STRINGdb Package Vignette

Andrea Franceschini and Damian Szklarczyk

1 INTRODUCTION

STRING (https://www.string-db.org) is a database of known and predicted protein-protein interactions. The interactions include direct (physical) and indirect (functional) associations. The database contains information from numerous sources, including experimental repositories, computational prediction methods and public text collections. Each interaction is associated with a combined confidence score that integrates the various evidences. STRING v12.0 covers 59.3 million proteins from 12,535 organisms and more than 20 billion interactions.

As you will learn in this guide, the STRING database can be useful to add meaning to lists of genes, for example the best hits coming out from a screen or the most differentially expressed genes coming out from a microarray or RNA-seq experiment.

We provide the STRINGdb R package in order to facilitate our users in accessing the STRING database from R. In this guide we explain, with examples, most of the package's features and functionalities.

In the STRINGdb R package we use the new ReferenceClasses of R (search for "ReferenceClasses" in the R documentation.). Besides we make use of the iGraph package (http://igraph.sourceforge.net) as a data structure to represent our protein-protein interaction network.

To begin, you should first know the NCBI taxonomy identifiers of the organism on which you have performed the experiment (e.g. 9606 for Human, 10090 for mouse). If you don't know that, you can search the NCBI Taxonomy (http://www.ncbi.nlm.nih.gov/taxonomy) or start looking at our species table (that you can also use to verify that your organism is represented in the STRING database).
Hence, if your species is not Human (i.e. our default species), you can find it and their taxonomy identifiers on STRING webpage under the 'organisms' section (https://string-db.org/cgi/input.pl?input_page_active_form=org
or download the full list in the download section of STRING website.

```
> library(STRINGdb)
> string_db <- STRINGdb$new( version="12.0", species=9606,
+                            score_threshold=400, network_type="full", link_data="detailed", input_di
```

As it has been shown in the above commands, you start instantiating the STRINGdb reference class. In the constructor of the class you can also define the STRING version to be used and a threshold for the combined scores of the interactions, such that any interaction below that threshold is not loaded in the object (by default the score threshold is set to 400).
You can also specify the network type "functional" for full functional STRING network or "physical"

for physical subnetwork, which link only the proteins which share a physical complex.
Besides, if you specify a local directory to the parameter input-directory, the database files will be downloaded into this directory and most of the methods can be used off-line. Otherwise, the database files will be saved and cached in a temporary directory that will be cleaned automatically when the R session is closed. By setting `link_data="detailed"` we also load the evidence score columns, such as the experimental score, in addition to the combined score.

For a better understanding of the package two other commands can be useful:

```
> STRINGdb$methods()              # To list all the methods available.

 [1] ".objectPackage"                        ".objectParent"
 [3] "add_diff_exp_color"                    "add_proteins_description"
 [5] "benchmark_ppi"                         "benchmark_ppi_pathway_view"
 [7] "callSuper"                             "copy"
 [9] "enrichment_heatmap"                    "export"
[11] "field"                                 "getClass"
[13] "getRefClass"                           "get_aliases"
[15] "get_annotations"                       "get_bioc_graph"
[17] "get_clusters"                          "get_enrichment"
[19] "get_enrichment_figure"                 "get_graph"
[21] "get_homologs"                          "get_homologs_besthits"
[23] "get_homology_graph"                    "get_interaction_partners"
[25] "get_interactions"                      "get_link"
[27] "get_neighbors"                         "get_paralogs"
[29] "get_pathways_benchmarking_blackList" "get_png"
[31] "get_ppi_enrichment"                    "get_ppi_enrichment_full"
[33] "get_proteins"                          "get_pubmed"
[35] "get_pubmed_interaction"                "get_subnetwork"
[37] "get_summary"                           "get_term_proteins"
[39] "import"                                "initFields"
[41] "initialize"                            "load"
[43] "load_all"                              "map"
[45] "mp"                                    "plot_network"
[47] "plot_ppi_enrichment"                   "post_payload"
[49] "ppi_enrichment"                        "remove_homologous_interactions"
[51] "set_background"                        "show"
[53] "show#envRefClass"                      "trace"
[55] "untrace"                               "usingMethods"


> STRINGdb$help("get_graph")      # To visualize their documentation.

Call:
$get_graph()



Description:
  Return an igraph object with the entire STRING network.
  We invite the user to use the functions of the iGraph package to conveniently
```

```
  search/analyze the network.

References:
  Csardi G, Nepusz T: The igraph software package for complex network research,
  InterJournal, Complex Systems 1695. 2006.
  http://igraph.sf.net

See Also:
  In order to simplify the most common tasks, we do also provide convenient functions
  that wrap some iGraph functions.
  get_interactions(string_ids)   # returns the interactions in between the input proteins
  get_neighbors(string_ids)      # Get the neighborhoods of a protein (or of a vector of proteins).
  get_subnetwork(string_ids)     # returns a subgraph from the given input proteins

Author(s):
   Andrea Franceschini
```

For all the methods that we are going to explain below, you can always use the help function in order to get additional information/parameters with respect to those explained in this guide.

