

# Package ‘dnet’

February 22, 2014

**Type** Package

**Title** Expression-active dynamic networks for stratifying and visualising samples

**Version** 0.99.0

**Date** 2014-2-20

**Author** Hai Fang and Julian Gough

**Maintainer** Hai Fang <hfang@cs.bris.ac.uk>

**Depends** R (>= 3.0.1), supraHex, igraph

## Imports

**Description** The 'dnet' package is initiated to fill in the need of an open-source tool for analysing biological networks and high-throughput biological data in an integrative manner. More specifically, dnet intends to analyse the biological network whose nodes/genes are associated with numeric properties such as expression patterns across a series of samples in subject. Also, dnet aims to deliver an eye-intuitive tool for network-based sample stratifications.

**URL** <http://dnet.r-forge.r-project.org>

**Collate** 'eCal.r' 'eView.r' 'eWrite.r' 'visRunES.r' 'ePvalAggregate.r' 'eGraphInduce.r' 'eBUMfit.r' 'eBUMscore.r' 'eGraphFind.r' 'eGraphPipeline.r' 'eGraphConfidence.r' 'visGraph.r' 'visGraphMul.r' 'visGraphAnimate.r' 'visGraphReorder.r' 'eGraphReorder.r' 'visGraphArc.r' 'visGraphCircle.r' 'eRWR.r'

**License** GPL-2

**biocViews** Bioinformatics

## R topics documented:

eBUMfit . . . . .	2
eBUMscore . . . . .	4
eCal . . . . .	5
eGraphConfidence . . . . .	8
eGraphFind . . . . .	9
eGraphInduce . . . . .	10
eGraphPipeline . . . . .	12
eGraphReorder . . . . .	13

ePvalAggregate . . . . .	15
eRWR . . . . .	16
eView . . . . .	17
eWrite . . . . .	18
visGraph . . . . .	19
visGraphAnimate . . . . .	21
visGraphArc . . . . .	23
visGraphCircle . . . . .	25
visGraphMul . . . . .	27
visGraphReorder . . . . .	29
visRunES . . . . .	31

<b>Index</b>	<b>33</b>
--------------	-----------

---

eBUMfit	<i>Function to fit a p-value distribution under beta-uniform mixture model</i>
---------	--

---

## Description

eBUMfit is supposed to take as input a vector of p-values for deriving their distribution under beta-uniform mixture model (see Note below). The density distribution of input p-values is expressed as a mixture of two components: one for the null hypothesis (the noise component) and the other for the alternative hypothesis (the signal component). The noise component is the uniform density, while the signal component is the remainder of the mixture distribution. It returns an object of class "BUM".

## Usage

```
eBUMfit(x, ntry = 1, hist.bum = T, contour.bum = T,
        verbose = T)
```

## Arguments

x	a vector containing input p-values
ntry	an integer specifying how many tries are used to find the optimised parameters by maximum likelihood estimation
hist.bum	logical to indicate whether the histogram graph should be drawn
contour.bum	logical to indicate whether a contour plot should be drawn to show the log likelihood as a function of two parameters (a and lambda) in the beta-uniform mixture model
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

## Value

an object of class "BUM", a list with following elements:

- lambda: estimated mixture parameter
- a: estimated shape parameter
- NLL: Negative log-likelihood

- pvalues: the input pvalues
- call: the call that produced this result

### Note

The probability density function of p-values under the Beta-Uniform Mixture model is formulated as:  $f(x|\lambda, a) = \lambda + (1 - \lambda) * a * x^{a-1}$ . The model names after mixing two distributions:

- the uniform distribution with the density function as  $\frac{1}{b-a} \Big|_{a=0}^{b=1} = 1$
- the beta distribution with the density function as  $\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} * x^{a-1} * (1-x)^{b-1} \Big|_{b=1} = a * x^{a-1}$

Both are mixed via  $\lambda$ . The mixture parameter  $\lambda$  measures the contribution from the uniform distribution. Accordingly,  $1 - \lambda$  measures the contribution from the beta distribution. Notably, the probability density function of the beta distribution can be splitted into two parts (rather than the exclusive signal):

- the constant part as noise:  $a * x^{a-1} \Big|_{x=1} = a$
- the rest part as signal:  $a * (x^{a-1} - 1)$

In other words, there is no signal at  $x = 1$  but all being noise. It is a conservative, upper bound estimation of the noise. Therefore, the probability density function in the model can be decomposed into signal-noise components:

