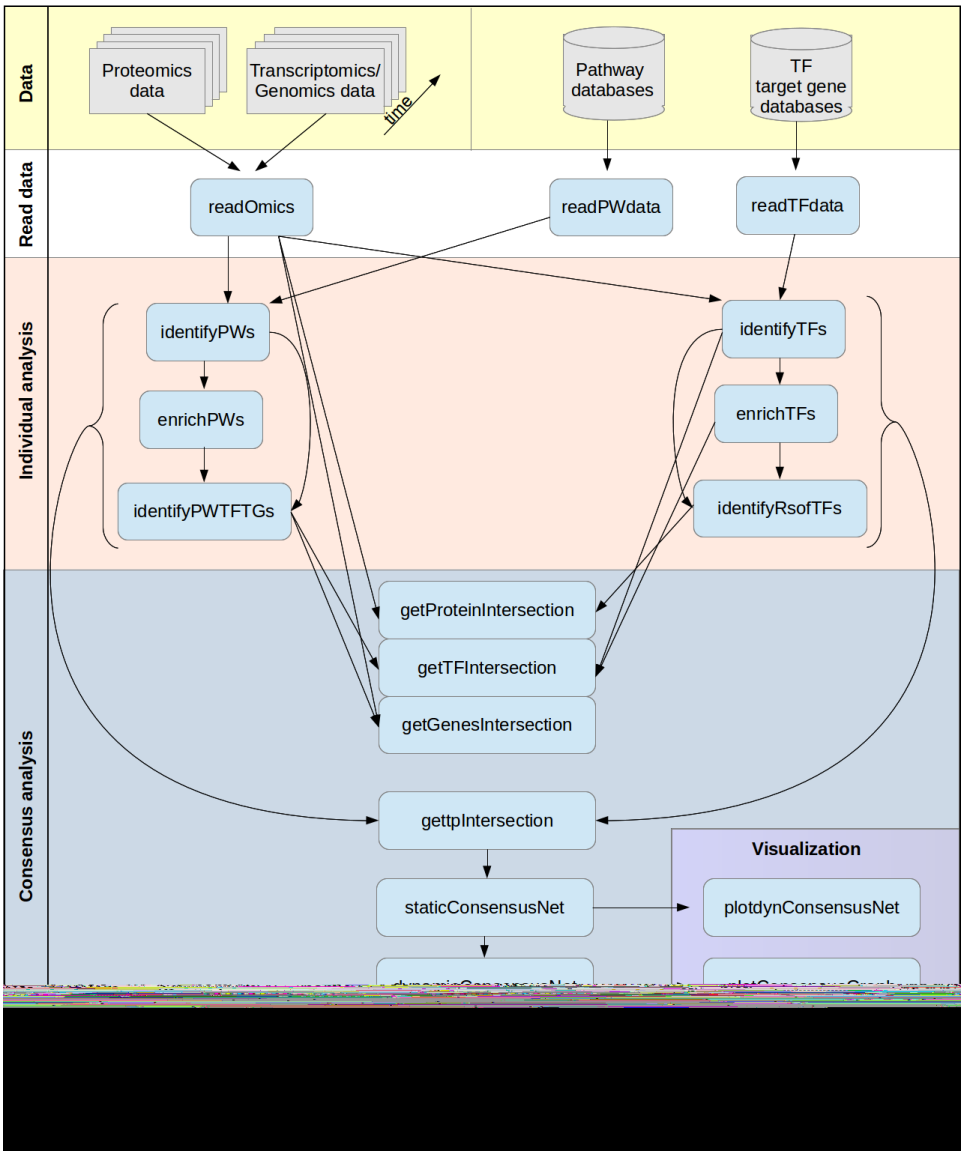

Figure 2: *pwOmics* workflow overview.

Up to this point analysis is restricted to human species. In future an expansion of the package is possible dependent on available online open access database information.

2 Databases

As *pwOmics* is a package for data integration based on prior pathway and transcription knowledge data, it is necessary to define the databases to work with. Three different kinds of databases are necessary to do all analyses steps:

1. Pathway databases:
   The user can choose from Biocarta [**nishimura_biocarta_2001**], Reactome [**milacic_annotating_2012**, **croft_reactome_2014**], PID [**schaefer_pid:_2009**] from the National Cancer Institute (NCI) and KEGG [**kanehisa_data_2014**, **kanehisa_kegg:_2000**].

