

# Package ‘simplifyEnrichment’

October 17, 2024

**Type** Package

**Title** Simplify Functional Enrichment Results

**Version** 1.14.1

**Date** 2024-09-11

**Depends** R (>= 3.6.0), BiocGenerics, grid

**Imports** GOSemSim, ComplexHeatmap (>= 2.7.4), circlize, GetoptLong, digest, tm, GO.db, org.Hs.eg.db, AnnotationDbi, slam, methods, clue, grDevices, graphics, stats, utils, proxyC, Matrix, cluster (>= 1.14.2), colorspace, GlobalOptions (>= 0.1.0)

**Suggests** knitr, ggplot2, cowplot, mclust, apcluster, MCL, dbscan, igraph, gridExtra, dynamicTreeCut, testthat, gridGraphics, clusterProfiler, msigdb, DOSE, DO.db, reactome.db, flexclust, BiocManager, InteractiveComplexHeatmap (>= 0.99.11), shiny, shinydashboard, cola, hu6800.db, rmarkdown, genefilter, gridtext, fpc

## Description

A new clustering algorithm, “binary cut”, for clustering similarity matrices of functional terms is implemented in this package. It also provides functions for visualizing, summarizing and comparing the clusterings.

**biocViews** Software, Visualization, GO, Clustering, GeneSetEnrichment

**URL** <https://github.com/jokergoo/simplifyEnrichment>,  
<https://simplifyEnrichment.github.io>

**VignetteBuilder** knitr

**License** MIT + file LICENSE

**git\_url** <https://git.bioconductor.org/packages/simplifyEnrichment>

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** 8c53ba8

**git\_last\_commit\_date** 2024-09-11

**Repository** Bioconductor 3.19

**Date/Publication** 2024-10-16

**Author** Zuguang Gu [aut, cre] (<<https://orcid.org/0000-0002-7395-8709>>)

**Maintainer** Zuguang Gu <z.gu@dkfz.de>

## Contents

all_clustering_methods . . . . .	3
anno_word_cloud . . . . .	4
anno_word_cloud_from_GO . . . . .	5
area_above_ecdf . . . . .	6
binary_cut . . . . .	7
cluster_by_apcluster . . . . .	8
cluster_by_dynamicTreeCut . . . . .	8
cluster_by_hdbscan . . . . .	9
cluster_by_igraph . . . . .	10
cluster_by_kmeans . . . . .	11
cluster_by_MCL . . . . .	11
cluster_by_mclust . . . . .	12
cluster_by_pam . . . . .	13
cluster_terms . . . . .	13
cmp_make_clusters . . . . .	15
cmp_make_plot . . . . .	16
compare_clustering_methods . . . . .	17
count_words . . . . .	18
dend_node_apply . . . . .	19
difference_score . . . . .	20
DO_similarity . . . . .	21
edit_node . . . . .	22
export_to_shiny_app . . . . .	23
GO_similarity . . . . .	23
guess_ont . . . . .	24
heightDetails.word_cloud . . . . .	25
ht_clusters . . . . .	26
keyword_enrichment_from_GO . . . . .	27
partition_by_hclust . . . . .	28
partition_by_kmeans . . . . .	29
partition_by_kmeanspp . . . . .	30
partition_by_pam . . . . .	30
plot_binary_cut . . . . .	31
random_DO . . . . .	32
random_GO . . . . .	32
register_clustering_methods . . . . .	33
remove_clustering_methods . . . . .	34
reset_clustering_methods . . . . .	34
scale_fontsize . . . . .	35
select_cutoff . . . . .	36
se_opt . . . . .	37
simplifyEnrichment . . . . .	37

simplifyGO . . . . .	38
simplifyGOFromMultipleLists . . . . .	40
subset_enrichResult . . . . .	42
summarizeGO . . . . .	43
term_similarity . . . . .	44
term_similarity_from_enrichResult . . . . .	45
term_similarity_from_gmt . . . . .	46
term_similarity_from_KEGG . . . . .	46
term_similarity_from_MSigDB . . . . .	47
term_similarity_from_Reactome . . . . .	48
widthDetails.word_cloud . . . . .	48
word_cloud_grob . . . . .	49

**Index****51**

all\_clustering\_methods

*All clustering methods***Description**

All clustering methods

**Usage**

all\_clustering\_methods()

**Details**

The default clustering methods are:

kmeans see [cluster\\_by\\_kmeans](#).  
dynamicTreeCut see [cluster\\_by\\_dynamicTreeCut](#).  
mclust see [cluster\\_by\\_mclust](#).  
apcluster see [cluster\\_by\\_apcluster](#).  
hdbscan see [cluster\\_by\\_hdbscan](#).  
fast\_greedy see [cluster\\_by\\_igraph](#).  
louvain see [cluster\\_by\\_igraph](#).  
walktrap see [cluster\\_by\\_igraph](#).  
MCL see [cluster\\_by\\_MCL](#).  
binary\_cut see [binary\\_cut](#).

**Value**

A vector of method names.

**See Also**

New methods can be added by [register\\_clustering\\_methods](#).

**Examples**

```
all_clustering_methods()
```

---

anno_word_cloud	<i>Word cloud annotations</i>
-----------------	-------------------------------

---

**Description**

Word cloud annotations

**Usage**

```
anno_word_cloud(align_to, term, exclude_words = NULL, max_words = 10,
  word_cloud_grob_param = list(), fontsize_range = c(4, 16), value_range = NULL,
  bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA"), side = c("right", "left"),
  add_new_line = FALSE, count_words_param = list(), ..., return_gbl = FALSE)
```

**Arguments**

- |                       |   |
|-----------------------|---|
| align_to              | How to align the annotations to the heatmap. Similar as in <a href="#">anno_link</a> , the value of <code>align_to</code> can be a list of row indices or a categorical vector where each vector in the list corresponds to a word cloud. If it is a categorical vector, rows with the same level correspond to a same word cloud. If <code>align_to</code> is a categorical vector and <code>term</code> is a list, names of <code>term</code> should have overlap to the levels in <code>align_to</code> . When <code>align_to</code> is set as a categorical vector, normally the same value is set to <code>row_split</code> in the main heatmap so that each row slice can correspond to a word cloud.   |
| term                  | The description text used for constructing the word clouds. The value should have the same format as <code>align_to</code> . If <code>align_to</code> is a list, <code>term</code> should also be a list. In this case, the length of vectors in <code>term</code> is not necessarily the same as in <code>align_to</code> . E.g. <code>length(term[[1]])</code> is not necessarily equal to <code>length(align_to[[1]])</code> . If <code>align_to</code> is a categorical vector, <code>term</code> should also be a character vector with the same length as <code>align_to</code> . To make it more general, when <code>align_to</code> is a list, <code>term</code> can also be a list of data frames where the first column contains keywords and the second column contains numeric values that will be mapped to font sizes in the word clouds. |
| exclude_words         | The words excluded for constructing word cloud.   |
| max_words             | Maximal number of words visualized in the word cloud.   |
| word_cloud_grob_param | A list of graphics parameters passed to <a href="#">word_cloud_grob</a> .   |
| fontsize_range        | The range of the font size. The value should be a numeric vector with length two. The font size interpolation is linear.  |

value_range	The range of values to map to font sizes.
bg_gp	Graphics parameters for controlling the background.
side	Side of the annotation relative to the heatmap.
add_new_line	Whether to add new line after every word? If TRUE, each word will be in a separated line.
count_words_param	A list of parameters passed to <a href="#">count_words</a> .
...	Other parameters.
return_gbl	Internally used.

### Details

The word cloud annotation is constructed by [anno\\_link](#).

If the annotation is failed to construct or no keyword is found, the function returns a [anno\\_empty](#) with 1px width.

English stop words, punctuation and numbers are removed by default when counting words. As specific stop words might coincide with gene or pathway names, and numbers in genes names might be meaningful it is recommended to adjust this behaviour by passing appropriate arguments to the [count\\_words](#) function using `count_words_param`.