- the signal component:  $(1 - \lambda) * a * (x^{a-1} - 1)$
- the noise component:  $\lambda + (1 - \lambda) * a$

It is misleading to simply view  $\lambda$  as the noise component and  $(1 - \lambda) * a * x^{a-1}$  as the signal component, just as wrongly do in the literatures (e.g. <http://www.ncbi.nlm.nih.gov/pubmed/18586718>)

### See Also

[eBUMscore](#)

### Examples

```
# 1) generate an vector consisting of random values from beta distribution
x <- rbeta(1000, shape1=0.5, shape2=1)

# 2) fit a p-value distribution under beta-uniform mixture model
fit <- eBUMfit(x)
fit$lambda
fit$a
```

eBUMscore

*Function to transform p-values into scores according to the fitted beta-uniform mixture model and/or after controlling false discovery rate*

### Description

eBUMscore is supposed to take as input a vector of p-values, which are transformed into scores according to the fitted beta-uniform mixture model. Also if the FDR threshold is given, it is used to make sure that p-values below this are considered significant and thus scored positively. Instead, those p-values above the given FDR are considered insignificant and thus scored negatively.

### Usage

```
eBUMscore(fit, method = c("pdf", "cdf"), fdr = NULL,
  scatter.bum = T)
```

### Arguments

fit	an object of class "BUM"
method	the method used for the transformation. It can be either "pdf" for the method based on the probability density function of the fitted model, or "cdf" for the method based on the cumulative distribution function of the fitted model
fdr	the given FDR threshold. By default, it is set to NULL, meaning there is no constraint. If given, those p-values with the FDR below this are considered significant and thus scored positively. Instead, those p-values with the FDR above this given FDR are considered insignificant and thus scored negatively
scatter.bum	logical to indicate whether the scatter graph of scores against p-values should be drawn. Also indicated is the p-value (called tau) corresponding to the given FDR threshold (if any)

### Value

- scores: a vector of scores

### Note

The transformation from the input p-value  $x$  to the score  $S(x)$  is based on the fitted beta-uniform mixture model with two parameters  $\lambda$  and  $a$ :  $f(x|\lambda, a) = \lambda + (1 - \lambda) * a * x^{a-1}$ . Specifically, it considers the log-likelihood ratio between the signal and noise component of the model. The probability density function (pdf) of the signal component and the noise component are  $(1 - \lambda) * a * (x^{a-1} - 1)$  and  $\lambda + (1 - \lambda) * a$ , respectively. Accordingly, the cumulative distribution function (cdf) of the signal component and the noise component are  $\int_0^x (1 - \lambda) * a * (x^{a-1} - 1) dx$  and  $\int_0^x \lambda + (1 - \lambda) * a dx$ . In order to take into account the significance of the p-value, the  $fdr$  threshold is also used for down-weighting the score. According to how to measure both components, there are two methods implemented for deriving the score  $S(x)$ :

- The method "pdf":  $S(x) = \log_2 \frac{(1-\lambda)*a*(x^{a-1}-1)}{\lambda+(1-\lambda)*a} - \log_2 \frac{(1-\lambda)*a*(\tau^{a-1}-1)}{\lambda+(1-\lambda)*a} = \log_2 \left( \frac{x^{a-1}-1}{\tau^{a-1}-1} \right)$ .  
For the purpose of down-weighting scores, it must ensure  $\log_2 \frac{(1-\lambda)*a*(\tau^{a-1}-1)}{\lambda+(1-\lambda)*a} \geq 0$ , that is, the constraint via  $\tau \leq \left( \frac{\lambda+2*a*(1-\lambda)}{a*(1-\lambda)} \right)^{\frac{1}{a-1}}$