As an example, we use the analyzed data of a microarray study taken from GEO (Gene Expression Omnibus, GSE9008). This study investigates the activity of Resveratrol, a natural phytoestrogen found in red wine and a variety of plants, in A549 lung cancer cells. Microarray gene expression profiling after 48 hours exposure to Revestarol has been performed and compared to a control composed by A549 lung cancer cells threated only with ethanol. This data is already analyzed for differential expression using the limma package: the genes are sorted by fdr corrected pvalues and the log fold change of the differential expression is also reported in the table.

```
> data(diff_exp_example1)
> head(diff_exp_example1)

     pvalue    logFC         gene
1 0.0001018 3.333461       VSTM2L
2 0.0001392 3.822383       TBC1D2
3 0.0001720 3.306056        LENG9
4 0.0001739 3.024605       TMEM27
5 0.0001990 3.854414 LOC100506014
6 0.0002393 3.082052       TSPAN1
```

As a first step, we map the gene names to the STRING database identifiers using the "map" method. In this particular example, we map from gene HUGO names, but our mapping function supports several other common identifiers (e.g. Entrez GeneID, ENSEMBL proteins, RefSeq transcripts ... etc.).

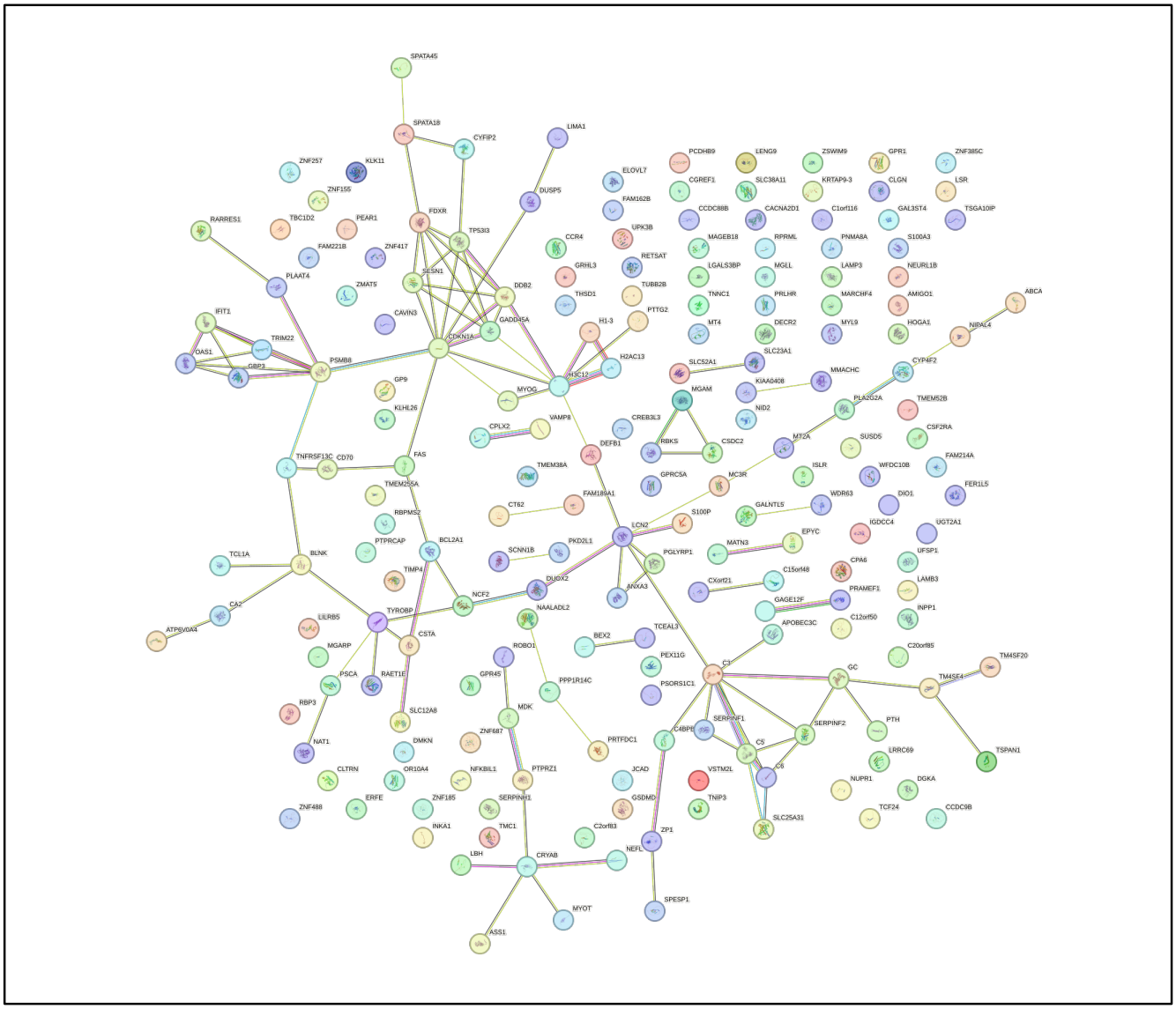
The map function adds an additional column with STRING identifiers to the dataframe that is passed as first parameter.

```
> example1_mapped <- string_db$map( diff_exp_example1, "gene", removeUnmappedRows = TRUE )

Warning:  we couldn't map to STRING 15% of your identifiers
```

