

STRINGdb Package Vignette

Andrea Franceschini

15 July 2013

1 INTRODUCTION

STRING (<http://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. We currently cover 5'214'234 proteins from 1133 organisms.

As you will learn in this guide, the STRING database can be useful to add meaning to list of genes (e.g. the best hits coming out from a screen or the most differentially expressed genes coming out from a Microarray/RNAseq 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 use this function to retrieve/search our species table:

```
> get_STRING_species(version="9_05", species_name=NULL)

> library(STRINGdb)
> string_db <- STRINGdb$new( version="9_05", species=9606,
+                             score_threshold=0, input_directory="" )
```

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 !).

Besides, if you specify a local directory to the parameter input-directory, all the database files will be downloaded into this directory and the package can then 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.

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

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

[1] "add_diff_exp_color"      "add_proteins_description"
[3] "callSuper"               "copy"
[5] "export"                  "field"
[7] "getClass"                "getRefClass"
[9] "get_aliases"             "get_annotations"
[11] "get_annotations_desc"    "get_bioc_graph"
[13] "get_clusters"            "get_enrichment"
[15] "get_graph"               "get_homologs"
[17] "get_homologs_besthits"    "get_interactions"
[19] "get_link"                "get_neighbors"
[21] "get_png"                 "get_ppi_enrichment"
[23] "get_ppi_enrichment_full" "get_proteins"
[25] "get_pubmed"              "get_pubmed_interaction"
[27] "get_subnetwork"          "get_summary"
[29] "get_term_proteins"        "import"
[31] "initFields"              "initialize"
[33] "load"                    "load_all"
[35] "map"                     "mp"
[37] "plot_network"            "plot_ppi_enrichment"
[39] "post_payload"            "set_background"
[41] "show"                    "show#envRefClass"
[43] "trace"                   "untrace"
[45] "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 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 17% of your identifiers