- The method "cdf":  $S(x) = \log_2 \frac{\int_0^x (1-\lambda)*a*(x^{a-1}-1) dx}{\int_0^x \lambda+(1-\lambda)*a dx} - \log_2 \frac{\int_0^\tau (1-\lambda)*a*(\tau^{a-1}-1) dx}{\int_0^\tau \lambda+(1-\lambda)*a dx} = \log_2 \frac{(1-\lambda)*(x^{a-1}-a)}{\lambda+(1-\lambda)*a} - \log_2 \frac{(1-\lambda)*(\tau^{a-1}-a)}{\lambda+(1-\lambda)*a} = \log_2 \left( \frac{x^{a-1}-a}{\tau^{a-1}-a} \right)$ . For the purpose of down-weighting scores, it must ensure  $\log_2 \frac{(1-\lambda)*(x^{a-1}-a)}{\lambda+(1-\lambda)*a} \geq 0$ , that is, the constraint via  $\tau \leq \left( \frac{\lambda+2*a*(1-\lambda)}{1-\lambda} \right)^{\frac{1}{a-1}}$
- Where  $\tau = \left[ \frac{\lambda+(1-\lambda)*a-fdr*\lambda}{fdr*(1-\lambda)} \right]^{\frac{1}{a-1}}$ , i.e. the p-value corresponding to the exact  $fdr$  threshold.  
It can be deduced from the definition of the false discovery rate:  $fdr \doteq \frac{\int_0^\tau \lambda+(1-\lambda)*a dx}{\int_0^\tau \lambda+(1-\lambda)*a*x^{a-1} dx}$ .  
Notably, if the calculated  $\tau$  exceeds the constraint, it will be reset to the maximum end of that constraint

## See Also

[eBUMfit](#)

## Examples

```
# 1) generate an vector consisting of random values from beta distribution
x <- rbeta(1000, shape1=0.5, shape2=1)

# 2) fit a p-value distribution under beta-uniform mixture model
fit <- eBUMfit(x)

# 3) calculate the scores according to the fitted BUM and fdr=0.01
# using "pdf" method
scores <- eBUMscore(fit, method="pdf", fdr=0.01)
# using "cdf" method
scores <- eBUMscore(fit, method="cdf", fdr=0.01)
```

---

eCal

*Function to conduct gene set enrichment analysis given the input data and the ontology in query*

---

## Description

eCal is supposed to conduct gene set enrichment analysis given the input data and the ontology in query. It returns an object of class "eTerm".

## Usage

```
eCal(data, identity = c("symbol", "entrez"),
      genome = c("mm", "hs"),
      ontology = c("GOBP", "GOMF", "GOCC", "MP", "DO", "PS"),
      sizeRange = c(10, 1000), which_distance = NULL,
      weight = 1, nperm = 1000, fast = T,
      sigTail = c("two-tails", "one-tail"), verbose = T)
```

## Arguments

data	a data frame or matrix of input data. It must have row names, either Entrez Gene ID or Symbol
identity	the type of gene identity (i.e. row names of input data), either "symbol" for gene symbols (by default) or "entrez" for Entrez Gene ID. The option "symbol" is preferred as it is relatively stable from one update to another; when gene symbols cannot be matched, synonyms will be searched against
genome	the genome identity. It can be either "mm" for mouse genome or "hs" for human genome
ontology	the ontology supported currently. For mouse genome, it can be "GOBP" for Gene Ontology Biological Process, "GOMF" for Gene Ontology Molecular Function, "GOCC" for Gene Ontology Cellular Component, or "MP" for Mammalian Phenotype. For human genome, it can be "GOBP" for Gene Ontology Biological Process, "GOMF" for Gene Ontology Molecular Function, "GOCC" for Gene Ontology Cellular Component, "HP" for Human Phenotype, or "DO" for Disease Ontology.
sizeRange	the minimum and maximum size of members of each gene set in consideration. By default, it sets to a minimum of 10 but no more than 1000
which_distance	which distance of terms in the ontology is used to restrict terms in consideration. By default, it sets to 'NULL' to consider all distances
weight	type of score weighth. It can be "0" for unweighted (an equivalent to Kolmogorov-Smirnov, only considering the rank), "1" for weighted by input gene score (by default), and "2" for over-weighted, and so on
nperm	the number of random permutations. For each permutation, gene-score associations will be permuted so that permutation of gene-term associations is realised
fast	logical to indicate whether to fast calculate expected results from permuted data. By default, it sets to true
sigTail	the tail used to calculate the statistical significance. It can be either "two-tails" for the significance based on two-tails or "one-tail" for the significance based on one tail
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display

## Value

an object of class "eTerm", a list with following components:

- **set\_info**: a matrix of nSet X 4 containing gene set information, where nSet is the number of gene set in consideration, and the 4 columns are "setID" (i.e. "Term ID"), "name" (i.e. "Term Name"), "namespace" and "distance"
- **gs**: a list of gene sets, each storing gene members. Always, gene sets are identified by "setID" and gene members identified by "Entrez ID"
- **data**: a matrix of nGene X nSample containing input data in consideration. It is not always the same as the input data as only those mappable are retained
- **es**: a matrix of nSet X nSample containing enrichment score, where nSample is the number of samples (i.e. the number of columns in input data)
- **nes**: a matrix of nSet X nSample containing normalised enrichment score. It is the version of enrichment score but after being normalised by gene set size

