Hipathia Package

***Marta R. Hidalgo*[*1], *Francisco Salavert*[2], *Alicia Amadoz*[3], *Çankut Cubuk*[4], *José Carbonell-Caballero*[5], *and Joaquín Dopazo*[4,6,7]**

[1]Unidad de Bioinformática y Bioestadística, Centro de Investigación Príncipe Felipe (CIPF), Valencia, 46012, Spain
[2]BioBam Bioinformatics S.L., Valencia, 46012, Spain
[3]Department of Bioinformatics, Igenomix S.A., Valencia, 46980, Spain
[4]Clinical Bioinformatics Area, Fundación Progreso y Salud (FPS), Hospital Virgen del Rocio, Sevilla, 41013, Spain
[5]Chromatin and Gene expression Lab, Gene Regulation, Stem Cells and Cancer Program, Centre de Regulació Genòmica (CRG), The Barcelona Institute of Science and Technology, PRBB, Barcelona, 08003, Spain
[6]Functional Genomics Node (INB), FPS, Hospital Virgen del Rocio, Sevilla, 41013, Spain.
[7]Bioinformatics in Rare Diseases (BiER), Centro de Investigación Biomédica en Red de Enfermedades Raras (CIBERER), FPS, Hospital Virgen del Rocio, Sevilla, 41013, Spain

*marta.hidalgo@outlook.es

2023-06-05

Abstract

Hipathia is a method for the computation of signal transduction along signaling pathways from transcriptomic data. The method is based on an iterative algorithm which is able to compute the signal intensity passing through the nodes of a network by taking into account the level of expression of each gene and the intensity of the signal arriving to it. It also provides a new approach to functional analysis allowing to compute the signal arriving to the functions annotated to each pathway.

Package

hipathia 3.0.2

Contents

1 Introduction

Hipathia package implements the Canonical Circuit Activity Analysis method for the quantification of the signaling pathways activity presented in Hidalgo et al. This method has been implemented in the webtool http://hipathia.babelomics.org, allowing the user to compare signal propagation in an experiment, and train and use a predictor based on the activation of the canonical circuits or subpathways. The package *hipathia* has been conceived as a functional tool for R users which allows more control on the analysis pipeline than the web implementation does.

This document will introduce you to the *hipathia* package and how to use it to analyze your data.

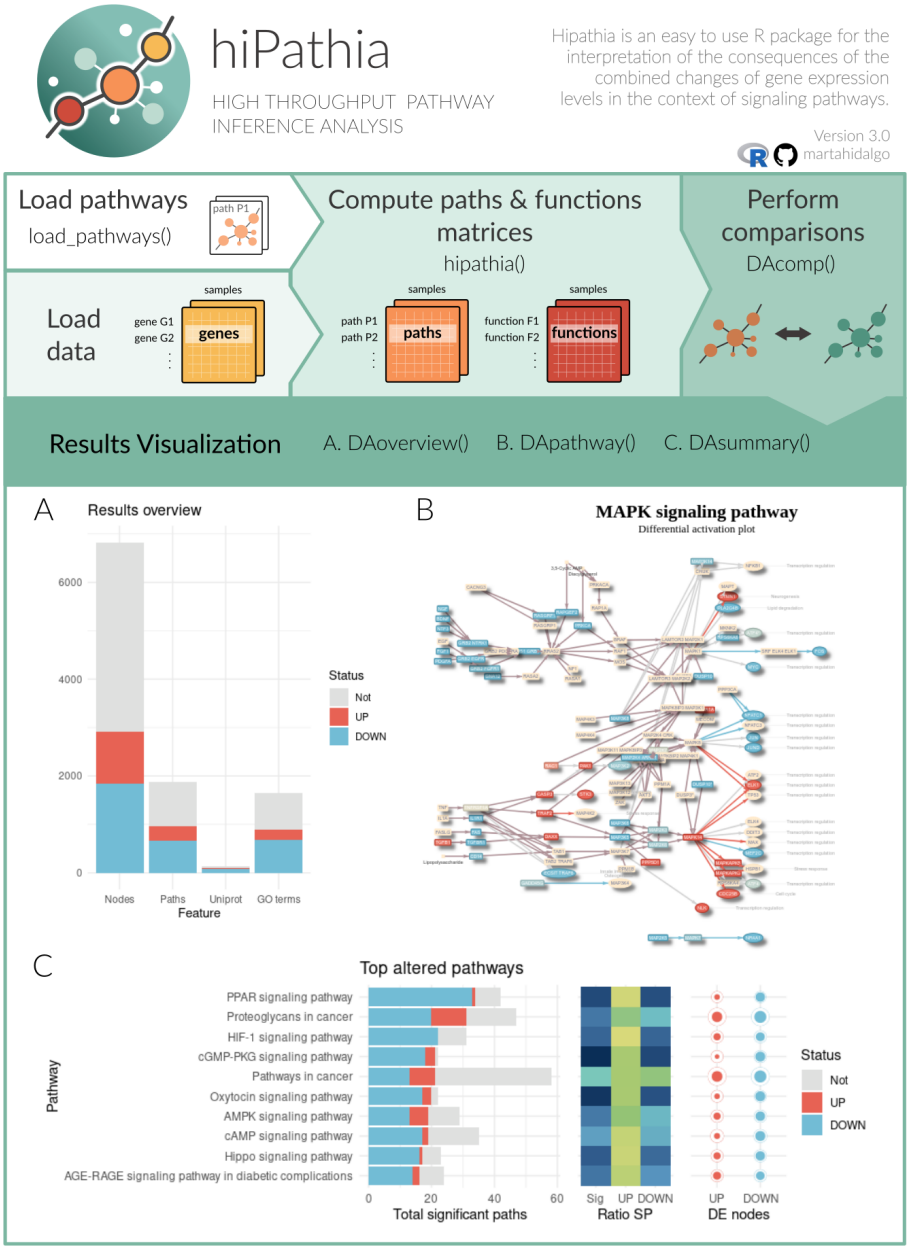


Figure 1: Visual representation of the package functionalities.

2 Previous considerations

Hipathia is a method for the computation of signal transduction along signaling pathways taking as input transcriptomics data. The method is independent on the pathways database, it only needs information about the topology of the graph and the genes included in each node.

However, due to computational cost, *hipathia* needs to preprocess the graphs to be fully efficient. In the current implementation we have developed a module with 145 preprocessed KEGG pathway KGML files, which are ready to be analyzed. In order to preprocess other pathways, see Section 7 on how to create a new pathways object to analyze your own graph pathways with *hipathia*.

Since version `3.0`, we have simplified the pipeline including the data normalization step and the computation of the functional matrices into the `hipathia()` function, introducing function `DAcomp()` to perform differential activation comparisons of nodes, pathways and functions at the same time, and function `DAreport()` to easily create a report. We have also introduced different visualization functions, such as `DAoverview()`, `DAsummary()`, `DAtop()` and `DApathway()`. All functions from previous versions remain available for further use (see Section 9), but we strongly recommend to use the new ones to simplify the use of the package.

2.1 Instalation

In order to install the *hipathia* package, type on your R console

```
## try http:// if https:// URLs are not supported
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("hipathia")
```

2.2 Example data

In order to illustrate the *hipathia* package functionalities an example dataset has been prepared. Data has been downloaded from The Cancer Genome Atlas data repository, from the BRCA-US project, release 20. 20 tumor and 20 normal samples of RNA-Seq data have been randomly selected and normalized.

Specifically, raw data has been corrected for batch effect using the `ComBat()` function from package *sva*, then corrected for RNA composition bias applying TMM normalization from package *edgeR*, and finally log-transformed.

```
library(hipathia)
data("brca")
brca
## class: SummarizedExperiment
## dim: 3187 40
## metadata(0):
## assays(1): raw
## rownames(3187): 2 8647 ... 3925 219699
## rowData names(0):
## colnames(40): TCGA.BH.A1FM.11B.23R.A13Q.07 TCGA.E2.A1LB.11A.22R.A144.07
##   ... TCGA.A2.A0CT.01A.31R.A056.07 TCGA.BH.A18U.01A.21R.A12D.07
## colData names(1): group
```

The dataset `brca` is a *SummarizedExperiment* object, including the gene expression of the 40 samples in the assay `raw`, and the information about whether each sample comes from *Tumor* or *Normal* tissues in the `group` columns of the `colData` dataFrame.

```
head(assay(brca), 4)
##      TCGA.BH.A1FM.11B.23R.A13Q.07 TCGA.E2.A1LB.11A.22R.A144.07
## 2                      10.5317320                     9.732938
## 8647                   -3.3266788                    -3.457515
## 5244                   -0.3600828                    -1.139309
## 1244                    2.2876961                     1.724625
##      TCGA.BH.A208.11A.51R.A157.07 TCGA.BH.A18K.11A.13R.A12D.07
## 2                       9.7958036                    10.868669
## 8647                   -2.5261155                    -3.584934
## 5244                   -0.7368491                    -1.257797
## 1244                    1.0217356                     1.467979
```

```
colData(brca)
## DataFrame with 40 rows and 1 column
##                                    group
##                                <character>
## TCGA.BH.A1FM.11B.23R.A13Q.07      Normal
## TCGA.E2.A1LB.11A.22R.A144.07      Normal
## TCGA.BH.A208.11A.51R.A157.07      Normal
## TCGA.BH.A18K.11A.13R.A12D.07      Normal
## TCGA.E9.A1RC.11A.33R.A157.07      Normal
## ...                                  ...
## TCGA.A0.A12A.01A.21R.A115.07       Tumor
## TCGA.AR.A0TR.01A.11R.A084.07       Tumor
## TCGA.A8.A07E.01A.11R.A034.07       Tumor
## TCGA.A2.A0CT.01A.31R.A056.07       Tumor
## TCGA.BH.A18U.01A.21R.A12D.07       Tumor
```

2.3 Accepted objects

Hipathia has been designed to work with matrices encapsulated as *SummarizedExperiment* objects, in which also the experimental design has been included. However, it is also possible to work in *hipathia* with matrix objects, as long as the experimental design is provided when needed.

Imagine we have the expression data stored in a matrix object called `brca_data` and the experimental design stored in a data frame with one column called `brca_design`. Then, in order to summarize this data in a *SummarizedExperiment* object we should only run:

```
brca <- SummarizedExperiment(assays=SimpleList(raw=brca_data),
                             colData=brca_design)
```

Note that the data frame object provided as `colData` parameter should be ordered as the columns in the matrix provided as assay. For further information on this kind of objects please refer to *SummarizedExperiment*.

When executing a function which needs as input parameter the experimental design (such as the comparison functions), parameter `group` may take two different objects. In case parameter `data` is a matrix, `group` should be a vector giving the class to which each sample belongs, in the same order than the data matrix. In case parameter `data` is a *SummarizedExperiment*, `group` may be either a vector as above, or the name of the column in the `colData` `dataFrame` of the *SummarizedExperiment* storing this information.

In general, functions accepting both SummarizedExperiment and matrix objects as input data and returning a data matrix object, will give as output the same kind of object received. That is, if we apply function `translate_data()` to a SummarizedExperiment object, we will obtain a SummarizedExperiment, while applying the same function to a matrix object will result in a matrix object as output.