### Examples

```
gm = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
go_id = rownames(gm)
go_term = AnnotationDbi::select(GO.db::GO.db, keys = go_id, columns = "TERM")$TERM

split = sample(letters[1:4], 100, replace = TRUE)
align_to = split(1:100, split)
term = lapply(letters[1:4], function(x) sample(go_term, sample(100:400, 1)))
names(term) = letters[1:4]

require(ComplexHeatmap)
mat = matrix(rnorm(100*10), nrow = 100)
Heatmap(mat, cluster_rows = FALSE, row_split = split,
right_annotation = rowAnnotation(foo = anno_word_cloud(align_to, term)))
```

---

anno\_word\_cloud\_from\_GO

*Word cloud annotations from GO*

---

### Description

Word cloud annotations from GO

**Usage**

```
anno_word_cloud_from_GO(align_to, go_id, stat = c("pvalue", "count"),
  min_stat = ifelse(stat == "count", 5, 0.05),
  term = NULL, exclude_words = NULL, ...)
```

**Arguments**

align_to	The same format as in <a href="#">anno_word_cloud</a> .
go_id	The value should be in the same format as align_to. If go_id is a vector, it should have the same length as align_to, and if go_id is a list, note, e.g. length(go_id[[1]]) is not necessarily equal to length(align_to[[1]]. If align_to is a categorical vector and go_id is a list, names of go_id should have overlap to the levels in align_to.
min_stat	Minimal value for stat for selecting keywords.
stat	What type of value to map to font sizes of the keywords. There are two possible values. "pvalue": enrichment is applied to keywords and $-\log_{10}(\text{p-value})$ is used to map to font size; "count": simply word frequency of keywords.
term	Alternatively the GO description can be set via the term argument. The same format as in <a href="#">anno_word_cloud</a> .
exclude_words	The words excluded for constructing word cloud. Some words are internally excluded: c("via", "protein", "factor", "side", "type", "specific").
...	All other arguments passed to <a href="#">anno_word_cloud</a> .

**Examples**

```
# There is no example
NULL
```

---

area_above_ecdf	<i>Area above the eCDF curve</i>
-----------------	----------------------------------

---

**Description**

Area above the eCDF curve

**Usage**

```
area_above_ecdf(x)
```

**Arguments**

x	A vector of similarity values.
---	--------------------------------

**Details**

Denote  $F(x)$  as the eCDF (empirical Cumulative Distribution Function) of the similarity vector  $x$ , this function calculates the area above the eCDF curve, which is  $1 - \int_0^1 F(x)dx$ .

**Value**

A numeric value.

**Examples**

```
# There is no example
NULL
```

---

binary_cut	<i>Cluster functional terms by recursively binary cutting the similarity matrix</i>
------------	---

---

**Description**

Cluster functional terms by recursively binary cutting the similarity matrix

**Usage**

```
binary_cut(mat, value_fun = area_above_ecdf, partition_fun = partition_by_pam,
           cutoff = 0.85, try_all_partition_fun = FALSE, partial = FALSE)
```

**Arguments**

mat	A similarity matrix.
value_fun	A function that calculates the scores for the four submatrices on a node.
partition_fun	A function to split each node into two groups. Pre-defined functions in this package are <a href="#">partition_by_kmeanspp</a> , <a href="#">partition_by_pam</a> and <a href="#">partition_by_hclust</a> .
cutoff	The cutoff for splitting the dendrogram.
try_all_partition_fun	Different partition_fun gives different clusterings. If the value of try_all_partition_fun is set to TRUE, the similarity matrix is clustered by three partitioning method: <a href="#">partition_by_pam</a> , <a href="#">partition_by_kmeanspp</a> and <a href="#">partition_by_hclust</a> . The clustering with the highest difference score is finally selected as the final clustering.
partial	Whether to generate the complete clustering or the clustering stops when submatrices cannot be split anymore.

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
binary_cut(mat)
```

---

cluster\_by\_apcluster *Cluster similarity matrix by apcluster*

---

**Description**

Cluster similarity matrix by apcluster

**Usage**

```
cluster_by_apcluster(mat, s = apcluster::negDistMat(r = 2), ...)
```

**Arguments**

mat	The similarity matrix.
s	Passed to the s argument in <a href="#">apcluster</a> .
...	Other arguments passed to <a href="#">apcluster</a> .

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example
NULL
```

---

cluster\_by\_dynamicTreeCut  
*Cluster similarity matrix by dynamicTreeCut*

---

**Description**

Cluster similarity matrix by dynamicTreeCut

**Usage**

```
cluster_by_dynamicTreeCut(mat, minClusterSize = 5, ...)
```



### Arguments

mat                    The similarity matrix.  
minClusterSize    Minimal number of objects in a cluster. Pass to [cutreeDynamic](#).  
...                    Other arguments passed to [cutreeDynamic](#).

### Value

A vector of cluster labels (in numeric).

### Examples

```
# There is no example  
NULL
```

---

cluster\_by\_hdbscan    *Cluster similarity matrix by hdbscan*

---

### Description

Cluster similarity matrix by hdbscan

### Usage

```
cluster_by_hdbscan(mat, minPts = 5, ...)
```

### Arguments

mat                    The similarity matrix.  
minPts                Passed to the minPts argument in [hdbscan](#).  
...                    Other arguments passed to [hdbscan](#).

### Value

A vector of cluster labels (in numeric).

### Examples

```
# There is no example  
NULL
```

---

cluster\_by\_igraph      *Cluster similarity matrix by graph community detection methods*

---

## Description

Cluster similarity matrix by graph community detection methods

## Usage

```
cluster_by_igraph(mat,  
  method = c("fast_greedy",  
    "leading_eigen",  
    "louvain",  
    "walktrap"),  
  ...)
```

## Arguments

mat	The similarity matrix.
method	The community detection method.
...	Other arguments passed to the corresponding community detection function, see Details.

## Details

The symmetric similarity matrix can be treated as an adjacency matrix and constructed as a graph/network with the similarity values as the weight of the edges. Thus, clustering the similarity matrix can be treated as detecting clusters/modules/communities from the graph.

Four methods implemented in igraph package can be used here:

fast\_greedy uses [cluster\\_fast\\_greedy](#).

leading\_eigen uses [cluster\\_leading\\_eigen](#).

louvain uses [cluster\\_louvain](#).

walktrap uses [cluster\\_walktrap](#).

## Value

A vector of cluster labels (in numeric).

## Examples

```
# There is no example  
NULL
```

---

cluster\_by\_kmeans      *Cluster similarity matrix by k-means clustering*

---

**Description**

Cluster similarity matrix by k-means clustering

**Usage**

```
cluster_by_kmeans(mat, max_k = max(2, min(round(nrow(mat)/5), 100)), ...)
```

**Arguments**

mat	The similarity matrix.
max_k	maximal k for k-means clustering.
...	Other arguments passed to <a href="#">kmeans</a> .

**Details**

The best number of k for k-means clustering is identified according to the "elbow" or "knee" method on the distribution of within-cluster sum of squares (WSS) at each k.

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example  
NULL
```

---

cluster\_by\_MCL      *Cluster similarity matrix by MCL*

---

**Description**

Cluster similarity matrix by MCL

**Usage**

```
cluster_by_MCL(mat, addLoops = TRUE, ...)
```

**Arguments**

mat	The similarity matrix.
addLoops	Passed to the addLoops argument in <a href="#">mcl</a> .
...	Other arguments passed to <a href="#">mcl</a> .

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example  
NULL
```

---

cluster_by_mclust	<i>Cluster similarity matrix by mclust</i>
-------------------	--

---

**Description**

Cluster similarity matrix by mclust

**Usage**

```
cluster_by_mclust(mat, G = seq_len(max(2, min(round(nrow(mat)/5), 100))), ...)
```

**Arguments**

mat	The similarity matrix.
G	Passed to the G argument in <a href="#">Mclust</a> .
...	Other arguments passed to <a href="#">Mclust</a> .