As you may have noticed, the previous command prints a warning showing the number of genes that we failed to map. In this particular example, we cannot map all the probes of the microarray that refer to position of the chromosome that are not assigned to a real gene (i.e. all the LOC genes). If we remove all these LOC genes before the mapping we obtain a much lower percentage of unmapped genes (i.e. < 6 %).
If you set to FALSE the "removeUnmappedRows" parameter, than the rows which corresponds to unmapped genes are left and you can manually inspect them.
Finally, we extract the genes with p-value < 0.05 and, for the examples below, we focus on the top 200 genes sorted by p-value. The image shows clearly the genes and how they are possibly functionally related. On the top of the plot, we insert a pvalue that represents the probability that you can expect such an equal or greater number of interactions by chance.

```
> example1_mapped_pval05 <- subset(example1_mapped, pvalue < 0.05)
> example1_mapped_pval05 <- example1_mapped_pval05[order(example1_mapped_pval05$pvalue), ]
> hits <- example1_mapped_pval05$STRING_id[1:200]

> string_db$plot_network( hits )
```

proteins: 200
interactions: 117
expected interactions: 62 (p−value: 4.2e−10)



If you want more control over the appearance of the network image, the `get_png` and `get_link` methods expose additional styling options. In particular, you can switch between `evidence` and `confidence` edge styles, hide node labels, hide disconnected proteins, disable the structure pictures inside bubbles, use the flat node design, center node labels, and change the label font size.

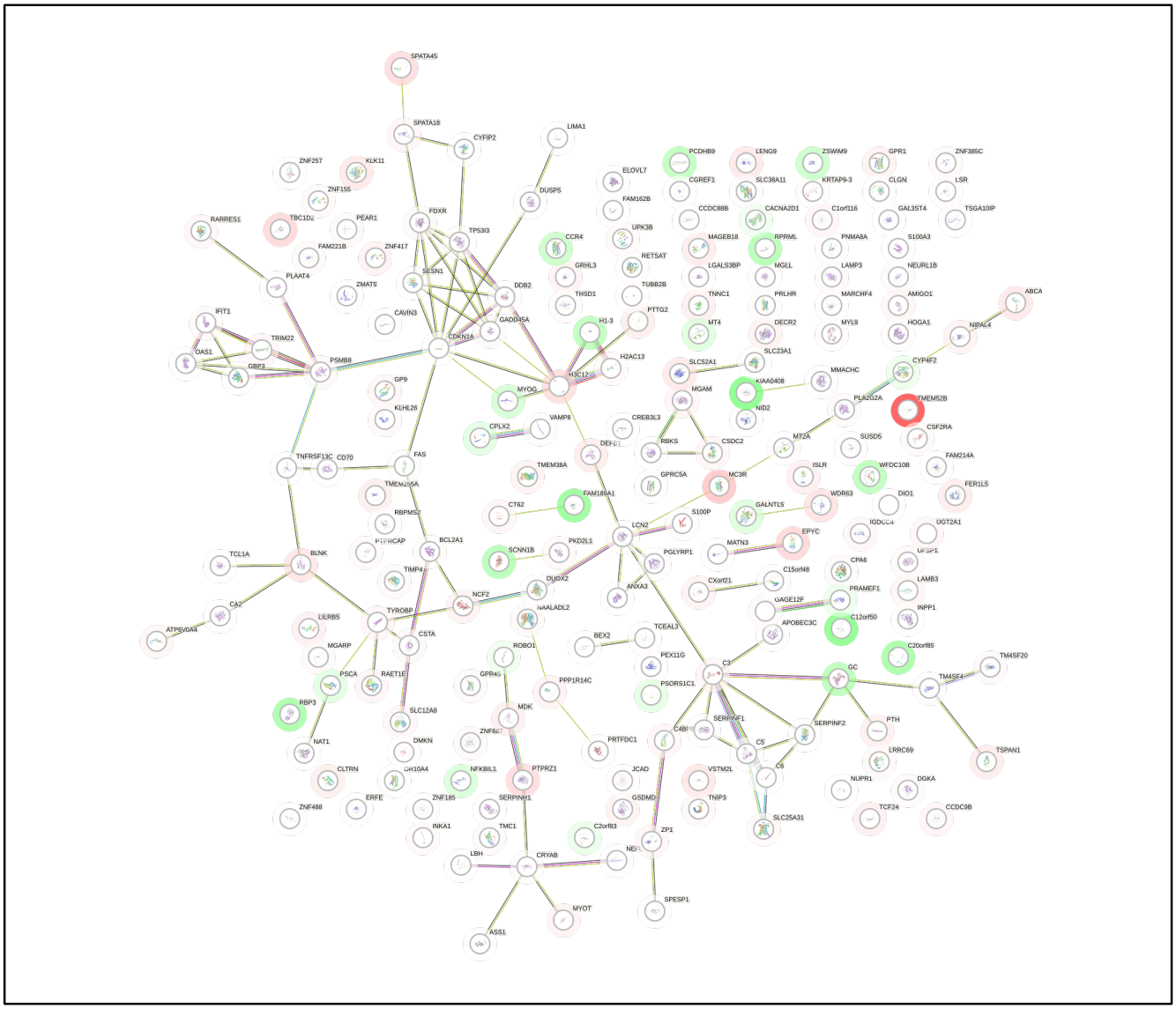
2 PAYLOAD MECHANISM

This R library provides the ability to interact with the STRING payload mechanism. The payload appears as an additional colored "halo" around the bubbles.