2.4 How to cite

Hipathia is a free open-source software implementing the result of a research work. If you use it, please support the research project by citing:

Hidalgo, M. R., Cubuk, C., Amadoz, A., Salavert, F., Carbonell-Caballero, J., & Dopazo, J. (2017). High throughput estimation of functional cell activities reveals disease mechanisms and predicts relevant clinical outcomes. Oncotarget, 8(3), 5160–5178. http://doi.org/10.18632/oncotarget.14107

3 Preprocessment

Hipathia accepts as input data a gene expression matrix. Expression may have been measured with any available sequencing technique. However, *hipathia* assumes that data has been already normalized for correcting any possible sequencing bias (which includes also batch effect correction).

3.1 Gene IDs translation

The gene expression matrix must include samples as columns and genes as rows, as shown in the `brca` dataset example. Rownames must be the Entrez IDs of the genes in the rows. In order to transform other gene IDs to Entrez IDs, function `translate_data()` can be used. Accepted IDs to be transformed to Entrez IDs include:

Human

- Affy HG U133A probeset
- Affy HG U133B probeset
- Affy HG U133-PLUS_2 probeset
- Agilent SurePrint G3 GE 8x60k
- Agilent SurePrint G3 GE 8x60k v2
- Agilent Whole Genome 4x44k
- Agilent Whole Genome 4x44k v2
- CCDS
- Ensembl gene
- Ensembl transcript
- Entrez ID
- GenBank EMBL
- GenBank PID
- HGNC symbol
- RefSeq mRNA
- RefSeq mRNA PRED
- RefSeq ncRNA
- RefSeq ncRNA PRED

Mouse

- Affy Mouse 430 2
- Ensembl gene
- Gene name
- Mouse Gene 1.0

Rat

- Ensembl gene
- Gene name

The parameters needed by this function are the data matrix and the species of the experiment.

```
data(brca_data)
trans_data <- translate_data(brca_data, "hsa")
## translated ids = 3184 (1)
## untranslated ids = 3 (0.00094)
## multihit ids = 0 (0)
```

4 Pathway activation computation

Hipathia aims to compute the level of activation of each subpathway in a pathway for each of the samples from the experiment. This is done by function `hipathia()`, which takes as inputs the matrix of gene expression, the pathways object and some additional parameters.

In this section we will see how to load the pathways object, how the *hipathia* method works and how to apply function `hipathia()` to the computation of the values of activation of the loaded pathways.

4.1 Loading pathways

Hipathia package works with a preprocessed pathway object. This object includes all the information that the different functions in the package need. Currently, a set of 146 preprocessed KEGG signaling pathways is available within the package. In order to load this object, use function `load_pathways()` and select the species to be analyzed. Available species include human (*hsa*), mouse (*mmu*) and rat (*rno*). To create your own set of preprocessed pathways, see Section 7.

```
pathways <- load_pathways(species = "hsa")
## Loaded 146 pathways
```

Parameter `pathways_list` allows the user to specify the pathways to be loaded. The different functions of the package will use all the pathways in the pathways object for its computations. In order to restrict the analysis to a particular set of pathways, load only the required pathways to the pathway object. By default, all pathways available for the specified species are loaded.

```
pathways_only2 <- load_pathways(species = "hsa", pathways_list = c("hsa03320",
                                                                   "hsa04014"))
## Loaded 2 pathways
```

In order to know which pathways are included in each pathways object, use function `get_pathways_list()`.

```
length(get_pathways_list(pathways))
## [1] 146
get_pathways_list(pathways)[1:10]
##  [1] "hsa03320" "hsa03460" "hsa04010" "hsa04012" "hsa04014" "hsa04015"
##  [7] "hsa04020" "hsa04022" "hsa04024" "hsa04062"
```

```
length(get_pathways_list(pathways_only2))
## [1] 2
get_pathways_list(pathways_only2)
## [1] "hsa03320" "hsa04014"
```

4.2 Computing the signal

In order for a protein to pass the signal, there are two important factors: first, the protein must be present, and second, some other protein must activate it. Therefore, *hipathia* is a method to compute signal transduction based on two steps. First, it quantifies the presence of a particular gene as a normalized value between 0 and 1. Then, it computes the signal

value passing through a node taking into account the level of expression of each gene inside the node and the intensity of the signal arriving to it. The signal value of the pathway is the signal value through the last node of the pathway.

4.2.1 Subpathways

Pathways are represented by directed graphs, which include different input and output nodes. The signal arrives to an initial node and is transmited along the pathway following the direction of the interactions up to an output node. Thus, the signal may follow many different paths along the pathway. *Hipathia* computes the intensity of this signal up to each output node of a pathway separately.

Genes in the output nodes are also called *effector proteins*, since they are the ones responsibles for performing the action the signal is seeking. We define the *effector subpathway* ending in node *G* as the subgraph including any node in a path leading to *G*. When applied to effector subpathways, *hipathia* returns the intensity of the signal arriving to the effector protein *G*.

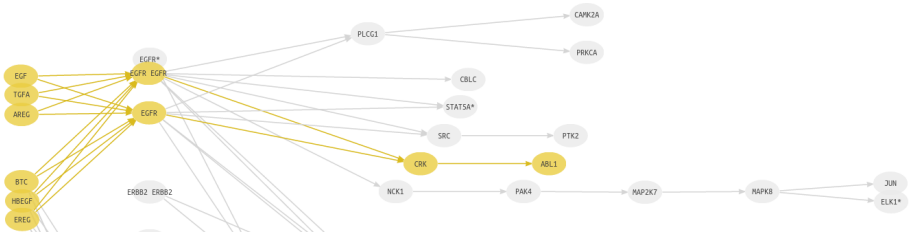


Figure 2: Effector subpathway depicted in yellow

Effector subpathways may have many different input nodes. In order to analyze in detail which of the possible paths leading to node *G* is responsible for the observed change, effector subpathays can be decomposed into several subpathways including only one input node. We define the *decomposed subpathway* from *H* to *G* as the subgraph including any node in a path from *H* to *G*.

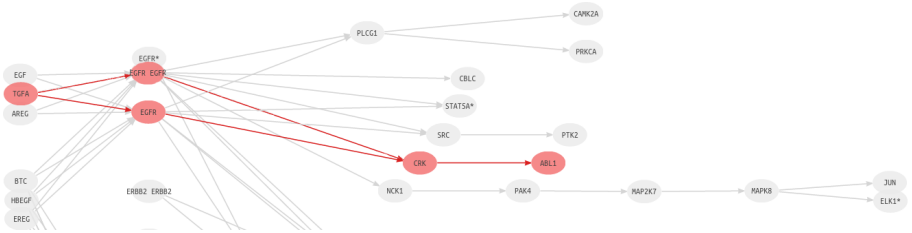


Figure 3: Decomposed subpathway depicted in red

4.2.2 Node expression

Pathways are represented by graphs and composed by nodes and relations among them. Some nodes may contain multiple genes representing different isoforms of the protein or members of the same gene familiy, among others. Since each gene has its own level of expression, the first step of the method is to summarize this information into a score representing the expression of the node as a whole.

4.2.3 Signal transduction

The computation of the signal intensity across the pathway is performed by means of an iterative algorithm beginning in the input nodes of the subpathway. In order to initialize the pathway signal we assume an incoming signal value of 1 in the input nodes of the subpathway. Then, for each node n of the network, the signal value S_n is propagated along the nodes according to the following rule

$$S_n = v_n \cdot (1 - \prod_{s_i \in A_n} (1 - s_i)) \cdot \prod_{s_j \in I_n} (1 - s_j) \quad (1)$$

where A_n is the set of signals arriving to the current node from an activation edge, I_n is the set of signals arriving to the current node from an inhibition edge, and v_n is the normalized value of expression of the current node.

4.3 Functional annotation

Each effector protein of a pathway is responsible for performing a particular function. Thus, from the matrix of effector subpathways we can infer the functions matrix, by computing an intensity value for each molecular function and for each sample.

Different effector subpathways of different pathways may end in the same effector protein, and also different effector proteins may have the same molecular function. Therefore, for a particular function f, we summarize the values of all the subpathways ending in an effector protein related to f with a mean value.

4.4 Using *Hipathia* to compute the signal

Function `hipathia()` computes the level of activation of the subpathways, taking as inputs the matrix of gene expression, the pathways object and some additional parameters.

The genes which are needed by `hipathia()` to compute the signal and are not present in the provided matrix are added by the function, assigning to each sample the median of the matrix. The number and percentage of added genes is shown by the function. A high level of *added missing genes* may indicate that the results are not representative of the actual analysis.

Parameter `decompose` indicates whether to use effector subpathways or decomposed subpathways. Option `decompose=FALSE` uses effector subpathways while option `decompose=TRUE` uses decomposed subpathways. For further information on this, see Section 4.2.1. For further information on the method used to compute the level of signal activity in the pathways, see Section 4.2 or refer to Hidalgo et al..

```
hidata <- hipathia(exp_data, pathways, uni.terms = TRUE, GO.terms = TRUE,
                   decompose = FALSE, verbose=TRUE)
## Added missing genes: 165 (4.93%)
## Computing pathways...
##
## Computing Uniprot terms...
## Quantified Uniprot terms: 142
##
## Computing GO terms...
## Quantified GO terms: 1654
## DONE
```

Since version `3.0`, the functional annotation is computed within the `hipathia()` function. Different function activity matrices can be computed depending on the functional annotation given to the effector nodes. Currently, `hipathia()` accepts any annotation defined by the user and includes two default annotations: Gene Ontology (GO) annotations and Uniprot keywords. For further information on the differences between GO and Uniprot keywords annotations please refer to this page.

Parameters `uni.terms` and `GO.terms` accept `TRUE` or `FALSE` values and indicate whether to compute the Uniprot keywords and GO terms activity matrices, respectively. Parameter `custom.terms` accepts a `data.frame` with the annotation of the genes to the functions. First column are gene symbols, second column the functions. Notice that functions annotated to genes which are not included in any effector node will be not computed.

The object resulting from `hipathia()` is a *MultiArrayExperiment* object, which includes up to five different *SummarizedExperiment* objects: `nodes`, `paths`, `uni.terms`, `GO.terms` and `custom.terms`.

```
hidata
## A MultiAssayExperiment object of 4 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 4:
##  [1] nodes: SummarizedExperiment with 6826 rows and 40 columns
##  [2] paths: SummarizedExperiment with 1876 rows and 40 columns
##  [3] uni.terms: SummarizedExperiment with 142 rows and 40 columns
##  [4] GO.terms: SummarizedExperiment with 1654 rows and 40 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

Rownames of the `paths` *SummarizedExperiment* are the IDs of the processed subpathways. Its `rowData` also stores the comprehensive names of the subpathways in column `path.name`.