- pvalue: a matrix of nSet X nSample containing nominal p value
- adjp: a matrix of nSet X nSample containing adjusted p value. It is the p value but after being adjusted for multiple comparisons
- gadjp: a matrix of nSet X nSample containing globally adjusted p value in terms of all samples
- fdr: a matrix of nSet X nSample containing false discovery rate (FDR). It is the estimated probability that the normalised enrichment score represents a false positive finding
- qvalue: a matrix of nSet X nSample containing q value. It is the monotonically increasing FDR
- call: the call that produced this result

### Note

The interpretation of returned components:

- "es": enrichment score for the gene set is the degree to which this gene set is overrepresented at the top or bottom of the ranked list of genes in each column of input data;
- "nes": normalised enrichment score for the gene set is enrichment score that has already normalised by gene set size. It is comparable across analysed gene sets;
- "pvalue": nominal p value is the statistical significance of the enrichment score. It is not adjusted for multiple hypothesis testing, and thus is of limited use in comparing gene sets;
- "adjp": adjusted p value by Benjamini & Hochberg method. It is comparable across gene sets;
- "gadjp": globally adjusted p value by Benjamini & Hochberg method. Unlike "adjp", it is adjusted in terms of all samples;
- "fdr": false discovery rate is the estimated probability that the normalised enrichment score represents a false positive finding. Unlike "adjp" or "gadjp" (also aliased as "fdr") that is derived from a list of p values, this version of fdr is directly calculate from the statistic (i.e. normalised enrichment score);
- "qvalue": q value is the monotonically increasing FDR so that the higher "nes", the lower "qvalue".

### See Also

[eView](#), [eWrite](#)

### Examples

```
#load("~/Databases/ChipSeq/supraHex/Bioinformatics_AN/RTiming/GSE18019/TableS1T.RData")
#data <- RT_LR[,1:2]
#eTerm <- eCal(data=data, identity="symbol", genome="mm", ontology="MP", sizeRange=c(10,1000), which_distar
```

---

eGraphConfidence	<i>Function to append the confidence information from the source graphs into the target graph</i>
------------------	---

---

## Description

eConsensusGraph is supposed to append the confidence information (extracted from a list of the source graphs) into the target graph. The confidence information is about how often a node (or an edge) in the target graph that can be found in the input source graphs. The target graph is an object of class "igraph" or "graphNEL", and the source graphs are a list of objects of class "igraph" or "graphNEL". It also returns an object of class "igraph" or "graphNEL"; specifically, the same as the input target graph but appended with the "nodeConfidence" attribute to the nodes and the "edgeConfidence" attribute to the edges.

## Usage

```
eGraphConfidence(target, sources, plot = F)
```

## Arguments

target	the target graph, an object of class "igraph" or "graphNEL"
sources	a list of the source graphs, each with an object of class "igraph" or "graphNEL". These source graphs will be used to calculate how often a node (or an edge) in the target graph that can be found with them.
plot	logical to indicate whether the returned graph (i.e. the target graph plus the confidence information on nodes and edges) should be plotted. If it sets true, the plot will display the returned graph with the size of nodes indicative of the node confidence (the frequency that a node appears in the source graphs), and with the width of edges indicative of the edge confidence (the frequency that an edge appears in the source graphs)

## Value

an object of class "igraph" or "graphNEL", which is a target graph but appended with the "nodeConfidence" attribute to the nodes and the "edgeConfidence" attribute to the edges

## Note

None

## See Also

[visGraph](#)

## Examples

```
# 1) generate a target graph according to the ER model
g <- erdos.renyi.game(100, 1/100)
target <- eGraphInduce(g, V(g), knn=0)

# 2) generate a list source graphs according to the ER model
sources <- lapply(1:100, function(x) erdos.renyi.game(100*runif(1), 1/10))
```



```
# 3) append the confidence information from the source graphs into the target graph
g <- eGraphConfidence(target=target, sources=sources)

# 4) visualise the confidence target graph
visGraph(g, vertex.size=V(g)$nodeConfidence/10, edge.width=E(g)$edgeConfidence)
```

eGraphFind

*Function to find heuristically maximum scoring module***Description**

eGraphFind is supposed to find the maximum scoring module from an input graph and scores imposed on its nodes. The input graph and the output module are both of "igraph" or "graphNEL" object. The input scores imposed on the nodes in the input graph can be divided into two parts: the positive nodes and the negative nodes. The searching for maximum scoring module is deduced to find the connected subgraph containing the positive nodes as many as possible, but the negative nodes as few as possible. To this end, a heuristic search is used (see Note below).

