piano

Platform for Integrative Analysis of Omics data

An R-package for enriched gene set analysis

Vignette

Leif Väremo, Intawat Nookaew

January 2, 2017

Systems and Synthetic Biology
Department of Chemical and Biological Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden

Contents

1 Introduction

The `piano` package is intended as a platform for performing gene set analysis (GSA) using a selection of available methods (some of which are also implemented in separate packages or other software). The advantage with using `piano` is that all of these methods can be run using the same setup, resulting in minimal effort for the user when switching between methods. Additionally, `piano` contains a new approach which divides the gene set results into directionality classes, giving deeper information. Further on, `piano` contains functions to perform consensus analysis using different GSA methods. The details are presented in [1] and are briefly discussed in Section 4 - Gene set analysis.

Piano also contains simple functions for basic microarray analysis. The purpose is to offer a seamless analysis workflow, starting from raw microarray data and ending with GSA results. Of course other packages and software can be used for the microarray processing and analysis prior to using `piano` for GSA, and of course other data than microarray data (e.g. RNA-seq data) can be used as input to the GSA.

Section 2 covers installation, Section 3 covers microarray analysis and Section 4 covers GSA. More specific information, than mentioned in this introduction, regarding these topics can be found in each section below.

1.1 Contact

For questions regarding `piano`, contact the authors, Leif Väremo and Intawat Nookaew at `piano.rpkg@gmail.com`.

1.2 Citation

If you use `piano` in your research please cite:

> Väremo, L., Nielsen, J., Nookaew, I. (2013) Enriching the gene set analysis of genome-wide data by incorporating directionality of gene expression and combining statistical hypotheses and methods. *Nucleic Acids Research.* 41 (8), 4378-4391. doi:10.1093/nar/gkt111

2 Getting started

A general introduction to R and apropriate tutorials can be found online, e.g. on the CRAN webpage (`http://cran.r-project.org/`).

2.1 Installation and first run

Install `piano` by typing in R:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("piano")
```

Once everything is installed you can load `piano` into your workspace:

```
> library(piano)
```

Note that this command has to be run every time you restart R in order to be able to use the functions in `piano`. To get immediate help on `piano` or any of its functions (e.g. `runGSA`), type:

```
> ?piano
> ?runGSA
```

2.2 Optional requirements

To install all optionally required packages, simply run:

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("piano", dependencies=TRUE)
```

This should install everything you need! If you only want to install selected packages, use the following guidelines:

If you plan to use `piano` for basic microarray analysis (e.g. normalization of raw-data, quality control and differential expression analysis) it is suggested that you also install the following packages for full functionality:

```
> pkgs <- c("yeast2.db", "limma", "affy", "plier", "affyPLM", "gtools", "plotrix")
> source("http://www.bioconductor.org/biocLite.R")
> biocLite(pkgs)
```

If you wish to import SBML models to be used to define gene sets you will need the Bioconductor package `rsbml`. Note that `rsbml` requires the open-source API LibSBML to be installed on your system (see `http://sbml.org/Downloads` and `rsbml` at Bioconductor).

In order to run all the examples in this vignette, you will also need the Bioconductor package `biomaRt`.

3 Microarray analysis

This section will cover the use of `piano` for basic microarray analysis. For instructions on the major purpose of `piano`, gene set analysis (GSA), please see the next section. A common starting point of gene set analysis is gene expression data, commonly provided by microarrays although the use of other technologies, such as RNA-sequencing, are increasing. The purpose with the microarray analysis functions in `piano` is to provide the user with basic and simple steps to transform raw data into statistics that are appropriate as input to the GSA. Of course it is perfectly possible to use other packages or software to analyze data prior to continuing with GSA using `piano`, and of course the input data to `runGSA` is not limited to gene expression data.

Essentially, the microarray analysis functions are wrappers around the popular and widely used packages `affy` and `limma`. Hence, for more advanced analysis the user is referred directly to those packages. The microarray analysis workflow in `piano` is as follows: (1) loading raw or normalized data for preprocessing (`loadMAdata`), (2) performing quality control (`runQC`) and (3) checking for differential expression (`diffExp`). After these steps it is possible, if one wishes, to continue with GSA.

3.1 Loading the data

The starting point for the microarray analysis is the `loadMAdata` function. Typically you want to load a set of CEL files. This also requires a setup file which describes the experimental setup and assigns each CEL file to a specific condition. This file can be created e.g. in Notepad or Excel and saved as a tab-delimited text file. If you name the file "setup.txt" `loadMAdata` will find it automatically. It is also possible to create the setup directly in R:

```
> # An example setup:
> oxygen <- c("aerobic","anaerobic")
> limitation <- c("Clim","Nlim")
> mySetup <- cbind(oxygen[c(1,1,1,2,2,2,1,1,1,2,2,2)],
+                  limitation[c(1,1,1,1,1,1,2,2,2,2,2,2)])
> # The rownames correspond to the CEL-file names (CAE1.CEL etc):
> rownames(mySetup) <- c("CAE1","CAE2","CAE3","CAN1","CAN2","CAN3",
+                        "NAE1","NAE2","NAE3","NAN1","NAN2","NAN3")
> colnames(mySetup) <- c("oxygen","limitation")
> # The final setup object can look like this:
> mySetup

     oxygen      limitation
CAE1 "aerobic"   "Clim"
CAE2 "aerobic"   "Clim"
CAE3 "aerobic"   "Clim"
CAN1 "anaerobic" "Clim"
CAN2 "anaerobic" "Clim"
CAN3 "anaerobic" "Clim"
NAE1 "aerobic"   "Nlim"
NAE2 "aerobic"   "Nlim"
NAE3 "aerobic"   "Nlim"
NAN1 "anaerobic" "Nlim"
```

```
NAN2 "anaerobic" "Nlim"
NAN3 "anaerobic" "Nlim"
```

No matter if you make the setup in R or in a tab delimited text file, the format should be as follows: The first column (the rownames if in R) should contain the names of the CEL files and additional columns should assign attributes in some category to each array.

Once you have your setup, either as a file or an object in R, make sure that you are in the directory of your CEL files, then run:

```
> # If you have the experimental setup in a file named setup.txt in the
> # working directory:
> myArrayData <- loadMAdata()
>
> # If you have the experimental setup in an object mySetup:
myArrayData <- loadMAdata(setup=mySetup)
```

The above commands are assuming that you are using yeast 2.0 CEL files, and in this case the correct annotation and filtering will be applied automatically. If you are using other arrays (or want to use a custom annotation) you can add the annotation separately. This is done through the argument `annotation`, pointed to either a `data.frame` or the file-name of a tab delimited text file containing the columns: Probeset ID (the rownames if you use a `data.frame`), Gene Name, Chromosome and Chromosome location. If only the Gene Names are available the two other columns can be set to arbitrary values (e.g. 1), but are still required. The Chromosome and Chromosome location info is only used in the polar plot (see below).

The `loadMAdata` function will normalize the raw data (by default using the `plier` algorithm). Further on, you can choose whether or not to filter the normalized data, through the `filter` argument. If you set this to `TRUE` only probesets that are included in the annotation will be kept for further analysis.

It is also possible to load already normalized data, either from a separate file or from a `data.frame` in R. The first column (or rownames if object in R) should be the probeset IDs and correspond to the ones used in the annotation. The column headers should match the names used in the setup. Note that if you are loading pre-normalized Yeast 2.0 data, you need to specify `platform="yeast2"` in order to get automatic annotation. In other cases just leave this argument to its default (`platform=NULL`) and supply separate optional annotation as described above.