```
rowData(hidata[["paths"]])
## DataFrame with 1876 rows and 4 columns
##                      path.ID             path.name            path.nodes
##                  <character>           <character>           <character>
## P-hsa03320-37   P-hsa03320-37 PPAR signaling pathw.. N-hsa03320-1, N-hsa0..
## P-hsa03320-61   P-hsa03320-61 PPAR signaling pathw.. N-hsa03320-1, N-hsa0..
## P-hsa03320-46   P-hsa03320-46 PPAR signaling pathw.. N-hsa03320-1, N-hsa0..
## P-hsa03320-57   P-hsa03320-57 PPAR signaling pathw.. N-hsa03320-1, N-hsa0..
## P-hsa03320-64   P-hsa03320-64 PPAR signaling pathw.. N-hsa03320-1, N-hsa0..
## ...                      ...                   ...                   ...
## P-hsa05321-74   P-hsa05321-74 Inflammatory bowel d.. N-hsa05321-47, N-hsa..
## P-hsa05321-81   P-hsa05321-81 Inflammatory bowel d.. N-hsa05321-47, N-hsa..
## P-hsa05321-138 P-hsa05321-138 Inflammatory bowel d.. N-hsa05321-47, N-hsa..
## P-hsa05321-75   P-hsa05321-75 Inflammatory bowel d.. N-hsa05321-47, N-hsa..
## P-hsa05321-152 P-hsa05321-152 Inflammatory bowel d.. N-hsa05321-101, N-hs..
##                decomposed
```

```
##                <logical>
## P-hsa03320-37       FALSE
## P-hsa03320-61       FALSE
## P-hsa03320-46       FALSE
## P-hsa03320-57       FALSE
## P-hsa03320-64       FALSE
## ...                   ...
## P-hsa05321-74       FALSE
## P-hsa05321-81       FALSE
## P-hsa05321-138      FALSE
## P-hsa05321-75       FALSE
## P-hsa05321-152      FALSE
```

In case you need to transform subpath IDs to comprehensive subpath names, see Section 8.1.3. However, it is not recommended to change the row names of the matrix of subpath values.

Notice that the matrix of subpathway activity values will include a value of activity for each sample and for each possible subpathway of the pathways in the pathway object. Depending on whether parameter `decompose` is set to `TRUE` or `FALSE`, and on the number of pathways included in the object of pathways given as attribute, the number of analyzed subpathways may vary. Currently *hipathia* includes up to the following number of pathways, effector subpathways and decomposed subpathways per species:

	Pathways	Effector subpathways	Decomposed subpathways
hsa	146	1876	8440
mmu	142	1857	8440
rno	142	1853	8251

It is recommended to perform an initial *hipathia* analysis with effector subpathways, and use decomposed subpathways only for specific pathways in which the user is highly interested.

5 Differential activation

Once the activation matrices have been computed with `hipathia()`, you may want to analyze differential activation. Since version 3.0, this can be done with function `DAcomp()`. For further information on the old pipeline, see Section 9.

Function `DAcomp()` computes the comparison of the nodes, paths and functional activation matrices included in the `hipathia()` output object. The test used for the comparison of each activation matrix can be set through parameters `path.method`, `node.method` and `function.method`. Options include `wilcoxon`, in which case a two groups comparison will be applied, or `limma`, in which case functions `lmFit()`, `contrasts.fit()` and `eBayes()` from the *limma* package will be used. By default, `node.method` is set to `limma` and `path.method` and `function.method`, set to `wilcoxon`.

The experimental design applied to the comparison must be the same in all cases. For a two case comparison, parameters `expdes` and `g2` can be used to specify case and control groups, respectively, either for the `wilcoxon` or `limma` options. For a contrast passed to function `makeContrasts()` in *limma* package, use the `expdes` parameter (only for the `limma` option).

```
# Perform comparisons
DAdata <- DAcomp(hidata, "group", "Tumor", "Normal")
DAdata <- DAcomp(hidata, "group", "Tumor - Normal", path.method = "limma",
                 fun.method = "limma")
```

Parameter `paired` (`FALSE` by default), `order` (`FALSE` by default) and `adjust` (`TRUE` by default) are boolean, and indicate whether samples are paired or not, whether the results should be ordered by p.value or not, and whether the p.values should be adjusted by the `p.adjust()` function using the FDR method. Parameter `conf.level` indicates the confidence level if `adjust` is `TRUE`.

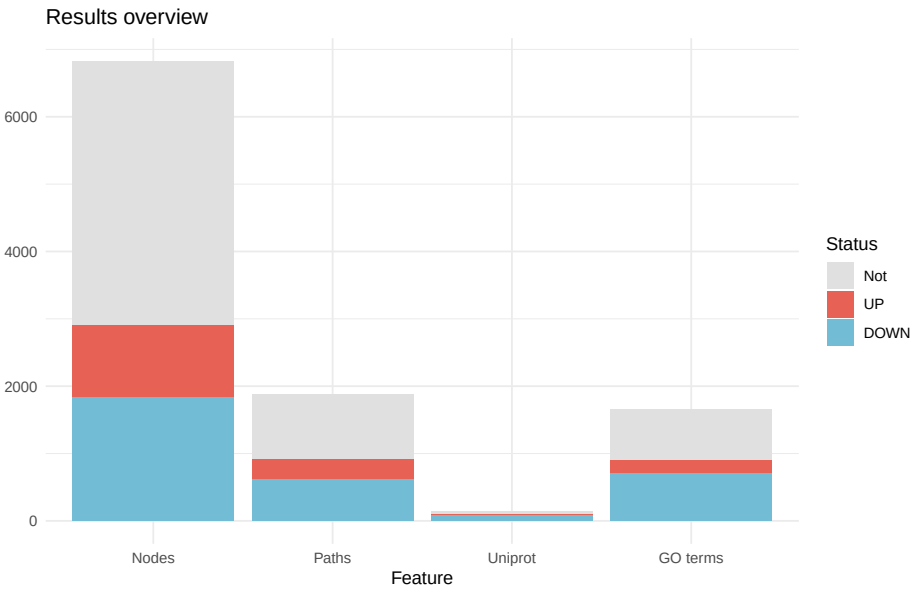
6 Results visualization

We have developed a set of functions to visualize the comparison results and different ways to summarize them.

6.1 Results overview

Function `DAoverview()` provides a `tibble` with the number of significant up- and down-activated nodes, paths and functions, and plots a bar chart with this info.

```
# Summary of UP & DOWN nodes, paths and functions
DAoverview(DAdata)
## # A tibble: 4 x 5
##   feature   total  sigs   UPs DOWNs
##   <chr>     <int> <int> <int> <int>
## 1 nodes      6826  2917  1081  1836
## 2 paths      1876   914   291   623
## 3 uni.terms   142   105    20    85
## 4 GO.terms   1654   900   192   708
```



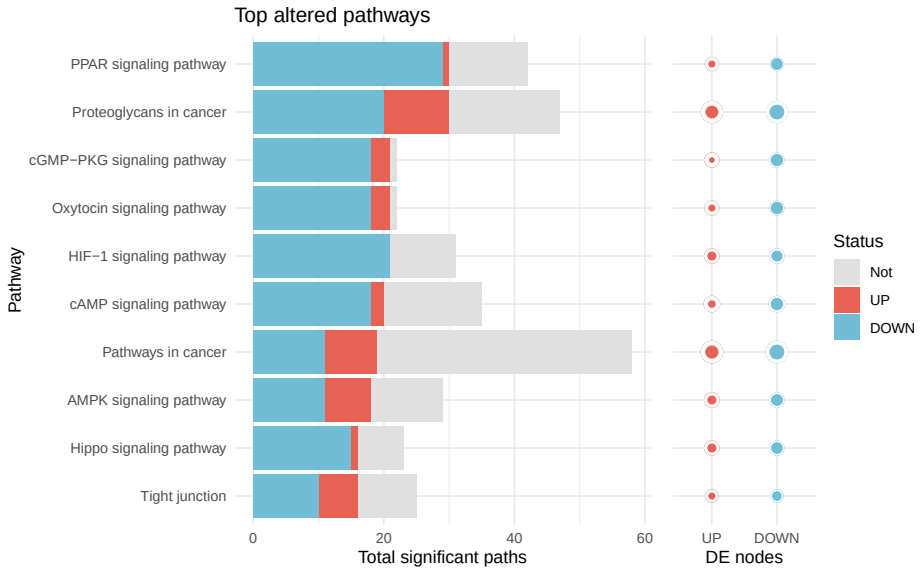
Parameter `conf.level` allows to set the significant cutoff, `adjust` indicates whether p.values should be adjusted, and `colors` allows to set the color scheme.

6.2 Results summary by pathway

Function `DAsummary()` provides a `tibble` with the summary of the number of paths altered in the `n` most altered pathways, and plots a complex bar and dot chart with this info.

```
# Summary of path alteration by pathway
DAsummary(DAdata)
## # A tibble: 146 x 14
##    ID       name         sigs   UPs DOWNs total ratio.sigs ratio.UPs ratio.DOWNs
```

```
##    <chr>    <chr>       <int> <int> <int> <int>      <dbl>     <dbl>       <dbl>
##  1 hsa03320 PPAR signa~    30     1    29    42      0.714    0.0238       0.690
##  2 hsa05205 Proteoglyc~    30    10    20    47      0.638    0.213        0.426
##  3 hsa04022 cGMP-PKG s~    21     3    18    22      0.955    0.136        0.818
##  4 hsa04921 Oxytocin s~    21     3    18    22      0.955    0.136        0.818
##  5 hsa04066 HIF-1 sign~    21     0    21    31      0.677    0            0.677
##  6 hsa04024 cAMP signa~    20     2    18    35      0.571    0.0571       0.514
##  7 hsa05200 Pathways i~    19     8    11    58      0.328    0.138        0.190
##  8 hsa04152 AMPK signa~    18     7    11    29      0.621    0.241        0.379
##  9 hsa04390 Hippo sign~    16     1    15    23      0.696    0.0435       0.652
## 10 hsa04530 Tight junc~    16     6    10    25      0.64     0.24         0.4
## # i 136 more rows
## # i 5 more variables: sig.nodes <int>, UP.nodes <int>, DOWN.nodes <int>,
## #   gene.nodes <int>, total.nodes <int>
```



The `DAsummary()` `tibble` includes columns: - ID: Pathway ID - name: Pathway name - sigs: Number of significant paths within each pathway - UPs: Number of significant up-activated paths within each pathway - DOWNs: Number of significant down-activated paths within each pathway - total: Number of total paths within each pathway - ratio.sigs: Ratio of significant paths with respect to the total paths within each pathway - ratio.UPs: Ratio of significant up-activated paths with respect to the total paths within each pathway - ratio.DOWNs: Ratio of significant down-activated paths with respect to the total paths within each pathway - sig.nodes: Number of significant nodes within each pathway - UP.nodes: Number of significant up-regulated nodes within each pathway - DOWN.nodes: Number of significant down-regulated nodes within each pathway - gene.nodes: Number of gene nodes (not metabolites or other compounds) within each pathway - total.nodes: Total number of nodes within each pathway.

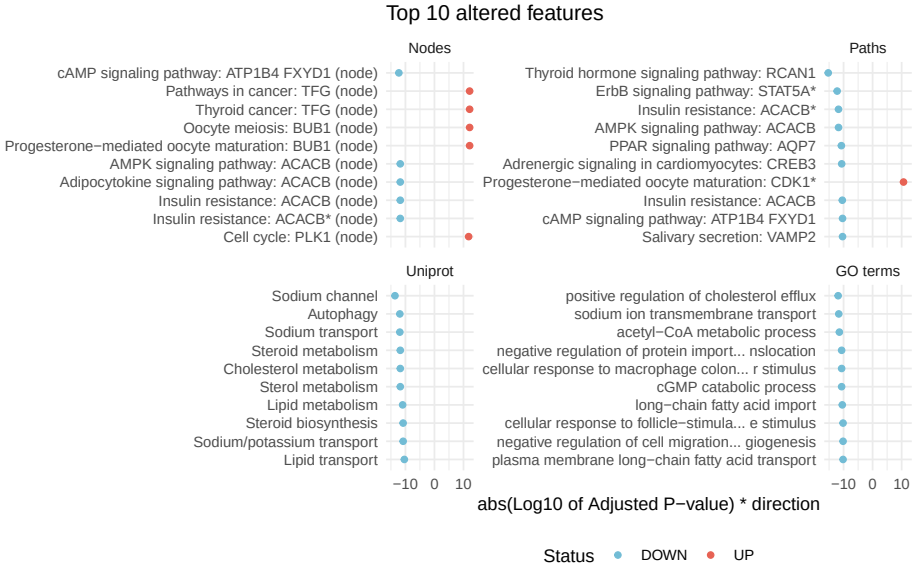
Parameter `ratio` indicates whether the ratio of altered paths should be included in the plot, and `order.by` indicates whether to select the top pathways by the number of significant paths, or by the ratio of significant paths with respect to the total number of paths. Also, parameter `conf.level` allows to set the significant cutoff, `adjust` indicates whether p.values should be adjusted, and `colors` allows to set the color scheme.

6.3 Top results per feature

Function `DAtop()` provides a `tibble` for each feature with the top `n` differentially activated nodes, paths and functions, and plots a dot plot with that info.