2. Protein-protein interaction (PPI) database:
   STRING [**franceschini_string_2013**].

3. Transcription factor (TF) - target gene databases:
   The user can choose from ChEA [**lachmann_chea:_2010**], Pazar [**portales-casamar_pazar_2009**, **portales-casamar_pazar:_2007**] and/or decide to specify an own file e.g. based on a commercial database.

The pathway database information is used to identify the pathways of the differentially abundant proteins in the downstream analysis as well as upstream protein regulators of TFs in the upstream analysis. The PPI database STRING [**franceschini_string_2013**] was chosen to define the protein net for the consensus analysis. The TF - target gene databases information is necessary for the TF identification in pathways in the downstream analysis. Additionally the upstream TFs of differentially expressed genes/transcripts are identified in the upstream analysis based on this information.
In downstream analysis the pathway gene set information is used, whereas in the upstream analysis also the pathway topology information is exploited.

The database information is downloaded internally via *STRINGdb* and *AnnotationHub* [**Morgan**] package. In case the author is interested also in the metadata of the pathway database and TF - target database it can be received by

```
> library(pwOmics)
> library(AnnotationHub)
> ah = AnnotationHub()
> #pathway databases
> pw = query(ah, "NIH Pathway Interaction Database")
> pw[1]

AnnotationHub with 1 record
# snapshotDate(): 2015-08-17
# names(): AH22329
# $dataprovider: NIH Pathway Interaction Database
# $species: Homo sapiens
# $rdataclass: biopax
# $title: BioCarta.owl.gz
# $description: BioCarta BioPax file from NCI Pathway Interaction Database
# $taxonomyid: 9606
# $genome: hg19
# $sourcetype: BioPax
# $sourceurl: ftp://ftp1.nci.nih.gov/pub/PID/BioPAX/BioCarta.owl.gz
# $sourcelastmodifieddate: 2009-09-09
# $sourcesize: 338343
```

```
# $tags: BioCarta, BioPax, Pathway Interaction Database
# retrieve record with 'object[["AH22329"]]'

> #TF-target databases
> chea = query(ah, "ChEA")
> chea[1]

AnnotationHub with 1 record
# snapshotDate(): 2015-08-17
# names(): AH22237
# $dataprovider: ChEA
# $species: NA
# $rdataclass: data.frame
# $title: chea-background.zip
# $description: ChEA background file, containing transcription factor data t...
# $taxonomyid: NA
# $genome: NA
# $sourcetype: Zip
# $sourceurl: http://amp.pharm.mssm.edu/result/kea/chea-background.zip
# $sourcelastmodifieddate: 2015-03-09
# $sourcesize: 3655103
# $tags: ChEA, Transcription Factors
# retrieve record with 'object[["AH22237"]]'

> pazar = query(ah, "Pazar")
> pazar[1]

AnnotationHub with 1 record
# snapshotDate(): 2015-08-17
# names(): AH22238
# $dataprovider: Pazar
# $species: NA
# $rdataclass: GRanges
# $title: pazar_ABS_20120522.csv
# $description: TF - Target Gene file from pazar_ABS_20120522
# $taxonomyid: NA
# $genome: NA
# $sourcetype: CSV
# $sourceurl: http://www.pazar.info/tftargets/pazar_ABS_20120522.csv
# $sourcelastmodifieddate: 2012-06-04
# $sourcesize: 120202
# $tags: Pazar, Transcription Factors
# retrieve record with 'object[["AH22238"]]'
```

In case you want to use TF - target gene information which is not part of the mentioned databases but e.g. part of a commercial database, a user-specified file can be used for the analysis. This file should be a '.txt' file with first column transcription factors and second column target gene symbols without a header, e.g.:

```
REST SUPT7L
REST SUV420H1
REST SUV420H2
REST SVOP
REST SYCN
REST SYN2
REST SYP
REST SYPL2
REST SYT1
...
```

The STRING PPI-information is downloaded automatically while processing and analyzing the data: The *STRINGdb* package [**franceschini_string_2013**] is used here.

