SetRank User Manual

Cedric Simillion

December 9, 2015

Contents

1 Introduction

SetRank is an advanced Gene Set Enrichment Analysis (GSEA) package. GSEA methods detect groups of genes that occur significantly more than expected by chance in a given list of genes. These groups of genes are very often genes that function together in the same biological process or pathway. Hence, many people refer to GSEA as pathway analysis. The latter name can however be misleading as there are many other computational methods to study biological pathways, such as network analysis. Moreover, GSEA is not only limited to detecting pathways but can be used to find any type of common characteristic between a group of genes, such as functional domains or localization in a tissue or cell.

The purpose of GSEA is to find general trends in the huge lists of genes or proteins that are generated by many modern functional genomics techniques. Performing a GSEA has become the typical first step in analysing these datasets. The goal of this approach is to help identify which genes are involved in the processes or mechanisms relevant to the biological question at hand. These genes can then be examined further on in greater detail while the rest can, for the time being, be ignored.

Many of the presently available GSEA methods however suffer from several shortcomings. The output of many algorithms is highly dependent on arbritary parameters such as the p-value cutoff applied when defining a list of significant genes. Moreover, results often contain many false positive entries, as a result of the overlap between many pathways and of bias introduced by the sample source. SetRank was developed specifically to address all these issues.

The exact details of the algorithm are described elsewhere. We need however to introduce a few concepts that are important to understand how SetRank works. The first one is a *gene set*, which is simply an unordered group of genes. As the name suggests, a gene set is the basic data structure underlying SetRank as well as any other GSEA method. Thus, when we want to detect pathways using SetRank, we first need to represent each of them as a gene set. We could be tempted to refer to gene sets as pathways but, as we have mentioned above, the latter name is too restrictive as we can also group genes based on other characteristics such as molecular function or cellular location. We shall therefore use the term *gene set* rather than pathway in the rest of this document.

A second concept is a *gene set collection* which refers to the ensemble of all gene sets used in a single analysis. Each gene set in a gene set collection will be tested for its significance when SetRank is run.

A *gene set database* is an external datasource that provides gene set descriptions. Examples are pathway databases such as Wikipathways, Reactome or KEGG or more general gene annotation databases such as the Gene Ontology annotation provided by Uniprot. As we shall see below, you can also define your own gene set database, if necessary. One of the strengths of SetRank is that the gene sets in a set collection can come from multiple gene set databases. Including multiple databases in one collection allows you to search these databases simultaneously and integrate all results together. The opposite approach, querying each gene set database separately, would produce different result sets rendering the interpretation afterwards a lot harder.

This manual describes the R/Bioconductor version of SetRank package, meaning that at least a basic understanding of the R programming language is required to use it. In this manual, we will explain all the steps required to install and use SetRank in detail to keep the amount of needed expertise to a minimum. If you are not familiar at all with

R or simply too impatient, you can also try the web-interface of SetRank available at `http://setrank.vital-it.ch`.

2 Installation

2.1 System requirements

In order to use SetRank, you need:

- A system running Linux, Mac OS X or any other Unix-based operating system. While SetRank will run on Microsoft Windows, you will not be able to run it multi-threaded which will severely impact computation times. See section 4.2.3 on page 16 for more details.
- R version 3.0 or above.
- A decent amount of RAM. It is hard to specify a lower limit since the amount of memory needed by SetRank depends on both your dataset and the number of CPU cores you want to use. As a rule of thumb, you need about 4GB to run SetRank on a single core and about 2GB for each additional core you want to use.

2.2 Installation steps

SetRank is distributed as a CRAN package. The easiest way to install it is to fetch the package directly from Bioconductor by opening an R session and typing the following commands:

```
source("http://bioconductor.org/biocLite.R")
biocLite("SetRank")
```

Alternatively, you can download and install the package file yourself. Open a Terminal window on your system and type

```
curl http://bioconductor.org/SetRank_0.6.0.tar.gz
```

to download the package.. Use this commands to install:

```
R CMD INSTALL SetRank_0.6.0.tar.gz
```

To make sure the installation was successful, type the following command in R:

```
library(SetRank)
```

If you get an error that says "`Error in library(SetRank) : there is no package called 'SetRank'`" then this means that the installation was not successful. If you don't get any messages, then the installation was successful and you can start using SetRank.

3 Preparing a set collection

Before you can run the SetRank on a dataset, you first need to build a set collection object. The `SetRank` package defines a function called `buildSetCollection` which takes care of this task. There are two important pieces of information you need to have ready though before you can build a gene set collection. First, you need one or more gene annotation tables which are used to define the gene sets. Next, you need to specify a reference set specific to your dataset. We will discuss the specifics of both elements in detail below.

3.1 Annotation tables

An annotation table is simply a data frame object that maps gene identifiers to one or more gene set identifiers and provides some additional information, such as a name and a description, about each gene set. Gene sets are created by taking all genes that map to the same gene set identifier and attaching the additional information to them.

There are several ways of obtaining annotation tables. You can either use some functions built in the `SetRank` package or use the `GeneSets` package to build a large collection of organism-specific annotation tables. Apart from these two options, you can also provide your own annotation table.

3.1.1 Using the GeneSets package

As mentioned before, there are many other publicly available resources that can be used as gene set databases. Unfortunately, the information in some of these databases may not be redistributed freely. To solve this problem, a stand-alone package, called `GeneSets`, was created. This package is a collection of scripts that download gene set information from various online resources and create sets of organism-specific annotation tables, conveniently made available as ready-to-use R-packages.

3.1.1.1 System requirements

The GeneSets package requires:

- A system running Linux, Mac OS X or any other Unix-based operating system.
- Perl version 5.12 or above, with the `Date::Format` CPAN package[1] installed.
- Python version 2.7 or above
- GNU make version 3.81 or above[2]
- R version 3.0 or above, with the `GO.db` Bioconductor package installed[3]
- The `wget` utility

[1] available from `http://search.cpan.org/~gbarr/TimeDate/lib/Date/Format.pm`

[2] For Perl, Python and GNU make, earlier versions will probably work as well, but these were not tested.

[3] Installed automatically as a dependency of the `SetRank` package

3.1.1.2 Installation The GeneSets package is available as a git repository at `https://github.com/C3c6e6/GeneSets`. If you are using the git version control manager, you can simply clone the repository to your own system using

```
git clone https://github.com/C3c6e6/GeneSets.git
```

Alternatively, you can download a `.zip` or `.tar.gz` archive from `https://github.com/C3c6e6/GeneSets/releases/latest`. All you have to do is to download the archive into an empty directory and extract its contents.