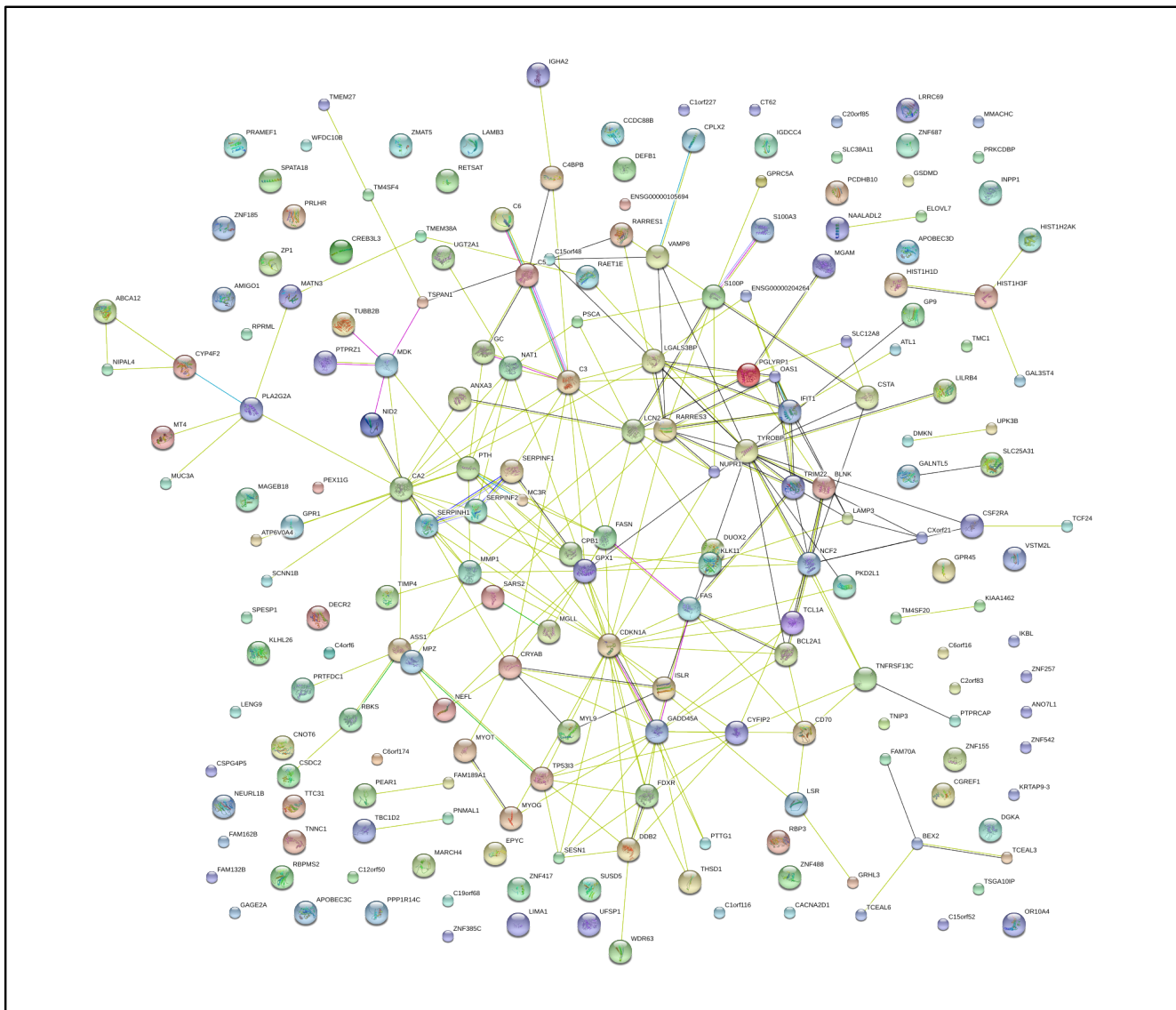
As you may have noticed, the previous command prints a warning showing the number of genes that we failed to map. In this particular example, we cannot map all the probes of the microarray that refer to position of the chromosome that are not assigned to a real gene (i.e. all the LOC genes). If we remove 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 best 200 genes and we produce an image of the STRING network for those. 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. Besides, on the bottom, there is a short-url that points to the relative page on our web-interface. You can copy and paste that url on your favorite browser and/or send it via e-mail to your colleagues or insert it in publications.

```
> hits <- example1_mapped$STRING_id[1:200]
> string_db$plot_network( hits )
```

proteins: 200
interactions: 234
expected interactions: 140 (p-value: 4.86055640180894e-13)



http://string905.embl.de/9_05/p/280242738

2 PAYLOAD MECHANISM

This R library provides the ability to interact with the STRING payload mechanism. The payload appears as an additional colored "halo" that starts from the border of the bubbles.