3 Example dataset

The example dataset used here for demonstration purposes is the one presented in [**Waters2012**], which comprises the mitogenic response of human mammary epithelial cells to epidermal growth factor (EGF). This dataset includes whole genome time course microarray data measured with NimbleGen whole genome 60-mer oligonucleotide arrays (Design Version 2003_10_27) at time points 0, 1, 4, 8, 13, 18 and 24 hr after EGF stimulation. The complementary proteomics data was measured with LC-FTICR (Fourier-transform ion cyclotron resonance-mass spectrometry coupled with advanced capillary liquid chromatography) at time points 0.25, 1, 4, 8, 13, 18 and 24 hours after EGF stimulation. Preprocessing of data was done as described in [**Waters2012**] resulting in lists of significant genes and proteins for each time point as log10 expression ratios relative to the time 0 hr controls.

4 Data pre-processing

pwOmics is a package for secondary data analysis, i.e. it needs already pre-processed data as input for the analysis. The input required is

1. a list of all protein IDs measured,
2. a list of all gene/transcript IDs measured,
3. a list of differentially abundant proteins + log fold changes,
4. a list of differentially expressed genes/transcripts + log fold changes.

The IDs need to be gene symbols, both for protein and gene/transcript data. In case time-series data is analyzed inputs 3. and 4. needs to be specified for each time point. It is absolutely necessary, that all proteins and genes/transcript in inputs 3. and 4. are part of the lists of all protein IDs and all gene/transcript IDs, respectively.

The OmicsData object is the format used for data analysis in *pwOmics* package. It contains a list of four main elements:

1. OmicsD - here the omics data set, its description and the results are stored

2. PathwayD - here the chosen pathway databases and the generated Biopax model is stored

3. TFtargetsD - here the chosen TF-target gene databases and the combined TF-target gene information is stored

4. Status - The status variable equals '1' in case not all information needed for the analysis is read in yet and '2' after identification of the first upstream/downstream signaling levels. As the enrichment step is not necessarily part of the analysis and dependent on the pathway database and the TF-target gene database the identification of signaling molecules in further levels might not be successful, the status variable is not used in the further analysis.

Thus *pwOmics* reads in the omics data set provided by the user to the first element of the OmicsData object and further on stores all the results in this part as well.

This is why the user has to provide the omics data set in a special format: A list needs to be generated with a protein list named 'P' as first element and a gene/transcript list named 'G' as second element. These lists contain as first element a data frame with all (unique) protein IDs and gene/transcript IDs in the first column, respectively, and as second element a list with data frames for each time point of measurement. The data frames have two columns with the first one containing the differentially abundant/expressed proteins or genes/transcripts as gene symbols and the second column containing the corresponding log fold changes, e.g.:

```
> data(OmicsExampleData)
> OmicsExampleData
```

Generated as in the following example:

```
OmicsExampleData = list(P = list(allPIDs,
                  list(PIDstp0.25, PIDstp1, PIDstp4, PIDstp8,
                       PIDstp13, PIDstp18, PIDstp24)),
                       G = list(allGIDs,
                  list(GIDstp1, GIDstp4, GIDstp8, GIDstp13,
                       GIDstp18, GIDstp24)))
```

```
> head(OmicsExampleData$P[[2]][[1]])

  GeneSymbol     X15min
1     MRPS17  0.6976049
2      RPS12 -1.0297977
```

```
3     SLC3A2 -1.2623327
4       RPL8  0.8304820
5       ACTB -2.4914461
6      ALDOA  0.8637013
```

In case the user only wants to analyze omics data from a single time point just one data frame has to be specified.
The time points do not have to be the same for protein and gene/transcript data and need to be specified when reading in the omics data set separately via the 'tp_prots' and 'tp_genes' parameters of the 'readOmics' function.

```
> data_omics = readOmics(tp_prots = c(0.25, 1, 4, 8, 13, 18, 24),
+                        tp_genes = c(1, 4, 8, 13, 18, 24),
+                        OmicsExampleData,
+                        PWdatabase = c("biocarta", "kegg", "nci",
+                                       "reactome"),
+                        TFtargetdatabase = c("chea", "pazar"))
```

If data from a single timepoint measurement should be analyzed the user simply assigns the experiment number '1' for these parameters:

```
#for single time point data set:
omics = list(P = list(allPIDs, list(PIDs_1)),
             G = list(allGIDs, list(GIDs_1)))
data_omics = readOmics(tp_prots = c(1),
                    tp_genes = c(1),
                    OmicsExampleData,
                    PWdatabase = c("biocarta", "kegg", "nci",
                                   "reactome"),
                    TFtargetdatabase = c("chea", "pazar"))
```

Additionally the selected databases have to be specified.

