

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 v12.5 covers 59.3 million proteins from 12,535 organisms and more than 28 billion interactions. It also provides directed regulatory networks alongside the functional and physical networks.

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="12.5", 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 "full" (or its alias "functional") for the full functional STRING network, "physical" for the physical subnetwork, or "regulatory" for the directed regulatory subnetwork. In a regulatory network, an edge points from the regulator to its target. Use `network_flavor="typed"` with a full/functional network to show the typed network view in the STRING web interface; this view is not available for a regulatory network. The optional `typed_physical_edges`, `typed_regulatory_edges`, and `show_regulatory_signs` arguments select which typed-edge details are shown.

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_geneset_description"
[21] "get_graph"               "get_homologs"
[23] "get_homologs_besthits"    "get_homology_graph"
[25] "get_interaction_partners" "get_interactions"
[27] "get_link"                "get_neighbors"
[29] "get_paralogs"            "get_pathways_benchmarking_blackList"
[31] "get_png"                 "get_ppi_enrichment"
[33] "get_ppi_enrichment_full" "get_proteins"
[35] "get_pubmed"              "get_pubmed_interaction"
[37] "get_subnetwork"          "get_summary"
[39] "get_term_proteins"        "import"
[41] "initFields"              "initialize"
[43] "load"                    "load_all"
[45] "map"                     "mp"
[47] "plot_network"            "plot_ppi_enrichment"
[49] "post_payload"            "ppi_enrichment"
[51] "remove_homologous_interactions" "set_background"
[53] "show"                    "show#envRefClass"
[55] "trace"                   "untrace"
[57] "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 Resveratrol has been performed and compared to a control composed by A549 lung cancer cells treated only with ethanol. This data is already analyzed for differential expression using the limma package: the genes are sorted by *fdr* corrected *p*values 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 %).

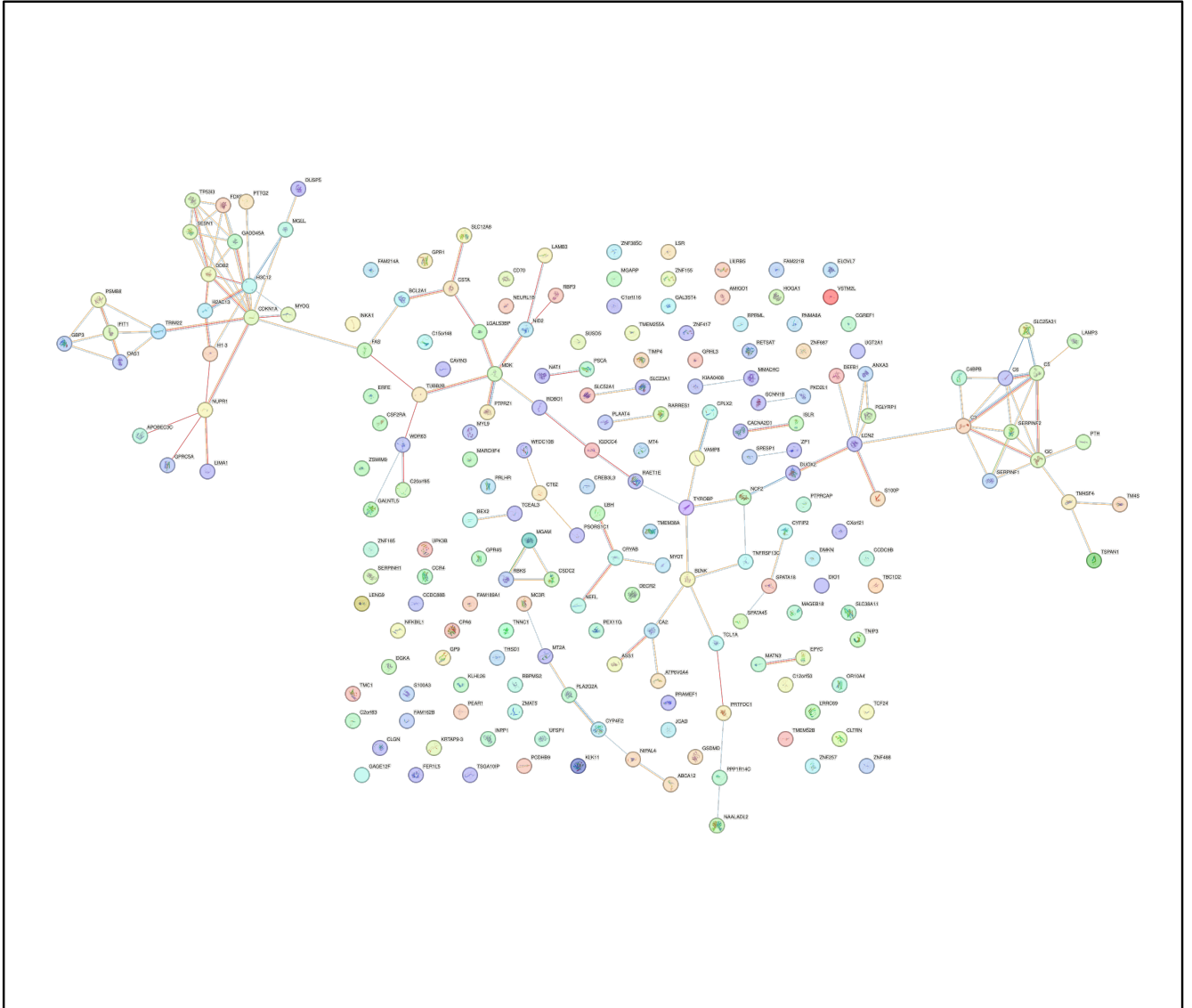
If you set to FALSE the "removeUnmappedRows" parameter, than the rows which corresponds to unmapped genes are left and you can manually inspect them.