For more details see the help page on `loadMAdata` (`> ?loadMAdata`). To have an example in this vignette we will load data included in the `piano` package, which is microarray data from yeast cultivated in aerobic and anaerobic conditions using either carbon limited or nitrogen limited media [2]. When we load this data we also use the `datadir` argument to specify the location of the data, since it is not in our working directory:

```
> # Get path to example data and setup files:
> dataPath <- system.file("extdata", package="piano")
> # Load pre-normalized data:
> myArrayData <- loadMAdata(datadir=dataPath,
+                           dataNorm="norm_data.txt.gz",
+                           platform="yeast2")
```

The `loadMAdata` function will return a list-like object of class `ArrayData`, in this case stored in `myArrayData`, containing the elements: `dataRaw`, `dataNorm`, `setup` and `annotation`:

```
> myArrayData
```

```
ArrayData object
  dataNorm:   data frame containing normalized expression values
              samples: 12
              genes: 5705
  setup:      data frame containing experimental setup
              samples: 12
              factor categories: 2
  annotation: data frame containing annotation information
              genes: 5705
```

You can examine these elements to see that they were correctly loaded:

```
> # Check the setup:
> myArrayData$setup
```

```
        oxygen limitation
CAE1   aerobic       Clim
CAE2   aerobic       Clim
CAE3   aerobic       Clim
CAN1 anaerobic       Clim
CAN2 anaerobic       Clim
CAN3 anaerobic       Clim
NAE1   aerobic       Nlim
NAE2   aerobic       Nlim
NAE3   aerobic       Nlim
NAN1 anaerobic       Nlim
NAN2 anaerobic       Nlim
NAN3 anaerobic       Nlim
```

```
> # Check the annotation (top 10 rows):
> myArrayData$annotation[1:10,]
```

```
              geneName chromosome    start
1769308_at     YKR009C         11  -454352
1769311_at     YDL157C          4  -174232
1769312_at     YER059W          5   272624
1769313_at     YGL059W          7   392223
1769314_at     YDR481C          4 -1418550
1769317_at     YML113W         13    44045
1769319_at     YIL157C          9   -46949
1769320_at     YPR003C         16  -561504
1769321_at     YNR074C         14  -777602
1769322_s_at YLR154C-H         12  -468827
```

Note that depending on the input `dataRaw` and/or `annotation` may be excluded.

3.2 Quality control

There are many quality control methods implemented in R. The `runQC` function collects a selection of these methods and lets you run them on your loaded and preprocessed data (Figure 1). The methods that are possible to run depends on if you started from raw data or not. To run all quality control methods possible for your data, simply type:

```
> runQC(myArrayData)
```

In this case a warning message will be given that `rnaDeg` and `nuseRle` could not be run due to that no raw data was available (since we started with pre-normalized data using `loadMAdata`). If you wish to run only selected quality control methods this can be specified by a true/false statement for each appropriate argument. For instance:

```
> # To only run the PCA:
> runQC(myArrayData, rnaDeg=FALSE, nuseRle=FALSE, hist=FALSE,
+       boxplot=FALSE, pca=TRUE)
> # Additionally, for the PCA you can specify other colors:
> runQC(myArrayData, rnaDeg=FALSE, nuseRle=FALSE, hist=FALSE,
+       boxplot=FALSE, pca=TRUE, colors=c("cyan","orange"))
```

The quality control may result in the decision to remove some bad arrays, in that case remove those and reload your data again with `loadMAdata`.

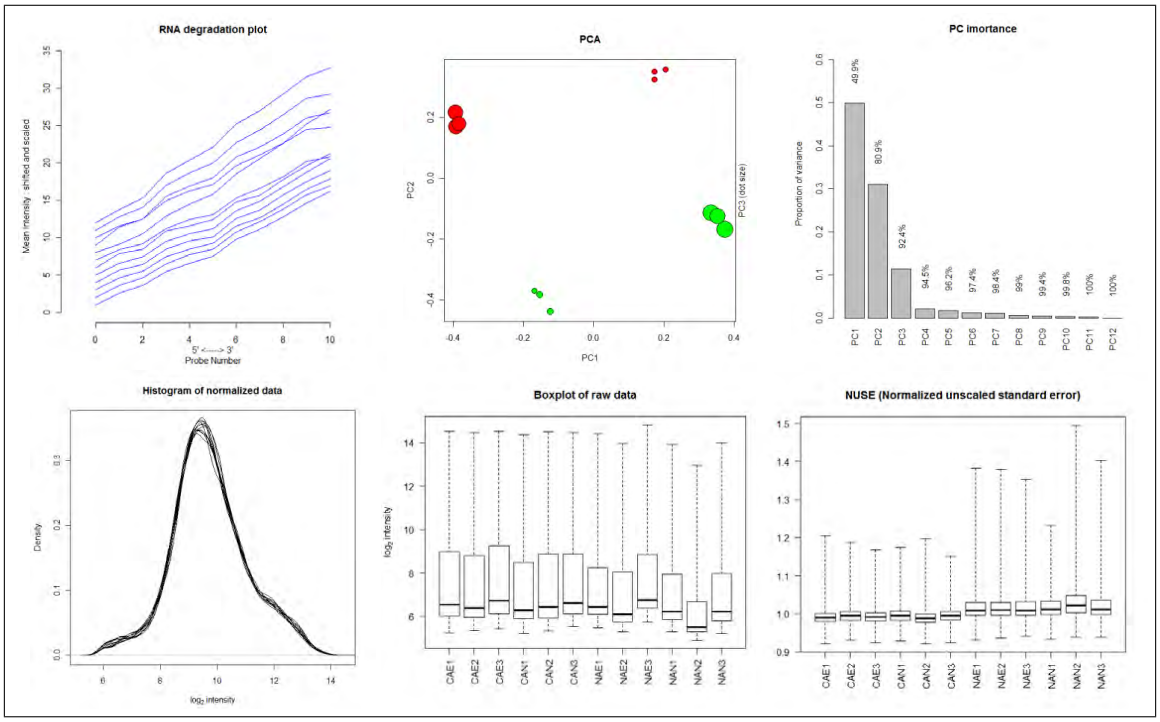

Figure 1: Example of figures that are produced by the `runQC` function.

3.3 Differential expression analysis

Once you have loaded and preprocessed the data and checked the quality, the next step is to check for differentially expressed genes. The `piano` package uses the linear models approach of the `limma` package [3]. Differential expression analysis can be carried out in many ways. Since the microarray analysis of `piano` aims to be simple and straight-forward, the options are limited to the most common approach. Here, you are required to define two conditions that are to be compared, and genes that differ in expression between these conditions will be detected. In order to see the conditions that are assigned to each microarray, use the `extractFactors` function:

```
> extractFactors(myArrayData)

$factors
            factors
CAE1   aerobic_Clim
CAE2   aerobic_Clim
CAE3   aerobic_Clim
CAN1 anaerobic_Clim
CAN2 anaerobic_Clim
CAN3 anaerobic_Clim
NAE1   aerobic_Nlim
NAE2   aerobic_Nlim
NAE3   aerobic_Nlim
NAN1 anaerobic_Nlim
NAN2 anaerobic_Nlim
NAN3 anaerobic_Nlim

$uniqueFactors
[1] "aerobic_Clim"   "anaerobic_Clim" "aerobic_Nlim"   "anaerobic_Nlim"
```

The conditions is simply a merge of the columns of the setup, if these get to specific, try to reduce the number of columns in the setup. In our case we have four unique conditions. As an example, we choose to compare yeast cultured in aerobic conditions to yeast in anaerobic conditions, for carbon limited and nitrogen limited media, separately. To do so, we will use the `diffExp` function:

```
> pfc <- diffExp(myArrayData, contrasts=c("aerobic_Clim - anaerobic_Clim",
+                                         "aerobic_Nlim - anaerobic_Nlim"))
```

Note that the compared conditions, called contrasts, are defined in a character vector, where each condition is separated from the condition it will be compared to, with a " - " (do not forget the spaces). The names of the conditions should match those returned by `extractFactors`. The `diffExp` function returns a list (stored in `pfc` in the example above) with the elements `pValues`, `foldChanges` and `resTable`, containing the statistics returned by `topTable` (in `limma`). By default the p-values are adjusted for multiple testing using FDR. To view e.g. the top five significant genes for each contrast:

```
> # Sort genes in "aerobic_Clim - anaerobic_Clim" according to adjusted p-value:
> ii <- sort(pfc$resTable[[1]]$adj.P.Val, index.return=TRUE)$ix
> pfc$resTable[[1]][ii[1:5],]
```