3.1.1.3 Building the annotation packages. The only thing you have to do is tell the package for which organisms you which to build annotation tables. In the root of the repository or archive, there is a file called `organisms`. Open this file with your favorite text editor and add the names of the organism(s) you want to build annotation tables for and remove any names you're not interested in. Make sure there is only one name per line and that the lines do not contain any leading or trailing white space characters. Also, make sure that the names you are using correspond exactly to the official names used by the NCBI taxonomy database, as this is the reference used by the `GeneSets` package. Using the correct name is especially important when working with bacterial strains and substrains. For instance, if you want to create annotation tables for *Escherichia coli* K12, substrain MG1655, you should write exactly:

```
Escherichia coli str. K-12 substr. MG1655
```

When in doubt, you can look up the name of your species on the NCBI website at `http://www.ncbi.nlm.nih.gov/Taxonomy/TaxIdentifier/tax_identifier.cgi` and simply copy-and-paste the name into your editor.

Once you've edited the `organisms` file, you are ready to build the packages. All you have to do is open your UNIX-shell, go to the directory where you've installed the package and type:

```
make
```

Issuing this command will start the build process, which might take a while to complete, mainly depending on your network connection speed.

Once the process is finished you should see, for each organism listed in the `organisms` file, a file appear named `GeneSets.`*`Genus.species`*`_`*`YY.MM.DD`*`.tar.gz`, with *`Genus.species`* the name of the organism and *`YY.MM.DD`* the date the file, e.g. `GeneSets.Homo.sapiens_13.10.18.tar.gz`. Each of these files is an R package – the date string functions as version number – that is ready to install.

To install several packages at once, you can use the following bash command:

```
for f in *_YY.MM.DD.tar.gz; do R CMD INSTALL $f; done
```

Of course, you have to replace *`YY.MM.DD`* with the actual date string.

3.1.1.4 Updating the annotation packages To keep your collection of annotation table packages up-to-date, all you have to do is to regularly run the `make` command. The `GeneSets` package will automatically detect if the gene set annotation data for a given organism has changed compared to the last time you ran `make` and create an updated version if necessary. Don't forget to install the updated packages afterwards.

3.1.1.5 Loading the annotation packages in R To start using the annotation tables created with `GeneSets`, simply load the desired package into R, e.g.:

```
library(GeneSets.Homo.sapiens)
```

Note that you don't have to specify the date/version string when loading a library. To see which annotation tables are provided, you can use the command:

```
ls("package:GeneSets.Homo.sapiens")
```

which will show the names of the available annotation tables, like this:

```
 [1] "BIOCYC"          "ENCODE"
 [3] "GOBP"            "GOCC"
 [5] "GOMF"            "ITFP"
 [7] "KEGG"            "Pathway_Interaction_Database"
 [9] "PhosphoSitePlus" "REACTOME"
[11] "WikiPathways"    "allDBs"
```

The actual table names may vary between different organisms, but the "`allDBs`" table will always be present as this contains the combination of all annotation tables.

3.1.2 Building your own annotation table

A third way of supplying gene set databases to SetRank is to define your own. This could be a set of pathways that your lab specializes in or another gene set database not provided by GeneSets. All you have to do is to create a data frame object in R that maps gene identifiers to one or more gene set descriptions. The object should contain the following columns:

geneID The identifier of the gene. This can be any type of identifier but bear in mind that that the annotation tables provided by SetRank and the GeneSets packages use Entrez Gene identifiers. Thus, if you want to combine your own annotation with the these predefined annotation tables, you need to use Entrez identifiers as well.

termID A unique identifier of the gene set. Make sure that it is unique to all gene set databases you plan to use.

termName A name for your gene set, usually a very short description, e.g. "glycolysis".

description A longer description of the gene set.

dbName The name of the gene set database the annotation is retrieved from. If you are compiling your own set of pathways, you can give it any name you like, of course.

3.2 Creating a reference set

3.2.1 Understanding sample source bias

Now that we have prepared our gene set database(s), it is time for the next step, preparing a reference gene set. Although this step is, strictly speaking, not necessary, it is

very important to avoid an artefact in your results that we call sample source bias. This type of bias can best be explained with an example. Suppose we want to perform GSEA on a transcriptomics dataset that compares peripheral blood mononuclear cells (PBMCs)[4] from diseased patients to those of healthy donors. The obvious aim of this analysis would be to know which biological processes are affected by the disease under investigation. Consequently, we want the GSEA method that we are using to only list processes relevant to the disease. Many GSEA algorithms however, will look for gene sets that are over-represented in the list of differentially expressed genes *compared to the entire genome*. The problem with this approach is that the list of gene sets returned will unavoidably contain gene sets that are unrelevant for the disease but simply characterise PBMCs compared to other cell or tissue types. The reason for this artefact is that the set of genes that are differentially expressed between the conditions tested will contain a number of genes that are only expressed in PBMCs and maybe a few related cell types but not in unrelated tissues such as lung or brain.

To avoid sample source bias in our example, we need to look for over-representation of gene sets in comparison to a *sample-specific reference set of genes*. Put in simpler words, we should not ask the question "*What is special about this set of genes?*" but we should ask "*What is special about this set of* PBMC *genes*".

3.2.2 Constructing reference sets for different types of data

In an ideal world, we would know for each developmental stage, tissue or cell type the entire list of genes that could be expressed and use this as a reference set. As this kind of knowledge will not be available for the forseeable future, we need a more pragmatic way to define reference sets. A convenient approach is to create a reference set by simply taking the union of all genes or proteins observed in all samples of your dataset. Below we will provide some detailed examples for specific data types.

3.2.2.1 RNA-seq data As RNA-seq technologies detect absolute transcript levels, we can create a reference set by taking the union of all transcripts that are detected at least once in our entire dataset. Most tools that calculate differential expression from RNA-seq data, such as `edgeR` or `DESeq`, return an output table in TAB-delimited format that looks like this:

geneID	counts_1	counts_2	log2FoldChange	pval	padj
ENSMUSG00000026208	78697	307	-7.444850797	0	0
ENSMUSG00000029321	8762	17	-8.5325216767	0.00E+00	2.73E-305
ENSMUSG00000034126	38479	712	-5.9106519753	8.26E-235	5.28E-231
ENSMUSG00000085862	4043	16	-7.9664275401	7.44E-229	3.57E-225
ENSMUSG00000061392	3169	6	-8.7073895562	9.62E-224	3.69E-220
ENSMUSG00000039323	21061	375	-5.8138818971	4.73E-217	1.51E-213
ENSMUSG00000058638	23305	457	-5.6433154574	9.56E-210	2.62E-206
ENSMUSG00000041328	2802	8	-8.503592771	5.47E-208	1.31E-204
ENSMUSG00000032207	7493	36	-6.3558689651	1.42E-206	3.02E-203

Suppose you have in a subdirectory called "`RNAseq_results`" of our working directory a set of files with the extension `.txt` that contain the results for the different com-

[4] PBMCs are any type of blood cell with having a round nucleus – as opposed to a lobed nucleus – and include lymphocytes monocytes and macrophages. PBMCs are widely used in research and clinical applications.

parisons, all formatted as the example above. You can then read these in using the following bit of R code:

```
topTableFiles = Sys.glob("RNAseq_results/*.txt")
topTables = lapply(topTableFiles, read.table, sep="\t", header=TRUE,
        comment.char="", quote="", stringsAsFactors=FALSE)
names(topTables) <- sub("\\.txt", "", basename(topTableFiles))
```

These commands will create a list object containing as many data frames as tables read. Each of these data frames is has the same name as its original file, but with leading directory name and trailing extension stripped from it.