The stored information can be easily accessed via the following functions:

```
> getOmicsTimepoints(data_omics)
> head(getOmicsallProteinIDs(data_omics))
> head(getOmicsallGeneIDs(data_omics))
> head(getOmicsDataset(data_omics, writeData = FALSE)[[1]])
```

5 Individual analysis

As shown in Figure **??** the analysis is based on an individual analysis of the proteomic and the genomic/transcriptomic data. The downstream analysis and upstream analysis are described in the following subsections.
Prior to that the database information has to be read in. In a first step the TF-target information can be made accessible to the OmicsData object by:

```
data_omics = readTFdata(data_omics)
```

Via the 'TF_target_path' parameter the path of the user-specified file can be given. This information can be used additionally to the selected database content.
Secondly, the 'readPWdata' function takes the OmicsData object with the provided information about the omics data set and the path of the prepared '.RData' files from the pathway databases (see section **??**) and automatically generates the corresponding genelists of the pathway data if 'loadgenelists = FALSE'. In this step the automatic definition of internal differing IDs for different pathway databases is necessary, which are stored in a new biopax model in the OmicsData object.

```
data_omicsPW = readPWdata(data_omics,
               loadgenelists = FALSE)
```

As the process of generating genelists with these IDs can take some time - especially for rather big databases such as Reactome [**milacic_annotating_2012**, **croft_reactome_2014**] - the genelists for the different databases are automatically stored in the working directory and can be reused in another analysis when the corresponding path containing these files is given to the 'readPWdata' function as loadgenelists parameter.

```
data_omics = readPWdata(data_omics,
             loadgenelists = "Genelist_reactome.RData")
```

Automatically the information of the selected databases and/or the corresponding user-specified file are merged. The file format (if this option is used) should be exactly as specified in section **??**.

5.1 Downstream analysis

The downstream analysis is starting with the provided proteomic data (either single time point data or time-series data). The first step is the identification of the pathways in which the differentially abundant proteins play a role. *pwOmics* performs this searching step on the basis of the provided proteomic data set and the selected pathway database(s).
After reading in these information the user can follow the workflow for downstream analysis and identify the pathways in which the differentially abundant proteins are present:

```
data_omics = identifyPWs(data_omics)
```

In a next step pathway enrichment can be conducted. The user can specify the multiple testing correction method as well as the significance level for this step. In case of few identified pathways this might result in too few pathways for further analysis. In this case the enrichment step should be skipped.

```
data_omics = enrichPWs(data_omics, "BH", alpha = 0.05)
```

Following the workflow the next step is the identification of the transcription factors in these (enriched) pathways, which is done with the information provided by the chosen TF-target gene database. The user can choose if only the enriched pathways or all pathways should be considered for further analysis:

```
data_omics = identifyPWTFTGs(data_omics, only_enriched = FALSE)
```

For use of this function the working directory should contain the previously generated genelists.

The results of the downstream analysis can be easily accessed by the following functions:

```
getDS_PWs(data_omics)
getDS_TFs(data_omics)
getDS_TGs(data_omics)

#Access biopax model generated newly on basis of selected
#pathway databases:
getBiopaxModel(data_omics)
```

5.2 Upstream analysis

The upstream analysis is starting with the provided gene/transcript data (either single time point data or time-series data). It first of all identifies the upstream TFs of the differentially expressed genes/transcripts. This step is done with the provided/selected information of TF-target gene pairs.
Given this information, the identification of upstream TFs can be done:

```
data_omics = identifyTFs(data_omics)
```

Similarly as in the downstream analysis also in the upstream analysis an optional enrichment step can be conducted, but here on the TF level.

```
data_omics = enrichTFs(data_omics, "BH", alpha = 0.05)
```

Upstream of the (enriched) TFs the regulator proteins can be identified with the following function:

```
data_omics = identifyRsofTFs(data_omics, only_enriched = FALSE,
             noTFs_inPW = 1, order_neighbors = 10)
```

Again, the user can specify if only the enriched TFs or all TFs should be considered for further analysis. The identification of upstream regulators is done in the following way:

1. Identification of the pathways the previously identified TFs are part of.

2. Selection of pathways according to the user-specified parameter 'noTFs_inPW': Only those pathways are considered in further analysis with at least this number of TFs in the pathway. Minimum number of TFs in the pathway is 2.