```
# Top 10 altered features per class (nodes, paths, functions)
DAtop(DAdata)
## $nodes
## # A tibble: 10 x 10
##    ID    name  label `UP/DOWN` statistic  p.value FDRp.value type  logPV feature
##    <chr> <chr> <chr> <chr>         <dbl>    <dbl>      <dbl> <chr> <dbl> <chr>
##  1 N-hs~ cAMP~ ATP1~ DOWN          -13.7 8.96e-17   5.48e-13 gene  -12.3 nodes
##  2 N-hs~ Path~ TFG   UP             13.0 5.26e-16   7.71e-13 gene   12.1 nodes
##  3 N-hs~ Thyr~ TFG   UP             13.0 5.26e-16   7.71e-13 gene   12.1 nodes
##  4 N-hs~ Oocy~ BUB1  UP             12.9 6.31e-16   7.71e-13 gene   12.1 nodes
##  5 N-hs~ Prog~ BUB1  UP             12.9 6.31e-16   7.71e-13 gene   12.1 nodes
##  6 N-hs~ AMPK~ ACACB DOWN          -12.4 2.58e-15   1.69e-12 gene  -11.8 nodes
##  7 N-hs~ Adip~ ACACB DOWN          -12.4 2.58e-15   1.69e-12 gene  -11.8 nodes
##  8 N-hs~ Insu~ ACACB DOWN          -12.4 2.58e-15   1.69e-12 gene  -11.8 nodes
##  9 N-hs~ Insu~ ACAC~ DOWN          -12.4 2.58e-15   1.69e-12 gene  -11.8 nodes
## 10 N-hs~ Cell~ PLK1  UP             12.2 3.86e-15   1.69e-12 gene   11.8 nodes
##
## $paths
## # A tibble: 10 x 15
##    ID         name   `UP/DOWN` statistic  p.value FDRp.value N.nodes N.gene.nodes
##    <chr>      <chr> <chr>         <dbl>    <dbl>      <dbl>   <int>        <int>
##  1 P-hsa0491~ Thyr~ DOWN          -16.7 2.79e-19   5.22e-16       2            2
##  2 P-hsa0401~ ErbB~ DOWN          -13.2 6.53e-16   6.12e-13       9            9
##  3 P-hsa0493~ Insu~ DOWN          -12.6 2.86e-15   1.78e-12       2            2
##  4 P-hsa0415~ AMPK~ DOWN          -12.5 3.89e-15   1.82e-12      12           11
##  5 P-hsa0332~ PPAR~ DOWN          -11.5 4.67e-14   1.75e-11       6            2
##  6 P-hsa0426~ Adre~ DOWN          -11.3 8.05e-14   2.24e-11      24           15
##  7 P-hsa0491~ Prog~ UP             11.3 8.35e-14   2.24e-11       2            2
##  8 P-hsa0493~ Insu~ DOWN          -11.1 1.57e-13   3.67e-11       7            6
##  9 P-hsa0402~ cAMP~ DOWN          -10.9 2.25e-13   4.42e-11      43           14
## 10 P-hsa0497~ Sali~ DOWN          -10.9 2.36e-13   4.42e-11       7            5
## # i 7 more variables: N.measured.nodes <int>, ratio.measured.gene.nodes <dbl>,
## #   nodes <chr>, N.DA.nodes <int>, DA.nodes <chr>, logPV <dbl>, feature <chr>
##
## $uni.terms
## # A tibble: 10 x 8
##    ID               name   `UP/DOWN` statistic  p.value FDRp.value logPV feature
##    <chr>            <chr> <chr>         <dbl>    <dbl>      <dbl> <dbl> <chr>
##  1 Sodium channel   Sodi~ DOWN          -13.6  1.75e-16   2.48e-14 -13.6 uni.te~
##  2 Autophagy        Auto~ DOWN          -11.7  1.81e-14   1.19e-12 -11.9 uni.te~
##  3 Sodium transport Sodi~ DOWN          -11.6  2.52e-14   1.19e-12 -11.9 uni.te~
##  4 Steroid metaboli~ Ster~ DOWN         -11.2  7.09e-14   1.72e-12 -11.8 uni.te~
##  5 Cholesterol meta~ Chol~ DOWN         -11.2  7.25e-14   1.72e-12 -11.8 uni.te~
##  6 Sterol metabolism Ster~ DOWN         -11.2  7.25e-14   1.72e-12 -11.8 uni.te~
##  7 Lipid metabolism  Lipi~ DOWN         -10.5  5.41e-13   1.10e-11 -11.0 uni.te~
##  8 Steroid biosynth~ Ster~ DOWN         -10.3  9.09e-13   1.61e-11 -10.8 uni.te~
##  9 Sodium/potassium~ Sodi~ DOWN         -10.2  1.04e-12   1.63e-11 -10.8 uni.te~
```

```
## 10 Lipid transport   Lipi~ DOWN          -9.88 3.00e-12   4.26e-11 -10.4 uni.te~
##
## $GO.terms
## # A tibble: 10 x 8
##    ID         name         `UP/DOWN` statistic  p.value FDRp.value logPV feature
##    <chr>      <chr>        <chr>         <dbl>    <dbl>      <dbl> <dbl> <chr>
##  1 GO:0010875 positive re~ DOWN          -13.0 8.12e-16   1.34e-12 -11.9 GO.ter~
##  2 GO:0035725 sodium ion ~ DOWN          -12.6 2.66e-15   2.20e-12 -11.7 GO.ter~
##  3 GO:0006084 acetyl-CoA ~ DOWN          -12.2 6.52e-15   3.59e-12 -11.4 GO.ter~
##  4 GO:0033159 negative re~ DOWN          -11.3 7.75e-14   2.14e-11 -10.7 GO.ter~
##  5 GO:0036006 cellular re~ DOWN          -11.3 7.75e-14   2.14e-11 -10.7 GO.ter~
##  6 GO:0046069 cGMP catabo~ DOWN          -11.3 7.75e-14   2.14e-11 -10.7 GO.ter~
##  7 GO:0044539 long-chain ~ DOWN          -11.0 1.53e-13   3.60e-11 -10.4 GO.ter~
##  8 GO:0071372 cellular re~ DOWN          -10.6 4.84e-13   6.60e-11 -10.2 GO.ter~
##  9 GO:0090051 negative re~ DOWN          -10.6 4.84e-13   6.60e-11 -10.2 GO.ter~
## 10 GO:0015911 plasma memb~ DOWN          -10.5 5.60e-13   6.60e-11 -10.2 GO.ter~
```

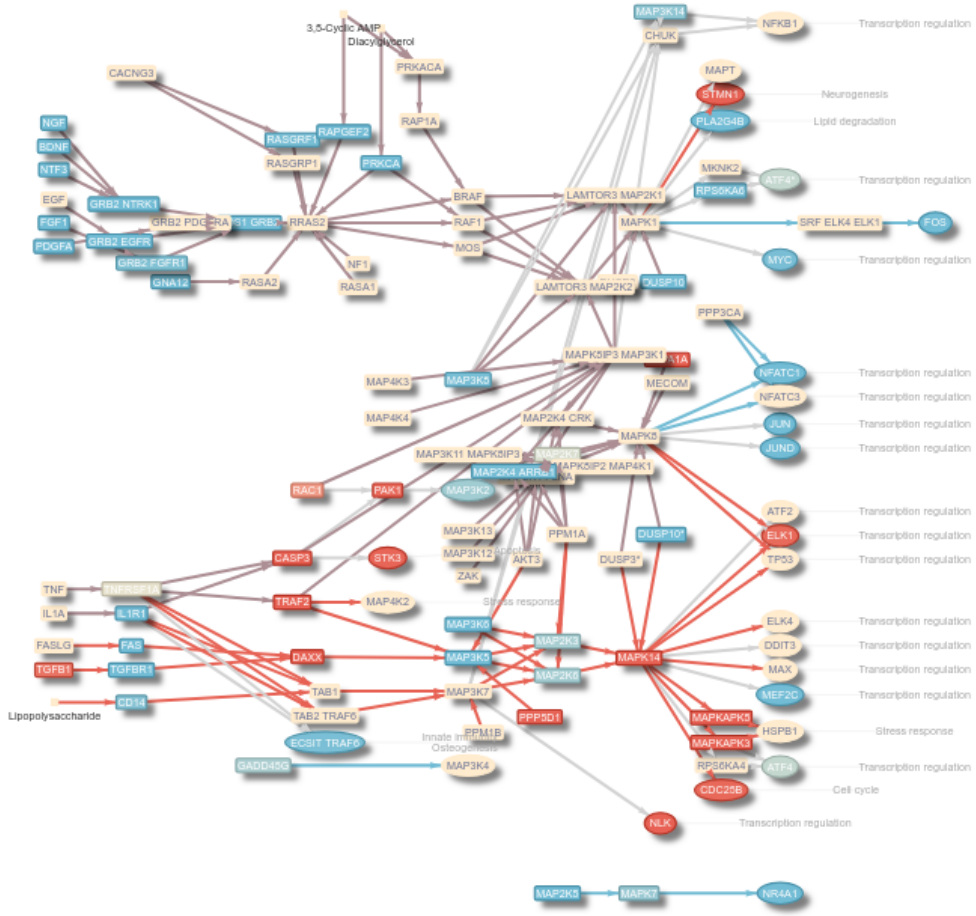


Parameter `conf.level` allows to set the significant cutoff, `adjust` indicates scheme.

6.4 Pathway differential activation plot

Function `DApathway()` uses the *visNetwork* CRAN package to plot an interactive graph visualization of the changes in the activation of a pathway, including node dysregulation and path dysactivation.