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: 127
 expected interactions: 77 (p-value: 1.19e-07)



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.

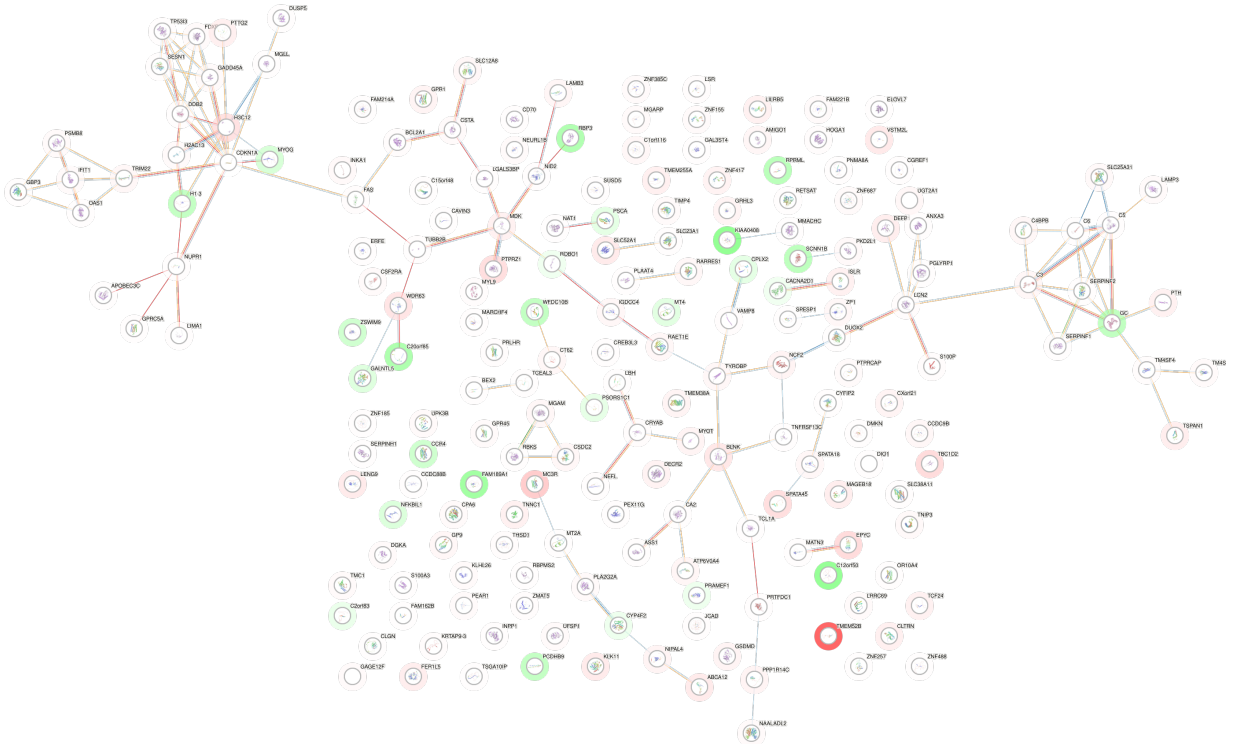
For example, this allows to color in green the genes that are down-regulated and in red the genes that are up-regulated. For this mechanism to work, we provide a function that posts the information on our web server.

```
> # filter by p-value and add a color column
> # (i.e. green down-regulated genes 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: 127