3. Upstream regulators are identified for these TFs. This is done by finding first for each TF the pathway neighborhood according to the user-specified parameter 'order_neighbors'. This parameter specifies the order of the identified pathway neighborhood. Then the intersection of all identified neighborhoods for all TFs in a pathway is determined. The resulting pathway node set is defined here as the set of regulator proteins.

In case the pathways under consideration do not have pathway components in the downloaded biopax model, this will be indicated with a warning. This warning can be ignored by the user in regard to the following analysis steps.
The results of the upstream analysis can be accessed with the following functions:

```
getUS_TFs(data_omics)
getUS_PWs(data_omics)
getUS_regulators(data_omics)
```

6 Consensus analysis

The consensus analysis combines the results from upstream and downstream analysis by constituting in particular the comparative analysis of the results of the two different data sets. The intersection analysis simply compares the results of the separate upstream and downstream analysis. The static consensus analysis enables setting up static consensus graphs for each time point measured in parallel. Finally, the dynamic consensus analysis provides the user with one final dynamic consensus graph obtained from the data changes over time based on dynamic bayesian network inference. The dynamic consensus analysis is self-evidently only conductable with time-series data sets measured for proteome and genome/transcriptome data in parallel.

6.1 Intersection analysis

The intersection analysis of *pwOmics* is a simple comparative analysis of the results of upstream and downstream analysis. Thus, it enables a comparison of single time point data and time-series data, the latter also for non-corresponding time points in the different data sets. The comparison is possible on the three different functional levels considered in this package: On the proteome level, the transcription factor level and gene/transcript level.

```
getProteinIntersection(data_omics,
                       tp_prot = 4,
                       tp_genes = 4)
getTFIntersection(data_omics,
                  tp_prot = 4,
                  tp_genes = 1)
getGenesIntersection(data_omics,
                     tp_prot = 4,
                     tp_genes = 13)
```

These functions not only enable a comparison of the same timepoints on the distinct levels, but for time-series data sets also for non-matching time points: With the time resolution of measuring omics data in most cases being predefined by expected signaling changes and financial limitations the potential in the interpretation of the results is strongly confined to the experimental design decisions. Thus, measured signaling changes, which naturally consist of a superposition of diverse time-scales of transcriptional and translational processes and comprehend diverse frequency patterns [**yosef_impulse_2011**], are dependent

on the sampling. This means for some of the signaling axes it might be the case, that

- changes are not detected at all as their rate is too high,
- hopefully most are represented in the data and
- some might be so slow that their change is not considered significant and thus are excluded from analysis.

As the corresponding signaling changes are not expected to be seen at the same time point in proteome data and gene/transcript data it is necessary to enable also the comparison of non-corresponding time points.
The possibility to compare such time points naturally cannot account for the changes not captured during measurement, however, it gives the possibility to consider also the time needed for regulatory control mechanisms in the interpretation of the measurement results - even if this shows considerable variations as well.
In case the user wants to compare the corresponding time points on the three levels simultaneously he can do so by using the following function:

```
gettpIntersection(data_omics)
```

6.2 Static consensus analysis

The static consensus analysis goes one step ahead and integrates the results gained from the comparative analysis of the corresponding time points to a consensus net for each time point. The change of this consensus net over time gives a first insight into the changes seen statically at the different time points. However, the static consensus nets do not yet include information gathered over time - as it is the case in the dynamic consensus analysis (see section **??**). This is why the static consensus analysis is also applicable for single time point measurements.

The static consensus analysis is conducted by generation of a Steiner tree [**Kleinberg_AlgorithmDesign_200**
on basis of matching proteins and TFs identified in downstream and upstream analysis for each corresponding time point. The underlying graph used is the protein-protein-interaction (PPI) graph from the STRING database reduced to the connected nodes. The matching proteins and TFs are considered as terminal nodes and are connected via a shortest path-based approximation of the Steiner tree algorithm [**Takahashi_Steiner_1980**, **Sadeghi2013**] across the reduced PPI-STRING-graph. Subsequently knowledge of TF-target gene pairs from the chosen database is used to expand the graph with matching genes/transcripts from both upstream and downstream analysis. In case the consensus graph contains corresponding proteins and genes/transcripts, feedback loops are added automically.

```
consensusGraphs = staticConsensusNet(data_omics)
```

6.3 Dynamic consensus analysis

Unlike the static consensus analysis, the dynamic consensus analysis takes into consideration also the signaling changes over time by applying dynamic bayesian

network inference. The packages used for the dynamic consensus analysis are *longitudinal* [**Opgen_longi_2006**, **rainer_opgen-rhein_inferring_2006**] to adjust the format of the data and the actual network inference part is done via the *ebdbNet* [**Rau_ebdbnet_2010**] package. This package includes an iterative empirical Bayesian procedure with a Kalman filter estimating the posterior distributions of the network parameters. The defined prior structure of the network is used for a straightforward estimation of hyperparameters via an expectation maximization (EM)-like algorithm and the dimension of the hidden states are determined via the singular value decomposition (SVD) of a block-Hangel matrix.