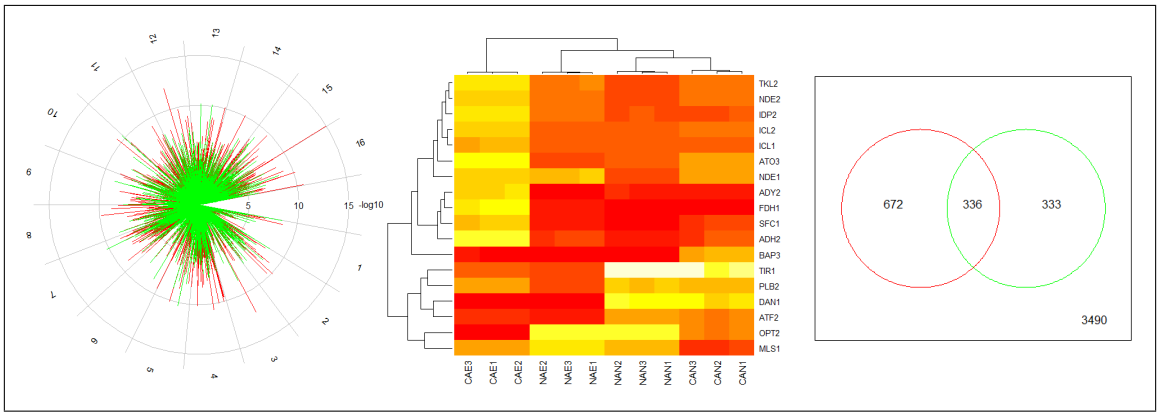

Figure 2: Example of figures that are produced by the `diffExp` function.

```
     ProbesetID GeneName     logFC   AveExpr         t      P.Value
2492 1774122_at   YOR388C  6.129208  8.014575  80.15804 4.527221e-19
655  1770580_at   YCR010C  4.859860  8.166250  46.06582 6.409365e-16
5398 1779660_at   YLR174W  3.661590  9.559445  45.50038 7.531924e-16
2233 1773640_at   YDR046C -4.193094  7.568842 -33.47018 4.125785e-14
5090 1779075_at   YPR194C -3.291721 10.668819 -34.04934 3.300794e-14
        adj.P.Val        B
2492 2.582780e-15 31.91323
655  1.432321e-12 26.56316
5398 1.432321e-12 26.42497
2233 4.707521e-11 22.80293
5090 4.707521e-11 23.01335

> # Sort genes in "aerobic_Nlim - anaerobic_Nlim" according to adjusted p-value:
> ii <- sort(pfc$resTable[[2]]$adj.P.Val, index.return=TRUE)$ix
> pfc$resTable[[2]][ii[1:5],]

     ProbesetID GeneName     logFC   AveExpr         t      P.Value
914  1771091_at   YJR150C -6.530596  9.474928 -34.05358 3.295453e-14
3318 1775717_at   YLR413W -4.154929  9.856378 -36.62184 1.279124e-14
4952 1778804_at   YER011W -6.305222 11.226470 -33.89155 3.506346e-14
965  1771190_at   YGR177C -3.399908  8.735167 -32.27216 6.625773e-14
2556 1774236_at   YMR006C -3.106606 10.280228 -31.05631 1.090939e-13
        adj.P.Val        B
914  6.667901e-11 22.62262
3318 6.667901e-11 23.41626
4952 6.667901e-11 22.56973
965  9.450008e-11 22.02138
2556 1.244761e-10 21.58456
```

The `diffExp` function also produces some plots (Figure 2): a Venn diagram, a polar plot, a heatmap and a volcano plot. The Venn diagram shows the number of differentially expressed genes for each contrast (by default at a significance level of 0.001, see argument `significance`). Which genes the number represent can be accessed through the returned object, in this example

`pfc$vennMembers`. The polar plot gives a visualization of differential expression by plotting the -log10(p-values) against chromosomal location of each gene. The heatmap shows the expression levels of significant genes (in at least one contrast, significance cutoff controlled by the argument `heatmapCutoff`) for all microarrays and clusters similar expression patterns together. The volcano plot shows the magnitude of change versus the significance. Each of these plots can be turned on/off through the argument `plot`.

The gene-level statistics, returned by `diffExp`, can be used to select genes of interest for further investigation but also passed on to additional global analyses. One common analysis in this context is gene set analysis which is the main part of `piano` and is presented in the next section.

4 Gene set analysis

This section will give an overview of how to run gene set analysis (GSA) using `piano`. To not get too technical, some details are left out here, but described more in detail in the documentation of each function (e.g. `> ?runGSA`) as well as in the `piano` publication and its supplementary material[1]. Please use these sources if interested in deeper details.

This section is organized in the following way: first a short introduction to GSA is given (4.1), followed by a description of input data (4.2) and available GSA methods (4.3). Next the main function `runGSA` is described (4.4). After this, some examples are presented to showcase what is possible to do (4.5). Finally, the last part describes how different GSA runs (using different settings) can be combined into a consensus result (4.6).

4.1 Introduction to gene set analysis

GSA is collective name of statistical methods that analyze the significance of genes, in terms of gene sets. To give a common example, genes can be grouped by GO terms so that a statistic for each GO term (gene sets in this case) can be calculated, representing the enrichment of significant genes within each GO term. One advantage with GSA is that the biological interpretation of gene-level data can be easier to assess using groups of genes that share some common property, rather than considering each single gene separately. Simply put, the GSA methods take gene-level statistics as input (i.e. a statistical value for each gene) and calculates gene set statistics for each defined group of genes (gene sets). The gene-level statistics are commonly taken from differential expression analysis of transcription data, but can in principle also be based on genome-wide association studies, proteomics or metabolomics data. For the examples in this vignette we will focus on gene-level statistics from gene expression studies.

4.2 GSA input data

To run GSA, two inputs are needed:

- Gene-level statistics
- Gene set collection (GSC)

The gene-level statistics are simply represented by a one-columned dataframe (or a named vector) of numeric values (usually p-values or t-values from a differential expression analysis) with some kind of gene IDs as rownames. Just to see an example of this we can use the gene-level p-values of one of the contrasts from the microarray analysis (described in the previous section):

```
> # Get p-values from the aerobic_Clim vs anaerobic_Clim comparison:
> myPval <- pfc$pValues["aerobic_Clim - anaerobic_Clim"]
> # Display the first values and gene IDs:
> head(myPval)
```

```
           aerobic_Clim - anaerobic_Clim
1769308_at                   2.662353e-08
1769311_at                   3.825827e-01
1769312_at                   2.300665e-01
1769313_at                   3.335551e-03
1769314_at                   7.228196e-05
1769317_at                   4.930221e-02
```

Apart from gene-level statistics, an optional but common input is `directions`. This simply gives the direction of expression change of each gene (e.g. fold-changes). This is used to separate up-regulated and down-regulated genes e.g. when p-values are used as gene-level statistics. If t-values are used as input, there is no need for `directions`, since this is given by the sign of each t-value.

The gene set collection should describe the grouping of genes into gene sets. This can be defined in many ways and is handled by the `loadGSC` function. This function loads a gene set collection into the correct format required by the `runGSA` function. A simple dummy example:

```
> # Custom gene to gene set mapping:
> genes2genesets <- cbind(paste("gene",c("A","A","A","B","B","C","C","C","D"),sep=""),
+                         paste("set",c(1,2,3,1,3,2,3,4,4),sep=""))
> genes2genesets

      [,1]    [,2]
 [1,] "geneA" "set1"
 [2,] "geneA" "set2"
 [3,] "geneA" "set3"
 [4,] "geneB" "set1"
 [5,] "geneB" "set3"
 [6,] "geneC" "set2"
 [7,] "geneC" "set3"
 [8,] "geneC" "set4"
 [9,] "geneD" "set4"

> # Load into correct format:
> myGsc <- loadGSC(genes2genesets)
> # View summary:
> myGsc

First 4 (out of 4) gene set names:
[1] "set1" "set2" "set3" "set4"

First 4 (out of 4) gene names:
[1] "geneA" "geneB" "geneC" "geneD"

Gene set size summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    2.00    2.00    2.25    2.25    3.00

No additional info available.

> # View all gene sets:
> myGsc$gsc

$set1
[1] "geneA" "geneB"

$set2
```

```
[1] "geneA" "geneC"

$set3
[1] "geneA" "geneB" "geneC"

$set4
[1] "geneC" "geneD"
```

After loading a gene set collection using `loadGSC` it is always recommended to look at the output (in this case `myGsc`) so that the given information was correctly interpreted, i.e. that the gene sets and genes were not swapped and that the distribution of gene set sizes (number of member genes) seem valid. Using a two-column mapping of all gene to gene set associations, as in the above example, is a simple way to load custom gene set collections, however some general formats are also recognized by `loadGSC`: gmt, sbml and sif (see `loadGSC` documentation for more info). Note that the gene names in the gene set collection have to match the gene names used for the gene-level statistics, i.e. an appropriate gene-level statistics format for the above example would be:

```
> myStats <- c(-1.5,-0.5,1,2)
> names(myStats) <- paste("gene",c("A","B","C","D"),sep="")
> myStats

geneA geneB geneC geneD
 -1.5  -0.5   1.0   2.0
```

Note that although the gene names have to match between the gene-level statistics and the gene set collection, not all genes have to be represented in both. Only genes that have a gene-level statistic and belong to at least one gene set are used in the GSA. (Detailed note: Actually, all given gene-level statistics are used for permutation-based significance estimation, regardless if they belong to a gene set.)