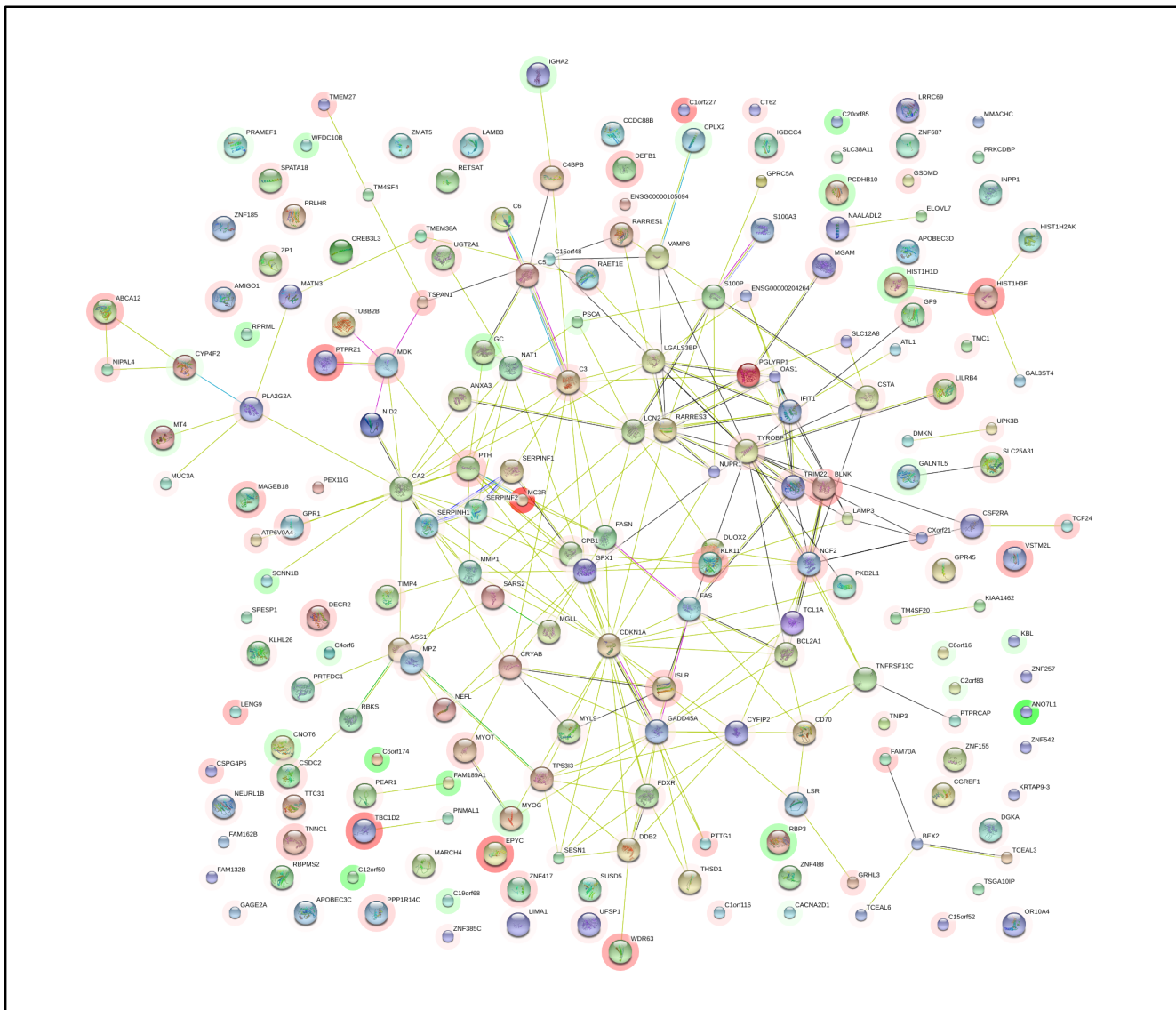
For example, this allows to color in green the genes that are down-regulated and in red the genes that are up-regulated. For this mechanism to work, we provide a function that posts the information on our web server.

```
> # 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: 234
expected interactions: 140 (p-value: 4.86055640180894e-13)

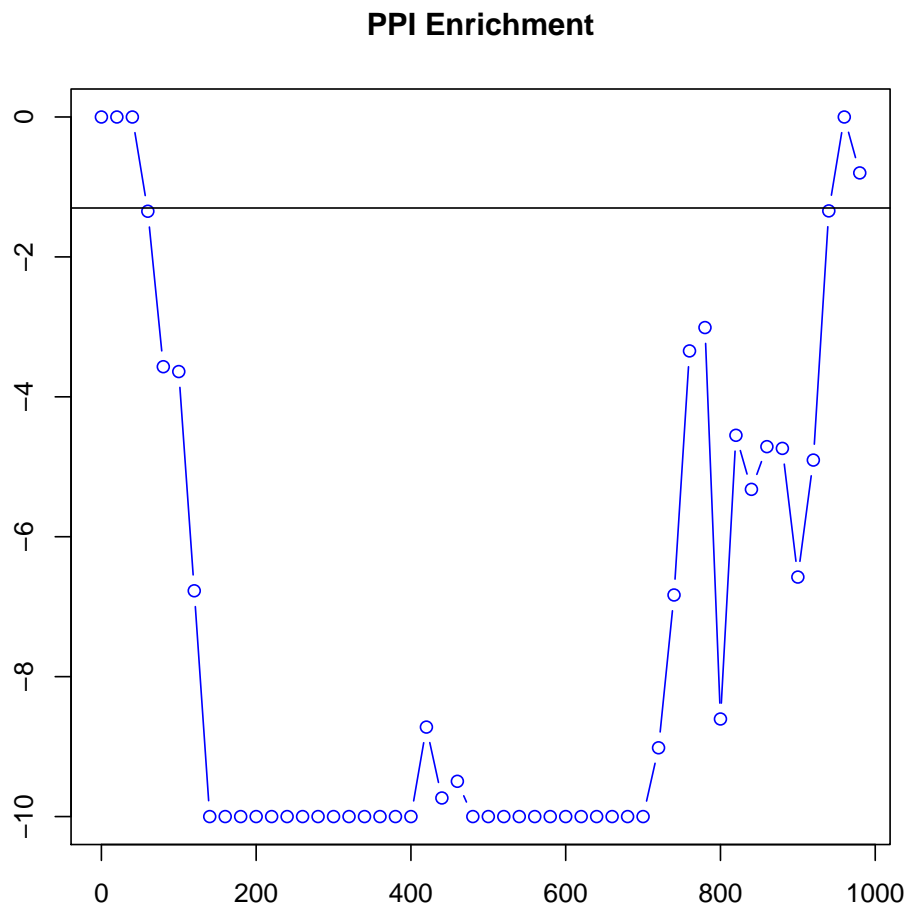


http://string905.embl.de/9_05/p/856242739

3 ENRICHMENT

We provide a method to compute (and visualize) the enrichment in protein-protein interactions along a sorted list of proteins. In the context of genome-wide screens (e.g. RNAi or proteomics), it can be useful to visualize the distribution of the enrichment in the resulting sorted list of genes. If the experiment was successful, and the top hits have protein-protein interactions, you should see more enrichment at the beginning of the list than at the end. Besides, you can also use the enrichment graph to help you to define a threshold on the number of proteins to consider (for example, if you see a strong enrichment up to position 600 on your list, this means that the signal is probably sparsed to cover the best 600 genes).