For example, this allows to color in green the genes that are down-regulated and in red the genes that are up-regulated. For this mechanism to work, we provide a function that posts the information on our web server.

```
> # filter by p-value and add a color column
> # (i.e. green down-regulated gened and red for up-regulated genes)
> example1_mapped_pval05 <- string_db$add_diff_exp_color( subset(example1_mapped, pvalue<0.05),
+                                                         logFcColStr="logFC" )

> # post payload information to the STRING server
> payload_id <- string_db$post_payload( example1_mapped_pval05$STRING_id,
+                                       colors=example1_mapped_pval05$color )

> # display a STRING network png with the "halo"
> string_db$plot_network( hits, payload_id=payload_id )
```

proteins: 200
interactions: 117
expected interactions: 62 (p−value: 4.2e−10)



3 ENRICHMENT

We provide a method to compute the enrichment in Gene Ontology (Process, Function and Component), KEGG and Reactome pathways, PubMed publications, UniProt Keywords, and PFAM/INTERPRO/SMART domains for your set of proteins all in one simple call. The enrichment itself is computed using an hypergeometric test and the FDR is calculated using Benjamini-Hochberg procedure.

```
> enrichment <- string_db$get_enrichment( hits )
> head(enrichment, n=20)
```

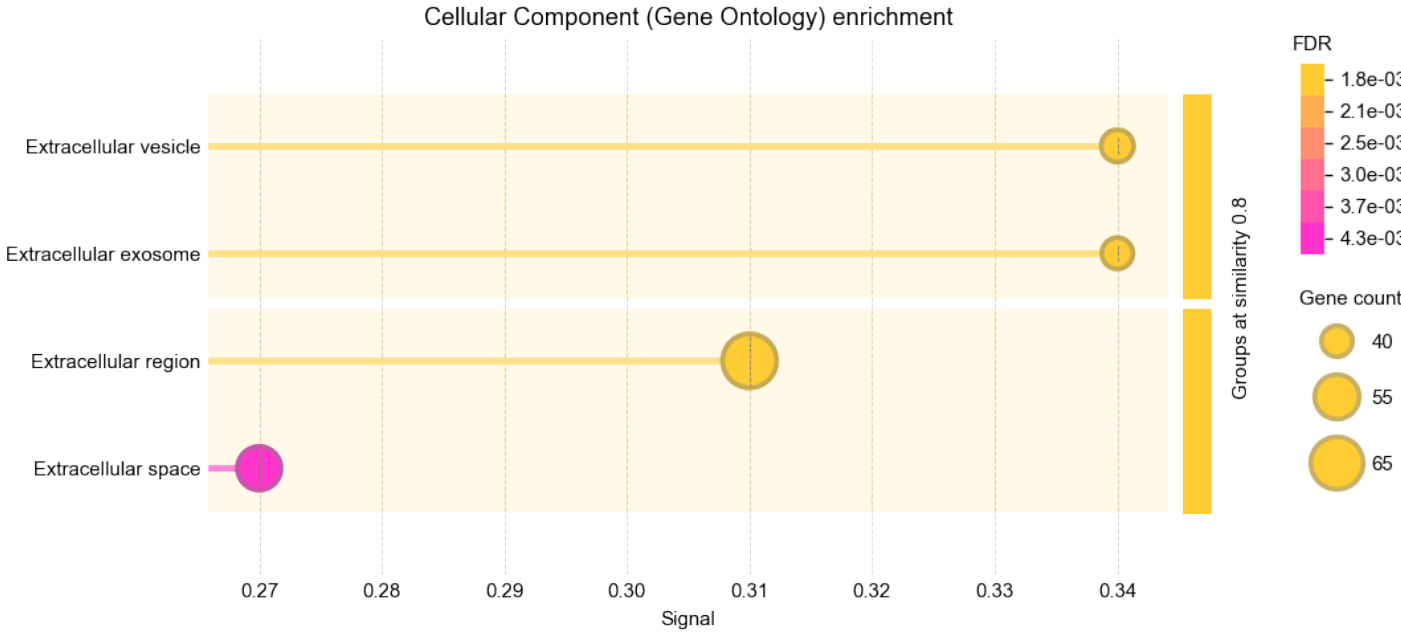
```
       category          term number_of_genes number_of_genes_in_background
1  COMPARTMENTS  GOCC:0005576              42                           2079
2  COMPARTMENTS  GOCC:0043230              17                            524
3       Process    GO:0006952              35                           1394
4     Component    GO:0005576              67                           4175
5     Component    GO:0070062              42                           2096
6     Component    GO:1903561              43                           2120
7     Component    GO:0005615              54                           3247
8       TISSUES   BTO:0000392              10                            171
9       Keyword       KW-0391              17                            537
10      Keyword       KW-0732              55                           3277
11         KEGG      hsa04115               6                             72
12         PMID PMID:39331229               6                              8
13 WikiPathways        WP4963               7                             90
   ncbiTaxonId
1         9606
2         9606
3         9606
4         9606
5         9606
6         9606
7         9606
8         9606
9         9606
10        9606
11        9606
12        9606
13        9606

1
2
3
4  9606.ENSP00000008938,9606.ENSP00000014914,9606.ENSP00000187762,9606.ENSP00000216286,9606.ENSP00000
5
6
7
8
9
10
11
12
13
```

```
1
2
3
4  PGLYRP1,GPRC5A,TMEM38A,NID2,C5,RARRES1,C4BPB,C3,ISLR,SERPINF1,THSD1,TUBB2B,EPYC,LGALS3BP,C6,VAMP8,
5
6
7                                                                                PGLYRP1,GPRC5A,TMEM38
8
9
10                                                                             PGLYRP1,NID2,C5,C4BPB,
11
12
13
    p_value    fdr
1  1.28e-05 0.0294
2  3.51e-05 0.0403
3  7.80e-07 0.0122
4  4.15e-05 0.0182
5  1.55e-05 0.0182
6  8.88e-06 0.0182
7  1.30e-04 0.0441
8  1.53e-05 0.0371
9  4.71e-05 0.0317
10 8.55e-05 0.0317
11 1.30e-04 0.0452
12 2.69e-09 0.0138
13 5.58e-05 0.0436

1
2                                                                                          Ext
3
4
5                                                                                            P
6                                                                                            P
7
8
9
10
11                                                                                           P
12 (2024) Identification of BBC3 as a novel indicator for predicting prostate cancer development and
13                                                                                 p53 transcri
```