4.3 Available GSA methods

Once you have gene-level statistics and have loaded a gene set collection using `loadGSC` you can perform a GSA using `runGSA`. This function collects a number of GSA methods into the same platform, meaning that it is easy to test different methods using the same settings, format and input. The available GSA methods are:

- Fisher's combined probability test [4]
- Stouffer's method [5]
- Reporter features [6, 7]
- Parametric analysis of gene set enrichment (PAGE, also implemented in Bioconductor package `PGSEA`) [8]
- Tail strength (also implemented in R package `samr`) [9]
- Wilcoxon rank-sum test (also used by `limma`)

- Gene set enrichment analysis (GSEA, R script also available at www.broadinstitute.org/gsea) [10, 11]
- Mean (also used by `limma` and R package `GSA`)
- Median
- Sum
- Maxmean (also implemented in `GSA`) [12]

For details on these methods, see corresponding references, R packages and the `piano` publication [1] (in particular the supplementary methods). The following GSA methods can only accept p-values as input: `fisher`, `stouffer`, `reporter` and `tailStrength`. The following methods can only accept t-values (or similarly formatted statistic): `maxmean`, `gsea` and `page`.

Each GSA method will calculate a statistic for each gene set in the gene set collection. The significance (gene set p-values) of these gene set statistics can generally be calculated in three ways: from a theoretical null distribution, by randomizing the gene labels or by randomizing the sample labels. For `fisher`, `stouffer`, `reporter`, `wilcoxon` and `page`, the gene set p-values can be calculated from a theoretical null-distribution. For all methods, gene sampling or sample permutation can be used.

4.4 The `runGSA` function

The main function of `piano` is `runGSA`. This section will go into how to use it and its different arguments and settings, and describe its output (see also the separate `runGSA` help documentation). Additional down-stream functions will be described as they are used in the examples in section 4.5.

4.4.1 A simple run

The `runGSA` function has many arguments, but a minimum requirement is only the arguments `geneLevelStats` and `gsc`. Using the gene set collection and gene-level statistics created in the dummy example in section 4.2, a minimal-input run would be executed like this:

```
> gsaRes <- runGSA(myStats, gsc=myGsc)
```

The gene names in `myStats` have to match the gene names used in `myGsc` and `myGsc` has to be an object of class `GSC`, as returned by `loadGSC`.

4.4.2 The output of `runGSA`

Let's start with looking at the output of `runGSA`. First, to get a summary of the GSA run and results stored in `gsaRes` (assuming you ran the previous line of code), type :

```
> gsaRes
```

```
Final gene/gene-set association: 4 genes and 4 gene-sets
  Details:
  Calculating gene set statistics from 4 out of 4 gene-level statistics
  Using all 4 gene-level statistics for significance estimation
```

```
  Removed 0 genes from GSC due to lack of matching gene statistics
  Removed 0 gene sets containing no genes after gene removal
  Removed additionally 0 gene sets not matching the size limits
  Loaded additional information for 0 gene sets

Gene statistic type: t-like
Method: mean
Gene-set statistic name: mean
Significance: Gene sampling
Adjustment: fdr
Gene set size limit: (1,Inf)
Permutations: 10000
Total run time: 0.03 min
```

Here we get details of the number of genes and gene sets that were used in the analysis, and a summary of the settings that were used as well as the total runtime. The full output structure is a list with the following elements (described in the `runGSA` help text):

```
> names(gsaRes)

 [1] "geneStatType"       "geneSetStat"        "signifMethod"
 [4] "adjMethod"          "info"               "gsSizeLim"
 [7] "gsStatName"         "nPerm"              "gseaParam"
[10] "geneLevelStats"     "directions"         "gsc"
[13] "nGenesTot"          "nGenesUp"           "nGenesDn"
[16] "statDistinctDir"    "statDistinctDirUp"  "statDistinctDirDn"
[19] "statNonDirectional" "statMixedDirUp"     "statMixedDirDn"
[22] "pDistinctDirUp"     "pDistinctDirDn"     "pNonDirectional"
[25] "pMixedDirUp"        "pMixedDirDn"        "pAdjDistinctDirUp"
[28] "pAdjDistinctDirDn"  "pAdjNonDirectional" "pAdjMixedDirUp"
[31] "pAdjMixedDirDn"     "runtime"
```

This list contains both the input and settings, as well as the result, in principle everything associated with the GSA run. We will first focus on the main result statistics. These can be accessed by using the `GSAsummaryTable` function. This is essentially a table of the number of genes in each gene set, the gene set statistics and their p-values (normal and adjusted). What you will notice is that there are three classes: distinct-directional, mixed-directional and non-directional.

These directionality classes aim to describe different aspects of direction of change of gene expression on the gene set level. The non-directional class disregards the direction of change and simply uses the absolute values of the gene-level statistics to calculate the gene set statistics and the associated p-values. The distinct-directional class takes direction of change into account, so that gene sets with genes both significantly up-regulated and significantly down-regulated will cancel out. Finally, the mixed-directional class considers the up-regulated subset and the down-regulated subset of a gene set separately. Each subset is scored according to the proportion of significant genes. Let's use an example to understand this.

Imagine a small gene set with four genes (all significant) and the following gene-level statistics:

```
[1] -4.0 -3.0  2.0  3.5
```

Now, the gene set statistic of the distinct-directional class would simply be (if using the mean method):

```
> mean(c(-4, -3, 2.5, 4.5))

[1] 0
```

We immediately see that the genes in the gene set do not change in a distinct direction, they show equal parts of up- and down-regulation. That is, the gene set as a whole does not show significant patterns of either distinct down- or up-regulation. However, disregarding the direction of change, using the non-directional class, the gene set suddenly appears to be highly significant:

```
> mean(abs(c(-4, -3, 2.5, 4.5)))

[1] 3.5
```

This is because all genes are displaying changes in expression levels between the compared conditions (albeit in different directions). Looking at the subsets of up- and down-regulated genes on their own also reveals that the gene set has significant parts of both up- and down regulation:

```
> mean(abs(c(-4, -3)))

[1] 3.5

> mean(abs(c(2.5, 4.5)))

[1] 3.5
```

By calculating and reporting the gene set statistics for all three directionality classes, we get a more detailed picture of each gene set.

This is just a dummy example with four genes, but in a real scenario we may end up with a gene set with the following p-values:

```
p (non.dir) p (dist.dir.up) p (dist.dir.dn)  p (mix.dir.up)  p (mix.dir.dn)
     0.0001          0.0001          0.9999          0.0001          0.0050
```

Here we see, due to the low non-directional p-value, that the gene set in general is enriched with significant genes (disregardful of direction of regulation), that the genes tend to be coordinately (or distinctly) up-regulated but that also the portion of down-regulated genes are on their own mostly significant. This is probably due to that the gene set contains many (e.g. 95) genes that are up-regulated and only a few (e.g. 5) that are down regulated.

```
Genes (tot)   Genes (up) Genes (down)
        100           95            5
```

These 5 genes can still be highly significant although the gene set as a whole shows distinct patterns of up-regulation. Again, taking all these values in to account, rather than basing our conclusions on only one of the p-values, we get a better picture of in what way the gene set is significant. The concept is outlined in Figure 3 and discussed more in detail in the `piano` publication [1].

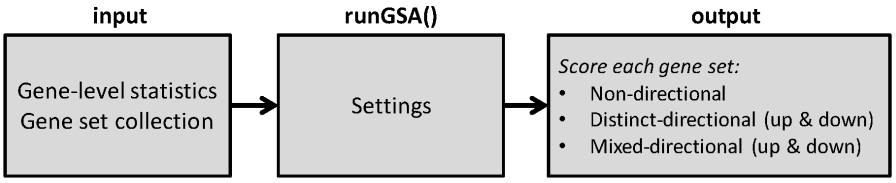

Figure 3: Conceptual overview of the `runGSA` workflow.