expected interactions: 77 (p-value: 1.19e-07)



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	44	1948
2	COMPARTMENTS	GOCC:1903561	19	552
3	Process	GO:0006952	35	1486
4	Component	GO:0070062	42	2045
5	Component	GO:1903561	43	2075
6	Component	GO:0005615	54	3120
7	Component	GO:0005576	64	3988
8	TISSUES	BTO:0000392	11	187
9	Keyword	KW-0732	58	3506
10	Keyword	KW-0964	28	1274
11	Keyword	KW-1015	46	2493
12	PMID	PMID:39331229	6	8
13	Hallmark	M5939	11	198

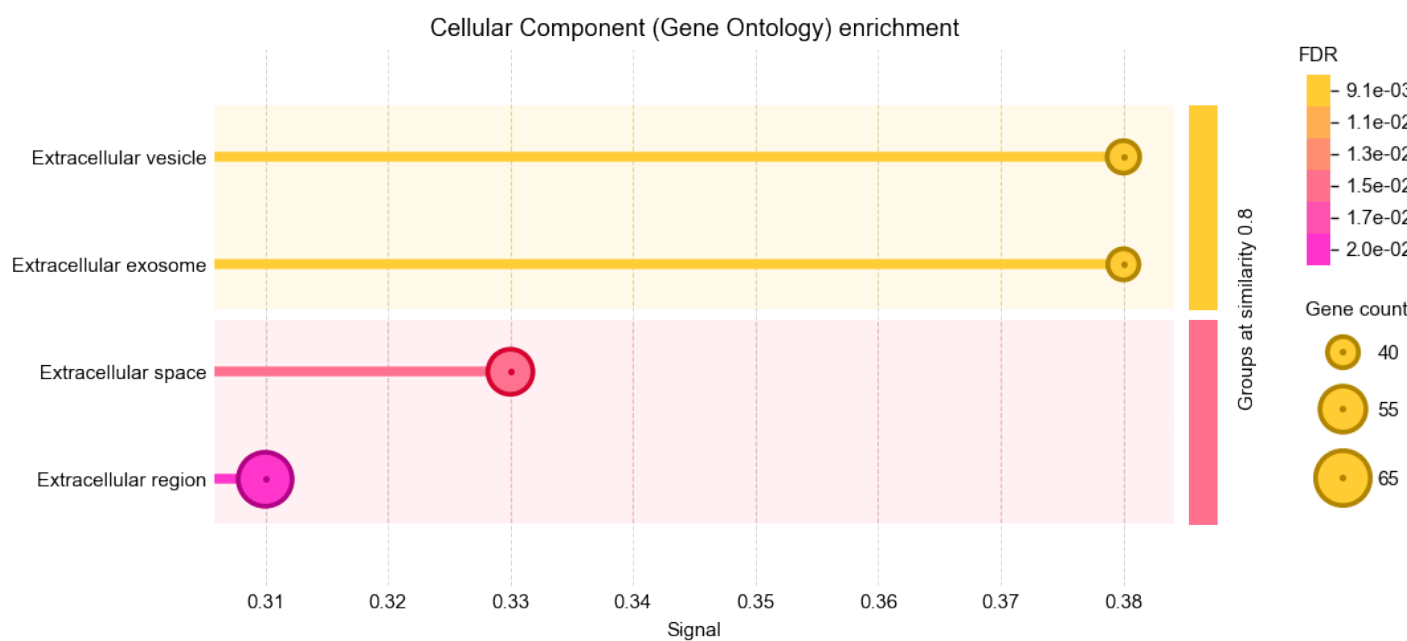
	ncbiTaxonId
1	9606
2	9606
3	9606
4	9606
5	9606
6	9606
7	9606
8	9606
9	9606
10	9606
11	9606
12	9606
13	9606