You can also retrieve an enrichment figure directly from STRING for a selected category. The image can be returned as a PNG array or as SVG markup, and you can customize the category, similarity-based grouping, color palette, the number of terms shown, and the variable displayed on the X-axis.

```
> enrichment_img <- string_db$get_enrichment_figure(
+   hits,
```

```
+   category = "Component",
+   group_by_similarity = 0.8,
+   color_palette = "yellow_pink",
+   x_axis = "signal"
+ )
> grid::grid.newpage()
> grid::grid.raster(enrichment_img, interpolate = FALSE)
```



If you have performed your experiment on a predefined set of proteins, it is important to run the enrichment statistics using that set as a background (otherwise you would get a wrong p-value !). Hence, before to launch the method above, you may want to set the background:

```
> backgroundV <- example1_mapped$STRING_id
> string_db$set_background(backgroundV)

IGRAPH 4b96b45 UN-- 17434 1702440 --
+ attr: name (v/c), neighborhood (e/n), fusion (e/n), cooccurence
| (e/n), coexpression (e/n), experimental (e/n), database (e/n),
| textmining (e/n), combined_score (e/n)
+ edges from 4b96b45 (vertex names):
 [1] 9606.ENSP00000000233--9606.ENSP00000493357
 [2] 9606.ENSP00000000233--9606.ENSP00000371175
 [3] 9606.ENSP00000000233--9606.ENSP00000354878
 [4] 9606.ENSP00000000233--9606.ENSP00000310226
 [5] 9606.ENSP00000000233--9606.ENSP00000258098
 [6] 9606.ENSP00000000233--9606.ENSP00000363232
+ ... omitted several edges
```

If you already have a `STRINGdb` object, calling `set_background` is sufficient and you do not need to instantiate a new object again. Passing `backgroundV` to the constructor is simply an alternative when you already know the background at initialization time:

```
> string_db <- STRINGdb$new( score_threshold=400, backgroundV = backgroundV )
```

Once the background has been set, the enrichment results can change substantially because the statistical test is now performed against the selected background rather than the whole proteome. In many cases the enrichment signals become smaller, which is expected: the test is now calibrated against the proteins that were actually measurable or considered in the experiment, rather than against all proteins in the organism.

```
> enrichment_background_corrected <- string_db$get_enrichment( hits )
> head(enrichment_background_corrected, n=20)

  category          term number_of_genes number_of_genes_in_background
1  Process    GO:0006952             35                          1298
2     PMID PMID:39331229              6                             8
  ncbiTaxonId
1        9606
2        9606

1 9606.ENSP00000008938,9606.ENSP00000223642,9606.ENSP00000224337,9606.ENSP00000243611,9606.ENSP000002
2

1 PGLYRP1,C5,BLNK,C4BPB,C3,TYROBP,LGALS3BP,C6,VAMP8,ANXA3,MGLL,DEFB1,SERPINF2,PTPRCAP,CCR4,WFDC10B,SC
2
   p_value    fdr
1 2.38e-06 0.0306
2 5.45e-09 0.0197

1
2 (2024) Identification of BBC3 as a novel indicator for predicting prostate cancer development and c
```