4.4.3 Adjusting settings

Now that we have an idea of the output (most importantly, p-values for each gene set in each directionality class), we will focus on the different settings. The default GSA method (i.e. the method of calculating the gene set statistics from the gene-level statistics) is the mean of the gene-level statistics. Which method to use is controlled by the `geneSetStat` argument. Be aware of that the following GSA methods can only accept p-values as input: `fisher`, `stouffer`, `reporter` and `tailStrength`, and that the following methods can only accept t-values (or similarly formatted statistic): `maxmean`, `gsea` and `page`.

Next, the gene set significance calculation method can be set using the `signifMethod`. The default is gene sampling, meaning that the gene-level statistics are randomly grouped into fake gene sets to generate a background distribution of gene set statistics. The number of time the genes are randomized is controlled by `nPerm`. The original gene set statistics are compared to this background in order to infer the gene set p-values. In a similar manner the sample labels can be randomized (sample permutation), but this has to be done by the user separately (see the `runGSA` help text for more info). For the GSA methods `fisher`, `stouffer`, `reporter`, `wilcoxon` and `page`, the gene set p-values can also be calculated from a theoretical null-distribution.

The gene set p-values are by default adjusted for multiple testing using FDR. Other methods can be applied through the `adjMethod` argument.

The `gsSizeLim` argument can be used to control the smallest and largest gene set sizes allowed in the analysis. For instance, you may wish to remove gene sets with fewer than five genes (for which you have gene-level statistics for) from the analysis.

The GSEA method is one of the most popular GSA methods and there is dedicated software for this (www.broadinstitute.org/gsea). In the statistical formulation there is a user-controlled parameter that defaults to 1. This can be set through the `gseaParam` argument. For more info on this see [11].

4.5 GSA examples

The applications of GSA using `piano` can be presented using a few examples. Here we will also introduce the down-stream functions `geneSetSummary`, `GSAsummaryTable` and `networkPlot`.

4.5.1 Example 1: A yeast microarray data set and GO terms

As an example of the seamless integration of the microarray analysis part of `piano` and the GSA part, we will start with an example using the data from section 3. At that point we ended up with an object, `pfc`, containing p-values and fold-changes from the differential expression analysis (using the `diffExp` function):

```
> # Get the p-values for the contrast aerobic_Clim - anaerobic_Clim
> myPval <- pfc$pValues["aerobic_Clim - anaerobic_Clim"]
```

```
> head(myPval)

           aerobic_Clim - anaerobic_Clim
1769308_at                   2.662353e-08
1769311_at                   3.825827e-01
1769312_at                   2.300665e-01
1769313_at                   3.335551e-03
1769314_at                   7.228196e-05
1769317_at                   4.930221e-02

> # Get the fold-changes for the contrast aerobic_Clim - anaerobic_Clim
> myFC <- pfc$foldChanges["aerobic_Clim - anaerobic_Clim"]
> head(myFC)

           aerobic_Clim - anaerobic_Clim
1769308_at                     1.8939411
1769311_at                    -0.1157152
1769312_at                     0.1319669
1769313_at                     0.3808714
1769314_at                     0.5884535
1769317_at                    -0.3538557
```

These values are for differentially expressed genes in yeast cultured in aerobic carbon-limited media compared to anaerobic carbon-limited media. As we see for the five first genes, the IDs used are the microarray probeset IDs. This is important to keep in mind when we select or gene set collection. In this example we will use GO terms as gene sets. The `biomaRt` package is useful to access available databases:

```
> library("biomaRt")
> # Select ensembl database and S. cerevisiae dataset:
> ensembl <- useMart("ENSEMBL_MART_ENSEMBL", dataset="scerevisiae_gene_ensembl",host="www.en
> # Map Yeast 2.0 microarray probeset IDs to GO:
> mapGO <- getBM(attributes=c('affy_yeast_2', 'name_1006'),
+                filters = 'affy_yeast_2',
+                values = rownames(myPval),
+                mart = ensembl)
> # Remove blanks ("")
> mapGO <- mapGO[mapGO[,2]!="",]
> # Check the 10 first rows to see what we got:
> mapGO[1:10,]

   affy_yeast_2                     name_1006
1  1769803_s_at             metal ion binding
2  1769803_s_at                       cytosol
3  1769803_s_at          antioxidant activity
4  1769803_s_at  superoxide dismutase activity
5  1769803_s_at removal of superoxide radicals
6  1769803_s_at             copper ion binding
7  1769803_s_at            cadmium ion binding
```

```
8  1769803_s_at   detoxification of copper ion
9  1769803_s_at  detoxification of cadmium ion
10 1769803_s_at         response to copper ion
```

Now that we have a mapping between our probeset IDs and GO terms we can load it into the correct format using `loadGSC`:

```
> myGsc <- loadGSC(mapGO)
```

We have all our input data and can move on with the `runGSA` function:

```
> gsaRes <- runGSA(myPval,myFC,gsc=myGsc,gsSizeLim=c(5,300))
```

```
Checking arguments...done!
Calculating gene set statistics...done!
Calculating gene set significance...done!
Adjusting for multiple testing...done!
```

```
> gsaRes
```

```
Final gene/gene-set association: 4257 genes and 981 gene sets
  Details:
  Calculating gene set statistics from 4257 out of 5705 gene-level statistics
  Using all 5705 gene-level statistics for significance estimation
  Removed 0 genes from GSC due to lack of matching gene statistics
  Removed 0 gene sets containing no genes after gene removal
  Removed additionally 2783 gene sets not matching the size limits
  Loaded additional information for 0 gene sets

Gene statistic type: p-like
Method: mean
Gene-set statistic name: mean
Significance: Gene sampling
Adjustment: fdr
Gene set size limit: (5,300)
Permutations: 10000
Total run time: 4.75 min
```

Here we limit the analysis to gene sets with at least 5 genes and at most 300 genes. We can save the results to a file (e.g. for editing in Microsoft Excel):

```
> GSAsummaryTable(gsaRes, save=TRUE, file="gsaResTab.xls")
```

To get info on a specific gene set:

```
> gs <- geneSetSummary(gsaRes, "oxidoreductase activity")
> gs$stats
```

```
                  Name     Value
1          Genes (tot) 243.00000
2   Stat (dist.dir.up)   0.47973
```

```
3      p (dist.dir.up)   0.03510
4  p adj (dist.dir.up)   0.40038
5   Stat (dist.dir.dn)   0.52027
6      p (dist.dir.dn)   0.96490
7  p adj (dist.dir.dn)   1.00000
8       Stat (non-dir)   0.15741
9          p (non-dir)   0.00000
10     p adj (non-dir)   0.00000
11          Genes (up) 127.00000
12   Stat (mix.dir.up)   0.15511
13      p (mix.dir.up)   0.00130
14  p adj (mix.dir.up)   0.10140
15          Genes (dn) 116.00000
16   Stat (mix.dir.dn)   0.15992
17      p (mix.dir.dn)   0.00250
18  p adj (mix.dir.dn)   0.10083
```

For this particular gene set we can see that it is enriched by significant genes (p-value in the non-directional class is low). However, the gene set as a whole is not regulated in a distinct direction (p-values in the distinct-directional class are not significant), although we see that both the subsets of up-regulated and down-regulated genes are slightly significant (mixed-directional class). This makes sense since roughly half of the genes are up-regulated and the other half are down-regulated (127 and 116 genes, respectively).