The nodes included into the network inference step are nodes which are part of the static consensus graphs from corresponding time points of the two different measurement types, i.e.

1. proteins identified in upstream and downstream analysis at the same time points,

2. Steiner nodes identified via static consensus analysis,

3. TFs identified in upstream and downstream analysis at the same time points and

4. genes/transcripts identified in upstream and downstream analysis at the same time points.

To apply dynamic network inference a reasonable number of measurements needs to be available. As in most cases of parallel protein and gene/transcript measurements only very few corresponding time steps are available it is necessary to artificially introduce additional time steps. This is done by generating smoothing splines applied on the log fold changes provided by the user under the simplifying assumption of a gradual change of signaling between the different time points.

This assumption, however, has to be applied consciously and carefully, as there might be higher frequency signaling components superimposed (see for a comprehensive analysis of temporal dynamics of gene expression [**yosef_impulse_2011**]). In theory a signal has to be sampled 2 times its maximal frequency in order to be able to reconstruct it exactly from time discrete measurements (Nyquist-Shannon sampling theorem [**Shannon_theorem_1949**, **Nyquist_theorem_1928**]). This means only exact interpretation of those signaling axes are possible that have a frequency which is smaller than half of the sampling frequency. However, under certain preconditions on signal structure and the sampling operator reconstruction of the original signal can be done with a lower sampling rate [**DBLP:journals/tit/BlumensathD09**]. This is an interesting starting point for a more comprehensive dynamic analysis of the expected signals and the sampling needed for an extensive data mining of omics data sets measured in parallel, but exceeds the scope of this package.

The number of time points generated additionally via smoothing splines is based

on simulation results of *ebdbNet* analysis for median area under the curve (AUC) values of receiver operating characteristic (ROC) curves: In their results it was shown that a plateau at around 50 to 75 time points was reached. Thus in *pwOmics* 50 time points are predicted with smoothing splines in order to apply dynamic bayesian network inference on omics data sets measured in parallel.

After generation of these time points a block-Hankel matrix of autocovariances is constructed based on the time series abundance/expression data. For this the user needs to provide the *laghankel* parameter, specifying the maximum relevant time lag to be used in constructing the block-Hankel matrix. With a singular value decomposition (see function 'hankel' of *ebdbNet* package) the number of hidden states can be determined. Here, the user can specify the *cutoffhankel* parameter to choose the cutoff to determine the desired percent of total variance explained by the singular values. Additional parameters on convergence criteria and iterations performed can be specified. For further details the user is referred to [**Rau_ebdbnet_2010**].

```
library(ebdbNet)
library(longitudinal)
dynInferredNet = dynamicConsensusNet(data_omics,
                 laghankel = 3,
                 cutoffhankel = 0.9)
```

7 Time profile clustering

An additional analysis option is clustering of co-regulation patterns over time. It provides information about the signaling molecules with common dynamic behaviour and thus allows to draw conclusions in terms of signaling chronology. Time profile clustering is performed as soft clustering based on the *Mfuzz* package [**Mfuzz2012**]. The advantage of this clustering method is that a protein, TF or gene/transcript can be assigned to several clusters, thus reducing the sensitivity to noise and the information loss hard clustering exhibits. It is implemented as fuzzy c-means algorithm [**Hathaway_Pattern1986**] and iteratively optimizes the objective function to minimize the variation of objects within the clusters. The user needs to provide a 'min.std' threshold parameter if proteins or genes/transcripts with a low standard deviation should be excluded. In addition the maximum number of cluster centers which should be tested in the 'minimum distance between cluster centroid test' has to by given. This number is used as initial number to determine the data-specific maximal cluster number based on the number of distinct data points. For more details see [**Mfuzz2012**] and [**schwammle_simple_2010**].

```
library(Mfuzz)
fuzzyClusters = clusterTimeProfiles(dynConsensusNet,
                                    min.std = 0,
                                    ncenters = 12)
```