```
# Pathway visualization
DApathway("hsa04010", pathways, DAdata)
```



Parameter `conf.level` allows to set the significant cutoff, `adjust` indicates whether p.values should be adjusted, `colors` allows to set the color scheme, and `no.col` allows to set the color of non-significant nodes. Parameters `main` and `submain` allow to set the title and subtitle of the plot.

6.5 Visualization through a local server

Hipathia results can be interactively visualized on a web browser. Use function `DAreport()` to create a report of the `DAdata` obtained from function `DAcomp()`, and function `visualize_report()` to serve the report to a browser. For the interpretation of the results in this visualization, see Section 6.6.

```
# Save and serve all results to browser
HPreport <- DAreport(DAdata, pathways)
visualize_report(HPreport)
## Serving the directory /tmp/RtmpWGYNhq/hipathia_report_1/pathway-viewer at http://127.0.0.1:4000
## Open a web browser and go to URL http://127.0.0.1:4000
```

Due to cross-origin security restrictions (CORS), a web server is needed to serve the result files correctly. The function `visualize_report()` uses the *servr* package to this end, please refer to the package documentation for further information. The report is served to the default URL http://127.0.0.1:4000. Port 4000 may be changed through parameter `port`. Notice that you can not serve to a port if it is already used.

The servers will be active until function `daemon_stop()` from package `servr` is executed. Information about how to stop each server individually is given as an output of each `visualize_report()` function. To stop all servers at a time, use

```
servr::daemon_stop()
```

Alternatively, if you have already a web server installed in your computer, just link or move the output folder to your web server http document root and then open it on your web browser.

6.6 Interpreting HTML results

The interactive visualization of *hipathia* results includes three panels and a legend. The legend is on top of the page resuming the main information depicted in the images. The left panel is the pathways panel, where the currently selected pathway is shown. The layout of the pathway is similar to the layout shown in KEGG.

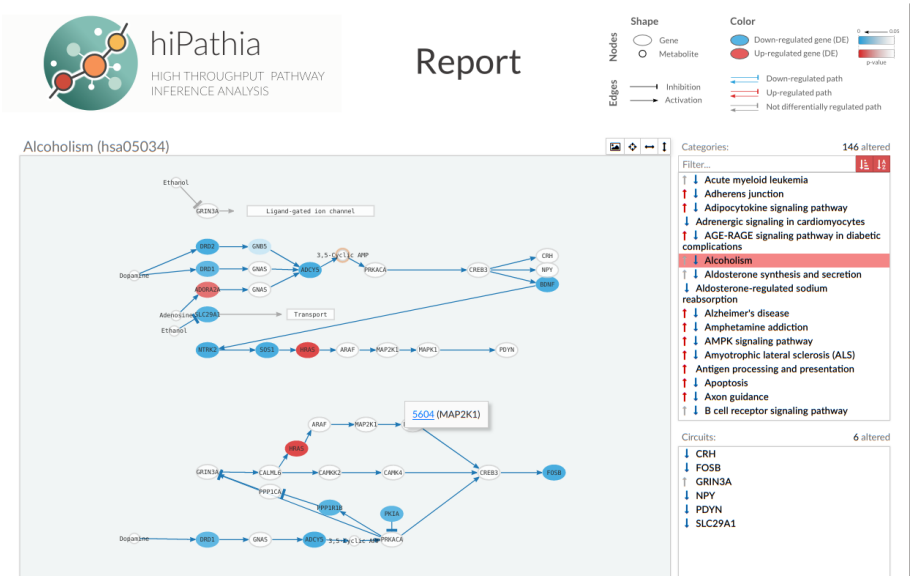


Figure 4: Interactive Hipathia report visualization

As before, edges belonging to significant down-activated pathways are depicted in blue, those belonging to significant up-activated subpathways are depicted in red, and those belonging to non-significant subpathways are depicted in grey. Similarly, when nodes are colored by their differential expression, down-regulated nodes are colored in blue, up-regulated nodes are colored in red and non-significant nodes are colored in white. Different shades of the colors indicate different levels of significance with respect to the p-value of the differential expression.

The selected pathway to be shown can be modified through the pathway list in the top right panel. Arrows pointing up and down to the left of the names of the pathways indicates that the pathways contain up- or down-activated subpathways, respectively. When the arrows are

colored in red or blue, it means that there are significant up- or down-regulated subpathways, respectively. The pathways list can be filtered through the *Filter...* box, or ordered by means of the buttons in the top right part of the panel.

All computed subpathways of the currently selected pathway are listed in the subpathways list in the bottom right panel. Arrows pointing up and down by the names of the subpathways indicates that they are up- or down-activated, respectively. When the arrows are colored in red or blue, it means that they are significantly up- or down-regulated, respectively. When a subpathway is selected from the list, only the arrows and nodes belonging to this subpathway will be highlighted. Clicking again on this subpathway will deselect it.

7 Creating a new Pathways object

7.1 Creating a new pathways object with Hipathia

Hipathia is able to read and analyze custom graphs from SIF files with attributes. No species restriction is applied in this case. See Section 7.2 for further details on file specifications.

The function used for that purpouse is `mgi_from_sif()`, which takes as parameter `sif.folder` the path to the folder where the pathway files are stored, and as `spe` parameter the modeled species. Optionally, the function can add the name of the functions to which the effector nodes are related to increase the readability of the output infromation. For that, the user must include as `entrez_symbol` parameter a data.frame with two columns, first column with the EntrezGene ID, second column with the gene Symbol of the included genes, and as parameter `dbannot` the functional annotation of the included genes.

```
newmgi <- mgi_from_sif(system.file("extdata/SIF_ATT_example",
                                   package = "hipathia"),
                       spe = "hsa")
## Loading graphs...
## Creating MGI...
## Created MGI with 1 pathway(s)
```

7.2 Pathway SIF + ATT specifications

Hipathia is able to read and include graphs from SIF files with attributes with the following features:

- Each pathway should be saved in two different files: .att (ATT file) and .sif (SIF file).
- The SIF and ATT files should have the same name, i.e. hsa00.sif and hsa00.att for the pathway with ID hsa00.
- Functions are not included in this files, but annotated "a posteriori" following a file of annotations from genes to functions.
- There must also be a file including the readable names of the pathways in the same folder, named: name (dot) pathways (underscore) (species) (dot) txt.

7.2.1 SIF File

The SIF file must fulfill the following requirements:

- Text file with three columns separated by tabulars.
- Each row represents an interaction in the pathway. First column is the source node, third column the target node, and the second is the type of relation between them.
- Only activation and inhibition interactions are allowed.
- The name of the nodes in this file will be stored as the IDs of the nodes.
- The nodes IDs should have the following structure: N (dash) pathway ID (dash) node ID.
- Hipathia distinguish between two types of nodes: simple and complex.

- Simple nodes may include many genes, but only one is needed to perform the function of the node.
- Complex nodes include different simple nodes and represent protein complexes. Each simple node within the complex represents one protein in the complex. This node requires the presence of all their simple nodes to perform its function.

- Node IDs from simple nodes do not include any space, i.e. N-hsa00-A.
- Node IDs from complex nodes are the juxtaposition of the included simple node IDs, separated by spaces, i.e. N-hsa00-D E.

An example of SIF file as described above is shown here (hashtags must not be included in the file):

```
##
##       N-hsa00-A activation   N-hsa00-C
##       N-hsa00-B inhibition   N-hsa00-C
##       N-hsa00-C activation N-hsa00-D E
##     N-hsa00-D E activation   N-hsa00-F
```

7.2.2 ATT File

The ATT file must fulfill the following requirements:

- Text file with twelve columns separated by tabulars.
- Each row represents a node (either simple or complex).
- The columns included are:
  - **ID**: Node ID as explained above.
  - **label**: Name to be shown in the picture of the pathway. Generally, the gene name of the first included EntrezID gene is used as label. For complex nodes, we juxtapose the gene names of the first genes of each simple node included (see genesList column below).
  - **X**: X-coordinate of the position of the node in the pathway.
  - **Y**: Y-coordinate of the position of the node in the pathway.
  - **color**: Default color of the node.
  - **shape**: Shape of the node. "rectangle" should be used for genes and "circle" for metabolites.
  - **type**: Type of the node, either "gene" for genes or "compound" for metabolites. For complex nodes, the type of each of their included simple nodes is juxtaposed separated by commas, i.e. gene,gene.
  - **label.cex**: Amount by which plotting label should be scaled relative to the default.
  - **label.color**: Default color of the node.
  - **width**: Default width of the node.
  - **height**: Default height of the node.
  - **genesList**: List of genes included in each node, with EntrezID:

- Simple nodes: EntrezIDs of the genes included, separated by commas (",") and no spaces, i.e. 1432,5880,842 for node N-hsa00-C.
- Complex nodes: GenesList of the simple nodes included, separated by a slash ("/") and no spaces, and in the same order as in the node ID. For instance, node N-hsa00-D E includes two simple nodes: D and E. Its genesList column is 5747,/,9047,5335, meaning that the gene included in node D is 5747, and the genes included in node E are 9047 and 5335.

- **tooltip**: Tooltip to be shown in the pathway visualization. HTML code may be included.

An example of ATT file as described above is shown here (hashtags must not be included in the file):

```
##               ID label   X  Y color     shape type label.cex label.color width
##        N-hsa00-A     A   0 40 white rectangle gene       0.5       black    46
##        N-hsa00-B     B   0  0 white rectangle gene       0.5       black    46
##        N-hsa00-C     C  50 20 white rectangle gene       0.5       black    46
##      N-hsa00-D E   D E 100 20 white rectangle gene       0.5       black    46
##        N-hsa00-F     F 150 20 white rectangle gene       0.5       black    46
##      height        genesList tooltip
##          17              998       A
##          17             5530       B
##          17    1432,5880,842       C
##          17 5747,/,9047,5335     D E
##          17              572       F
```

7.2.3 Pathway names file

The file including the real names of the patwhays must fulfill the following requirements:

- Text file with two columns separated by tabulars.
- Each row represents a pathway. First column is the ID of the pathway, and the second is the real name of the pathway.
- Only activation and inhibition interactions are allowed.

An example of ATT file as described above is shown here (hashtags must not be included in the file):

```
##
##  0 New pathway
```

8 Utilities

8.1 Functions

We have developed some simple functions to ease the use of data in *hipathia*.

8.1.1 hhead

Function `hhead()` has been conceived as a generalization of function `head()` to matrices, dataframes and SummarizedExperiment objects. It returns the values of the n first rows and columns of the matrix. In case the object is a SummarizedExperiment, it returns the values of the n first rows and columns of the (first) assay included in it. In case the object is not a matrix, dataframe or SummarizedExperiment object, it returns the result of applying function `head()` to the object.