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example  
NULL
```

---

cluster_by_pam	<i>Cluster similarity matrix by pam clustering</i>
----------------	--

---

**Description**

Cluster similarity matrix by pam clustering

**Usage**

```
cluster_by_pam(mat, max_k = max(2, min(round(nrow(mat)/10), 100)), ...)
```

**Arguments**

mat	The similarity matrix.
max_k	maximal k for pam clustering.
...	Other arguments passed to <a href="#">pamk</a> .

**Details**

PAM is applied by [pamk](#) which can automatically select the best k.

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example
NULL
```

---

cluster_terms	<i>Cluster functional terms</i>
---------------	---------------------------------

---

**Description**

Cluster functional terms

**Usage**

```
cluster_terms(mat, method = "binary_cut", control = list(), catch_error = FALSE,
  verbose = TRUE)
```

## Arguments

mat	A similarity matrix.
method	Method for clustering the matrix.
control	A list of parameters passed to the corresponding clustering function.
catch_error	Internally used.
verbose	Whether to print messages.

## Details

The following methods are the default:

kmeans see [cluster\\_by\\_kmeans](#).  
pam see [cluster\\_by\\_pam](#).  
dynamicTreeCut see [cluster\\_by\\_dynamicTreeCut](#).  
mclust see [cluster\\_by\\_mclust](#).  
apcluster see [cluster\\_by\\_apcluster](#).  
hdbscan see [cluster\\_by\\_hdbscan](#).  
leading\_eigen see [cluster\\_by\\_igraph](#).  
louvain see [cluster\\_by\\_igraph](#).  
walktrap see [cluster\\_by\\_igraph](#).  
MCL see [cluster\\_by\\_MCL](#).  
binary\_cut see [binary\\_cut](#).

Also the user-defined methods in [all\\_clustering\\_methods](#) can be used here.

New clustering methods can be registered by [register\\_clustering\\_methods](#).

Please note it is better to directly call [cluster\\_terms](#) for clustering while not the individual `cluster_by_*` functions because [cluster\\_terms](#) does additional cluster label adjustment.

## Value

A numeric vector of cluster labels (in numeric).

If `catch_error` is set to TRUE and if the clustering produces an error, the function returns a try-error object.

## Examples

```
# There is no example  
NULL
```

---

cmp_make_clusters	<i>Apply various clustering methods</i>
-------------------	---

---

**Description**

Apply various clustering methods

**Usage**

```
cmp_make_clusters(mat, method = setdiff(all_clustering_methods(), "mclust"),  
  verbose = TRUE)
```

**Arguments**

mat	The similarity matrix.
method	Which methods to compare. All available methods are in <a href="#">all_clustering_methods</a> . A value of all takes all available methods. By default mclust is excluded because its long runtime.
verbose	Whether to print messages.

**Details**

The function compares following default clustering methods by default:

kmeans see [cluster\\_by\\_kmeans](#).  
pam see [cluster\\_by\\_pam](#).  
dynamicTreeCut see [cluster\\_by\\_dynamicTreeCut](#).  
mclust see [cluster\\_by\\_mclust](#). By default it is not included.  
apcluster see [cluster\\_by\\_apcluster](#).  
hdbscan see [cluster\\_by\\_hdbscan](#).  
fast\_greedy see [cluster\\_by\\_igraph](#).  
louvain see [cluster\\_by\\_igraph](#).  
walktrap see [cluster\\_by\\_igraph](#).  
MCL see [cluster\\_by\\_MCL](#).  
binary\_cut see [binary\\_cut](#).

Also the user-defined methods in [all\\_clustering\\_methods](#) are also compared.

**Value**

A list of cluster label vectors for different clustering methods.

## Examples

```
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
clt = cmp_make_clusters(mat)

## End(Not run)
```

---

cmp\_make\_plot

*Make plots for comparing clustering methods*

---

## Description

Make plots for comparing clustering methods

## Usage

```
cmp_make_plot(mat, clt, plot_type = c("mixed", "heatmap"), nrow = 3)
```

## Arguments

mat	A similarity matrix.
clt	A list of clusterings from <a href="#">cmp_make_clusters</a> .
plot_type	What type of plots to make. See Details.
nrow	Number of rows of the layout when plot_type is set to heatmap.

## Details

If plot\_type is the default value mixed, a figure with three panels generated:

- A heatmap of the similarity matrix with different classifications as row annotations.
- A heatmap of the pair-wise concordance of the classifications of every two clustering methods.
- Barplots of the difference scores for each method (calculated by [difference\\_score](#)), the number of clusters (total clusters and the clusters with size  $\geq 5$ ) and the mean similarity of the terms that are in the same clusters.

If plot\_type is heatmap. There are heatmaps for the similarity matrix under clusterings from different methods. The last panel is a table with the number of clusters under different clusterings.

## Value

No value is returned.



## Examples

```
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
clt = cmp_make_clusters(mat)
cmp_make_plot(mat, clt)
cmp_make_plot(mat, clt, plot_type = "heatmap")

## End(Not run)
```

---

compare\_clustering\_methods

*Compare clustering methods*

---

## Description

Compare clustering methods

## Usage

```
compare_clustering_methods(mat, method = setdiff(all_clustering_methods(), "mclust"),
  plot_type = c("mixed", "heatmap"), nrow = 3, verbose = TRUE)
```

## Arguments

mat	The similarity matrix.
method	Which methods to compare. All available methods are in <a href="#">all_clustering_methods</a> . A value of all takes all available methods. By default mclust is excluded because its long runtime.
plot_type	See explanation in <a href="#">cmp_make_plot</a> .
nrow	Number of rows of the layout when plot_type is set to heatmap.
verbose	Whether to print messages.

## Details

The function compares following clustering methods by default:

kmeans see [cluster\\_by\\_kmeans](#).  
pam see [cluster\\_by\\_pam](#).  
dynamicTreeCut see [cluster\\_by\\_dynamicTreeCut](#).  
mclust see [cluster\\_by\\_mclust](#). By default it is not included.  
apcluster see [cluster\\_by\\_apcluster](#).  
hdbscan see [cluster\\_by\\_hdbscan](#).  
fast\_greedy see [cluster\\_by\\_igraph](#).

louvain see [cluster\\_by\\_igraph](#).

walktrap see [cluster\\_by\\_igraph](#).

MCL see [cluster\\_by\\_MCL](#).

binary\_cut see [binary\\_cut](#).

This function is basically a wrapper function. It calls the following two functions:

- [cmp\\_make\\_clusters](#): applies clustering with different methods.
- [cmp\\_make\\_plot](#): makes the plots.

### Value

No value is returned.

### Examples

```
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
compare_clustering_methods(mat)
compare_clustering_methods(mat, plot_type = "heatmap")

## End(Not run)
```

---

count\_words

*Calculate word frequency*

---

### Description

Calculate word frequency

### Usage

```
count_words(term,
  exclude_words = NULL, stop_words = stopwords(),
  min_word_length = 1, tokenizer = 'words', transform_case = tolower,
  remove_numbers = TRUE, remove_punctuation = TRUE, custom_transformer = NULL,
  stemming = FALSE, dictionary = NULL)
```

### Arguments

term	A vector of description texts.
exclude_words	The words that should be excluded.
stop_words	The stop words that should be removed.
min_word_length	Minimum length of the word to be counted.

tokenizer        The tokenizer function, one of the values accepted by `tm::termFreq`.  
 transform\_case   The function normalizing lettercase of the words.  
 remove\_numbers   Whether to remove numbers.  
 remove\_punctuation        Whether to remove punctuation.  
 custom\_transformer        Custom function that transforms words.  
 stemming         Whether to only keep the roots of inflected words.  
 dictionary        A vector of words to be counted (if given all other words will be excluded).

### Details

The text preprocessing followings the instructions from <http://www.sthda.com/english/wiki/word-cloud-generator-in-r-one-killer-function-to-do-everything-you-need>.