```
1
2
3
4
5
6
7 9606.ENSPP00000008938,9606.ENSPP00000014914,9606.ENSPP00000187762,9606.ENSPP00000216286,9606.ENSPP00000
8
9
10
11
12
13
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13

```
> 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 d6d0c47 UN-- 17440 1990788 --
+ attr: name (v/c), neighborhood (e/n), fusion (e/n), cooccurrence
| (e/n), coexpression (e/n), experimental (e/n), database (e/n),
| textmining (e/n), combined_score (e/n)
+ edges from d6d0c47 (vertex names):
[1] 9606.ENSP00000000233--9606.ENSP00000054666
[2] 9606.ENSP00000000233--9606.ENSP00000158762
[3] 9606.ENSP00000000233--9606.ENSP00000160827
[4] 9606.ENSP00000000233--9606.ENSP00000216446
[5] 9606.ENSP00000000233--9606.ENSP00000217402
[6] 9606.ENSP00000000233--9606.ENSP00000219479
+ ... 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	COMPARTMENTS	GOCC:0005576	44	1867
2	COMPARTMENTS	GOCC:1903561	19	541
3	TISSUES	BTO:0000392	11	183
4	PMID	PMID:39331229	6	8
5	Hallmark	M5939	11	196
	ncbiTaxonId			
1	9606			
2	9606			
3	9606			
4	9606			
5	9606			

```
1 9606.ENSP00000008938,9606.ENSP00000014914,9606.ENSP00000216286,9606.ENSP00000223642,9606.ENSP00000000000
2
3
4
5
```

```

1 PGLYRP1,GPRC5A,NID2,C5,RARRES1,C4BPB,C3,ISLR,SERPINF1,THSD1,LGALS3BP,C6,VAMP8,CSTA,ANXA3,ZP1,CA2,TI
2
3
4
5

```

```

      p_value    fdr
1 3.26e-06 0.0056
2 2.12e-05 0.0181
3 1.35e-05 0.0117
4 5.45e-09 0.0197
5 2.47e-05 0.0012

```

```

1
2
3
4 (2024) Identification of BBC3 as a novel indicator for predicting prostate cancer development and c
5

```

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:0000139	2	0.010	9606
2	COMPARTMENTS	GOCC:0000151	2	0.010	9606
3	COMPARTMENTS	GOCC:0000323	6	0.030	9606
4	COMPARTMENTS	GOCC:0000502	1	0.005	9606
5	COMPARTMENTS	GOCC:0000785	1	0.005	9606
6	COMPARTMENTS	GOCC:0000791	1	0.005	9606
7	COMPARTMENTS	GOCC:0001533	1	0.005	9606
8	COMPARTMENTS	GOCC:0001650	1	0.005	9606
9	COMPARTMENTS	GOCC:0001669	1	0.005	9606
10	COMPARTMENTS	GOCC:0001725	2	0.010	9606
11	COMPARTMENTS	GOCC:0001726	2	0.010	9606
12	COMPARTMENTS	GOCC:0005576	44	0.220	9606
13	COMPARTMENTS	GOCC:0005577	1	0.005	9606
14	COMPARTMENTS	GOCC:0005579	3	0.015	9606
15	COMPARTMENTS	GOCC:0005581	1	0.005	9606
16	COMPARTMENTS	GOCC:0005604	2	0.010	9606
17	COMPARTMENTS	GOCC:0005610	1	0.005	9606
18	COMPARTMENTS	GOCC:0005615	23	0.115	9606
19	COMPARTMENTS	GOCC:0005622	102	0.510	9606
20	COMPARTMENTS	GOCC:0005634	34	0.170	9606