**Usage**

```
eGraphFind(g, scores)
```

**Arguments**

g	an object of class "igraph" or "graphNEL"
scores	a vector of scores. For each element, it must have the name that could be mapped onto the input graph

**Value**

a module with a maximum score, an object of class "igraph" or "graphNEL"

**Note**

The search procedure is heuristic to find the module with the maximum score:

- i) transform the input graph into a new graph by collapsing connected positive nodes into a meta-node. As such, meta-nodes are isolated to each other but are linked via negative nodes (single-nodes). Clearly, meta-nodes have positive scores, and negative scores for the single-nodes.
- ii) append the weight attribute to the edges in the transformed graph. There are two types of edges: 1) the single-single edge with two single-nodes as two ends, and 2) single-meta edge with a single-node as one end and a meta-node as the other end. The weight for a single-single edge is the absolute sum of the scores in its two-end single-nodes but normalised by their degrees. The weight for a single-meta edge is simply the absolute score in its single-node end normalised by the degree. As such, weights are all non-negative.
- iii) find minimum spanning tree (MST) in the weighted transformed graph using Prim's greedy algorithm. A spanning tree of the weighted graph is a subgraph that is tree and connects all the node together. The MST is a spanning tree with the sum of its edge weights minimised among all possible spanning trees.

- iv) find all shortest paths between any pair of meta-nodes in the MST. Within the weighted transformed graph in ii), a subgraph is induced containing nodes (only occurring in these shortest paths) and all edges between them.
- v) within the induced subgraph, identify single-nodes that are direct neighbors of meta-nodes. For each of these single-nodes, also make sure it has the absolute scores no more than the sum of scores in its neighboring meta-nodes. These single-nodes meeting both criteria are called "linkers".
- vi) still within the induced subgraph in v), find the linker graph that contains only linkers and edges between them. Similarly to iii), find MST of the linker graph, called 'linker MST'. Notably, this linker MST serves as the scaffold, which only contains linkers but has meta-nodes being directly attached to.
- vii) in linker MST plus its attached meta-nodes, find the optimal path that has the sum of scores of its nodes and attached meta-nodes maximised amongst all possible paths. Nodes along this optimal path plus their attached meta-nodes are called 'module nodes'.
- viii) finally, from the input graph extract a subgraph (called 'module') that only contains module nodes and edges between them. This module is the maximum scoring module containing the positive nodes as many as possible, but the negative nodes as few as possible.

## See Also

[eGraphFind](#)

## Examples

```
# 1) generate an vector consisting of random values from beta distribution
x <- rbeta(1000, shape1=0.5, shape2=1)

# 2) fit a p-value distribution under beta-uniform mixture model
fit <- eBUMfit(x, ntry=1, hist.bum=TRUE, contour.bum=TRUE)

# 3) calculate the scores according to the fitted BUM and fdr=0.01
# using "pdf" method
scores <- eBUMscore(fit, method="pdf", fdr=0.05, scatter.bum=TRUE)
names(scores) <- as.character(1:length(scores))

# 4) generate a random graph according to the ER model
g <- erdos.renyi.game(1000, 1/100)

# 5) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 6) find the module with the maximum score
module <- eGraphFind(subg, scores)
```

---

eGraphInduce

*Function to generate a subgraph induced by given vertices and their k nearest neighbors*

---

## Description

eGraphInduce is supposed to produce a subgraph induced by given vertices and its k nearest neighbors. The input is a graph of "igraph" or "graphNEL" object, a list of the vertices of the graph, and a k value for finding k nearest neighbors for these vertices. The output is a subgraph induced by given vertices plus their k neighbours. The resultant subgraph inherits the class from the input one. The induced subgraph contains exactly the vertices of interest, and all the edges between them.