### Value

A data frame with words and frequencies.

### Examples

```
gm = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
go_id = rownames(gm)
go_term = AnnotationDbi::select(GO.db::GO.db, keys = go_id, columns = "TERM")$TERM
count_words(go_term)
```

---

dend\_node\_apply        *Apply functions on every node in a dendrogram*

---

### Description

Apply functions on every node in a dendrogram

### Usage

```
dend_node_apply(dend, fun)
```

### Arguments

dend                A dendrogram.  
 fun                 A self-defined function.

**Details**

The function returns a vector or a list as the same length as the number of nodes in the dendrogram. The self-defined function can have one single argument which is the sub-dendrogram at a certain node. E.g. to get the number of members at every node:

```
dend_node_apply(dend, function(d) attr(d, "members"))
```

The self-defined function can have a second argument, which is the index of current sub-dendrogram in the complete dendrogram. E.g. `dend[[1]]` is the first child node of the complete dendrogram and `dend[[c(1, 2)]]` is the second child node of `dend[[1]]`, et al. This makes that at a certain node, it is possible to get information of its child nodes and parent nodes.

```
dend_node_apply(dend, function(d, index) {
  dend[[c(index, 1)]] # is the first child node of d, or simply d[[1]]
  dend[[index[-length(index)]]] # is the parent node of d
  ...
})
```

Note for the top node, the value of `index` is `NULL`.

**Value**

A vector or a list, depends on whether `fun` returns a scalar or more complex values.

**Examples**

```
mat = matrix(rnorm(100), 10)
dend = as.dendrogram(hclust(dist(mat)))
# number of members on every node
dend_node_apply(dend, function(d) attr(d, "members"))
# the depth on every node
dend_node_apply(dend, function(d, index) length(index))
```

---

difference_score	<i>Difference score</i>
------------------	-------------------------

---

**Description**

Difference score

**Usage**

```
difference_score(mat, cl)
```

**Arguments**

mat	The similarity matrix.
cl	Cluster labels.

### Details

This function measures the different between the similarity values for the terms that belong to the same clusters and in different clusters. The difference score is the Kolmogorov-Smirnov statistic between the two distributions.

### Value

A numeric scalar.

### Examples

```
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
cl = binary_cut(mat)
difference_score(mat, cl)
```

---

DO\_similarity

*Calculate Disease Ontology (DO) semantic similarity matrix*

---

### Description

Calculate Disease Ontology (DO) semantic similarity matrix

### Usage

```
DO_similarity(do_id, measure = "Rel", remove_orphan_terms = FALSE)
```

### Arguments

`do_id` A vector of DO IDs.

`measure` Semantic measure for the DO similarity, pass to [doSim](#).

`remove_orphan_terms` Whether to remove terms that have zero similarity to all other terms?

### Details

This function is basically a wrapper on [doSim](#).

### Value

A symmetric matrix.

### Examples

```
require(DOSE)
do_id = random_DO(10)
DO_similarity(do_id)
```

---

edit_node	<i>Modify nodes in a dendrogram</i>
-----------	-------------------------------------

---

**Description**

Modify nodes in a dendrogram

**Usage**

```
edit_node(dend, fun = function(d, index) d)
```

**Arguments**

dend	A dendrogram.
fun	A self-defined function.

**Details**

if fun only has one argument, it is basically the same as [dendapply](#), but it can have a second argument which is the index of the node in the dendrogram, which makes it possible to get information of child nodes and parent nodes for a specific node.

As an example, we first assign random values to every node in the dendrogram:

```
mat = matrix(rnorm(100), 10)
dend = as.dendrogram(hclust(dist(mat)))
dend = edit_node(dend, function(d) {attr(d, 'score') = runif(1); d})
```

Then for every node, we take the maximal absolute difference to all its child nodes and parent node as the attribute `abs_diff`

```
dend = edit_node(dend, function(d, index) {
  n = length(index)
  s = attr(d, "score")
  if(is.null(index)) { # d is the top node
    s_children = sapply(d, function(x) attr(x, "score"))
    s_parent = NULL
  } else if(is.leaf(d)) { # d is the leaf
    s_children = NULL
    s_parent = attr(dend[[index[-n]]], "score")
  } else {
    s_children = sapply(d, function(x) attr(x, "score"))
    s_parent = attr(dend[[index[-n]]], "score")
  }
  abs_diff = max(abs(s - c(s_children, s_parent)))
  attr(d, "abs_diff") = abs_diff
  return(d)
})
```

**Value**

A dendrogram object.

**Examples**

```
# There is no example
NULL
```

---

export\_to\_shiny\_app     *Interactively visualize the similarity heatmap*

---

**Description**

Interactively visualize the similarity heatmap

**Usage**

```
export_to_shiny_app(mat, cl = binary_cut(mat))
```

**Arguments**

**mat**                    A similarity matrix.

**cl**                     Cluster labels inferred from the similarity matrix, e.g. from [cluster\\_terms](#) or [binary\\_cut](#).

**Examples**

```
if(interactive()) {
  mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
    package = "simplifyEnrichment"))
  cl = binary_cut(mat)
  export_to_shiny_app(mat, cl)
}
```

---

GO\_similarity             *Calculate Gene Ontology (GO) semantic similarity matrix*

---

**Description**

Calculate Gene Ontology (GO) semantic similarity matrix

**Usage**

```
GO_similarity(go_id, ont = NULL, db = 'org.Hs.eg.db', measure = "Rel",
  remove_orphan_terms = FALSE)
```

**Arguments**

go_id	A vector of GO IDs.
ont	GO ontology. Value should be one of "BP", "CC" or "MF". If it is not specified, the function automatically identifies it by random sampling 10 IDs from go_id (see <a href="#">guess_ont</a> ).
db	Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a> . The value can also directly be a OrgDb object.
measure	Semantic measure for the GO similarity, pass to <a href="#">termSim</a> .
remove_orphan_terms	Whether to remove terms that have zero similarity to all other terms?

**Details**

This function is basically a wrapper on [termSim](#).

**Value**

A symmetric matrix.

**Examples**

```
go_id = random_GO(100)
mat = GO_similarity(go_id)
```

---

guess_ont	<i>Guess the ontology of the input GO IDs</i>
-----------	---

---

**Description**

Guess the ontology of the input GO IDs

**Usage**

```
guess_ont(go_id, db = 'org.Hs.eg.db')
```

**Arguments**

go_id	A vector of GO IDs.
db	Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a> . The value can also directly be a OrgDb object.

**Details**

10 GO IDs are randomly sampled and checked.



**Value**

A single character scalar of "BP", "CC" or "MF".

If there are more than one ontologies detected. It returns NULL.

**Examples**

```
go_id = random_GO(100)
guess_ont(go_id)
```

---

`heightDetails.word_cloud`  
*Height for word\_cloud grob*

---

**Description**

Height for word\_cloud grob

**Usage**

```
## S3 method for class 'word_cloud'
heightDetails(x)
```

**Arguments**

x                    The word\_cloud grob returned by [word\\_cloud\\_grob](#).

**Value**

A `unit` object.