```
class(brca)
## [1] "SummarizedExperiment"
## attr(,"package")
## [1] "SummarizedExperiment"
hhead(brca, 4)
##      TCGA.BH.A1FM.11B.23R.A13Q.07 TCGA.E2.A1LB.11A.22R.A144.07
## 2                      10.5317320                     9.732938
## 8647                   -3.3266788                    -3.457515
## 5244                   -0.3600828                    -1.139309
## 1244                    2.2876961                     1.724625
##      TCGA.BH.A208.11A.51R.A157.07 TCGA.BH.A18K.11A.13R.A12D.07
## 2                       9.7958036                    10.868669
## 8647                   -2.5261155                    -3.584934
## 5244                   -0.7368491                    -1.257797
## 1244                    1.0217356                     1.467979
```

```
class(assay(brca))
## [1] "matrix" "array"
hhead(assay(brca), 4)
##      TCGA.BH.A1FM.11B.23R.A13Q.07 TCGA.E2.A1LB.11A.22R.A144.07
## 2                      10.5317320                     9.732938
## 8647                   -3.3266788                    -3.457515
## 5244                   -0.3600828                    -1.139309
## 1244                    2.2876961                     1.724625
##      TCGA.BH.A208.11A.51R.A157.07 TCGA.BH.A18K.11A.13R.A12D.07
## 2                       9.7958036                    10.868669
## 8647                   -2.5261155                    -3.584934
## 5244                   -0.7368491                    -1.257797
## 1244                    1.0217356                     1.467979
```

8.1.2 get_path_name

The results object returned by function `hipathia()` includes the names of the subpathways. However, in case we need to transform subpath IDs to comprehensive subpath names, we can use `get_path_names()` function:

```
get_path_names(pathways, c("P-hsa03320-37", "P-hsa04010-15"))
## [1] "PPAR signaling pathway: HMGCS2" "MAPK signaling pathway: NFKB1"
```

8.1.3 get_paths_data

The `paths` object in the *MultiAssayExperiment* includes as assay a matrix with the level of activity of the signal in each subpathway. In order to extract the object of signal activity values from this object use function `get_paths_data()`. By default, this function returns a *SummarizedExperiment* object, but it can return just the matrix of subpaths values if parameter `matrix` is set to `TRUE`.

```
path_vals <- get_paths_data(results, matrix = TRUE)
path_vals <- get_paths_data(results)
hhead(path_vals, 4)
##               TCGA.BH.A1FM.11B.23R.A13Q.07 TCGA.E2.A1LB.11A.22R.A144.07
## P-hsa03320-37                    0.3971915                    0.3558425
## P-hsa03320-61                    0.2078705                    0.2107836
## P-hsa03320-46                    0.1250563                    0.1234348
## P-hsa03320-57                    0.1348015                    0.1330537
##               TCGA.BH.A208.11A.51R.A157.07 TCGA.BH.A18K.11A.13R.A12D.07
## P-hsa03320-37                    0.3757608                    0.3851692
## P-hsa03320-61                    0.1847616                    0.2032974
## P-hsa03320-46                    0.1242326                    0.1269767
## P-hsa03320-57                    0.1339136                    0.1645470
```

9 Pipeline before 'v3.0'

Since version `3.0`, we have simplified the pipeline including the data normalization step and the computation of the functional matrices into the `hipathia()` function, introducing function `DAcomp()` to perform differential activation comparisons of nodes, pathways and functions at the same time, and function `DAreport()` to create easily a report. All previous functions remain available for further use, and are explained in this section. However, we strongly recommend to use the new ones to simplify the use of the package.

9.1 Data scaling & normalization

Apart from the necessary bias corrections, the expression data matrix must be scaled between 0 and 1 before computing the subpaths activation values. Function `normalize_data()` is designed to this purpouse.

```
exp_data <- normalize_data(trans_data)
```

```
boxplot(trans_data)
```

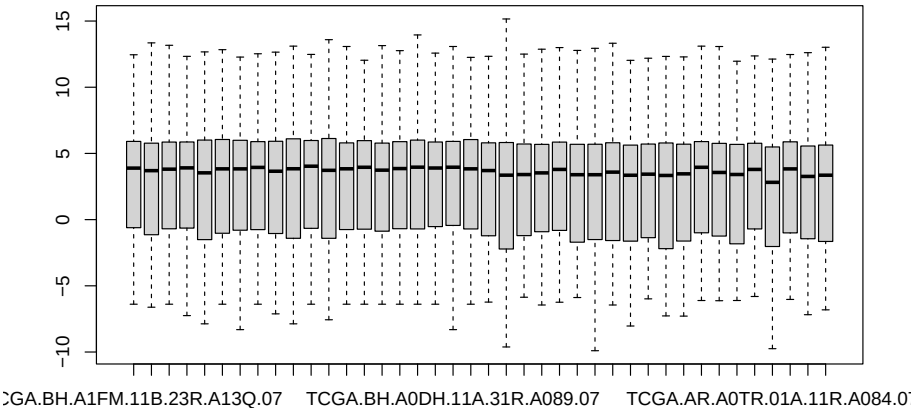


Figure 5: BRCA data before scaling

```
boxplot(exp_data)
```

Function `normalize_data()` includes different parameters for normalization. If option `by_quantiles` is `TRUE`, a previous normalization by quantiles is performed.

```
exp_data <- normalize_data(trans_data, by_quantiles = TRUE)
boxplot(exp_data)
```

Other parameters of this function affect the way in which scaling to the interval [0,1] is performed. Parameter `by_gene` indicates whether to perform the scaling to [0,1] to each row of the matrix. If the option `by_gene` is set to `TRUE`, the normalization between 0 and 1 is done for each row of the matrix, meaning that the expression of each gene will have a range between 0 and 1. If it is set to `FALSE`, the normalization is done for the whole matrix, meaning that only the genes with the maximum value of the matrix will have a normalized value of 1. It is recommended to keep it set to `FALSE`, as the default value.

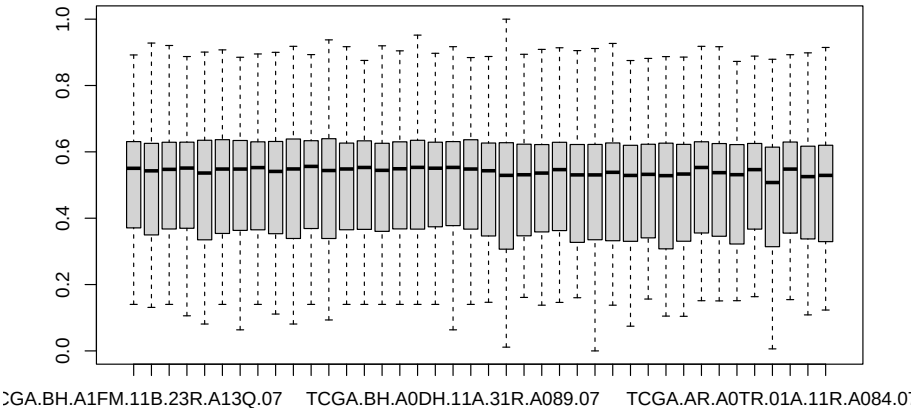


Figure 6: BRCA data after scaling

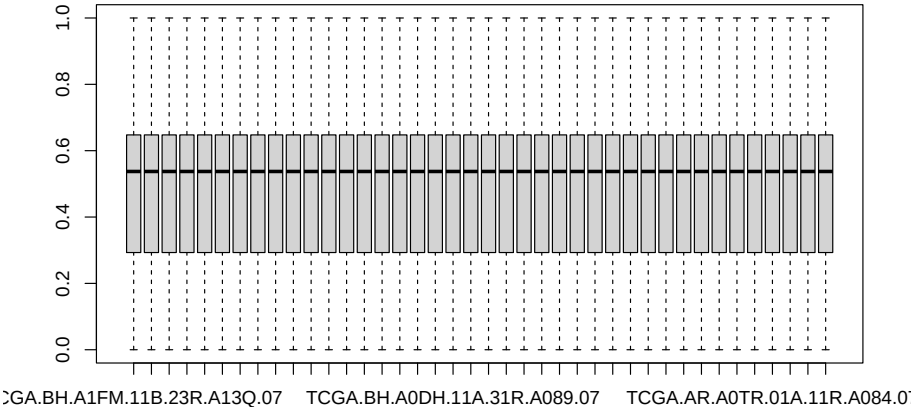


Figure 7: BRCA data after a Quantiles normalization

Parameter `percentil` indicates whether to use the percentil to compute the normalized value between 0 and 1. If it is set to `TRUE`, the function takes as a value for the position `(i,j)` of the matrix the percentil of sample `j` in the ditribution of gene `i`. If it is set to `FALSE`, the function applies a direct transformation from the original interval to [0,1]. It is recommended to keep it set to `FALSE` except for heavy-tailed distributions of the genes.

```
exp_data <- normalize_data(trans_data, percentil = TRUE)
boxplot(exp_data)
```

Parameter `truncation_percentil` gives the value of percentil `p` from which all further values are truncated to percentil `p`. Symmetrically, values beyond percentil `1-p` are also truncated to `1-p`.

```
exp_data <- normalize_data(trans_data, truncation_percentil = 0.95)
boxplot(exp_data)
```

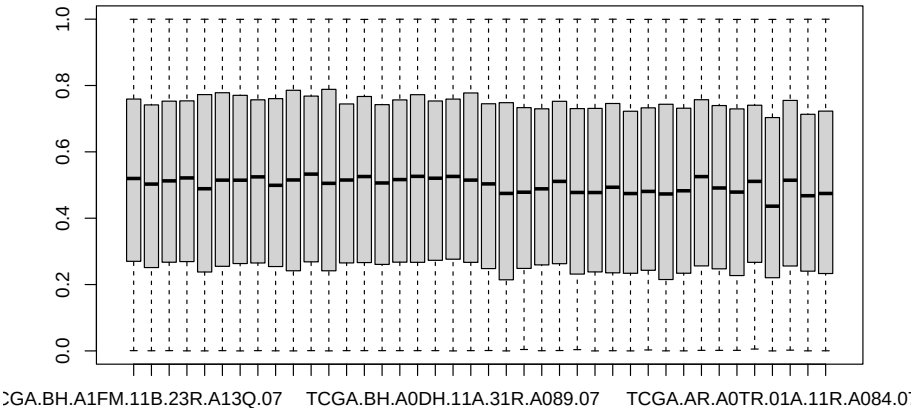


Figure 8: BRCA data after normalizing by percentil

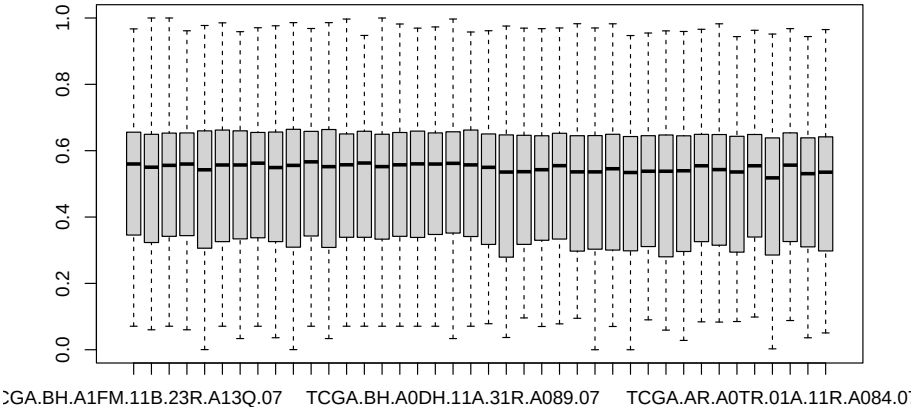


Figure 9: BRCA data after truncating by percentil 0.95

9.2 Function activation computation

Each effector protein of a pathway is responsible for performing a particular function. Thus, from the matrix of effector subpathways we can infer the functions matrix with the function `quantify_terms()`, which computes an intensity value for each molecular function and for each sample.

Different effector subpathways of different pathways may end in the same effector protein, and also different effector proteins may have the same molecular function. Therefore, for a particular function f, `quantify_terms()` summarizes the values of all the subpathways ending in an effector protein related to f with a mean value.

Different function activity matrices can be computed depending on the functional annotation given to the effector nodes. Function `quantify_terms()`, through parameter `dbannot`, accepts any annotation defined by the user and it has also two default annotations: Gene Ontology functions and Uniprot keywords. For further information on the differences between Gene Ontology and Uniprot keywords annotations please refer to this page.