## Usage

```
eGraphInduce(g, nodes_query, knn = 0, remove.loops = T,
             largest.comp = T)
```

## Arguments

g	an object of class "igraph" or "graphNEL"
nodes_query	the vertices for which the calculation is performed
knn	an integer specifying how many k steps are used to find the nearest neighbours of the given vertices. By default, knn is set to zero; it means no neighbors will be considered. When knn is 1, the immediate neighbors of the given vertices will be also considered for inducing the subgraph. The same is true when knn is 2, etc
remove.loops	logical to indicate whether the loop edges are to be removed. By default, it sets to true for self-loops being removed
largest.comp	logical to indicate whether the largest component is only retained. By default, it sets to true for the largest component being left

## Value

- subg: an induced subgraph, an object of class "igraph" or "graphNEL"

## Note

The given vertices plus their k nearest neighbors will be used to induce the subgraph.

## See Also

[eGraphInduce](#)

## Examples

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) select the first 10 vertices as the query nodes
nodes_query <- V(g)[1:10]

# 3) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, nodes_query, knn=0)

# 4) produce the induced subgraph based on the nodes in query and their immediate neighbours
subg <- eGraphInduce(g, nodes_query, knn=1)
```

---

eGraphPipeline	<i>Function to setup the pipeline for finding maximum-scoring module from an input graph and the significance imposed on its nodes</i>
----------------	--

---

### Description

eGraphPipeline is supposed to finish ab initio maximum-scoring module identification for the input graph with the node information on the significance (p-values). It returns an object of class "igraph" or "graphNEL".

### Usage

```
eGraphPipeline(g, pval, method = c("pdf", "cdf"),
  fdr = NULL, nsize = NULL, plot = F, verbose = T)
```

### Arguments

g	an object of class "igraph" or "graphNEL"
pval	a vector containing input p-values. For each element, it must have the name that could be mapped onto the input graph
method	the method used for the transformation. It can be either "pdf" for the method based on the probability density function of the fitted model, or "cdf" for the method based on the cumulative distribution function of the fitted model
fdr	the given FDR threshold. By default, it is set to NULL, meaning there is no constraint. If given, those p-values with the FDR below this are considered significant and thus scored positively. Instead, those p-values with the FDR above this given FDR are considered insignificant and thus scored negatively
nsize	the desired number of nodes constrained to the resulting module. It is not null, a wide range of FDR will be scanned to find the FDR threshold leading to the desired number of nodes in the resulting module. Notably, the given FDR threshold will be overwritten.
plot	logical to indicate whether the histogram plot, contour plot and scatter plot should be drawn. By default, it sets to false for no plotting
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

### Value

a module with a maximum score, an object of class "igraph" or "graphNEL"

### Note

The pipeline sequentially consists of:

- i) [eBUMfit](#) used to fit the p-value distribution under beta-uniform mixture model.
- ii) if there is the desired number of nodes constrained to the resulting module, a wide range of FDR (including rough stage with large intervals, and finetune stage with smaller intervals) will be scanned to find the FDR threshold to meet the desired number of nodes.
- iii) [eBUMscore](#) used to calculate the scores according to the fitted BUM and FDR threshold.
- iv) [eGraphFind](#) used to find maximum-scoring module from the input graph and scores imposed on its nodes.

**See Also**

[eBUMfit](#), [eBUMscore](#), [eGraphFind](#)

**Examples**

```
# 1) generate an vector consisting of random values from beta distribution
x <- rbeta(1000, shape1=0.5, shape2=1)
names(x) <- as.character(1:length(x))

# 2) generate a random graph according to the ER model
g <- erdos.renyi.game(1000, 1/100)

# 3) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 4) find maximum-scoring module based on fdr=0.1 threshold
module <- eGraphPipeline(g=subg, pval=x, fdr=0.1)

# 5) find maximum-scoring module with the desired node number nsize=20
module <- eGraphPipeline(g=subg, pval=x, nsize=20)
```

---

eGraphReorder	<i>Function to reorder the multiple graph colorings within a sheet-shape rectangle grid</i>
---------------	---

---

**Description**

eGraphReorder is reorder the multiple graph colorings within a sheet-shape rectangle grid

**Usage**