```

1

```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

```

11                      Ruffle
12          Extracellular region
13          Fibrinogen complex
14          Membrane attack complex
15          Collagen trimer
16          Basement membrane
17          laminin-5 complex
18          Extracellular space
19 Intracellular anatomical structure
20                      Nucleus

```

4 CLUSTERING

The iGraph package provides several clustering/community algorithms: "fastgreedy", "walktrap", "springlass", "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])

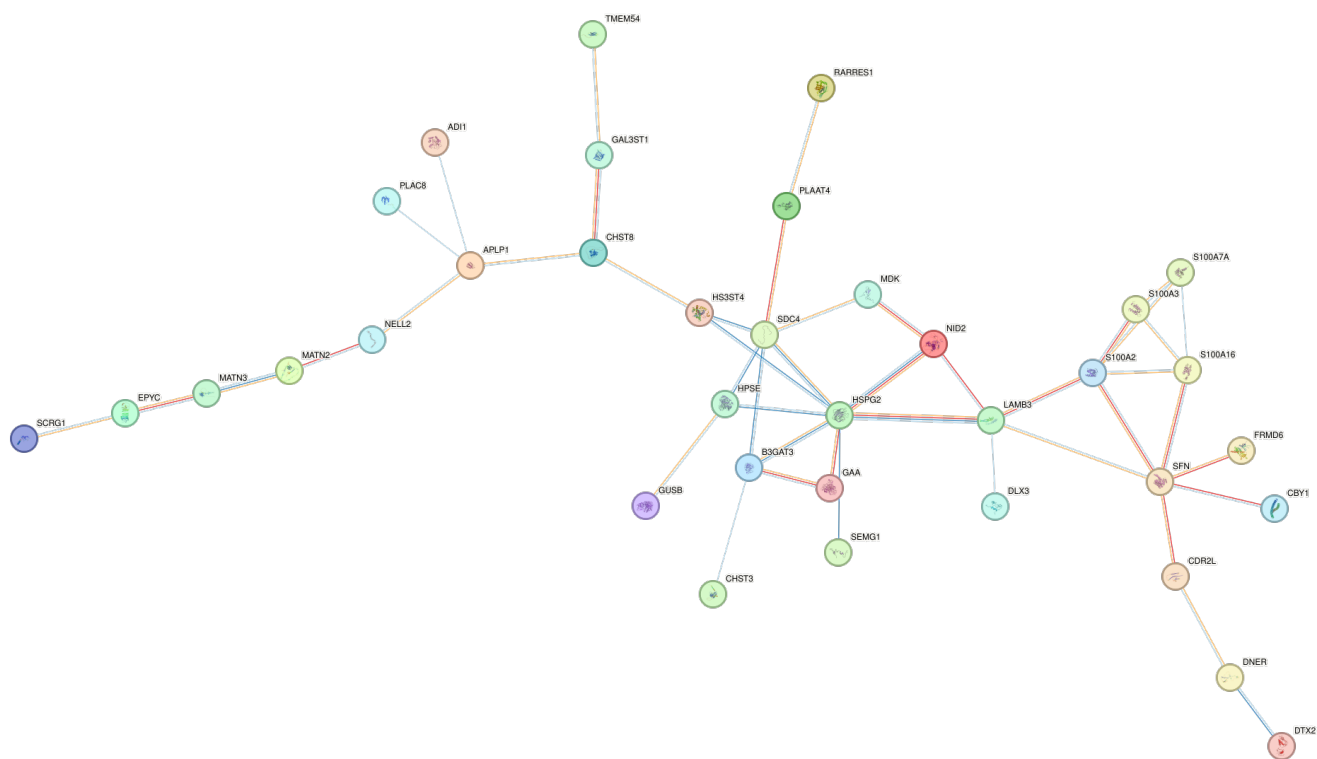
```

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)

```



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" )
> mdm2 = string_db$mp( "mdm2" )
```

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.

6 TYPED NETWORK VIEWS AND REGULATORY EDGES

The full functional network, physical network, and regulatory network have different edge semantics. The functional and physical networks are undirected, while the regulatory network is directed from regulator to target. Create a separate **STRINGdb** object for each network type that you need; they can use the same cache directory because their downloaded link files have distinct names. The **string_db** object created above is the functional network object.

```

> functional_db <- string_db
> physical_db <- STRINGdb$new(
+   version = "12.5", species = 9606, score_threshold = 400,
+   network_type = "physical", link_data = "detailed",
+   input_directory = functional_db$input_directory
+ )
> regulatory_db <- STRINGdb$new(
+   version = "12.5", species = 9606, score_threshold = 400,
+   network_type = "regulatory", link_data = "detailed",
+   input_directory = functional_db$input_directory
+ )

```

Use the typed view on the functional object to overlay the physical and regulatory edge types. The three typed-edge arguments control which edge types and regulatory signs are shown:

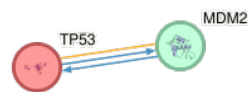
```

> functional_db$plot_network(
+   c(tp53, mdm2),
+   network_flavor = "typed",
+   typed_physical_edges = TRUE,
+   typed_regulatory_edges = TRUE,
+   show_regulatory_signs = TRUE
+ )

```

proteins: 2

round too large! -- due to technical limitations STRING does not compute PPI enrichment for query and/or background
expected interactions: (p-value:)



For analysis of regulation, use the regulatory object itself. Its interaction table retains direction: **from** is the regulator and **to** is its target. For example, the following retrieves regulatory links within the TP53–MDM2 pair and lists them as **from_name** -> **to_name**.