Now we need to obtain for each of these data frames the list of all gene identifiers for which at least one transcript was identified. We could do this by searching each of the columns with sequence-count values for non-zero values but most RNA-seq analysis tools allow us to take a short cut. If you look in any of the dataframes for a gene with zero sequence counts in all samples, you will notice that p-value and adjusted p-value are assigned the `NA`[5] value. Thus, we can simply filter out any rows with `NA` values in adjusted p-value column, like so:

```
topTables = lapply(topTables, function(t) t[!is.na(t$padj),])
```

You might have to replace `padj` with the name given to the adjusted p-value column by your RNAseq tool of choice. Now we can take the union of all genes with at least one transcript detected:

```
allGenes = unique(unlist(lapply(topTables, function(t) t$geneID),
        use.names=FALSE))
```

The only problem left is that the gene identifiers used in the example file are Ensembl gene IDs whereas gene set collections used by `SetRank` typically use NCBI Entrez Gene identifiers. So, before we can do something useful with them, we have to convert the identifiers in the `allGenes` vector to the latter type[6]. For this task, you can use `createIDConverter`, a convenience function provided by the `SetRank` package. This function takes as input the name of an `organismDBI` package and the types of identifiers to convert between. Rather than directly converting between identifiers, `createIDConverter` returns a new function which do the actual conversion. In the case of our sample input file show above, you need to convert the mouse Ensembl gene identifiers into Entrez gene identifiers. This example shows how it works:

```
library(SetRank)
ensembl2EntrezID = createIDConverter("Mus.musculus", "ENSEMBL",
        "ENTREZID")
referenceSet = ensembl2EntrezID(allGenes)
```

We have put the library command that loads the SetRank module within this snippet but it is considered good practice to put all these statements at the top of your script. Also, you will need to replace `"Mus.musculus"` and `"ENSEMBL"` with respectively the correct name of the `organismDBI` module and type of identifier used in your input file. See appendix B on page 26 for more information on this topic.

[5] `NA` is R's way of indicating missing or incalculable values.

[6] Of course, if you are using a gene set collection that uses a different type of identifier, you will have to convert to the latter instead

3.2.2.2 Shotgun proteomics data With shotgun proteomics data we mean any non-targeted proteomics mass spectrometry technology that tries to identify as many proteins as possible in a given sample. Just as with RNA-seq data, we can construct a reference set by taking the union of all proteins detected in all samples of a dataset.

To get the set of detected proteins, you will need the output from either search engines such as MASCOT or X!Tandem or from quantitive proteomics tools such as MaxQuant in TAB-delimited format. Most of these tools allow you to export their results in one or more TAB-delimited files or as a spreadsheet. In the latter case, you will need to open the file first in your spreadsheet program and export the data in TAB-delimited text format. If your proteomics tool of choice does not provide any option to export the results in tabular format, you use the open-source tool PeptideShaker. This tool allows you to import the results from different search engines and, amongst other things, export them in TAB-delimited format.

Suppose we now have a directory called "`proteomics_data`" which contains a number files with the extension `.txt`, each of them listing the proteins identified in a specific sample. We will assume for now these files are all exported with PeptideShaker, meaning they look like this:

```
Accession  Other Protein(s)        Complete Protein Group
P11021                             P11021
Q15149                             Q15149
P49327                             P49327
P62736     P63267, P68032, P68133  P62736, P63267, P68032, P68133
P29401                             P29401
P63104                             P63104
P68363     Q71U36, Q9BQE3          P68363, Q71U36, Q9BQE3
P27797                             P27797
P68363     P68366                  P68363, P68366
```

Only the first three columns of the PeptideShaker output are shown here as these are the only ones relevant to us now. Typically, proteomics results files list UniProt identifiers and PeptideShaker is no exception to this rule. If your software happens to also export gene identifiers you can use these directly of course, but try to avoid the using gene symbols. See appendix on page 26 for more information on the different types of sequence identifiers. For now, we will assume we can only rely on the provided UniProt identifiers.

As you can see, some lines contain multiple protein identifiers. These are cases where no single protein could be unambiguously identified meaning that our sample contains at least one of the listed proteins. To take these ambiguous cases into account, we must parse the comma-separated lists of identifiers in the column labeled "`Complete Protein Group`". The following bit of R code takes care of this task:

```
searchResultFiles = Sys.glob("proteomics_data/*.txt")
searchResults = lapply(searchResultFiles, read.table, sep="\t",
        header=TRUE, quote="", fill=TRUE, stringsAsFactors=FALSE)
names(searchResults) <- sub("\\.txt", "",
        basename(searchResultFiles))
allProteins = unique(unlist(lapply(searchResults, function(t)
        strsplit(t$Complete.Protein.Group, ",\\s+")),
        use.names=FALSE))
```

Just a with our RNA-seq example, we first create a named list of tables corresponding to our different input files. We then use the built-in R function `strsplit` to split the contents of the column "`Complete Protein Group`" into separate Uniprot identifiers. After calling the `unique` function, we obtain the variable `allProteins` which contains the UniProt identifiers of all proteins found in all samples in our dataset. The only thing left to do is to convert these to Entrez Gene identifiers. For this step, we can again use `createIDConverter` to create function that will do this conversion for us. Applying the latter to `allProteins` then gives us our reference set for our proteomics dataset.

```
library(SetRank)
uniprot2EntrezID = createIDConverter("Homo.sapiens", "UNIPROT",
        "ENTREZID")
referenceSet = uniprot2EntrezID(allProteins)
```

3.2.2.3 Microarray data Contrary to RNA-seq and mass spectrometry proteomics technologies, where the presence of a transcript or peptide is determined in absolute terms, RNA microarrays only allow to determine relative changes in expression intensity between different sample groups. As a result, constructing a reference set is a bit more tricky for microarray datasets. We suggest to define the reference set for this type of data as the set of genes which vary significantly across any of the conditions in the experiment. The rationale for this approach is that if a gene varies significantly across different conditions, it should be expressed in at least one of these. The drawback is that genes which are expressed at a constant level across all conditions, will not be included in the reference set.

Conveniently, the `limma` package, which is widely used to analyse microarray data in R, provides exactly the function we need. The `topTableF` function performs a type of one-way ANOVA for each gene in the dataset. We will assume that you have already fitted a linear model on your data using the `lmFit` function from the `limma` package. For details on how to use the latter function, we refer to the documentation of the `limma` package.