```
> # plot the enrichment for the best 1000 genes  
> string_db$plot_ppi_enrichment( example1_mapped$STRING_id[1:1000], quiet=TRUE )
```



We also provide a method to compute the enrichment in Gene Ontology, KEGG pathway and Interpro domains, similar to the "enrichment" widget in our web-interface. The enrichment is computed using an hypergeometric test. The multiple testing correction method can be changed (setting the methodMT parameter) and it is also possible to specify whether to use the "Inferred from Elettronic Annotations" or only the manually curated annotations.

```
> enrichmentGO <- string_db$get_enrichment( hits, category = "Process", methodMT = "fdr", iea = TRUE )
> enrichmentKEGG <- string_db$get_enrichment( hits, category = "KEGG", methodMT = "fdr", iea = TRUE )
> head(enrichmentGO, n=7)
```

	term_id	proteins	hits	pvalue	pvalue_fdr
1	GO:0002252	332	13	3.500911e-05	0.04780871
2	GO:0001970	2	2	1.030360e-04	0.04780871
3	GO:0045917	2	2	1.030360e-04	0.04780871
4	GO:2000259	2	2	1.030360e-04	0.04780871
5	GO:0002443	162	8	2.526033e-04	0.07795678
6	GO:0045071	32	4	2.976325e-04	0.07795678
7	GO:0001969	3	2	3.070372e-04	0.07795678

	term_description
1	immune effector process
2	positive regulation of activation of membrane attack complex
3	positive regulation of complement activation
4	positive regulation of protein activation cascade
5	leukocyte mediated immunity
6	negative regulation of viral genome replication
7	regulation of activation of membrane attack complex