If you just want to know which terms are assigned to your set of proteins, and not necessarily which ones are enriched, you can use "get_annotations" method. This method outputs the terms from most categories (the exceptions are KEGG terms due to licensing issues and PubMed due to the size of the output) that are associated with your set of proteins.

```
> annotations <- string_db$get_annotations( hits )
> head(annotations, n=20)
```

```
       category      term_id number_of_genes ratio_in_set species
1  COMPARTMENTS GOCC:0000109               1        0.005    9606
2  COMPARTMENTS GOCC:0000139               1        0.005    9606
3  COMPARTMENTS GOCC:0000151               2        0.010    9606
4  COMPARTMENTS GOCC:0000152               1        0.005    9606
5  COMPARTMENTS GOCC:0000228               1        0.005    9606
6  COMPARTMENTS GOCC:0000307               1        0.005    9606
7  COMPARTMENTS GOCC:0000323               6        0.030    9606
8  COMPARTMENTS GOCC:0000502               1        0.005    9606
9  COMPARTMENTS GOCC:0000785               2        0.010    9606
10 COMPARTMENTS GOCC:0000786               1        0.005    9606
11 COMPARTMENTS GOCC:0000791               1        0.005    9606
12 COMPARTMENTS GOCC:0001533               1        0.005    9606
13 COMPARTMENTS GOCC:0001650               1        0.005    9606
14 COMPARTMENTS GOCC:0001669               1        0.005    9606
15 COMPARTMENTS GOCC:0001725               2        0.010    9606
16 COMPARTMENTS GOCC:0001726               2        0.010    9606
17 COMPARTMENTS GOCC:0002133               1        0.005    9606
18 COMPARTMENTS GOCC:0005576              42        0.210    9606
19 COMPARTMENTS GOCC:0005577               1        0.005    9606
20 COMPARTMENTS GOCC:0005579               3        0.015    9606

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 9606.ENSP00000008938,9606.ENSP00000216286,9606.ENSP00000223642,9606.ENSP00000243611,9606.ENSP00000
19
20
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 PGLYRP1,NID2,C5,C4BPB,C3,SERPINF1,THSD1,TUBB2B,LGALS3BP,C6,CSTA,ANXA3,ZP1,CA2,TIMP4,DEFB1,PSCA,SER
19
20
```

	description
1	Nucleotide-excision repair complex
2	Golgi membrane
3	Ubiquitin ligase complex
4	Nuclear ubiquitin ligase complex
5	Nuclear chromosome
6	Cyclin-dependent protein kinase holoenzyme complex
7	Lytic vacuole
8	Proteasome complex
9	Chromatin
10	Nucleosome
11	Euchromatin
12	Cornified envelope
13	Fibrillar center
14	Acrosomal vesicle
15	Stress fiber
16	Ruffle
17	Polycystin complex
18	Extracellular region
19	Fibrinogen complex
20	Membrane attack complex