**Examples**

```
# There is no example
NULL
```

ht\_clusters

*Visualize the similarity matrix and the clustering***Description**

Visualize the similarity matrix and the clustering

**Usage**

```
ht_clusters(
  mat,
  cl,
  dend = NULL,
  col = c("white", "red"),

  # arguments that control the word cloud annotation
  draw_word_cloud = TRUE,
  min_term = round(nrow(mat)*0.01),
  order_by_size = FALSE,
  stat = "pvalue",
  min_stat = ifelse(stat == "count", 5, 0.05),
  exclude_words = character(0),
  max_words = 10,
  word_cloud_grob_param = list(),
  fontsize_range = c(4, 16),
  bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA"),

  # arguments that control the heatmaps
  column_title = NULL,
  ht_list = NULL,
  use_raster = TRUE,
  run_draw = TRUE,
  ...)
```

**Arguments**

mat	A similarity matrix.
cl	Cluster labels inferred from the similarity matrix, e.g. from <a href="#">cluster_terms</a> or <a href="#">binary_cut</a> .
dend	Used internally.
col	A vector of colors that map from 0 to the 97.5 <sup>th</sup> percentile of the similarity values. The value can also be a color mapping function generated by <a href="#">colorRamp2</a> .
draw_word_cloud	Whether to draw the word clouds.
min_term	Minimal number of functional terms in a cluster. All the clusters with size less than <code>min_term</code> are all merged into one separated cluster in the heatmap.

order_by_size	Whether to reorder clusters by their sizes. The cluster that is merged from small clusters (size < min_term) is always put to the bottom of the heatmap.
stat	Type of value for mapping to the font size of keywords in the word clouds. There are two options: "count": simply number of keywords; "pvalue": enrichment on keywords is performed (by fisher's exact test) and $-\log_{10}(\text{pvalue})$ is used to map to font sizes.
min_stat	Minimal value for stat for selecting keywords.
exclude_words	Words that are excluded in the word cloud.
max_words	Maximal number of words visualized in the word cloud.
word_cloud_grob_param	A list of graphic parameters passed to <a href="#">word_cloud_grob</a> .
fontsize_range	The range of the font size. The value should be a numeric vector with length two. The font size interpolation is linear.
bg_gp	Graphics parameters for controlling word cloud annotation background.
column_title	Column title for the heatmap.
ht_list	A list of additional heatmaps added to the left of the similarity heatmap.
use_raster	Whether to write the heatmap as a raster image.
run_draw	Internally used.
...	Other arguments passed to <a href="#">draw, HeatmapList-method</a> .

**Value**

A [HeatmapList-class](#) object.

**Examples**

```
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
cl = binary_cut(mat)
ht_clusters(mat, cl, word_cloud_grob_param = list(max_width = 80))
ht_clusters(mat, cl, word_cloud_grob_param = list(max_width = 80),
  order_by_size = TRUE)

## End(Not run)
```

---

keyword\_enrichment\_from\_GO

*Keyword enrichment for GO terms*

---

**Description**

Keyword enrichment for GO terms

**Usage**

```
keyword_enrichment_from_GO(go_id, min_bg = 5, min_term = 2)
```

**Arguments**

go_id	A vector of GO IDs.
min_bg	Minimal number of GO terms (in the background, i.e. all GO terms in the GO database) that contain a specific keyword.
min_term	Minimal number of GO terms (GO terms in go_id) that contain a specific keyword.

**Details**

The enrichment is applied by Fisher's exact test. For a keyword, there is the following 2x2 contingency table:

	contains the keyword	does not contain the keyword
In the GO set	s11	s12
Not in the GO set	s21	s22

where s11, s12, s21 and s22 are number of GO terms in each category.

**Value**

A data frame with keyword enrichment results.

**Examples**

```
## Not run:
go_id = random_GO(100)
keyword_enrichment_from_GO(go_id)

## End(Not run)
```

---

partition\_by\_hclust    *Partition by hclust*

---

**Description**

Partition by hclust

**Usage**

```
partition_by_hclust(mat)
```

**Arguments**

mat	The similarity matrix.
-----	------------------------

**Details**

The "ward.D2" clustering method was used.

This function is used to set to the `partition_fun` argument in [binary\\_cut](#).

**Examples**

```
# There is no example  
NULL
```

---

partition_by_kmeans	<i>Partition by kmeans</i>
---------------------	----------------------------

---

**Description**

Partition by kmeans

**Usage**

```
partition_by_kmeans(mat, n_repeats = 10)
```

**Arguments**

<code>mat</code>	The similarity matrix.
<code>n_repeats</code>	Number of repeated runs of k-means.

**Details**

Since k-means clustering brings randomness, this function performs k-means clustering several times (controlled by `n_repeats`) and uses the final consensus partitioning.

This function is used to set to the `partition_fun` argument in [binary\\_cut](#).

**Examples**

```
# There is no example  
NULL
```

---

partition\_by\_kmeanspp *Partition by kmeans++*

---

**Description**

Partition by kmeans++

**Usage**

```
partition_by_kmeanspp(mat)
```

**Arguments**

mat                    The similarity matrix.

**Details**

This function is used to set to the partition\_fun argument in [binary\\_cut](#).

**Examples**

```
# There is no example  
NULL
```

---

partition\_by\_pam        *Partition by PAM*

---

**Description**

Partition by PAM

**Usage**

```
partition_by_pam(mat)
```

**Arguments**

mat                    The similarity matrix.

**Details**

The clustering is performed by [pam](#) with setting pamonce argument to 5.

This function is used to set to the partition\_fun argument in [binary\\_cut](#).

**Examples**

```
# There is no example
NULL
```

---

plot_binary_cut	<i>Visualize the process of binary cut</i>
-----------------	--

---

**Description**

Visualize the process of binary cut

**Usage**

```
plot_binary_cut(mat, value_fun = area_above_ecdf, cutoff = 0.85,
  partition_fun = partition_by_pam, dend = NULL, dend_width = unit(3, "cm"),
  depth = NULL, show_heatmap_legend = TRUE, ...)
```

**Arguments**

mat	The similarity matrix.
value_fun	A function that calculates the scores for the four submatrices on a node.
cutoff	The cutoff for splitting the dendrogram.
partition_fun	A function to split each node into two groups. Pre-defined functions in this package are <a href="#">partition_by_kmeanspp</a> , <a href="#">partition_by_pam</a> and <a href="#">partition_by_hclust</a> .
dend	A dendrogram object, used internally.
depth	Depth of the recursive binary cut process.
dend_width	Width of the dendrogram on the plot.
show_heatmap_legend	Whether to show the heatmap legend.
...	Other arguments.

**Details**

After the functions which perform clustering are executed, such as [simplifyGO](#) or [binary\\_cut](#), the dendrogram is temporarily saved and [plot\\_binary\\_cut](#) directly uses this dendrogram.

**Examples**

```
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
plot_binary_cut(mat, depth = 1)
plot_binary_cut(mat, depth = 2)
plot_binary_cut(mat)
```

---

random_DO	<i>Generate random Disease Ontology (DO) IDs</i>
-----------	--

---

**Description**

Generate random Disease Ontology (DO) IDs

**Usage**

```
random_DO(n)
```

**Arguments**

n	Number of DO IDs.
---	-------------------

**Details**

DO.db package should be installed.

**Value**

A vector of DO IDs.

**Examples**

```
random_DO(100)
```

---

random_GO	<i>Generate random GO IDs</i>
-----------	-------------------------------

---

**Description**

Generate random GO IDs

**Usage**

```
random_GO(n, ont = "BP", db = 'org.Hs.eg.db')
```

**Arguments**

n	Number of GO IDs.
ont	GO ontology. Value should be one of "BP", "CC" or "MF".
db	Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a>



**Value**

A vector of GO IDs.

**Examples**

```
random_GO(100)
```

---

```
register_clustering_methods  
  Register new clustering methods
```

---

**Description**

Register new clustering methods

**Usage**

```
register_clustering_methods(...)
```

**Arguments**

... A named list of clustering functions, see Details.

**Details**

The user-defined functions should accept at least one argument which is the input matrix. The second optional argument should always be ... so that parameters for the clustering function can be passed by control argument from [cluster\\_terms](#), [simplifyGO](#) or [simplifyEnrichment](#). If users forget to add ..., it is added internally.

Please note, the user-defined function should automatically identify the optimized number of clusters.

The function should return a vector of cluster labels. Internally it is converted to numeric labels.

**Value**

No value is returned.

**Examples**

```
register_clustering_methods(  
  # assume there are 5 groups  
  random = function(mat, ...) sample(5, nrow(mat), replace = TRUE)  
)  
all_clustering_methods()  
remove_clustering_methods("random")
```

---

remove\_clustering\_methods  
*Remove clustering methods*

---

**Description**

Remove clustering methods

**Usage**

```
remove_clustering_methods(method)
```

**Arguments**

method            A vector of method names.