8 Visualization

To complement the results from the different comparisons and analyses (accessible via the 'get...' functions) the *pwOmics* package provides visualization functions for the different analyses. The consensus graphs of the static analysis for one or more corresponding time points can be plotted with the following function (see Figure **??**):

```
plotConsensusGraph(consensusGraphs, data_omics)
```

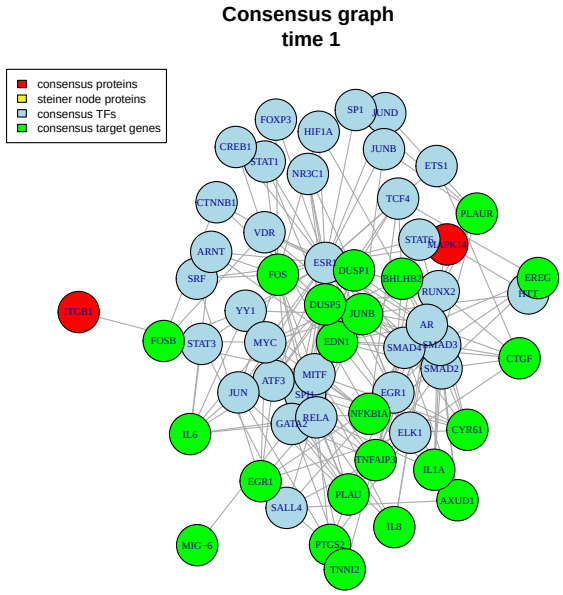

Figure 3: *pwOmics* static consensus graph: Time point 1 hr.

The dynamic consensus analysis result can be visualized as follows (see Figure **??**):

```
plotdynConsensusNet(dynInferredNet, sig.level = 0.65)
```

Here, the parameter 'sig.level' is the significance level used as cutoff for plotting edges in the network and has to be specified in the range between 0 and 1. Furthermore the user can indicate if unconnected nodes should be removed and provide additional *igraph* [**igraph2006**] layout parameters.

However, as the user can access the networks easily *tkplot* from the *igraph* R package is a nice interactive graph drawing alternative. In addition plot parameters can be easily changed as the result networks are of class 'igraph'.
In order to plot the results from time profile clustering (see Figure **??**) the following function can be used:

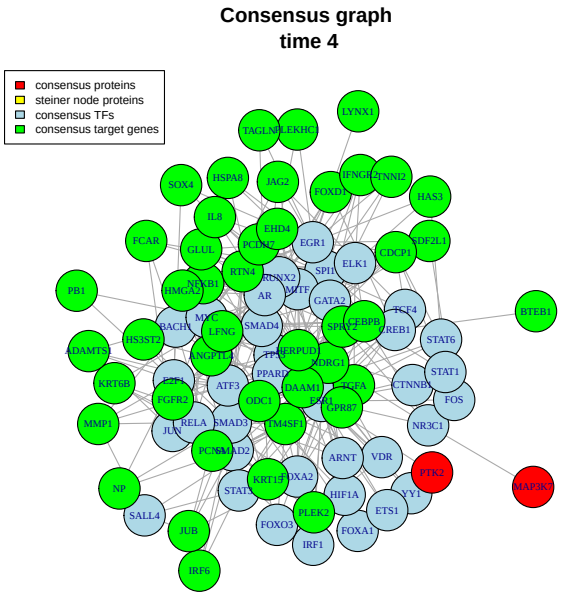

Figure 4: *pwOmics* static consensus graph: Time point 4 hrs.

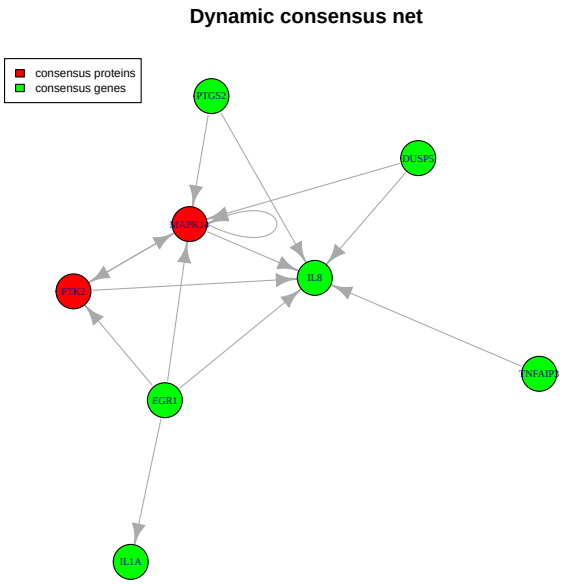

Figure 5: *pwOmics* dynamic consensus graph.

```
plotTimeProfileClusters(fuzzyClusters)
```