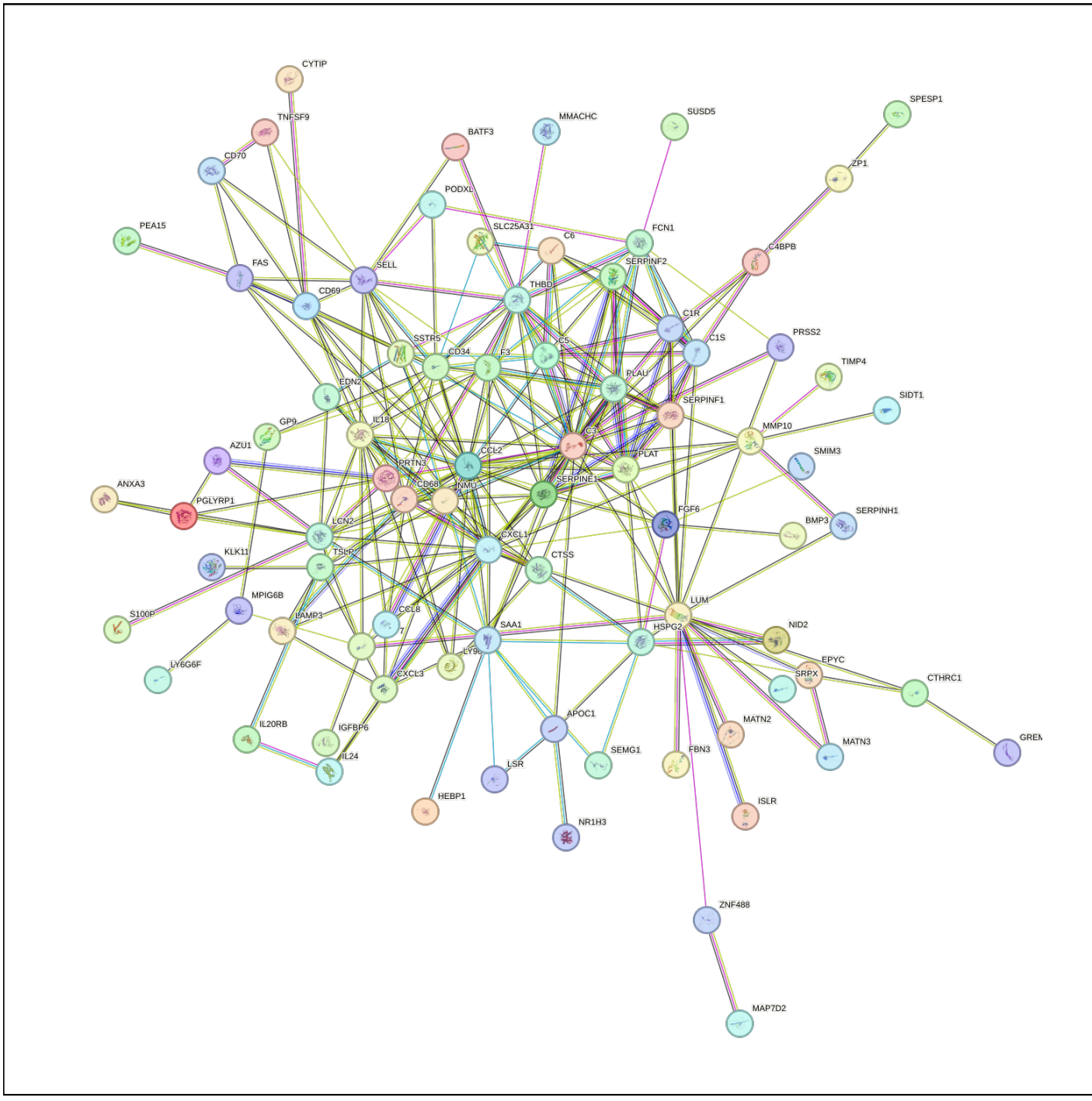
4 CLUSTERING

The iGraph package provides several clustering/community algorithms: "fastgreedy", "walktrap", "spinglass", "edge.betweenness". We encapsulate this in an easy-to-use function that returns the clusters in a list.

```
> # get clusters
> clustersList <- string_db$get_clusters(example1_mapped_pval05$STRING_id[1:600])
```

Now we visualize the first four clusters one by one.

```
> par(mar = c(0, 0, 0.5, 0))
> string_db$plot_network(clustersList[[1]], add_summary = FALSE)
> par(mar = c(0, 0, 0.5, 0))
> string_db$plot_network(clustersList[[2]], add_summary = FALSE)
> par(mar = c(0, 0, 0.5, 0))
> string_db$plot_network(clustersList[[3]], add_summary = FALSE)
> par(mar = c(0, 0, 0.5, 0))
> string_db$plot_network(clustersList[[4]], add_summary = FALSE)
```


CYTIP
TNFSF9
CD70
BATF3
MMACHC
SUSD5
SPESP1
ZP1
PODXL
PEA15
SLC25A31
C6
FCN1
C4BPB
FAS
SELL
SERPINF2
CD69
THBD
C1R
C1S
PRSS2
SSTR5
CD34
F3
C5
TIMP4
EDN2
PLAU
SERPINF1
SIDT1
GP9
IL18
C3
MMP10
AZU1
CCL2
PLAT
SMIM3
PRTN3
CD68
NMU
SERPINE1
ANXA3
PGLYRP1
FGF6
BMP3
SERPINH1
LCN2
CXCL1
KLK11
TSLP
CTSS
LUM
MPIG6B
S100P
LAMP3
CCL8
SAA1
HSPG2
NID2
LY6G6F
LY96
EPYC
SRPX
CTHRC1
CXCL3
APOC1
MATN2
IL20RB
IGFBP6
SEMG1
FBN3
MATN3
IL24
LSR
HEBP1
ISLR
NR1H3
ZNF488
MAP7D2

5 ADDITIONAL PROTEIN INFORMATION

You can get a table that contains all the proteins that are present in our database of the species of interest. The protein table also includes the preferred name, the size and a short description of each protein.

```
> string_proteins <- string_db$get_proteins()
> head(string_proteins)
```

```
   protein_external_id preferred_name protein_size
1 9606.ENSP00000000233           ARF5          180
2 9606.ENSP00000000412           M6PR          277
3 9606.ENSP00000001008          FKBP4          459
4 9606.ENSP00000001146        CYP26B1          512
5 9606.ENSP00000002125        NDUFAF7          441
6 9606.ENSP00000002165          FUCA2          467

1
2
3 Peptidyl-prolyl cis-trans isomerase FKBP4, N-terminally processed; Immunophilin protein with PPIase
4 Cytochrome P450 26B1; Involved in the metabolism of retinoic acid (RA), rendering this classical mo
5
6
```

In the following section we will show how to query STRING with R on some specific proteins. In the examples, we will use the famous tumor proteins TP53 and ATM .

First we need to get the STRING identifier of those proteins, using our mp method:

```
> tp53 = string_db$mp( "tp53" )
> atm = string_db$mp( "atm" )
```

The mp method (i.e. map proteins) is an alternative to our map method, to be used when you need to map only one or few proteins.
It takes in input a vector of protein aliases and returns a vector with the STRING identifiers of those proteins.