```
> head(enrichmentKEGG, n=7)
```

	term_id	proteins	hits	pvalue	pvalue_fdr
1	hsa04115	67	6	7.347871e-05	0.007494828
2	hsa04610	89	5	2.565305e-03	0.130830543
3	hsa00590	61	3	2.759402e-02	0.770321039
4	hsa04966	33	2	4.877055e-02	0.770321039
5	hsa00061	6	1	6.300321e-02	0.770321039
6	hsa00232	6	1	6.300321e-02	0.770321039
7	hsa05219	41	2	7.173413e-02	0.770321039

	term_description
1	p53 signaling pathway
2	Complement and coagulation cascades
3	Arachidonic acid metabolism
4	Collecting duct acid secretion
5	Fatty acid biosynthesis
6	Caffeine metabolism
7	Bladder cancer

If you have performed your experiment on a pre-defined 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 methods explained above, you should set the background:

```
> backgroundV <- example1_mapped$STRING_id[1:2000] # as an example, we use the first 2000 genes
> string_db$set_background(backgroundV)
```

You can also set the background when you instantiate the STRINGdb object:

```
> string_db <- STRINGdb$new( score_threshold=0, backgroundV = backgroundV )
```

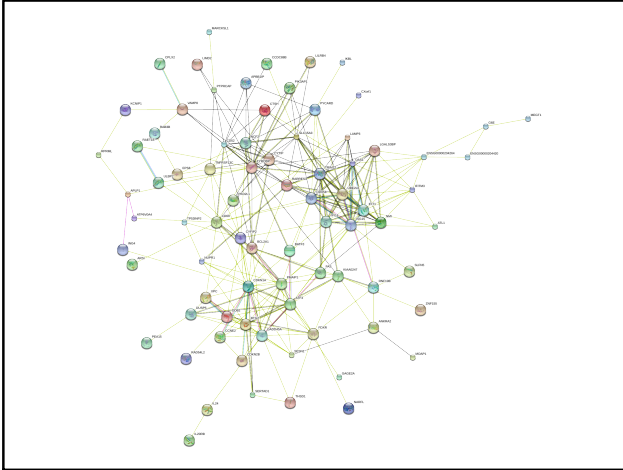
4 CLUSTERING

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$get_clusters(example1_mapped$STRING_id[1:600])

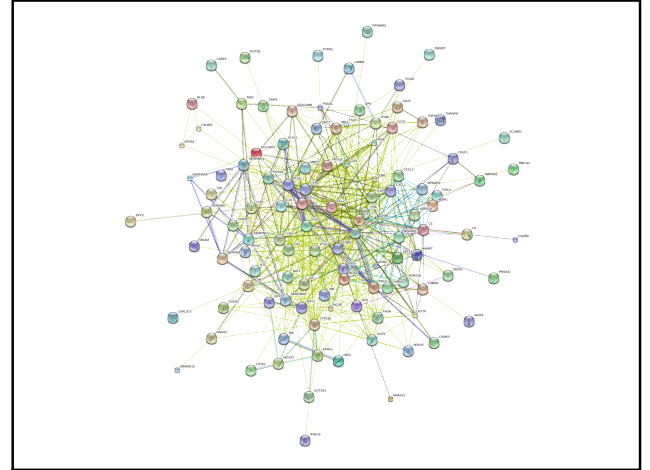
> # plot first 4 clusters
> par(mfrow=c(2,2))
> for(i in seq(1:4)){
+   string_db$plot_network(clustersList[[i]])
+ }
```

proteins: 80
interactions: 278
expected interactions: 39 (p-value: 0)