The different colours represent the different clusters. The legend is only shown if the number of genes and proteins is not too large. Otherwise the user can easily access this information by having a look to the output of the 'clusterTimeProfiles' function which provides information about cluster centers, the number of data points in each cluster of the closest hard clustering, cluster indices, and additional parameters explained in detail in the 'mfuzz' documentation. In the legend the attachments '_g' and '_p', respectively, indicate, if the node originally derives from protein or gene/transcript measurements.

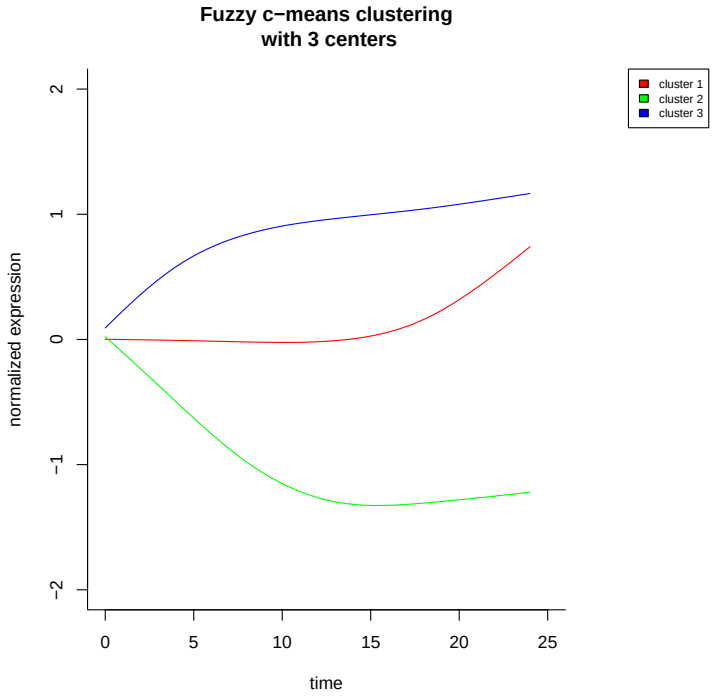

Figure 6: *pwOmics* time profile clusters.

9 Session Information

- R Under development (unstable) (2015-03-11 r67980), Platform: x86_64-unknown-linux-gnu (64-bit), Running under: Ubuntu precise (12.04.5 LTS)

- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=en_US.UTF-8, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C

- Attached base packages: tcltk, parallel, stats, graphics, grDevices, utils, datasets, methods, base

- Other attached packages: ebdbNet_1.2.3, Mfuzz_2.27.1, DynDoc_1.45.0, widgetTools_1.45.0, e1071_1.6-4, Biobase_2.27.3, BiocGenerics_0.13.11, longitudinal_1.1.11, corpcor_1.6.7, igraph_0.7.1, pwOmics_0.99.2

- Loaded via namespace (and not attached): Rcpp_0.11.5, XVector_0.7.4, BiocInstaller_1.17.7, GenomeInfoDb_1.3.19, plyr_1.8.1, AnnotationHub_1.99.82, tkWidgets_1.45.0, class_7.3-12, bitops_1.0-6, biomaRt_2.23.5, digest_0.6.8, RSQLite_1.0.0, shiny_0.11.1, DBI_0.3.1, rBiopaxParser_2.5.0, stringr_0.6.2, httr_0.6.1, S4Vectors_0.5.23, gtools_3.4.2, caTools_1.17.1, IRanges_2.1.44, stats4_3.2.0, data.table_1.9.4, R6_2.0.1, AnnotationDbi_1.29.24, XML_3.98-1.1, RJSONIO_1.3-0, gdata_2.13.3, reshape2_1.4.1, STRINGdb_1.7.0, gplots_2.16.0, htmltools_0.2.6, GenomicRanges_1.19.54, mime_0.3, interactiveDisplayBase_1.5.6, xtable_1.7-4, httpuv_1.3.2, KernSmooth_2.23-14, RCurl_1.95-4.5, chron_2.3-45