You can retrieve the interactions that connect certain input proteins between each other.
The "get_interactions" method returns a data frame describing the interactions among the queried proteins. The table contains the STRING identifiers in the `from` and `to` columns, the corresponding preferred names in `from_name` and `to_name`, the `combined_score`, and any additional evidence scores available for the current `link_data` setting. This makes it easy to inspect both the interacting proteins and the supporting scores for each interaction.

```
> string_db$get_interactions( c(tp53, atm) )
```

```
                 from                   to combined_score from_name to_name
1 9606.ENSP00000269305 9606.ENSP00000278616            999      TP53     ATM
```

```
  neighborhood fusion cooccurence coexpression experimental database textmining
1            0      0           0          103          929      900        975
```

If you want to retrieve the first-shell interaction partners of one or more proteins from the locally loaded graph, you can use the "get_interaction_partners" method.
The output keeps the same interaction-table structure as "get_interactions".

```
> string_db$get_interaction_partners( c(tp53, atm), limit=5 )[, c("from", "to", "combined_score", "ex
```

```
                      from                   to combined_score experimental
30659  9606.ENSP00000269305 9606.ENSP00000212015            999          887
195401 9606.ENSP00000269305 9606.ENSP00000254719            999          982
219212 9606.ENSP00000269305 9606.ENSP00000258149            999          999
270101 9606.ENSP00000269305 9606.ENSP00000262367            999          998
286703 9606.ENSP00000269305 9606.ENSP00000263253            999          999
121197 9606.ENSP00000278616 9606.ENSP00000233146            999          324
126836 9606.ENSP00000278616 9606.ENSP00000234420            999          324
336588 9606.ENSP00000278616 9606.ENSP00000265433            999          990
363176 9606.ENSP00000278616 9606.ENSP00000269305            999          929
401207 9606.ENSP00000278616 9606.ENSP00000325863            999          907
       from_name to_name
30659       TP53   SIRT1
195401      TP53    RPA1
219212      TP53    MDM2
270101      TP53  CREBBP
286703      TP53   EP300
121197       ATM    MSH2
126836       ATM    MSH6
336588       ATM     NBN
363176       ATM    TP53
401207       ATM   MRE11
```

You can also output only interactions supported by a particular type of evidence. Here we retrieve only experimentally supported interactions, keeping only the protein names together with the experimental score:

```
> interaction_partners <- string_db$get_interaction_partners( c(tp53, atm), limit=5 )
> interaction_partners_exp <- subset(interaction_partners, experimental > 0,
+                                    select = c("from_name", "to_name", "experimental"))
> interaction_partners_exp
```

```
       from_name to_name experimental
30659       TP53   SIRT1          887
195401      TP53    RPA1          982
219212      TP53    MDM2          999
270101      TP53  CREBBP          998
286703      TP53   EP300          999
121197       ATM    MSH2          324
126836       ATM    MSH6          324
```

```
336588       ATM     NBN          990
363176       ATM    TP53          929
401207       ATM   MRE11          907
```

STRING provides a way to get homologous proteins: in our database we store ALL-AGAINST-ALL alignments within species. You can retrieve all paralogs of the protein using "get_paralogs" method.

```
> # Get all homologs of TP53 in human.
> string_db$get_paralogs(tp53)

  ncbiTaxonId_A           stringId_A ncbiTaxonId_B           stringId_B
1          9606 9606.ENSP00000269305          9606 9606.ENSP00000269305
  bitscore
1    815.8
```

STRING also stores best hits (as measured by bitscore) between proteins from different species. "get_homologs_besthits" lets you retrieve these homologs.

```
> # get the best hits of TP53 in all STRING species
> tp53_besthits <- string_db$get_homologs_besthits(tp53)
> tp53_besthits_100 <- subset(tp53_besthits, bitscore > 100)
> nrow(tp53_besthits_100)

[1] 318


> head(tp53_besthits_100, n=10)

   ncbiTaxonId_A           stringId_A ncbiTaxonId_B                stringId_B
1           9606 9606.ENSP00000269305          9597   9597.ENSPPAP00000011040
2           9606 9606.ENSP00000269305          9598   9598.ENSPTRP00000014836
3           9606 9606.ENSP00000269305          9606      9606.ENSP00000269305
4           9606 9606.ENSP00000269305          9601   9601.ENSPPYP00000008923
5           9606 9606.ENSP00000269305         61621  61621.ENSRBIP00000028632
6           9606 9606.ENSP00000269305         61622  61622.ENSRROP00000026843
7           9606 9606.ENSP00000269305        591936 591936.ENSPTEP00000010553
8           9606 9606.ENSP00000269305         60711  60711.ENSCSAP00000007551
9           9606 9606.ENSP00000269305          9541   9541.ENSMFAP00000038526
10          9606 9606.ENSP00000269305          9544   9544.ENSMMUP00000011334
   bitscore
1     815.8
2     815.8
3     815.8
4     799.3
5     786.6
6     786.6
7     785.8
```

```
8     785.0
9     783.9
10    783.9
```

... or you can specify the species of interest:

```
> # get the homologs of the following two proteins in the mouse (i.e. species_id=10090)
> string_db$get_homologs_besthits(c(tp53, atm), target_species_id=10090)

  ncbiTaxonId_A           stringId_A ncbiTaxonId_B               stringId_B
1          9606 9606.ENSP00000278616         10090 10090.ENSMUSP00000156344
2          9606 9606.ENSP00000269305         10090 10090.ENSMUSP00000104298
  bitscore
1   5328.8
2    598.2
```

6 CITATION

Please cite:

Szklarczyk D, Kirsch R, Koutrouli M, Nastou K, Mehryary F, Hachilif R, Gable AL, Fang T, Doncheva NT, Pyysalo S, Bork P, Jensen LJ, von Mering C. The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Res. 2023 Jan 6;51(D1):D638-D646. doi: 10.1093/nar/gkac1000.