**Value**

No value is returned.

**Examples**

```
# There is no example  
NULL
```

---

reset\_clustering\_methods  
*Reset to default clustering methods*

---

**Description**

Reset to default clustering methods

**Usage**

```
reset_clustering_methods()
```

**Details**

The default methods are:

kmeans see [cluster\\_by\\_kmeans](#).

pam see [cluster\\_by\\_pam](#).

dynamicTreeCut see [cluster\\_by\\_dynamicTreeCut](#).

mclust see [cluster\\_by\\_mclust](#).  
 apcluster see [cluster\\_by\\_apcluster](#).  
 hdbscan see [cluster\\_by\\_hdbscan](#).  
 fast\_greedy see [cluster\\_by\\_igraph](#).  
 louvain see [cluster\\_by\\_igraph](#).  
 walktrap see [cluster\\_by\\_igraph](#).  
 MCL see [cluster\\_by\\_MCL](#).  
 binary\_cut see [binary\\_cut](#).

### Value

No value is returned.

### Examples

```

all_clustering_methods()
remove_clustering_methods(c("kmeans", "mclust"))
all_clustering_methods()
reset_clustering_methods()
all_clustering_methods()

```

---

scale_fontsize	<i>Scale font size</i>
----------------	------------------------

---

### Description

Scale font size

### Usage

```
scale_fontsize(x, rg = c(1, 30), fs = c(4, 16))
```

### Arguments

x	A numeric vector.
rg	The range.
fs	Range of the font size.

### Value

A numeric vector.

### Details

It is a linear interpolation.

## Examples

```
x = runif(10, min = 1, max = 20)
# scale x to fontsize 4 to 16.
scale_fontsize(x)
```

---

select_cutoff	<i>Select the cutoff for binary cut</i>
---------------	---

---

## Description

Select the cutoff for binary cut

## Usage

```
select_cutoff(mat, cutoff = seq(0.6, 0.98, by = 0.01), verbose = TRUE, ...)
```

## Arguments

mat	A similarity matrix.
cutoff	A list of cutoffs to test. Note the range of the cutoff values should be inside [0.5, 1].
verbose	Whether to print messages.
...	Pass to <a href="#">binary_cut</a> .

## Details

Binary cut is applied to each of the cutoff and the clustering results are evaluated by following metrics:

- difference score, calculated by [difference\\_score](#).
- number of clusters.
- block mean, which is the mean similarity in the blocks in the diagonal of the heatmap.

## Examples

```
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
  package = "simplifyEnrichment"))
select_cutoff(mat)
```

---

se_opt	<i>Global parameters</i>
--------	--------------------------

---

**Description**

Global parameters

**Usage**

```
se_opt(..., RESET = FALSE, READ.ONLY = NULL, LOCAL = FALSE, ADD = FALSE)
```

**Arguments**

...	Arguments for the parameters, see "details" section.
RESET	Whether to reset to default values.
READ.ONLY	Please ignore.
LOCAL	Please ignore.
ADD	Please ignore.

**Details**

There are the following global options:

verbose Whether to print messages.

**Examples**

```
# There is no example
NULL
```

---

simplifyEnrichment	<i>Simplify functional enrichment results</i>
--------------------	---

---

**Description**

Simplify functional enrichment results

**Usage**

```
simplifyEnrichment(mat, method = "binary_cut", control = list(),
  plot = TRUE, verbose = TRUE,
  column_title = qq("{nrow(mat)} terms clustered by '{method}'"),
  ht_list = NULL, ...)
```

**Arguments**

mat	A similarity matrix.
method	Method for clustering the matrix. See <a href="#">cluster_terms</a> .
control	A list of parameters for controlling the clustering method, passed to <a href="#">cluster_terms</a> .
plot	Whether to make the heatmap.
column_title	Column title for the heatmap.
verbose	Whether to print messages.
ht_list	A list of additional heatmaps added to the left of the similarity heatmap.
...	Arguments passed to <a href="#">ht_clusters</a> .

**Details**

The usage is the same as [simplifyGO](#).

**Examples**

```
# There is no example
NULL
```

---

simplifyGO

*Simplify Gene Ontology (GO) enrichment results*


---

**Description**

Simplify Gene Ontology (GO) enrichment results

**Usage**

```
simplifyGO(mat, method = "binary_cut", control = list(),
  plot = TRUE, verbose = TRUE,
  column_title = qq("@{nrow(mat)} GO terms clustered by '{method}'"),
  ht_list = NULL, ...)
```

**Arguments**

mat	A GO similarity matrix.
method	Method for clustering the matrix. See <a href="#">cluster_terms</a> .
control	A list of parameters for controlling the clustering method, passed to <a href="#">cluster_terms</a> .
plot	Whether to make the heatmap.
column_title	Column title for the heatmap.
verbose	Whether to print messages.
ht_list	A list of additional heatmaps added to the left of the similarity heatmap.
...	Arguments passed to <a href="#">ht_clusters</a> .

## Details

This is basically a wrapper function that it first runs `cluster_terms` to cluster GO terms and then runs `ht_clusters` to visualize the clustering.

The arguments in `simplifyGO` passed to `ht_clusters` are:

`draw_word_cloud` Whether to draw the word clouds.

`min_term` Minimal number of GO terms in a cluster. All the clusters with size less than `min_term` are all merged into one single cluster in the heatmap.

`order_by_size` Whether to reorder GO clusters by their sizes. The cluster that is merged from small clusters (`size < min_term`) is always put to the bottom of the heatmap.

`stat` What values of keywords are used to map to font sizes in the word clouds.

`exclude_words` Words that are excluded in the word cloud.

`max_words` Maximal number of words visualized in the word cloud.

`word_cloud_grob_param` A list of graphic parameters passed to `word_cloud_grob`.

`fontsize_range` The range of the font size. The value should be a numeric vector with length two. The minimal font size is mapped to word frequency value of 1 and the maximal font size is mapped to the maximal word frequency. The font size interpolation is linear.

`bg_gp` Graphic parameters for controlling the background of word cloud annotations.

## Value

A data frame with two columns: GO IDs and cluster labels.

## See Also

[simplifyGOFromMultipleLists](#) which performs `simplifyGO` analysis with multiple lists of GO IDs.

## Examples

```
set.seed(123)
go_id = random_GO(500)
mat = GO_similarity(go_id)
df = simplifyGO(mat, word_cloud_grob_param = list(max_width = 80))
head(df)
```

---

simplifyGOFromMultipleLists

*Perform simplifyGO analysis with multiple lists of GO IDs*


---

## Description

Perform simplifyGO analysis with multiple lists of GO IDs

## Usage

```
simplifyGOFromMultipleLists(lt, go_id_column = NULL,
  padj_column = NULL, padj_cutoff = 1e-2,
  filter = function(x) any(x < padj_cutoff), default = 1,
  ont = NULL, db = 'org.Hs.eg.db', measure = "Rel",
  heatmap_param = list(NULL), show_barplot = TRUE,
  method = "binary_cut", control = list(),
  min_term = NULL, verbose = TRUE, column_title = NULL, ...)
```

## Arguments

lt	A data frame, a list of numeric vectors (e.g. adjusted p-values) where each numeric vector has GO IDs as names, or a list of GO IDs.
go_id_column	Column index of GO ID if lt contains a list of data frames.
padj_column	Column index of adjusted p-values if lt contains a list of data frames.
padj_cutoff	Cut off for adjusted p-values
filter	A self-defined function for filtering GO IDs. By default it requires GO IDs should be significant in at least one list.
default	The default value for the adjusted p-values. See Details.
ont	GO ontology. Value should be one of "BP", "CC" or "MF". If it is not specified, the function automatically identifies it by random sampling 10 IDs from go_id (see <a href="#">guess_ont</a> ).
db	Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a>
measure	Semantic measure for the GO similarity, pass to <a href="#">termSim</a> .
heatmap_param	Parameters for controlling the heatmap, see Details.
show_barplot	Whether draw barplots which shows numbers of significant GO terms in clusters.
method	Pass to <a href="#">simplifyGO</a> .
control	Pass to <a href="#">simplifyGO</a> .
min_term	Pass to <a href="#">simplifyGO</a> .
verbose	Pass to <a href="#">simplifyGO</a> .
column_title	Pass to <a href="#">simplifyGO</a> .
...	Pass to <a href="#">simplifyGO</a> .