If we want to visualize the results we can use the function `networkPlot` (Figure 4):

```
> nw <- networkPlot(gsaRes,class="non")
```

Here we select the gene sets that are significant ($p<0.001$) in the non-directional class (i.e. GO terms that are enriched with significant genes, but may contain a mix of up- and down-regulated genes). The network plot shows the relation between gene sets by connecting those that share member genes. The thickness of the edges correspond to the number of shared genes. The size of the nodes correspond to the size of the gene sets. We can see that the GO term we checked in detail above (oxidoreductase activity) is included in the plot and that it is highly overlapping with the GO term oxidation-reduction process. We can access the full names of all gene sets included in the plot through the returned object:

```
> nw$geneSets
```

4.5.2 Example 2: Reporter metabolites

An alternative to using GO terms as gene sets is to use so called Reporter Metabolites [6]. Here, a genome-scale metabolic network reconstruction is used to group genes into sets so that genes that belong to the same gene set are coding for enzymes that catalyze reactions where a specific metabolite takes part. In other words, each gene set is named after a specific metabolite and contains the enzyme coding genes that catalyze reactions in which the metabolite takes part. Enriched, or significant, gene sets (Reporter Metabolites) can be interpreted as metabolic hotspots, or metabolites around which important transcriptional changes occur. We can use the `loadGSC` function to load a gene set collection from a genome-scale metabolic model:

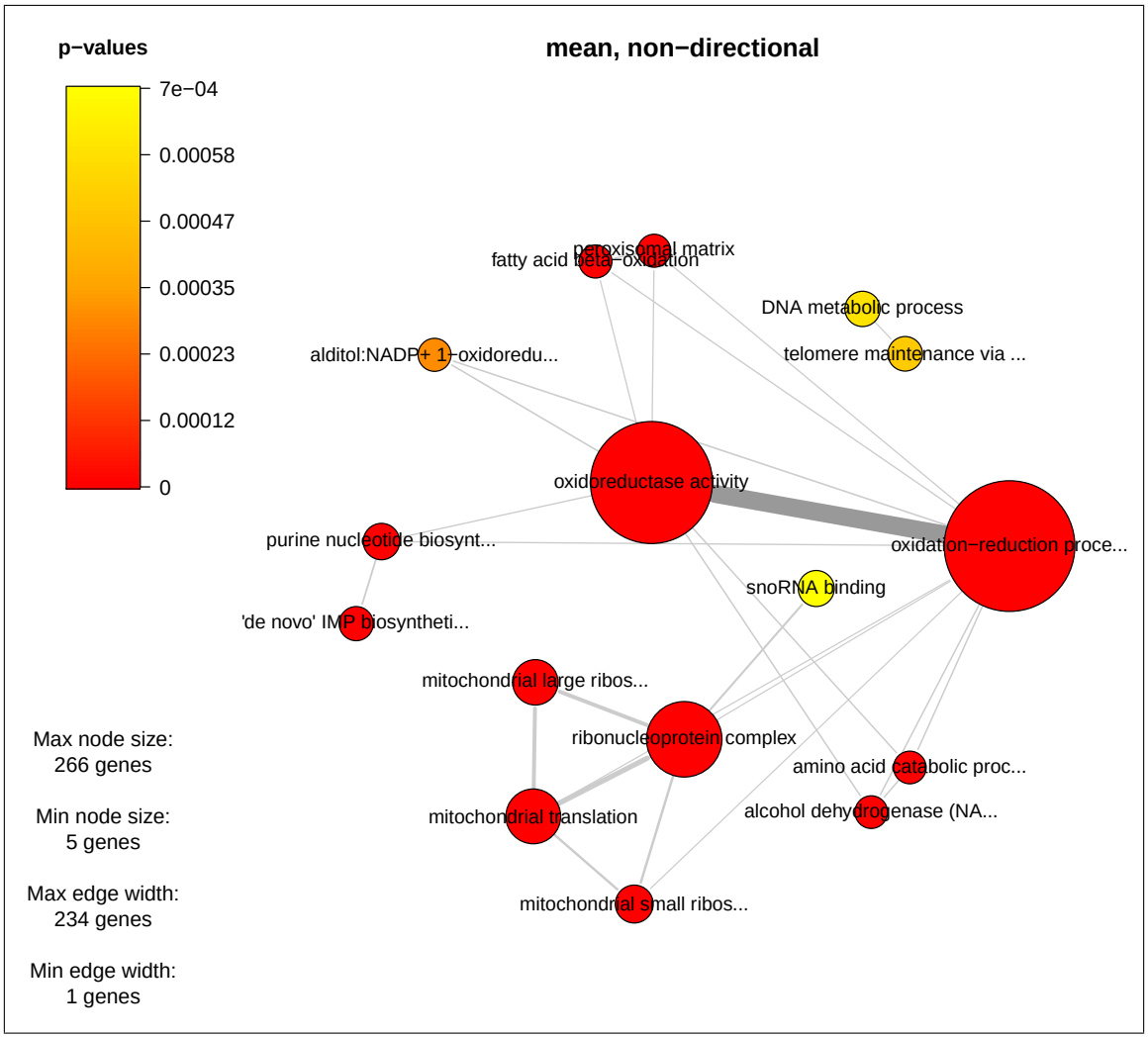

Figure 4: The figure produced by `networkPlot`.

```
> # An example usage:
> myGsc <- loadGSC("myModel.xml")
```

In order to save time and space (since the model file size can be large and the extraction of gene sets can take a while), we have already loaded the gene to gene set associations from two *S. cerevisiae* metabolic networks [13, 14] and included them in `piano`. Let's load:

```
> # To load iTO977:
> metMap <- system.file("extdata", "probe2metabolites_iTO977.txt.gz",
+                       package="piano")
> # To load iIN800:
> metMap <- system.file("extdata", "probe2metabolites_iIN800.txt.gz",
+                       package="piano")
> # Convert into piano format:
> metMap <- read.delim(metMap)
> myGsc <- loadGSC(metMap)
```

Again, we will use the gene-level statistics (p-values and fold-changes) from section 3:

```
> gsaRes <- runGSA(myPval,myFC,gsc=myGsc,
+                  geneSetStat="reporter",
+                  signifMethod="nullDist", nPerm=1000,
+                  gsSizeLim=c(5,100))

Checking arguments...done!
Calculating gene set statistics...done!
Calculating gene set significance...done!
Adjusting for multiple testing...done!

> gsaRes

Final gene/gene-set association: 643 genes and 163 gene-sets
  Details:
  Calculating gene set statistics from 643 out of 5705 gene-level statistics
  Removed 0 genes from GSC due to lack of matching gene statistics
  Removed 0 gene sets containing no genes after gene removal
  Removed additionally 656 gene sets not matching the size limits
  Loaded additional information for 0 gene sets

Gene statistic type: p-like
Method: reporter
Gene-set statistic name: Z
Significance: Theoretical null distribution
Adjustment: fdr
Gene set size limit: (5,100)
Permutations: 1000
Total run time: 0.1 min
```

Here we decrease the number of permutations to 1000 and only include gene sets that contain between 5 and 100 genes (this is only to make the example run faster). As you can see, we choose the gene set statistics to be calculated according to the Reporter features algorithm (`geneSetStat="reporter"`). This is the original algorithm for calculating Reporter Metabolites [6] (but of course other GSA methods can be used as well).

Now, we will use the network plot again, but with some different settings than those in the previous example:

```
> nw <- networkPlot(gsaRes, class="distinct", direction="both",
+                   significance=0.005, label="numbers")
```

Here we plot the gene sets that are significant ($p<0.005$) in either the up-regulated or down-regulated distinct-directional class. Instead of using the gene set names (metabolites in this case) as node labels, we use numbers instead, in order to make it less messy (Figure 5). In order to map the numbers to the names we can type:

```
> nw$geneSets
```