Assuming we have a variable `fittedData` which contains the output of the `lmFit` function, calling the `topTableF` function is straightforward:

```
fTable = topTableF(fittedData, number=Inf, p.value=0.05)
```

The `number` argument specifies the maximum number of genes we want to have returned. It is important to set this to infinity (`Inf`), as we want the list of returned genes only to be limited by the p-value. The `p.value` argument specifies cutoff to be applied on the adjusted p-values. As we are constructing a reference set, we want to avoid false negatives rather than false positives, so we can apply a permissive cutoff such as 0.05 or even 0.1. Here is what the relevant columns in the returned `fTable` data frame look like:

```
PROBEID    ENTREZID  SYMBOL  GENENAME
81641_at   81641     Anpep   alanyl (membrane) aminopeptidase
56611_at   56611     Anxa2   annexin A2
25661_at   25661     Fn1     fibronectin 1
445415_at  445415    S100a11 S100 calcium binding protein A11
64193_at   64193     Pttg1   pituitary tumor-transforming 1
25211_at   25211     Lyz2    lysozyme 2
```

We are in luck here, as the data frame already contains a column containig Entrez Gene identifiers. All we have to do is remove any missing (NA) values from that column and we have our reference set, like so:

```
referenceSet = fTable[!is.na(fTable$ENTREZID),]$ENTREZID
```

The actual type of sequence identifiers include in the output of topTableF depends on how the raw microarray data was processed and can thus vary between datasets. However, most of the times, one or more types of gene identifiers are provided. As we have seen in the previous examples, a function to convert these to Entrez identifiers is readily created with the `createIDConverter` function. Additionnally, most BioConductor `AnnotationDBI` packages also allow to convert probe set identifiers directly to Entrez identifiers. See appendix B on page 26 for more information on this topic.

3.2.2.4 Targeted technologies With targeted technologies, we mean any technology that determines the expression level of a limited set of proteins or transcripts. Examples of these technologies are targeted MS/MS proteomics, NanoString nCount or RT-qPCR.

Of course, a dataset should contain a sufficiently large number of genes or proteins for a SetRank analysis to make sense. It is hard to define an absolute minimum but once you have something in the order of 100 genes or proteins, you should be fine. There is of course no harm in trying to run SetRank on smaller datasets. All you risk is obtaining an empty result set.

Defining a reference set for this type of analysis is straightforward, simply take the entire set of genes you are measuring. As always, make sure to convert all identifiers to Entrez Gene identifiers (see appendix B on page 26).

3.2.2.5 Gene groups Many bioinformatics analyses return discrete groups of genes. Clustering algorithms, for instance, detect groups of co-expressed genes or proteins in large expression datasets or communities of closely interacting genes in large gene networks. Other analyses predict transcription factor targets, kinase substrates or any other of type of molecule interacting with a given gene.

With each of these analyses, the typical next step is to perform GSEA on each of the resulting groups. In most cases, the best approach is to use the union of all returned groups as the reference set.

3.3 Creating the set collection object

Now that we have our annotation tables and reference set ready, we can finally build our set collection object. This step is very simple, all we have to do is to call the `buildSetCollection` function provided by the `SetRank` module. This function is used as follows:

```
buildSetCollection(..., referenceSet = NULL, maxSetSize = 500)
```

The `...` argument contains one or more annotation tables, as described in section 3.1 on page 5. When you are using a `GeneSets` package, you can either use the `allDBs` table or any combination of the individual tables representing single gene set databases.

The `referenceSet` argument is the reference gene set, as described in section 3.2 on page 7. When omitted, the union of all genes present in all supplied annotation tables will be used as reference set. Using this default option is almost certainly a bad idea and should be avoided.

The `maxSetSize` argument specifies the maximum number of elements in a gene set. Any gene sets present in the collection with more elements than this number will be ignored during the analysis. This limitation is to both speed up the analysis and to avoid large The default value of 500 is recomended, but feel free to experiment with this value.

Because `buildSetCollection` pre-calculates all intersections between any two gene sets in the collection, as well as some critical values, it takes quite a while to run. Fortunately, its execution time can be sped up by using multiple CPU-cores. The number of cores to use can be set before running `buildSetCollection` using the command:

```
options(mc.cores=4)
```

It is not recommended to use more than 16 cores.

Once we have our collection, we can save it so that we can use it in our later analysis. The code snippet shows a sample run of `buildSetCollection`, with the reference set construction omitted.

```
library(SetRank)
library(GeneSets.Homo.sapiens)

# ...
# (build your reference set, store it in variable "reference")
# ...

options(mc.cores=4) #Change this if you want to use more or less cores
collection = buildSetCollection(GOBP, KEGG, Reactome, Wikipathways,
        referenceSet=reference, maxSetSize = 500)
save(collection, file="collection.Rda")
```

Here we have chosen to only use the GO Biological Process (GOBP), KEGG, Reactome and Wikipathways databases. Remember that if we want to use all gene set databases present in the `GeneSets.Homo.sapiens` package, we can simply use the `allDBs` table, like this:

```
collection = buildSetCollection(allDBs, referenceSet=reference,
        maxSetSize = 500)
```

4 Running SetRank

4.1 Ranked vs. unranked analysis

Now that we have our dataset ready, we can finally perform our SetRank analysis. An important decision to make is if we want to run SetRank in ranked or unranked mode. The difference between these two modes lies in how the input vector of gene identifiers (the `geneIDs` argument) is treated and how the primary gene set p-values are calculated.

In ranked mode, the gene identifiers in the input vector are assumed to be ranked by increasing p-value. The p-value of a gene set is then calculated by checking if its genes rank higher than what would be expected by chance. The main reason for choosing ranked mode is that it does not require you to rely on an arbitrary p-value cutoff to divide your input data set into a set of significant and one of non-significant genes. Although the latter approach is widely used, it has been shown that the results of GSEA vary greatly depending on the applied cutoff.

An additional advantage of the way gene set p-values are calculated in ranked mode, is increased sensitivity. It is possible to obtain a significant p-value for a gene set although none of its individual genes have a p-value that would normally be considered significant. As a result, you can even detect significant gene sets in a dataset where no single gene is found to be significant. Because of these reasons, SetRank should always be run in ranked mode whenever the genes in your input dataset have p-values, or a similar score that reflects significance, associated to them.

In unranked mode, the significance of a gene set is calculated by comparing its relative abundance in the input vector compared to the remainder of the reference set. The more abundant the gene set is in the input vector, the lower the obtained p-value will be. Unranked mode should only be used when the input gene identifiers cannot be ranked in a meaningful way. Examples of such cases include qualitative proteomics datasets or results from clustering analyses.

4.2 Running your analysis

4.2.1 Single input file

In the simplest case, your dataset only consists of a single input file, typically the comparison results of a set of treatment samples to a set of control samples. Assuming that our file is called `results.txt` and that it is formatted in the same way as the example in 3.2.2.1, we can read the data in like this:

```
inputData = read.table("results.txt", sep="\t", header=TRUE,
        comment.char="", quote="", stringsAsFactors=FALSE)
```

Remember that the input data for SetRank is a vector of NCBI Entrez Gene identifiers, ranked by increasing p-value. When your input file already contains Entrez Gene identifiers, you have to make sure R treats them as character values and not as integer numbers. You can force the `read.table` function to read in a certain column as character values using the `colClasses` argument. When the column containing the Entrez identifiers is called `geneID`, simply add the argument

```
colClasses=c(geneID="character")
```

to the `read.table` call.