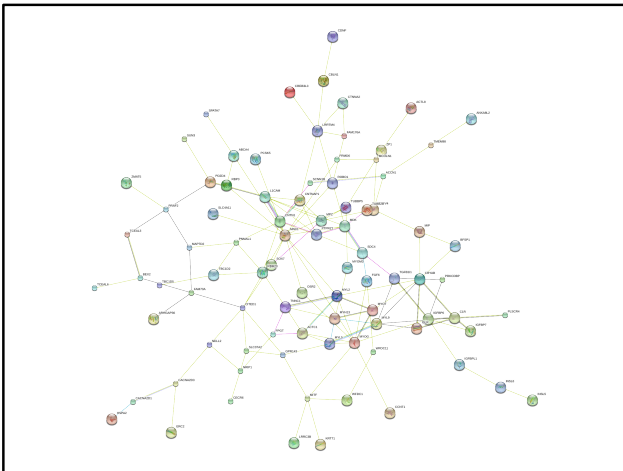
http://string905.embl.de/9_05/p/483142740

proteins: 112
interactions: 673
expected interactions: 180 (p-value: 0)



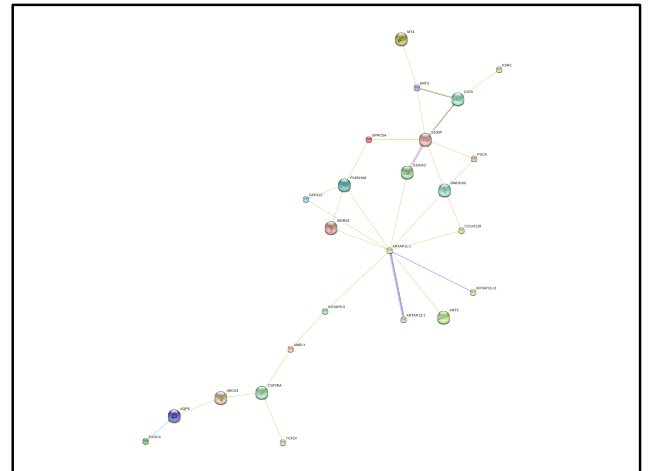
http://string905.embl.de/9_05/p/797642741

proteins: 87
interactions: 138
expected interactions: 21 (p-value: 0)



http://string905.embl.de/9_05/p/431042742

proteins: 24
interactions: 30
expected interactions: 0 (p-value: 0)



http://string905.embl.de/9_05/p/473442743

5 ADDITIONAL PROTEIN INFORMATION

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

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

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.

Using the following method, you can see the proteins that interact with one or more of your proteins:

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

It is also possible to retrieve the interactions that connect certain input proteins between each other. Using the "get-interactions" method we can clearly see that TP53 and ATM interact with each other with a good evidence/score.

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

	from	to	neighborhood	fusion	cooccurrence
1	9606.ENSP00000269305	9606.ENSP00000278616	0	0	0
	coexpression	experimental	database	textmining	combined_score
1	0	999	900	965	999

Using the get-pubmed-interactions method we can retrieve the pubmed identifiers of the articles that contain the name of both the proteins (if any). Such articles could support to the interaction.

```
> string_db$get_pubmed_interaction( tp53, atm )
```

STRING also provides a way to get homologous proteins: in our database we store an ALL-AGAINST-ALL blast alignment of all our 5 million proteins. The method "get-homologs-besthits" can be used to get the best hits of your proteins in all the >1000 STRING species (with the "sybets" parameter you can limit the search to the reciprocal-best-hits. This increase the confidence to get orthologs and not paralogous proteins)

```
> # get the reciprocal best hits of the following protein in all the STRING species
> string_db$get_homologs_bests hits(tp53, symbets = TRUE)
```

In addition, you can get all the homologous in a given species (i.e. all the blast hits):

```
> # get the homologs of the following two proteins in the mouse (i.e. species_id=10090)
> string_db$get_homologs(c(tp53, atm), target_species_id=10090, bitscore_threshold=60 )
```

6 CITATION

Please cite:

Franceschini, A et al. (2013). STRING v9.1: protein-protein interaction networks, with increased coverage and integration. In: Nucleic Acids Res. 2013 Jan;41(Database issue):D808-15. doi: 10.1093/nar/gks1094. Epub 2012 Nov 29