## Details

The input data can have three types of formats:

- A list of numeric vectors of adjusted p-values where each vector has the GO IDs as names.
- A data frame. The column of the GO IDs can be specified with `go_id_column` argument and the column of the adjusted p-values can be specified with `padj_column` argument. If these columns are not specified, they are automatically identified. The GO ID column is found by checking whether a column contains all GO IDs. The adjusted p-value column is found by comparing the column names of the data frame to see whether it might be a column for adjusted p-values. These two columns are used to construct a numeric vector with GO IDs as names.
- A list of character vectors of GO IDs. In this case, each character vector is changed to a numeric vector where all values take 1 and the original GO IDs are used as names of the vector.

Now let's assume there are  $n$  GO lists, we first construct a global matrix where columns correspond to the  $n$  GO lists and rows correspond to the "union" of all GO IDs in the lists. The value for the  $i$ th GO ID and in the  $j$ th list are taken from the corresponding numeric vector in `lt`. If the  $j$ th vector in `lt` does not contain the  $i$ th GO ID, the value defined by `default` argument is taken there (e.g. in most cases the numeric values are adjusted p-values, `default` is set to 1). Let's call this matrix as  $M_0$ .

Next step is to filter  $M_0$  so that we only take a subset of GO IDs of interest. We define a proper function via argument `filter` to remove GO IDs that are not important for the analysis. Functions for `filter` is applied to every row in  $M_0$  and `filter` function needs to return a logical value to decide whether to remove the current GO ID. For example, if the values in `lt` are adjusted p-values, the `filter` function can be set as `function(x) any(x < padj_cutoff)` so that the GO ID is kept as long as it is significant in at least one list. After the filter, let's call the filtered matrix  $M_1$ .

GO IDs in  $M_1$  (row names of  $M_1$ ) are used for clustering. A heatmap of  $M_1$  is attached to the left of the GO similarity heatmap so that the group-specific (or list-specific) patterns can be easily observed and to corresponded to GO functions.

Argument `heatmap_param` controls several parameters for heatmap  $M_1$ :

- `transform`: A self-defined function to transform the data for heatmap visualization. The most typical case is to transform adjusted p-values by  $-\log_{10}(x)$ .
- `breaks`: break values for color interpolation.
- `col`: The corresponding values for breaks.
- `labels`: The corresponding labels.
- `name`: Legend title.

## Examples

```
# perform functional enrichment on the signatures genes from cola analysis
require(cola)
data(golub_cola)
res = golub_cola["ATC:skmeans"]
require(hu6800.db)
```

```
x = hu6800ENTREZID
mapped_probes = mappedkeys(x)
id_mapping = unlist(as.list(x[mapped_probes]))
lt = functional_enrichment(res, k = 3, id_mapping = id_mapping) # you can check the value of `lt`

# a list of data frames
simplifyGOFromMultipleLists(lt, padj_cutoff = 0.001)

# a list of numeric values
lt2 = lapply(lt, function(x) structure(x$p.adjust, names = x$ID))
simplifyGOFromMultipleLists(lt2, padj_cutoff = 0.001)

# a list of GO IDS
lt3 = lapply(lt, function(x) x$ID[x$p.adjust < 0.001])
simplifyGOFromMultipleLists(lt3)
```

---

subset\_enrichResult    *Subset method of the enrichResult class*

---

## Description

Subset method of the enrichResult class

## Usage

```
subset_enrichResult(x, i)
```

## Arguments

x	A enrichResult object from 'clusterProfiler' or other related packages.
i	Row indices.

## Value

Still a enrichResult object but with the selected subset of rows.

## Examples

```
# There is no example
NULL
```

---

summarizeGO

*A simplified way to visualize enrichment in GO clusters*


---

## Description

A simplified way to visualize enrichment in GO clusters

## Usage

```
summarizeGO(go_id, value = NULL, aggregate = mean,
  method = "binary_cut", control = list(), verbose = TRUE,
  axis_label = "Value", title = "", legend_title = axis_label,

  min_term = round(nrow(mat)*0.01),
  stat = "pvalue",
  min_stat = ifelse(stat == "count", 5, 0.05),
  exclude_words = character(0),
  max_words = 6,
  word_cloud_grob_param = list(),
  fontsize_range = c(4, 16),
  bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA")
)
```

## Arguments

go_id	A vector of GO IDs.
value	A list of numeric value associate with go_id. We suggest to use $-\log_{10}(\text{p.adjust})$ or $-\log_2(\text{fold enrichment})$ as the values.
aggregate	Function to aggregate values in each GO cluster.
method	Method for clustering the matrix. See <a href="#">cluster_terms</a> .
control	A list of parameters for controlling the clustering method, passed to <a href="#">cluster_terms</a> .
verbose	Whether to print messages.
axis_label	X-axis label.
title	Title for the whole plot.
legend_title	Title for the legend.
min_term	Minimal number of functional terms in a cluster. All the clusters with size less than min_term are all merged into one separated cluster in the heatmap.
stat	Type of value for mapping to the font size of keywords in the word clouds. There are two options: "count": simply number of keywords; "pvalue": enrichment on keywords is performed (by fisher's exact test) and $-\log_{10}(\text{pvalue})$ is used to map to font sizes.
min_stat	Minimal value for stat for selecting keywords.
exclude_words	Words that are excluded in the word cloud.

max_words	Maximal number of words visualized in the word cloud.
word_cloud_grob_param	A list of graphic parameters passed to <a href="#">word_cloud_grob</a> .
fontsize_range	The range of the font size. The value should be a numeric vector with length two. The font size interpolation is linear.
bg_gp	Graphics parameters for controlling word cloud annotation background.

### Details

There are several other ways to specify GO IDs and the associated values.

1. specify value as a named vector where GO IDs are the names.
2. specify value as a list of numeric named vectors. In this case, value contains multiple enrichment results.

### Examples

```
# There is no example
NULL
```

---

term_similarity	<i>Similarity between terms based on the overlap of genes</i>
-----------------	---

---

### Description

Similarity between terms based on the overlap of genes

### Usage

```
term_similarity(gl, method = c("kappa", "jaccard", "dice", "overlap"), all = NULL, remove_negative = TR
```

### Arguments

gl	A list of genes that are in the terms.
method	The similarity measurement.
all	The universe set.
remove_negative	If the value is negative, reset to zero

### Details

The definition of the four similarity measurements can be found at [https://jokergoo.github.io/simplifyEnrichment\\_supplementary/supplS01\\_coefficient\\_definition/supplS01\\_coefficient\\_definition.html](https://jokergoo.github.io/simplifyEnrichment_supplementary/supplS01_coefficient_definition/supplS01_coefficient_definition.html).

### Value

A symmetric matrix.

**Examples**

```
# There is no example
NULL
```

---

```
term_similarity_from_enrichResult
Similarity between terms in the enrichResult class
```

---

**Description**

Similarity between terms in the enrichResult class

**Usage**

```
term_similarity_from_enrichResult(x, ...)
```

**Arguments**

x	A enrichResult object from 'clusterProfiler' or other related packages.
...	Pass to <a href="#">term_similarity</a> .

**Details**

The object is normally from the 'clusterProfiler', 'DOSE', 'meshes' or 'ReactomePA' package.

**Value**

A symmetric matrix.