Just as with the construction of the reference set, when your input file uses a different type of gene identifier, such as Ensembl gene identifiers, you need to convert these to Entrez Gene identifiers. Again, we can use the `createIDConverter` convenience function provided by the `SetRank` package for this task. To add a new column with the Entrez Gene IDs to the `inputData` data frame, you can use

```
ensembl2Entrez = createIDConverter("Homo.sapiens", "ENSEMBLID",
```

```
        "ENTREZID")
inputData$entrezID = ensembl2Entrez(inputData$geneID, na.rm=FALSE,
        drop.ambiguous=TRUE)
```

The `na.rm` and `drop.ambiguous` arguments are added to ensure that the vector returned is of the same length as the input vector. To finally start the SetRank analysis, you have to call the `setRankAnalysis` function, which is used like this:

```
setRankAnalysis(geneIDs, setCollection, use.ranks, setPCutoff = 0.01,
        fdrCutoff = 0.05)
```

The `geneIDs` argument is the vector of Entrez Gene IDs, sorted by increasing p-value. The second argument is the gene set collection object that we created in section 3. The `use.ranks` argument is a logical value. When set to `TRUE`, SetRank will be run in ranked and when `FALSE`, in unranked mode. Ranked and unranked mode were discussed in section 4.1 on page 13. The arguments `setPCutoff` and `fdrCutoff` arguments set the statistical parameters of the SetRank algorithm. The most important of these is `setPCutoff`, which is the p-value cutoff above which a gene set should be rejected. This cutoff is used to determine both which gene sets are discarded after the primary gene set calculation and to decide if a gene set is a false positive after the intersection of another gene set has been substracted from it. See the SetRank paper for more details on the algorithm. We recommend to use a value between 0.01 and 0.05. The `fdrCutoff` controls the false discovery rate in the final set of results. We advise you to use a rather liberal cutoff here, as it is always possible to perform a more stringent filtering on the result set whereas the reverse is not possible. See section 5.1.1.3 on page 25 for some pointers on how to filter SetRank results.

To simply use the default parameters on our example input file with the gene set collection object we prepared in section 3 on page 5, we can start SetRank like this:

```
network = setRankAnalysis(inputData$entrezID, collection, TRUE)
```

The `setRankAnalysis` function returns a gene set network as an igraph object, which we store in the variable `network`. Bear in mind that running this function might take a couple of hours, depending on how big your input dataset is and how many gene sets are initially found significant. In section 4.2.3 on the following page, we will discuss how to speed up computation using multi-threading.

4.2.2 Multiple input files

There are different ways to run SetRank on multiple input files. You could, for instance, wrap an R script that runs SetRank on a single file in a shell script that iterates over multiple input files. While this approach would work fine, we however recommend to process all input files that make up a single dataset within a single R run. The most efficient way is to create a list object with all input files and then use the `lapply` function to run SetRank on each of these files to obtain a list of gene set networks. The reason for choosing this method is that it will give nicer results with the visualisation methods discussed in section 4.3 on page 17. The code examples below show how this task can be done.

Assuming we use the same input files as in section 3.2.2.1 on page 8, we can use the same code again to read in multiple input files in a list object called `topTables`:

```
topTableFiles = Sys.glob("RNAseq_results/*.txt")
topTables = lapply(topTableFiles, read.table, sep="\t", header=TRUE,
        comment.char="", quote="", stringsAsFactors=FALSE)
names(topTables) <- sub("\\.txt", "", basename(topTableFiles))
topTables = lapply(topTables, function(t) t[!is.na(t$padj),])
topTables = lapply(topTables, function(t) t[order(t$padj),])
```

The last line in the above snippet makes sure the input tables are sorted by increasing p-value. Assuming the gene identifiers are in a column called "geneID", we can extract the input gene lists like so:

```
genes = lapply(topTables, function(t) t$geneID)
```

If our gene identifiers are not Entrez Gene identifiers, we need to convert them first by again creating a converter function (see appendix B on page 26) and applying it to our list of input genes.

```
ensembl2Entrez = createIDConverter("Homo.sapiens", "ENSEMBLID",
        "ENTREZID")
genes = lapply(genes, ensembl2Entrez)
```

We can now call the `setRankAnalysis` function on each of our input gene lists to obtain our final list of gene set networks.

```
networks = lapply(setRankAnalysis, genes, collection, TRUE)
```

4.2.3 Using multi-threading

As the SetRank method is quite CPU-intensive, running an analysis, especially on multiple input files, can take a long time. To speed up the process, SetRank has the built-in option to run an analysis multi-threaded, i.e. using multiple CPU cores in parallel, by virtue of the R `parallel` package. To specify the number of cores to use, simply set the global `mc.cores` option to the desired number using the `options` function before you call `setRankAnalysis`.

```
options(mc.cores=4)
```

There are a few caveats to consider when you want to run SetRank multi-threaded:

- Setting the `mc.cores` option will have no effect when running on Microsoft Windows. This limitation is because SetRank creates multiple threads using a system process called forking, which does not exist on Windows[7].
- Bear in mind that SetRank is also quite memory-hungry. On big datasets, each additional thread can take up an additional 2 GB on top of the initial 4GB or so that SetRank uses. Thus, make sure you have enough memory on your machine for the number of cores you want to use.

[7] Look, don't expect the scientific community to take you seriously when you use Windows for bioinformatics analysis. :-)

- Using all available cores on your machine might actually be slower than using half of them, even if you have enough memory when your processor uses hyper-threading. Try using half to three-quarters of available cores for optimal results.
- It is not advisable to use the `mclapply` function instead of `lapply` when you call the `setRankAnalysis` function on multiple input sets, as shown in 4.2.2 on page 15. Doing so will result in n^2 rather than n threads being created, where n is the value you assigned to `mc.cores`.

4.3 Exporting and visualising your results

4.3.1 Exporting your results

The `setRankAnalysis` function only returns an network[8] object which can only be used inside R. Fortunately, the `SetRank` package contains two functions, `exportSingleResult` and `exportMultipleResults` to export your results. As their names suggests, these functions can be used when you have a single gene set network or a list of multiple networks respectively.

4.3.1.1 Single result The usage and arguments of `exportSingleResult` are discussed below.

```
exportSingleResult(network, selectedGenes, collection, networkName,
        IDConverter = NULL, outputPath = "./")
```

`network` is the gene set network object returned by `setRankAnalysis`.

`selectedGenes` should be the same list of gene identifiers that you passed to `setRankAnalysis`

`collection` is the gene set collection that you passed to `setRankAnalysis`.

`networkName` is a string used to name the different output files generated by this function.