```
eGraphReorder(g, data, feature = c("node", "edge"),
  node.normalise = c("none", "degree"), xdim = NULL,
  ydim = NULL, amplifier = NULL,
  metric = c("none", "pearson", "spearman", "kendall", "euclidean", "manhattan", "cos", "mi"),
  init = c("linear", "uniform", "sample"),
  algorithm = c("sequential", "batch"),
  alphaType = c("invert", "linear", "power"),
  neighKernel = c("gaussian", "bubble", "cutgaussian", "ep", "gamma"))
```

**Arguments**

g	an object of class "igraph" or "graphNEL"
data	an input data matrix used to color-code vertices/nodes. One column corresponds to one graph node coloring. The input matrix must have row names, and these names should include all node names of input graph, i.e. V(g)\$name, since there is a mapping operation. After mapping, the length of the pattern vector should be the same as the number of nodes of input graph. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: colormap, ncolors, zlim and colorbar)
feature	the type of the features used. It can be one of either 'edge' for the edge feature or 'node' for the node feature.

<code>node.normalise</code>	the normalisation of the nodes. It can be one of either 'none' for no normalisation or 'degree' for a node being penalised by its degree.
<code>xdim</code>	an integer specifying x-dimension of the grid
<code>ydim</code>	an integer specifying y-dimension of the grid
<code>amplifier</code>	an integer specifying the amplifier (3 by default) of the number of component planes. The product of the component number and the amplifier constitutes the number of rectangles in the sheet grid
<code>metric</code>	distance metric used to define the similarity between component planes. It can be "none", which means directly using column-wise vectors of codebook/data matrix. Otherwise, first calculate the covariance matrix from the codebook/data matrix. The distance metric used for calculating the covariance matrix between component planes can be: "pearson" for pearson correlation, "spearman" for spearman rank correlation, "kendall" for kendall tau rank correlation, "euclidean" for euclidean distance, "manhattan" for cityblock distance, "cos" for cosine similarity, "mi" for mutual information.
<code>init</code>	an initialisation method. It can be one of "uniform", "sample" and "linear" initialisation methods
<code>algorithm</code>	the training algorithm. Currently, only "sequential" algorithm has been implemented
<code>alphaType</code>	the alpha type. It can be one of "invert", "linear" and "power" alpha types
<code>neighKernel</code>	the training neighbor kernel. It can be one of "gaussian", "bubble", "cutgaussian", "ep" and "gamma" kernels

### Value

an object of class "sReorder", a list with following components:

- `nHex`: the total number of rectangles in the grid
- `xdim`: x-dimension of the grid
- `ydim`: y-dimension of the grid
- `uOrder`: the unique order/placement for each component plane that is reordered to the "sheet"-shape grid with rectangular lattice
- `coord`: a matrix of `nHex` x 2, with each row corresponding to the coordinates of each "uOrder" rectangle in the 2D map grid
- `call`: the call that produced this result

### Note

According to which features are used and whether nodes should be penalised by degrees, the feature data are constructed differently from the input data and input graph. When the node features are used, the feature data is the input data (or penalised data) with the same dimension. When the edge features are used, each entry (i.e. given an edge and a sample) in the feature data is the absolute difference between its two-end nodes (or after being penalised). Then, the constructed feature are subject to sample correlation analysis by `supraHex`. That is, a map grid (with sheet shape consisting of a rectangular lattice) is used to train either column-wise vectors of the feature data matrix or the covariance matrix thereof. As a result, similar samples are placed closer to each other within this map grid. More precisely, to ensure the unique placement, each sample mapped to the "sheet"-shape grid with rectangular lattice is determined iteratively in an order from the best matched to the next compromised one. If multiple samples are hit in the same rectangular lattice, the worse one is always sacrificed by moving to the next best one till all samples are placed somewhere exclusively on their own.

**See Also**[visGraphReorder](#)**Examples**

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) reorder the module with vertices being color-coded by input data
nnodes <- vcount(subg)
nsamples <- 10
data <- matrix(runif(nnodes*nsamples), nrow=nnodes, ncol=nsamples)
rownames(data) <- V(subg)$name
sReorder <- eGraphReorder(g=subg, data, feature="node", node.normalise="none")
```

ePvalAggregate

*Function to aggregate p values***Description**

ePvalAggregate is supposed to aggregate a input matrix p-values into a vector of aggregated p-values. The aggregate operation is applied to each row of input matrix, each resulting in an aggregated p-value. The method implemented can be based on the order statistics of p-values or according to Fisher's method.

**Usage**

```
ePvalAggregate(pmatrix,
  method = c("orderStatistic", "fishers"),
  order = ncol(pmatrix))
```

**Arguments**

pmatrix	a data frame or matrix of p-values
method	the method used. It can be either "orderStatistic" for the method based on the order statistics of p-values, or "fishers" for Fisher's method
order	an integer specifying the order used for the aggregation according to on the order statistics of p-values

**Value**

- ap: a vector with the length nrow(pmatrix), containing aggregated p-values

**Note**

For each row of input matrix with the  $c$  columns, there are  $c$  p-values that are uniformly independently distributed over  $[0,1]$  under the null hypothesis (uniform distribution). According to the order statistics, they follow the Beta distribution with the parameters  $a = order$  and  $b = nc - order + 1$ . According to the Fisher's method, after transformation by  $-2 * \sum^c \log(pvalue)$ , they follow Chi-Squared distribution.