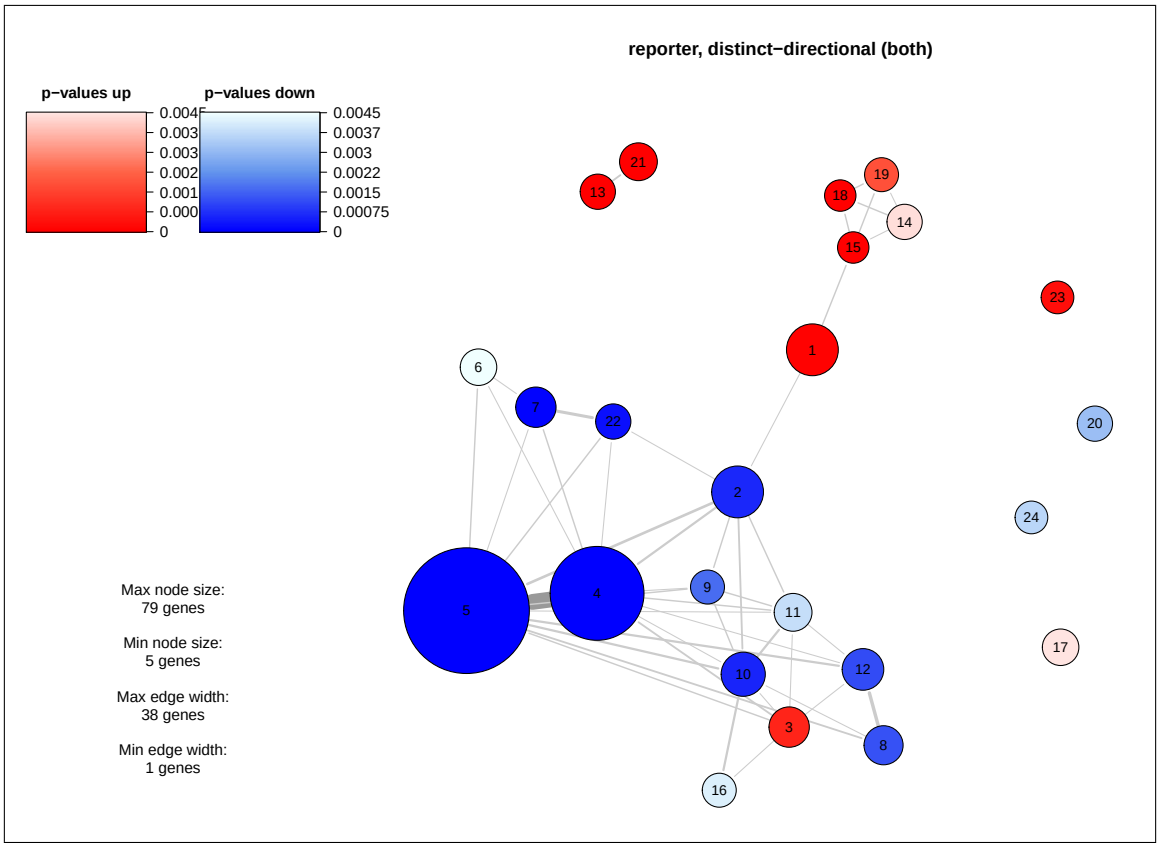

Figure 5: The figure produced by `networkPlot`.

```
                     1                      2
"2-oxoglutarate[c]"          "L-glutamine[c]"
                     3                      4
         "acetate[c]"              "AMP[c]"
                     5                      6
       "diphosphate[c]"              "GTP[c]"
                     7                      8
             "IMP[c]"              "AMP[m]"
                     9                     10
       "L-histidine[c]"       "L-methionine[c]"
                    11                     12
       "diphosphate[m]"   "hydrogen peroxide[c]"
                    13                     14
          "malate[c]"        "isocitrate[c]"
                    15                     16
    "homocysteine[c]"  "glycerone phosphate[c]"
                    17                     18
        "glyoxylate[c]"         "succinate[c]"
                    19                     20
"dolichyl phosphate[c]"       "glutathione[c]"
                    21                     22
```

```
"xanthosine 5-phosphate[c]"  "6-phospho-D-gluconate[c]"
                          23
 "3-phospho-D-glycerate[c]"
```

If we wish to get a list of all gene sets that are significant above some cutoff we can use the `GSAsummaryTable` function:

```
> gsaResTab <- GSAsummaryTable(gsaRes)
> # Which columns contain p-values:
> grep("p \\(",colnames(gsaResTab),value=T)

[1] "p (dist.dir.up)" "p (dist.dir.dn)" "p (non-dir.)"    "p (mix.dir.up)"
[5] "p (mix.dir.dn)"

> grep("p \\(",colnames(gsaResTab))

[1]  4  7 10 14 18
```

We see that the p-values in the non-directional class (significant gene sets if enriched by significant genes, disregardful of up- or down-regulation) are found in column 10. To get a list of gene sets with non-directional p<0.0001:

```
> ii <- which(gsaResTab[,10]<0.0001)
> gsaResTab$Name[ii]

 [1] "L-glutamate[c]"     "L-tyrosine[c]"      "CO2[c]"
 [4] "ethanol[c]"         "NAD(+)[c]"          "NADH[c]"
 [7] "H(+)[e]"            "L-glutamate[e]"     "L-tyrosine[e]"
[10] "L-valine[e]"        "L-valine[c]"        "L-serine[e]"
[13] "L-alanine[e]"       "L-alanine[c]"       "L-histidine[c]"
[16] "L-isoleucine[e]"    "L-isoleucine[c]"    "L-leucine[e]"
[19] "L-leucine[c]"       "L-methionine[e]"    "L-phenylalanine[e]"
[22] "L-phenylalanine[c]" "L-cysteine[e]"      "L-cysteine[c]"
[25] "L-tryptophan[e]"    "isocitrate[c]"      "glyoxylate[c]"
[28] "succinate[c]"
```

To get a list of gene sets that are significant (p<0.0001) in at least one directionality class:

```
> # Get minimum p-value for each gene set:
> minPval <- apply(gsaResTab[,c(4,7,10,14,18)],1,min,na.rm=TRUE)
> # Select significant gene sets:
> ii <- which(minPval<0.0001)
> gsaResTabSign <- gsaResTab[ii,c(1,4,7,10,14,18)]
> # Look at the first 10 gene sets:
> gsaResTabSign[1:10,]

                Name p (dist.dir.up) p (dist.dir.dn) p (non-dir.)
13    L-glutamate[c]      2.9376e-01      7.0624e-01   7.6441e-06
14 2-oxoglutarate[c]      9.9368e-05      9.9990e-01   1.9537e-04
15     L-tyrosine[c]      8.0723e-01      1.9277e-01   1.2151e-06
```

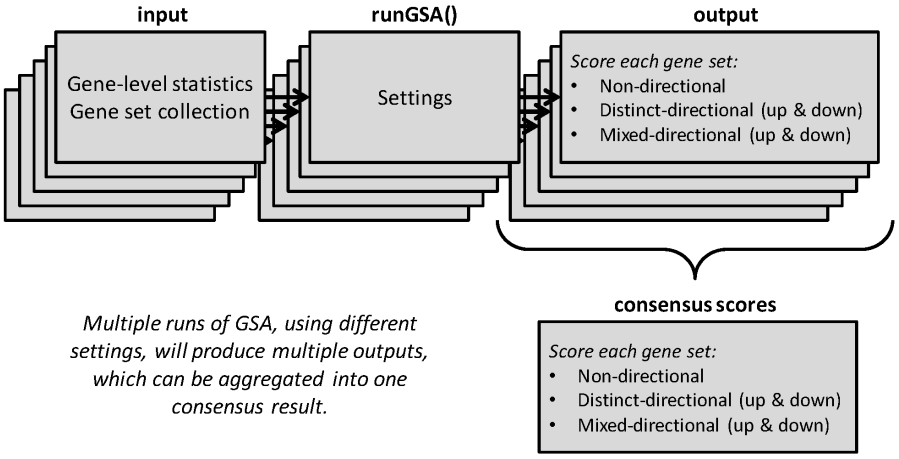

Figure 6: Multiple results from different GSA runs can be combined using the `consensusScores` function.

```
21            CO2[c]      4.5038e-01      5.4952e-01   8.5152e-05
30            AMP[c]      1.0000e+00      9.8569e-07   9.3764e-04
31    diphosphate[c]      1.0000e+00      3.8578e-08   4.0351e-04
33            IMP[c]      9.9995e-01      5.1924e-05   1.3912e-03
42        ethanol[c]      5.1688e-01      4.8312e-01   5.5234e-05
43         NAD(+)[c]      8.6732e-02      9.1323e-01   1.3441e-09
45           NADH[c]      2.9593e-01      7.0399e-01   2.0384e-10
   p (mix.dir.up) p (mix.dir.dn)
13     4.0324e-04     3.4036e-03
14     8.4100e-04     1.5952e-01
15     2.0644e-03     2.9135e-05
21     1.2919e-04     3.4741e-02
30     3.0381e-01     7.1731e-05
31     6.4533e-01     1.5272e-06
33     2.6302e-01     5.3269e-04
42     1.2139e-03     8.4627e-03
43     1.2838e-07     3.1274e-04
45     8.2159e-07     1.0488e-05
```

4.6 Consensus scoring of gene sets

When you are familiar with running different GSAs using `runGSA`, you may ask yourself which settings are the best. We will let you decide on that matter, but offer here a new approach to combine the results from different runs of GSA into a consensus result (Figure 6). This approach is discussed in detail in the `piano` publication [1] and will not be further presented here, except for some examples of how to apply it.