`IDConverter` is an ID converter function generated by the `createIDConverter` function (see appendix B on page 26). The purpose of this function is to convert the Entrez Gene identifiers used by SetRank to more human-readable gene symbols or names. Note that this converter function will be different from the one you might have used before to convert your input gene identifiers to Entrez identifiers. If you omit this argument, the output files will simply contain Entrez Gene identifiers.

`outputPath` is the directory where the output files will be created. The default is to use the current working directory. If the last element of the path doesn't exist, a directory will be created.

Using the results from the example in section 4.2.1 on page 14, we can export our results using the following bit of code:

[8] Actually an `igraph` object from the package with the same name

```
library(SetRankVisualization)
IDConverter = createIDConverter("Homo.sapiens", "ENTREZID", "SYMBOL")
exportSingleResult(network, inputData$entrezID, collection,
        "myResults", IDConverter, "setrank_results")
```

The `IDConverter` function we created here converts Entrez identifiers to more human-readable gene symbols. The call to `exportSingleResult` will create the following files in a subdirectory called "`setrank_results`" of the current working directory, with *<name>* replaced by the value of `networkName`.

<name>`_pathways.txt` A tab-delimited file listing the gene sets found to be significant by SetRank. This file can be imported into a spreadsheet program.

<name>`_membership.txt` A tab-delimited file listing which genes belong to each of the pathways mentioned in `myResults_pathways.txt`, can also be imported into a spreadsheet.

<name>`.net.xml` A file containing the gene set network returned by `setRankAnalysis`. This file uses the graphML[9] format to describe a network and can be imported into Cytoscape (see section 4.3.3 on page 20).

`setrank.xml` A Cytoscape visualisation style, to be used in Cytoscape. See section 4.3.3 on page 20 for details.

4.3.1.2 Multiple results

The `exportMultipleResults` function is used as follows:

```
exportMultipleResults(networkList, selectedGenesList, collection,
        IDConverter = NULL, outputPath = "./")
```

`networkList` is a list of gene set network objects

`selectedGenesList` should the same list of input gene identifier vectors that was passed to `setRankAnalysis`. The names should correspond to those of `networkList`.

`collection` is the same gene set collection that was passed to `setRankAnalysis`.

`IDConverter` same as for `exportSingleResult`.

`outputPath` same as for `exportSingleResult`.

Note that compared to `exportSingleResult`, there is no `networkName` argument. The `exportMultipleResults` function uses the names from the `networkList` argument instead.

Using the example from section 4.2.2 on page 15, we can export our results like so:

```
library(SetRankVisualization)
IDConverter = createIDConverter("Homo.sapiens", "ENTREZID", "SYMBOL")
exportMultipleResults(networks, genes, collection, IDConverter,
        "setrank_results")
```

[9] `http://graphml.graphdrawing.org/`

For each network object in `networkList`, `exportMultipleResults` will create the same `<name>_pathways.txt`, `<name>_membership.txt`, and `<name>.net.xml` files as `exportSingleResult`, as well as the `setrank.xml` style file. Note that there is no `networkName` argument, `exportMultipleResults` function uses the names from the `networkList` argument instead. Apart from these files, an additional file called `pathways.txt` is generated, which gives an overview of all pathways detected in the different input gene lists.

4.3.2 Importing the output tables in a spreadsheet

Most spreadsheet programs have the option to import data from TAB-delimited files.

4.3.2.1 Microsoft Excel

1. Choose the "Data" ribbon and click the icon that says "From Text".
2. A file dialog box will appear. Choose the file you wish to import, e.g. `myResults_pathways.txt`.
3. The "Text Import Wizard" is opened now. On the first window, make sure the "Delimited" option is chosen. Then click "Next".
4. In the second window, make sure TAB is the only option selected under "Delimiters". Click Finish.
5. In the Import Data dialog, choose "New Worksheet" and click OK

Repeat these steps for each file you need to import.

4.3.2.2 OpenOffice/LibreOffice Calc

1. From the "Insert" menu, choose "Sheet from File..."
2. A file dialog box will appear. Choose the file you wish to import, e.g. `myResults_pathways.txt`.
3. A "Text Import" dialog will now appear. Under the heading "Separator options", make sure the "Separated By" option is chosen and that "Tab" is the only separator option chosen. Click OK
4. A dialog "Insert Sheet" will now appear. Just click OK

Repeat these steps for each file you need to import.

4.3.2.3 Gnumeric Whereas you can use "File" → "Open" to import TAB-delimited files into Gnumeric[10], this spreadsheet program also comes with a command line utility called `ssconvert` which allows you to combine several files into a single Excel or LibreOffice Calc file. To create a single spreadsheet from the output of a typical SetRank run, just run the following command (all on one line) from your terminal:

```
ssconvert --merge-to setrank.xls pathways.txt *_pathways.txt
        *_membership.txt
```

[10] `http://www.gnumeric.org/`

If you only used the `exportSingleResult` function to export your data, you have to leave out `pathways.txt`. If you want to have an LibreOffice file instead of an Excel file, just type `setrank.ods` instead of `setrank.xls`.

4.3.3 Importing the gene set network in Cytoscape

To visualize the gene set networks returned by SetRank, we can use Cytoscape, an Open Source network analysis tool. Cytoscape can be downloaded for free at `http://www.cytoscape.org`.

To import a gene set network, go to "File" → "Import" → "Network" → "File" and select the file you want to load from the file dialog, e.g. `myResults.net.xml`. The network will appear in the "Network" tab of the Control Panel and a default network view will be created.

Repeat the above procedure to import all network files you need. Next, you need to import the SetRank style file, which is called "`setrank.xml`". Go to "File" → "Import" → "Style" and choose the `setrank.xml` file from the file dialog.

To activate the SetRank visualisation style, select the "Style" tab of the Control Panel and select the style called "SetRank" from the drop-down list. You will see the network appearance change. By default, Cytoscape lays out a network in a grid arrangement, which is hard to interpret. Fortunately Cytoscape comes with different graph layout algorithms. For large networks, the Edge-Weighted Spring Embedded algorithm gives generally the best results. To apply this layout, go to "Layout" → "Edge-Weighted Spring Embedded" → "jaccard". Very often, the resulting layout will still look very 'cluttered', with many nodes on top of each other. You can change this by scaling the layout. Go to "Layout" → "Scale". A layout panel will now appear. You can drag the slider to the right to increase the distance between nodes. Make sure the "Scale Selected Nodes Only" option is not selected.

After you have imported and layed out all your networks, you might want to save your session so you don't have to repeat all these steps every time you want to look at your results. Simply go to "File" → "Save" and choose a file name.

4.4 Creating gene set-specific gene interaction networks

Once you have identified the most important pathways affected by your experiment, you will probably want to know what genes are playing a role in these pathways and how these relate to one another. To help you answer these questions, the SetRank package contains the function `exportGeneNets` to create detailed gene interaction networks for each of the pathways the were returned in a SetRank run and visualises your original expression data on top of these networks. This function needs three pieces of information:

- The results from one or more SetRank runs, i.e. one or more gene set network objects returned by the `setRankAnalysis` function.
- For each of the above network objects, the original input table that contains per gene the p-value and the log-fold change.
- A genome-wide interaction network or *interactome*, for your species of interest. You will need to prepare this beforehand, as explained in the next section.