**Examples**

```
# There is no example
NULL
```

---

`term_similarity_from_gmt`*Similarity between terms from a gmt file*

---

**Description**

Similarity between terms from a gmt file

**Usage**

```
term_similarity_from_gmt(term_id, gmt, extract_term_id = NULL, ...)
```

**Arguments**

<code>term_id</code>	A vector of terms.
<code>gmt</code>	The path of the gmt file.
<code>extract_term_id</code>	If the term ID in the first column only as a substring, setting a function to extract this substring.
<code>...</code>	Pass to <code>term_similarity</code> .

**Value**

A symmetric matrix.

**Examples**

```
# There is no example  
NULL
```

---

`term_similarity_from_KEGG`*Similarity between KEGG terms*

---

**Description**

Similarity between KEGG terms

**Usage**

```
term_similarity_from_KEGG(term_id, ...)
```

**Arguments**

term\_id            A vector of KEGG IDs, e.g., hsa001.  
...                Pass to [term\\_similarity](#).

**Value**

A symmetric matrix.

**Examples**

```
# There is no example  
NULL
```

---

term\_similarity\_from\_MSigDB  
*Similarity between MSigDB terms*

---

**Description**

Similarity between MSigDB terms

**Usage**

```
term_similarity_from_MSigDB(term_id, category = NULL, subcategory = NULL, ...)
```

**Arguments**

term\_id            A vector of MSigDB gene set names.  
category           E.g., 'C1', 'C2', pass to [msigdb](#).  
subcategory        E.g., 'CGP', 'BP', pass to [msigdb](#).  
...                Pass to [term\\_similarity](#).

**Value**

A symmetric matrix.

**Examples**

```
# There is no example  
NULL
```

term\_similarity\_from\_Reactome

*Similarity between Reactome terms*

---

### Description

Similarity between Reactome terms

### Usage

```
term_similarity_from_Reactome(term_id, ...)
```

### Arguments

term\_id            A vector of Reactome IDs.  
...                Pass to [term\\_similarity](#).

### Value

A symmetric matrix.

### Examples

```
# There is no example  
NULL
```

---

widthDetails.word\_cloud

*Width for word\_cloud grob*

---

### Description

Width for word\_cloud grob

### Usage

```
## S3 method for class 'word_cloud'  
widthDetails(x)
```

### Arguments

x                    The word\_cloud grob returned by [word\\_cloud\\_grob](#).

### Value

A [unit](#) object.



**Examples**

```
# There is no example
NULL
```

---

word_cloud_grob	<i>A simple grob for the word cloud</i>
-----------------	---

---

**Description**

A simple grob for the word cloud

**Usage**

```
word_cloud_grob(text, fontsize,
  line_space = unit(4, "pt"), word_space = unit(4, "pt"), max_width = unit(80, "mm"),
  col = function(fs) circlize::rand_color(length(fs), luminosity = "dark"),
  add_new_line = FALSE, test = FALSE)
```

**Arguments**

text	A vector of words.
fontsize	The corresponding font size. With the frequency of the words known, <a href="#">scale_fontsize</a> can be used to linearly interpolate frequencies to font sizes.
line_space	Space between lines. The value can be a <a href="#">unit</a> object or a numeric scalar which is measured in mm.
word_space	Space between words. The value can be a <a href="#">unit</a> object or a numeric scalar which is measured in mm.
max_width	The maximal width of the viewport to put the word cloud. The value can be a <a href="#">unit</a> object or a numeric scalar which is measured in mm. Note this might be larger than the final width of the returned grob object.
col	Colors for the words. The value can be a vector, in numeric or character, which should have the same length as text. Or it is a self-defined function that takes the font size vector as the only argument. The function should return a color vector. See Examples.
add_new_line	Whether to add new line after every word? If TRUE, each word will be in a separated line.
test	Internally used. It basically adds borders to the words and the viewport.

**Value**

A [grob](#) object. The width and height of the grob can be get by [grobWidth](#) and [grobHeight](#).

**Examples**

```
# very old R versions do not have strrep() function
if(!exists("strrep")) {
  strrep = function(x, i) paste(rep(x, i), collapse = "")
}
words = sapply(1:30, function(x) strrep(sample(letters, 1), sample(3:10, 1)))
require(grid)
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100)
grid.newpage(); grid.draw(gb)

# color as a single scalar
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = 1)
grid.newpage(); grid.draw(gb)

# color as a vector
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = 1:30)
grid.newpage(); grid.draw(gb)

# color as a function
require(circlize)
col_fun = colorRamp2(c(5, 17, 30), c("blue", "black", "red"))
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = function(fs) col_fun(fs))
grid.newpage(); grid.draw(gb)
```

# Index

`all_clustering_methods`, 3, 14, 15, 17  
`anno_empty`, 5  
`anno_link`, 4, 5  
`anno_word_cloud`, 4, 6  
`anno_word_cloud_from_GO`, 5  
`apcluster`, 8  
`area_above_ecdf`, 6

`binary_cut`, 3, 7, 14, 15, 18, 23, 26, 29–31, 35, 36

`cluster_by_apcluster`, 3, 8, 14, 15, 17, 35  
`cluster_by_dynamicTreeCut`, 3, 8, 14, 15, 17, 34  
`cluster_by_hdbscan`, 3, 9, 14, 15, 17, 35  
`cluster_by_igraph`, 3, 10, 14, 15, 17, 18, 35  
`cluster_by_kmeans`, 3, 11, 14, 15, 17, 34  
`cluster_by_MCL`, 3, 11, 14, 15, 18, 35  
`cluster_by_mclust`, 3, 12, 14, 15, 17, 35  
`cluster_by_pam`, 13, 14, 15, 17, 34  
`cluster_fast_greedy`, 10  
`cluster_leading_eigen`, 10  
`cluster_louvain`, 10  
`cluster_terms`, 13, 14, 23, 26, 33, 38, 39, 43  
`cluster_walktrap`, 10  
`cmp_make_clusters`, 15, 16, 18  
`cmp_make_plot`, 16, 17, 18  
`colorRamp2`, 26  
`compare_clustering_methods`, 17  
`count_words`, 5, 18  
`cutreeDynamic`, 9

`dend_node_apply`, 19  
`dendrapply`, 22  
`difference_score`, 16, 20, 36  
`DO_similarity`, 21  
`doSim`, 21

`edit_node`, 22  
`export_to_shiny_app`, 23

`GO_similarity`, 23  
`grob`, 49  
`grobHeight`, 49  
`grobWidth`, 49  
`guess_ont`, 24, 24, 40

`hdbscan`, 9  
`heightDetails.word_cloud`, 25  
`ht_clusters`, 26, 38, 39

`keyword_enrichment_from_GO`, 27  
`kmeans`, 11

`mcl`, 12  
`Mclust`, 12  
`msigdb`, 47

`pam`, 30  
`pamk`, 13  
`partition_by_hclust`, 7, 28, 31  
`partition_by_kmeans`, 29  
`partition_by_kmeanspp`, 7, 30, 31  
`partition_by_pam`, 7, 30, 31  
`plot_binary_cut`, 31, 31

`random_DO`, 32  
`random_GO`, 32  
`register_clustering_methods`, 4, 14, 33  
`remove_clustering_methods`, 34  
`reset_clustering_methods`, 34

`scale_fontsize`, 35, 49  
`se_opt`, 37  
`select_cutoff`, 36  
`simplifyEnrichment`, 33, 37  
`simplifyGO`, 31, 33, 38, 38, 39, 40  
`simplifyGOFromMultipleLists`, 39, 40  
`subset_enrichResult`, 42  
`summarizeGO`, 43

`term_similarity`, 44, 45–48

`term_similarity_from_enrichResult`, 45  
`term_similarity_from_gmt`, 46  
`term_similarity_from_KEGG`, 46  
`term_similarity_from_MSigDB`, 47  
`term_similarity_from_Reactome`, 48  
`termSim`, 24, 40

`unit`, 25, 48, 49

`widthDetails.word_cloud`, 48  
`word_cloud_grob`, 4, 25, 27, 39, 44, 48, 49