```
uniprot_vals <- quantify_terms(hidata, pathways, dbannot = "uniprot")
## Quantified Uniprot terms: 142
go_vals <- quantify_terms(hidata, pathways, dbannot = "GO")
## Quantified GO terms: 1654
```

The result of this function is a data object with the level of activity of each annotated function for each sample. As before, the returned object is a *SummarizedExperiment*, unless parameter `matrix` is set to `TRUE` in which case a matrix is returned.

Notice that functions annotated to genes which are not included in any effector node will be not computed.

9.3 Two classes comparison

Once the object data of desired features has been computed, either subpath values or function values, any kind of analysis may be performed on it, in the same way as if it were the matrix of gene expression. Specifically, comparison of the features across different groups of samples is one of the keys. We can perform a comparison of two groups applying the Wilcoxon test using function `do_wilcoxon()`.

```
data(brca_design)
sample_group <- brca_design[colnames(path_vals),"group"]
comp <- do_wilcoxon(path_vals, sample_group, g1 = "Tumor", g2 = "Normal")
hhead(comp)
##                                        name UP/DOWN  statistic     p.value
## P-hsa03320-37 PPAR signaling pathway: HMGCS2    DOWN -2.2722075 0.022718728
## P-hsa03320-61  PPAR signaling pathway: APOA1    DOWN -1.1361037 0.264831806
## P-hsa03320-46  PPAR signaling pathway: APOA2    DOWN -2.5968085 0.008711881
## P-hsa03320-57  PPAR signaling pathway: APOC3    DOWN -2.4615581 0.013194383
## P-hsa03320-64  PPAR signaling pathway: APOA5    DOWN -0.5680519 0.583114228
##               FDRp.value
## P-hsa03320-37 0.03170055
## P-hsa03320-61 0.30557516
## P-hsa03320-46 0.01340289
## P-hsa03320-57 0.01884912
## P-hsa03320-64 0.62476524
```

Function `get_pathways_summary()` returns a summary by pathway of the results from the Wilcoxon test, summaryzing the number of significant up- or down-activated features.

```
pathways_summary <- get_pathways_summary(comp, pathways)
head(pathways_summary, 4)
##                         id_pathways num_total_paths num_significant_paths
## PPAR signaling pathway    hsa03320              42                   35
## ErbB signaling pathway    hsa04012              18                   10
## Fanconi anemia pathway    hsa03460               0                    0
## MAPK signaling pathway    hsa04010               0                    0
##                         percent_significant_paths num_up_paths percent_up_paths
## PPAR signaling pathway                      83.33            1            2.38
## ErbB signaling pathway                      55.56            1            5.56
## Fanconi anemia pathway                        NaN            0             NaN
## MAPK signaling pathway                        NaN            0             NaN
##                         num_down_paths percent_down_paths
## PPAR signaling pathway              34              80.95
## ErbB signaling pathway               9              50.00
## Fanconi anemia pathway               0                NaN
## MAPK signaling pathway               0                NaN
```

In order to visualize the results of the comparison, see Section 6.

9.4 Pathway comparison

The results of a comparison are sometimes difficult to summarize. An easy way to understand these results is to visualize them as an image. Function `pathway_comparison_plot()` creates an image of a pathway, with the same layout from KEGG, including a color code representing the significant up- and down-activated subpathways, and, if desired, the significant up- and down-regulated nodes.

```
pathway_comparison_plot(comp, metaginfo = pathways, pathway = "hsa03320")
```

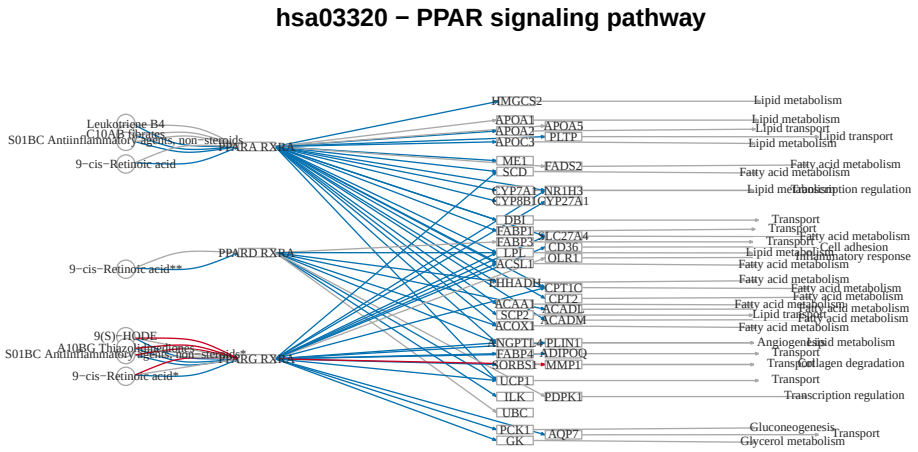


Figure 10: Pathway comparison plot without node colors

In these plots, colored edges represent significant subpathways. Edges belonging to subpathways which are significantly down-activated will be depicted in blue and those belonging to subpathways which are significantly up-activated will be depicted in red (as default). The *up* and *down* colors may be changed by the user through the parameter `colors` by giving a vector with three colors (representing down-activation, non-significance and up-activation respectively) or a color scheme (either *classic* or *hipathia*).

In order to visualize the effect of the nodes expression differences in the pathways, nodes can be colored by its differential expression. The color of each node with respect to its differential expression must be previously computed using function `node_color_per_de()`. Note that this fucntion computes differential expression on the nodes, not on the genes. It uses function `eBayes()` from package *limma*, see the package vignette for further information.

When computed, the resulting object must be provided to the `pathway_comparison_plot()` function as parameter `node_colors`.

```
colors_de <- node_color_per_de(results, pathways, sample_group, "Tumor",
                               "Normal")
pathway_comparison_plot(comp, metaginfo = pathways, pathway = "hsa03320",
                        node_colors = colors_de)
```

```
colors_de_hipathia <- node_color_per_de(results, pathways, sample_group,
                                        "Tumor", "Normal", colors = "hipathia")
```

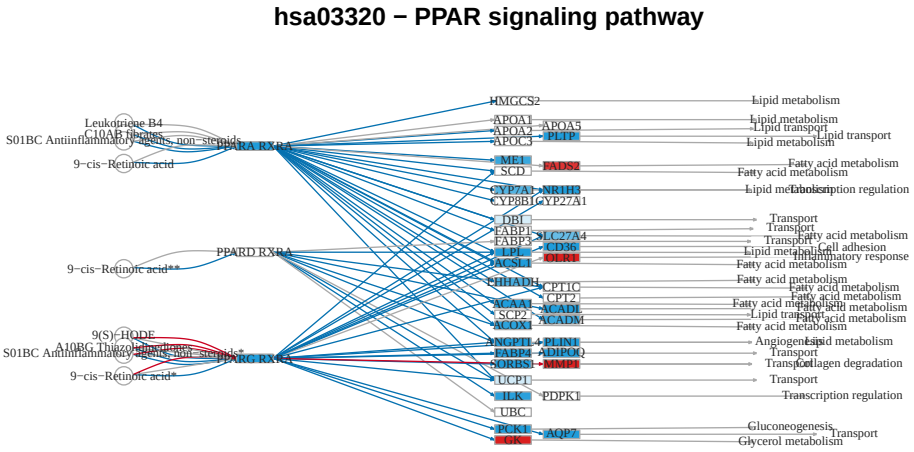


Figure 11: Pathway comparison plot with node colors: 'classic'

```
pathway_comparison_plot(comp, metaginfo = pathways, pathway = "hsa03320",
                        node_colors = colors_de_hipathia, colors = "hipathia")
```

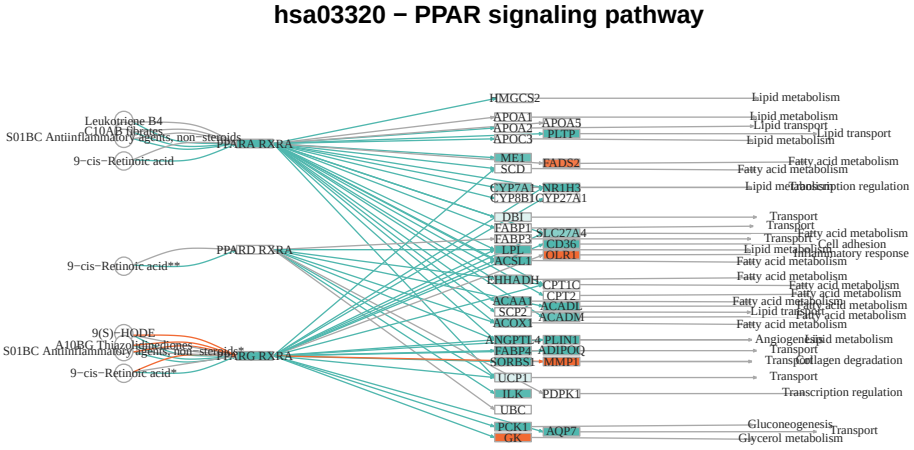


Figure 12: Pathway comparison plot with node colors: 'hipathia'

9.5 Heatmap

Function `heatmap_plot()` plots a heatmap with the values of the given data object. This object may be a SummarizedExperiment object or a matrix. The experimental design can be provided to assign a class to each sample by means of the parameter `group`. Notice that the classes must be in the same order as the columns of the provided matrix. One can select whether to cluster samples or variables setting parameters `variable_clust` and `sample_clust` to `TRUE`.

The colors of the different classes of samples can be selected through parameter `sample_colors` with a vector of colors named after the classes. The colors inside the heatmap can be also selected with parameter `colors`. Personalized colors can be provided as a vector, or preselected color schemes *classic* (default), *hipathia* or *redgreen* may be chosen.

```
heatmap_plot(path_vals, group = sample_group)
## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.

## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.
```

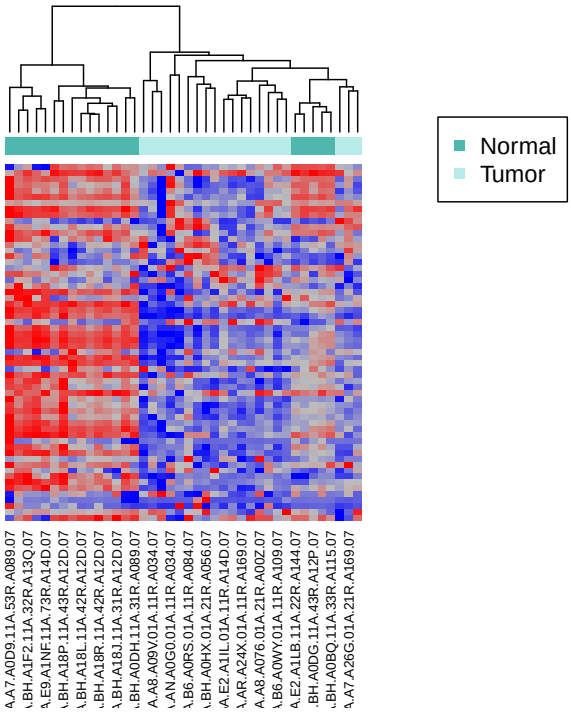


Figure 13: Heatmap plot

```
heatmap_plot(uniprot_vals, group = sample_group, colors="hipathia",
          variable_clust = TRUE)
## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.

## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.
```

```
heatmap_plot(go_vals, group = sample_group, colors="redgreen",
          variable_clust = TRUE)
## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.

## Warning: useNames = NA is deprecated. Instead, specify either useNames = TRUE
## or useNames = TRUE.
```

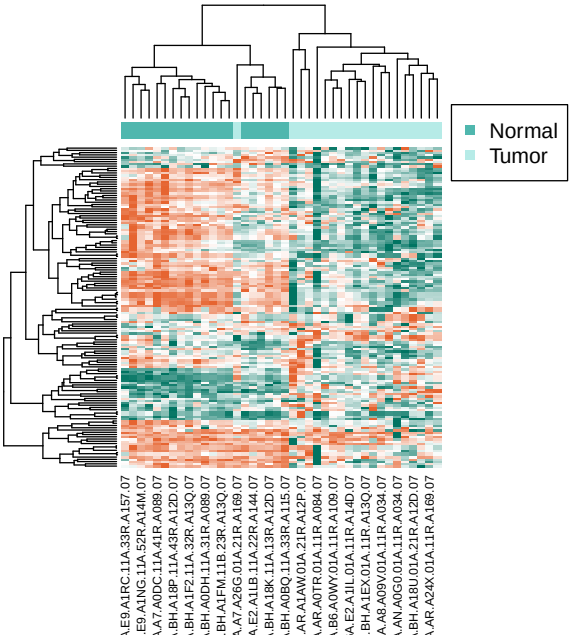


Figure 14: Heatmap plots with variable clustering

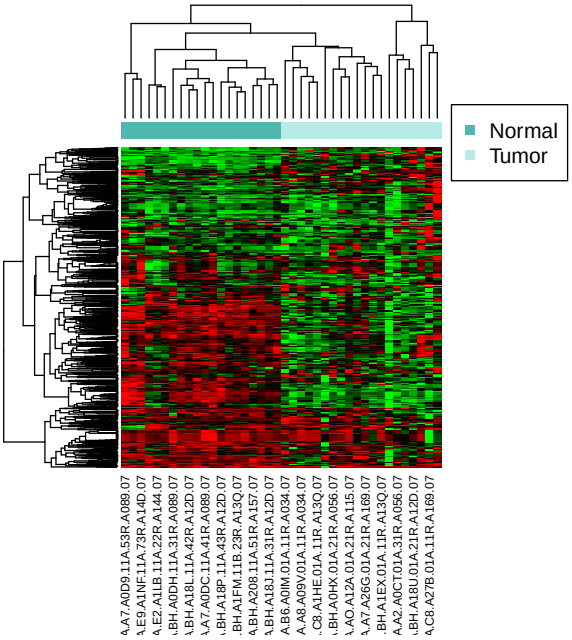


Figure 15: Different colors of heatmaps: 'redgreen'

9.6 Principal Components Analysis

Principal Components Analysis can be also performed by using function `do_pca()`. Notice that the number of rows must not exceed the number of columns of the input matrix.

```
ranked_path_vals <- path_vals[order(comp$p.value, decreasing = FALSE),]
pca_model <- do_pca(ranked_path_vals[1:ncol(ranked_path_vals),])
```

PCA models can be visualized with the function `pca_plot()`. Function `pca_plot()` plots two components of a PCA model computed with function `do_pca()`. The experimental design can be provided to assign a class to each sample by means of the parameter `group`. Notice that the classes must be in the same order as the columns of the matrix provided to the PCA model. The colors of the different classes of samples can be selected through parameter `sample_colors` with a vector of colors named after the classes. If no such parameter is provided, a predefined set of colors will be assigned. A main title may be given to the plot through parameter `main`. The components to be plotted can be selected through parameters `cp1` and `cp2` giving integer number. If parameter `legend` is set to TRUE, the legend will be plotted.

```
pca_plot(pca_model, sample_group, legend = TRUE)
```

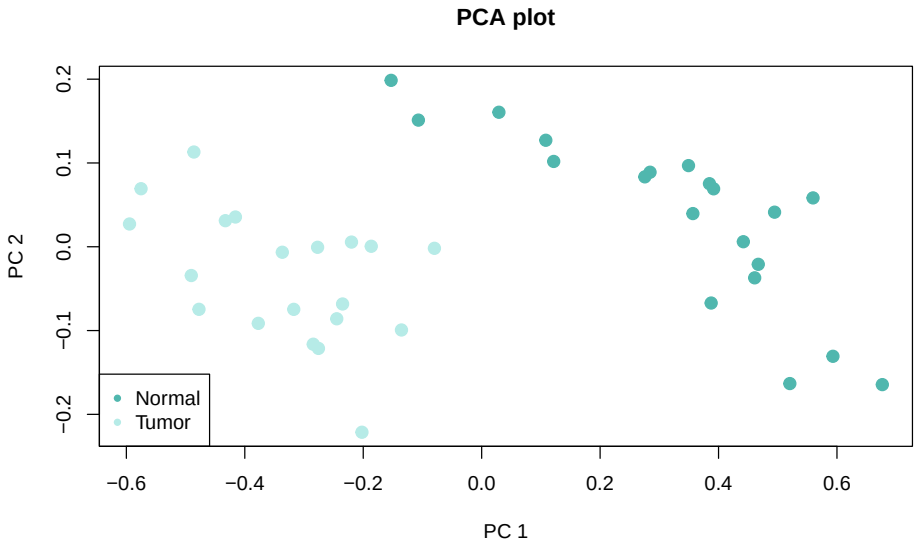


Figure 16: PCA plot

```
pca_plot(pca_model, group = rep(1:5, 8), main = "Random types",
      legend = TRUE)
```

Function `multiple_pca_plot()` plots n PCA components given by parameter `comps=n` as an integer vector. By default, $n = 3$. As before, the experimental design can be provided to assign a class to each sample by means of the parameter `group`. Notice that the classes must be in the same order as the columns of the matrix provided to the PCA model. The colors of the different classes of samples can be selected through parameter `sample_colors` with a vector of colors named after the classes. If no such parameter is provided, a predefined set of colors will be assigned. The cumulative explained variance can be represented by setting `plot_variance` parameter to TRUE. If parameter `legend` is set to TRUE, the legend will be plotted. A main title may be given to the plot through parameter `main`.

```
multiple_pca_plot(pca_model, sample_group, cex=3, plot_variance = TRUE)
```

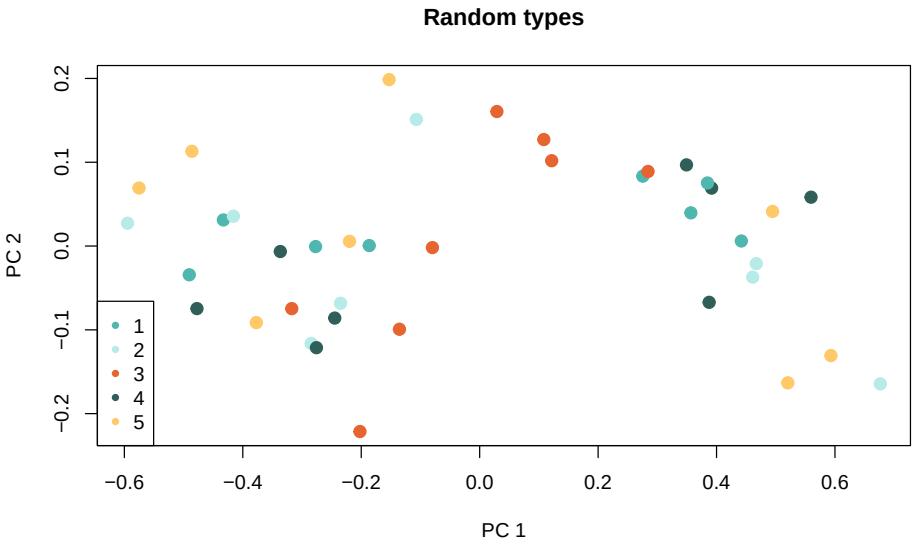


Figure 17: PCA plot with 5 random colors

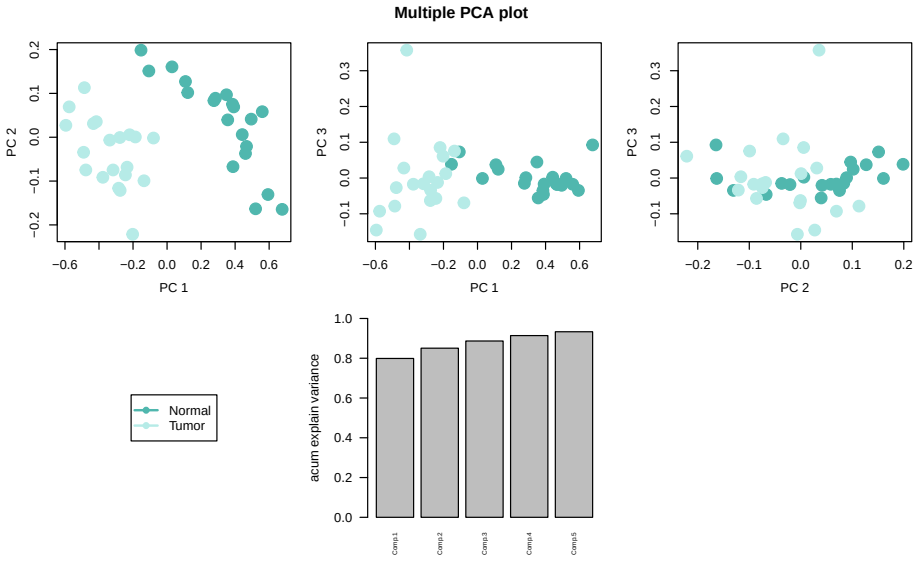


Figure 18: Multiple PCA plot with acumulated explained variance