4.4.1 Building interactomes

4.4.1.1 Using the SetRank_interactomes package The easiest way to create an interactome for your species of interest, is to use the SetRank_interactomes package. This package is similar to the GeneSets package that we discussed in section 3.1.1 on page 5 in that it downloads external data and packages it in a way so that it is ready for use with SetRank. You can downlad the SetRank_interactomes package by either cloning the git repository using

```
git clone https://github.com/C3c6e6/SetRank_interactomes.git
```

or by downloading the `.tar.gz` or `.zip` archive at `https://github.com/C3c6e6/SetRank_interactomes/releases/latest`.

The package has the following system requirements:

- A system running Linux, Mac OS X or any other Unix-based operating system.
- Perl version 5.12 or above
- Python version 2.7 or above
- GNU make version 3.81 or above
- R version 3.0 or above
- The `wget` utility

If you managed to get the GeneSets package working, you should have no problem running the SetRank_interactomes package.

Just like the GeneSets package, the only thing you need to do is to edit the included `organisms` file to specify for which organisms you wish to construct an interactome. You can check section 3.1.1 on page 5 for details on how to do this, but you'll probably just want to copy the `organisms` file you've created for the GeneSets package.

Again, to start building the interactomes, just type

```
make
```

at the command line. The whole build process can take a few hours, as the included scripts will first download the entire list of interactions from the STRING database. This data will automatically be parsed and for each organism listed in the organisms file, an R data file will be created in a subdirectory called output. Each file has a file name in in the format of `<Genus.species>_<YY.MM.DD>.Rda` where `<Genus.species>` is the name of the species and `<YY.MM.DD>` is a datestamp/version number and contains a single R object named `string`. This object can be passed on to the `exportGeneNets` function. Note that you don't need to repeat the build process for every time you wish to use `exportGeneNets`. Of course, it is good practice to re-run the `make` command every other months or so to have updated interactomes.

4.4.1.2 Building your own interactome If you prefer to use another interaction database, you can use the `graph_from_data_frame` function from the `igraph` package. For this function, you need to prepare two data frames, one with node attributes and one with edge attributes. The node attributes data frame must have the Entrez

Gene Identifier of the nodes as first column and another column called "symbol" which contains the gene symbol. You may specify any number of other attributes as additional columns. The first two columns of the edge attributes data frame must be the Entrez Gene Identifiers of the nodes connect by an edge. Additionally, you may specify any number of edge attributes. You can pass the igraph object created by the graph_from_data_frame function to the exportGeneNets function. See the igraph documentation for more details.

4.4.2 Integrating the interactome with SetRank results and expression data

We can now use exportGeneNets to create gene interaction files for each of the significant gene sets returned by SetRank. We first need to load the previously created interactome file. Here we will assume you have used the SetRank_interactomes package and that we need to load the interactome for human. We simply type

```
load("path/to/SetRank_interactomes/output/Homo.sapiens_15.11.22.Rda")
```

to load the string object.

If you only have a single gene set network, as described in section 4.2.1 on page 14, you first need to wrap both our input data and our gene set network into list objects. Assuming we have the same objects as at the end of section 4.2.1 on page 14, we can do

```
topTables = list(setrank=inputData)
networks = list(setrank=network)
```

If you have multiple gene sets and you have followed the procedure described in section 4.2.2 on page 15, we already have our topTables and networks list object ready.

We also need to tell the function what columns in our input data frame contain the Entrez Gene Identifiers, the gene symbols, the log-fold change values and the (corrected) p-values. This information is passed through a named vector with the names "geneID", "symbol", "logFC" and "p", like this:

```
fieldNames = c("geneID"="entrezID", "symbol"="SYMBOL",
    "logFC"="log2FoldChange", "p"="padj")
```

Now, we can call the exportGeneNets function

```
exportGeneNets(topTables, networks, collection, string,
    "gene_nets", fields = fieldNames)
```

This will create a directory called gene_nets, where, for each gene set present in any of the gene set networks in networkList, a file with extension .net.xml will be created. Additionally, a file called gene_net_styles.viz.xml is created as well, which contains visualisation styles for use with Cytoscape (see section 4.4.3 on the next page).

The full usage of exportGeneNets is:

```
exportGeneNets(topTables, networks, collection, string, outDir,
    geneSetIDs = NULL, fields = c(geneID = "ENTREZID",
        symbol = "SYMBOL", logFC = "logFC", p = "adj.P.Val"))
```

We have already discussed most of the arguments. The `geneSetIDs` argument allows you to specify for which specific gene sets you want to create network files. If this argument is omitted, all gene set identifers from all networks will be used.

4.4.3 Importing the gene interaction networks into Cytoscape

To import a gene interaction network, go to "File" → "Import" → "Network" → "File" and select the file you want to load from the file dialog, e.g. `sodium.ion.transport.GOBP.net.xml`. The network will appear in the "Network" tab of the Control Panel and a default network view will be created. Just as with the gene set network, you can apply a layout to make the network easier to interpret.

Repeat the above procedure to import all network files you want. Next, you need to import the gene nets style file, which is called "`gene_net_styles.viz.xml`". Go to "File" → "Import" → "Style" and choose the `gene_net_styles.viz.xml` file from the file dialog.

In the Style tab of the Control Panel, there should now some new styles, with the same names as your `topTables` objects had in the previous section. If you select one of these styles, your network will show the data from that table onto your interaction network.

5 Interpreting SetRank results

Now that we have ran SetRank and imported its results into a spreadsheet and Cytoscape, the real work begins: analysing and interpreting the results! In this section, we will explain on what all the numbers in the pathways and membership tables mean as well as how to look at gene set networks in Cytoscape. We will also discuss how to examine significant gene sets in detail by visualising its gene interactions.

5.1 Understanding the pathway and membership tables

As mentioned in section 4.3.1 on page 17, for each gene set network returned by SetRank, two tables, the pathway table (`<name>_pathways.txt`) and the membership table (`<name>_membership.txt`), are created. Additionally, when multiple gene set networks were created from the same dataset, a pathway overview table (`pathways.txt`) is generated as well.

5.1.1 The pathway table

5.1.1.1 Column descriptions

The pathway table contains the following columns:

name The identifier of a gene set, e.g. GO or KEGG identifier

description A human-readable description of the gene set.

database The database the gene set was defined in.