In order to compute consensus scores we first need to run multiple GSAs. Let's use the p-values and t-values as gene-level statistics from the example in section 3:

```
> myTval <- pfc$resTable[["aerobic_Clim - anaerobic_Clim"]]$t
> names(myTval) <- pfc$resTable[["aerobic_Clim - anaerobic_Clim"]]$ProbesetID
```

Now we will run multiple GSAs with different settings, using the GO gene sets from one of the previous examples:

```
> myGsc <- loadGSC(mapGO)
> gsaRes1 <- runGSA(myTval,geneSetStat="mean",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes2 <- runGSA(myTval,geneSetStat="median",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes3 <- runGSA(myTval,geneSetStat="sum",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes4 <- runGSA(myTval,geneSetStat="maxmean",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes5 <- runGSA(myPval,myFC,geneSetStat="fisher",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes6 <- runGSA(myPval,myFC,geneSetStat="stouffer",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
> gsaRes7 <- runGSA(myPval,myFC,geneSetStat="tailStrength",gsc=myGsc,
+                   nPerm=1000,gsSizeLim=c(10,800))
```

Once we have several GSA results that we wish to combine we can use the `consensusHeatmap` function:

```
> resList <- list(gsaRes1,gsaRes2,gsaRes3,gsaRes4,gsaRes5,gsaRes6,gsaRes7)
> names(resList) <- c("mean","median","sum","maxmean","fisher",
+                     "stouffer","tailStrength")
> ch <- consensusHeatmap(resList,cutoff=30,method="mean")
```

Figure 7 shows the resulting plot with the gene set consensus scores. In this case, the consensus score is the mean rank given each gene set by the different GSA runs. This means that a low score (e.g. 1) is a gene set that is ranked high by most of the GSA methods. To also get an idea of the significance level of the consensus scored gene sets, the median p-values can be accessed through:

```
> ch$pMat
```

We can see from the heatmap that the gene sets clustered at the upper part are showing patterns of mostly down-regulation whereas the gene sets in the lower part are showing patterns of mainly up-regulation. Please see [1] for more information about the consensus scoring approach.

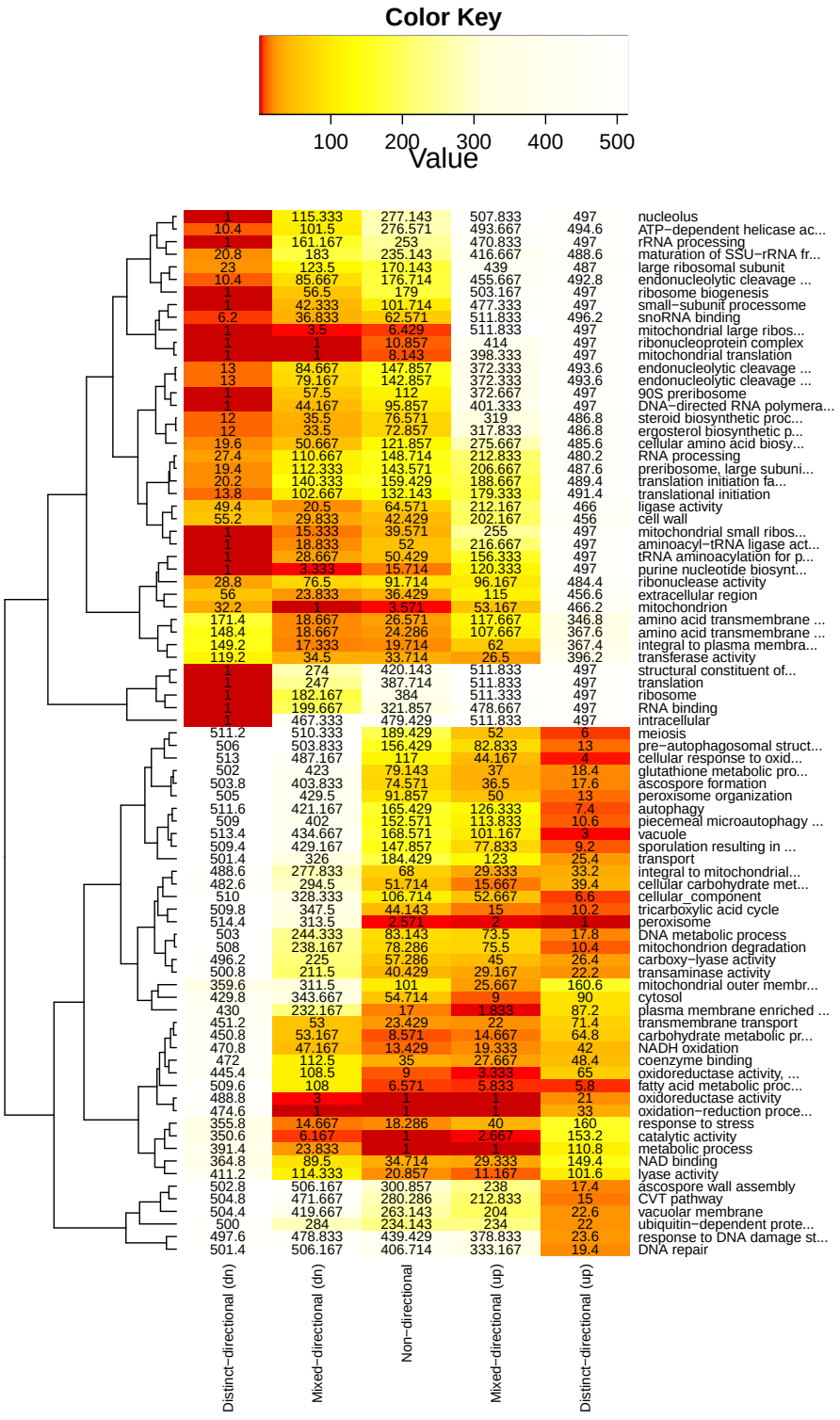

Figure 7: Heatmap of consensus scores.

References

1. Väremo, L., Nielsen, J., and Nookaew, I. Enriching the gene set analysis of genome-wide data by incorporating directionality of gene expression and combining statistical hypotheses and methods. *Nucleic Acids Research* 41(8), 4378–4391 (2013).

2. Nookaew, I., Papini, M., Pornputtpong, N., Scalcinati, G., Fagerberg, L., Uhlén, M., and Nielsen, J. A comprehensive comparison of rna-seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: a case study in saccharomyces cerevisiae. *Nucleic Acids Research* 40(20) (2012).

3. Smyth, G. K. Limma: linear models for microarray data. In Bioinformatics and Computational Biology Solutions using R and Bioconductor, Gentleman, R., Carey, V., Dudoit, S., Irizarry, R., and Huber, W., editors, 397–420. Springer, New York (2005).

4. Fisher, R. *Statistical methods for research workers.* Oliver and Boyd, Edinburgh, (1932).

5. Stouffer, S., Suchman, E., Devinney, L., Star, S., and Williams Jr, R. *The American soldier: adjustment during army life.* Princeton University Press, Oxford, England, (1949).

6. Patil, K. and Nielsen, J. Uncovering transcriptional regulation of metabolism by using metabolic network topology. *Proceedings of the National Academy of Sciences of the United States of America* 102(8), 2685 (2005).

7. Oliveira, A., Patil, K., and Nielsen, J. Architecture of transcriptional regulatory circuits is knitted over the topology of bio-molecular interaction networks. *BMC Systems Biology* 2(1), 17 (2008).

8. Kim, S. and Volsky, D. Page: parametric analysis of gene set enrichment. *BMC Bioinformatics* 6(1), 144 (2005).

9. Taylor, J. and Tibshirani, R. A tail strength measure for assessing the overall univariate significance in a dataset. *Biostatistics* 7(2), 167–181 (2006).

10. Mootha, V., Lindgren, C., Eriksson, K., Subramanian, A., Sihag, S., Lehar, J., Puigserver, P., Carlsson, E., Ridderstråle, M., Laurila, E., et al. Pgc-1α-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nature Genetics* 34(3), 267–273 (2003).

11. Subramanian, A., Tamayo, P., Mootha, V., Mukherjee, S., Ebert, B., Gillette, M., Paulovich, A., Pomeroy, S., Golub, T., Lander, E., et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences of the United States of America* 102(43), 15545–15550 (2005).

12. Efron, B. and Tibshirani, R. On testing the significance of sets of genes. *The Annals of Applied Statistics* 1, 107–129 (2007).

13. Nookaew, I., Jewett, M., Meechai, A., Thammarongtham, C., Laoteng, K., Cheevadhanarak, S., Nielsen, J., and Bhumiratana, S. The genome-scale metabolic model iin800 of saccharomyces cerevisiae and its validation: a scaffold to query lipid metabolism. *BMC Systems Biology* 2(1), 71 (2008).

14. Österlund, T., Nookaew, I., Bordel, S., and Nielsen, J. Mapping condition-dependent regulation of metabolism in yeast through genome-scale modeling. *BMC systems biology* 7(1), 36 (2013).