This pair illustrates a well-characterized negative-feedback circuit: TP53 transcriptionally activates MDM2, whereas MDM2 promotes proteasome-dependent degradation of TP53. Both directions are therefore expected in the regulatory graph.

```
> regulatory_edges <- regulatory_db$get_interactions(c(tp53, mdm2))
> regulatory_edges$regulatory_edge <- paste(
+   regulatory_edges$from_name, "->", regulatory_edges$to_name
+ )
> regulatory_edges[, c("regulatory_edge", "combined_score")]
```

	regulatory_edge	combined_score
1	MDM2 -> TP53	997
2	TP53 -> MDM2	996

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	cooccurrence	coexpression	experimental	database	textmining
1	0	0	0	0	999	900	954

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
32004	9606.ENSP00000269305	9606.ENSP00000212015	999	999
37406	9606.ENSP00000269305	9606.ENSP00000215832	999	999
53636	9606.ENSP00000269305	9606.ENSP00000216911	999	999
55816	9606.ENSP00000269305	9606.ENSP00000217244	999	999
66287	9606.ENSP00000269305	9606.ENSP00000219548	999	999
386993	9606.ENSP00000278616	9606.ENSP00000265433	999	999
419128	9606.ENSP00000278616	9606.ENSP00000269305	999	999
461927	9606.ENSP00000278616	9606.ENSP00000325863	999	992
461988	9606.ENSP00000278616	9606.ENSP00000343741	999	972
462088	9606.ENSP00000278616	9606.ENSP00000361423	999	999

	from_name	to_name
32004	TP53	SIRT1
37406	TP53	MAPK1
53636	TP53	AURKA
55816	TP53	CSNK2A1
66287	TP53	STUB1
386993	ATM	NBN
419128	ATM	TP53

461927	ATM	MRE11
461988	ATM	ATR
462088	ATM	ABL1

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
32004	TP53	SIRT1	999
37406	TP53	MAPK1	999
53636	TP53	AURKA	999
55816	TP53	CSNK2A1	999
66287	TP53	STUB1	999
386993	ATM	NBN	999
419128	ATM	TP53	999
461927	ATM	MRE11	992
461988	ATM	ATR	972
462088	ATM	ABL1	999

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.ENSEP00000269305	9606	9606.ENSEP00000269305
	bitscore			
1	815.8			

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

```
> # get the best hits of TP53 in all STRING species
> tp53_bests <- string_db$get_homologs_bests(tp53)
> tp53_bests_100 <- subset(tp53_bests, bitscore > 100)
> nrow(tp53_bests_100)
```

```
[1] 318
```

```
> head(tp53_besthits_100, n=10)
```

	ncbiTaxonId_A	stringId_A	ncbiTaxonId_B	stringId_B
1	9606	9606.ENSPPAP000000269305	9597	9597.ENSPPAP00000011040
2	9606	9606.ENSPPAP000000269305	9598	9598.ENSPPAP00000014836
3	9606	9606.ENSPPAP000000269305	9606	9606.ENSPPAP000000269305
4	9606	9606.ENSPPAP000000269305	9601	9601.ENSPPAP00000008923
5	9606	9606.ENSPPAP000000269305	61621	61621.ENSPPAP00000028632
6	9606	9606.ENSPPAP000000269305	61622	61622.ENSPPAP00000026843
7	9606	9606.ENSPPAP000000269305	591936	591936.ENSPPAP00000010553
8	9606	9606.ENSPPAP000000269305	60711	60711.ENSPPAP00000007551
9	9606	9606.ENSPPAP000000269305	9541	9541.ENSPPAP00000038526
10	9606	9606.ENSPPAP000000269305	9544	9544.ENSPPAP00000011334

	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.ENSPPAP000000278616	10090	10090.ENSPPAP000000156344
2	9606	9606.ENSPPAP000000269305	10090	10090.ENSPPAP000000104298

	bitscore
1	5328.8
2	598.2

7 CITATION

Please cite:

Szklarczyk D, Nastou K, Koutrouli M, Kirsch R, Mehryary F, Hachilif R, Hu D, Peluso ME, Huang Q, Fang T, Doncheva NT, Pyysalo S, Bork P, Jensen LJ, von Mering C. The STRING database in 2025: protein networks with directionality of regulation. *Nucleic Acids Res.* 2025 Jan 6;53(D1):D730-D737. doi: 10.1093/nar/gkaf1113.