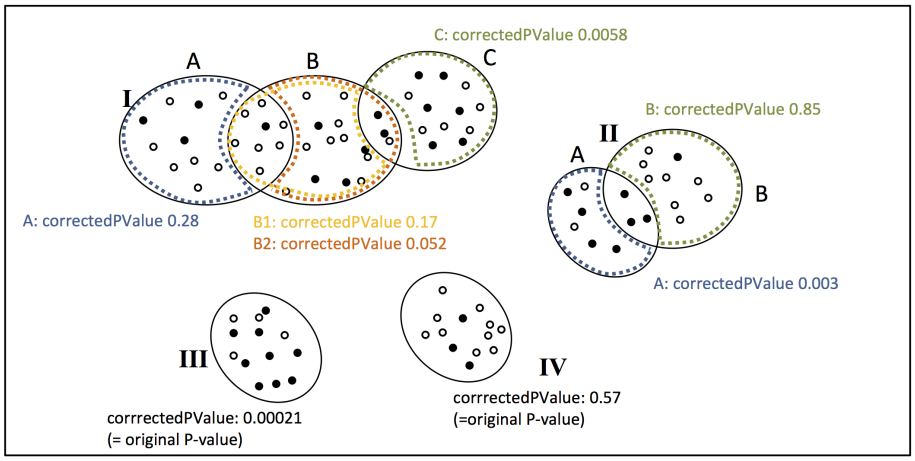

Figure 1: Calculation of corrected p-values.

size The number of genes from the reference set (see section 3.2 on page 7) that are part of the gene set.

setRank The SetRank value of a gene set is the PageRank value of the gene set in the gene set network (see section 5.2 on page 26). This value reflects the importance of the gene set in a gene set network, given the importance of other gene sets that point at it. The higher the SetRank value, the more important the gene set. However, without the associated p-value, the SetRank value itself is meaningless.

pSetRank The p-value associated with the SetRank value of the gene set. This p-value expresses the probability of observing a gene set with the same setRank value in a random network with the same number of nodes and edges as the observed gene set network.

correctedPValue The corrected p-value of the gene set. P-values from gene set enrichment tests are adjusted by SetRank to account for overlap with other gene sets. When the gene set only intersects with a single other gene set, (example II in figure 1), the corrected p-value is simply the value based on the genes unique to the former set. When a gene set overlaps with more than one other gene set (set B in example I on figure 1), multiple P-values are calculated based on the genes in e.g. B but not A (denoted as B2 on figure 1), B but not C (denoted as B1 on figure 1) etc. The reported corrected p-value is then the largest of these p-values.

adjustedPValue To account for the many statistical tests that are performed, SetRank does a multiple test correction as follows. For each connected component, i.e. each group of connected nodes, in the gene set network, the lowest corrected p-value is selected. These p-values are then adjusted for multiple testing using the Holm procedure and the adjusted values are assigned back to the gene sets within each component.

5.1.1.2 Understanding the p-values At this point, you might be wondering why there are three different types of p-values in the pathway table. The reason is that, rather than ranking gene sets by a single p-value, SetRank uses a more sophisticated way of sorting the results. The algorithm considers gene sets with a significant SetRank value as the most important of all. When the SetRank value is not significant, the most network components are first ranked according to significance using the adjusted p-value. The corrected p-value is then used to rank gene sets within a single component.

Thus, in short, the gene sets in a pathway table are ranked by importance, with the most important ones on top of the table.

5.1.1.3 Filtering results If you feel that the list of pathways is too long and you would like to shorten it, be careful not to discard interesting results. For instance, it is possible that a gene set has a very significant pSetRank value but a rater weak corrected p-value. To avoid excluding such cases, you can apply a filter that retains gene sets with a corrected and adjusted p-value below a certain threshold – e.g. 0.001 – or with a pSetRank below another threshold – e.g. 0.05.

You can either use the filtering functionality of your spreadsheet program or filter the results using R. In Microsoft Excel, applying a filter on two or more columns is unfortunately quite complicated[11]. Fortunately, in OpenOffice/LibreOffice, the procedure is very straightforward. Go to Data → Filter → Standard Filter. In the resulting dialog window, choose the pSetRank field in the first list box under "Field Name", choose "<=" under "Condition", and put 0.05 under "Value". In the next row, first choose the "OR" operator, before choosing "adjustedPValue", "<=", and 0.001 in the respective other boxes. In the third row, select the "AND" operator, and enter the values "correctedPValue", "<=", and 0.001. Click "OK" to apply the filter.

With R, you can filter the pathway table like so:

```
pathways = read.table("<name>_pathways.txt", sep="\t", quote="",
        header=TRUE, stringsAsFactors=FALSE)
filteredPathways = pathways[pathways$pSetRank <= 0.05 |
        (pathways$correctedPValue <= 0.001 &
        pathways$adjustedPValue <= 0.001),]
write.table(filteredPathways, "<name>_filtered_pathways.txt",
        sep="\t", col.names=TRUE, row.names=FALSE, quote=FALSE)
```

This will create a new file called *<name>*`_pathways.filtered.txt`, where *<name>* is of course replaced with whatever prefix you have used.

5.1.2 The membership table

Once we know the that a given gene set is significant, we typically want to know what genes are inside it. The membership table gives you exactly that information. Each column represents a gene set and is labelled using its identifier, i.e. the value in the column "name" of the pathway table. Likewise, each row represents a gene and is labelled with its gene symbol[12]. Whenever a gene is part of a given gene set, the cell in that row and column will contain an 'X', otherwise it will contain a dot ('.'). The rows

[11] It seems footnote 7 on page 16 also applies to Microsoft Excel

[12] or whatever identifier type the Entrez ID was converted to by the `IDConverter` function used in section 4.3.1 on page 17

and columns are ordered such that gene sets with a similar gene content are next to one another and genes that typically occur in the same gene sets are below one another.

5.1.3 The pathways overview table

When the data set you are analysing involves multiple comparisons, looking at multiple individual pathway tables quickly becomes confusing. The pathways overview table tries to help with this problem by aggregating all results into a single table. The pathways overview table contains a column for each individual result set. The column labels are taken from the names of the individual pathway tables. Each row corresponds to a gene set. The numbers in the columns reflect the rank of a gene set in a particular comparison. If the number is 1, it means the gene set was ranked first, if it is 0.50, it means that gene set was ranked halfway down the list. If it is 0, it means this gene set was not found significant for this particular comparison. The score column contains a global score, reflecting how important a gene set is across all comparisons. The score is calculated as the number of comparisons where the gene set was found to be significant + the average rank value – i.e. the number between 0 and 1.

5.2 Understanding the gene set network

5.3 Visualising the interactions within a gene set

A Gene Set databases in the GeneSets package

B Different types of sequence identifiers and how to convert between them

UniprotID Used by UniProt. Example: Q035233

TranscriptID Used by NCBI

Ensembl

Entrez GeneID E.g. 1042

Gene Symbol Approved by HUGO consortium. Warning: not stable, change over time. Only use for displaying purposes. Also, beware of spreadsheets automatically converting gene names like “SEPT7” into dates like “September 7th”.

Mention how to find between which types of sequences one can convert using the columns function. Also, show how you can find out what each identifier looks like using the keys function.

Explain annotationDBI packages, especially “columns” and “keytypes”. Explain use of createIDConverter and functions returned by it.