**See Also**[ePvalAggregate](#)**Examples**

```
# 1) generate an iid uniformly-distributed random matrix of 1000x3
pmatrix <- cbind(runif(1000), runif(1000), runif(1000))

# 2) aggregate according to the ordre statistics
ap <- ePvalAggregate(pmatrix, method="orderStatistic")

# 3) aggregate according to the Fishers method
ap <- ePvalAggregate(pmatrix, method="fishers")
```

eRWR

*Function to implement Random Walk with Restart (RWR) to pre-compute affinity matrix for the input graph*

**Description**

eRWR is supposed to implement Random Walk with Restart (RWR) to pre-compute affinity matrix for for nodes in the input graph with respect to the starting node (loop over every node in the graph)

**Usage**

```
eRWR(g,
      normalise = c("laplacian", "row", "column", "none"),
      restart = 0.75)
```

**Arguments**

<code>g</code>	an object of class "igraph" or "graphNEL"
<code>normalise</code>	the way to normalise the adjacency matrix of the input graph. It can be 'laplacian' for laplacian normalisation, 'row' for row-wise normalisation, 'column' for column-wise normalisation, or 'none'
<code>restart</code>	the restart probability used for RWR

**Value**

- `PTmatrix`: affinity matrix with the dimension of  $n \times n$ , where  $n$  is the number of nodes in the input graph. Columns stand for starting nodes walking from, and rows for ending nodes walking to. Therefore, a column for a starting node represents a steady-state affinity vector that the starting node will visit all the ending nodes in the graph

**Note**

The input graph will treat as an unweighted graph if there is no 'weight' edge attribute associated

**See Also**[eGraphInduce](#)



## Examples

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) calculate the affinity matrix
PTmatrix <- eRWR(subg, normalise="laplacian", restart=0.75)

# 4) visualise affinity matrix
graphics::image(PTmatrix, col=visColormap("wyr")(64), zlim=c(0,1))
```

---

eView

*Function to view enrichment results in a sample-specific manner*


---

## Description

eView is supposed to view results of gene set enrichment analysis but for a specific sample.

## Usage

```
eView(eTerm, which_sample = 1, top_num = 10,
      sortBy = c("adjp", "gadjp", "ES", "nES", "pvalue", "FWER", "FDR", "qvalue"),
      decreasing = NULL, details = F)
```

## Arguments

eTerm	an object of class "eTerm"
which_sample	which sample will be viewed
top_num	the maximum number of gene sets will be viewed
sortBy	which statistics will be used for sorting and viewing gene sets. It can be "adjp" for adjusted p value, "gadjp" for globally adjusted p value, "ES" for enrichment score, "nES" for normalised enrichment score, "pvalue" for p value, "FWER" for family-wise error rate, "FDR" for false discovery rate, "qvalue" for q value
decreasing	logical to indicate whether to sort in a decreasing order. If it is null, it would be true for "ES" or "nES"; otherwise it would be false
details	logical to indicate whether the detail information of gene sets is also viewed. By default, it sets to false for no inclusion

## Value

a data frame with following components:

- setID: term ID
- ES: enrichment score
- nES: normalised enrichment score
- pvalue: nominal p value
- adjp: adjusted p value

- gadjp: globally adjusted p value
- FDR: false discovery rate
- qvalue: q value
- setSize: the number of genes in the set; optional, it is only appended when "details" is true
- name: term name; optional, it is only appended when "details" is true
- namespace: term namespace; optional, it is only appended when "details" is true
- distance: term distance; optional, it is only appended when "details" is true

### Note

none

### See Also

[eCal](#)

### Examples

```
#eView(eTerm, which_sample=1, top_num=10, sortBy="adjp", decreasing=F, details=T)
```

---

eWrite	<i>Function to write out enrichment results</i>
--------	---

---

### Description

eWrite is supposed to write out enrichment results.

### Usage

```
eWrite(eTerm,
  which_content = c("gadjp", "adjp", "pvalue", "FWER", "FDR", "qvalue", "nES", "ES"),
  which_score = c("gadjp", "adjp", "FWER", "FDR", "qvalue"),
  cutoff = 0.1, filename = NULL, keep.significance = T)
```

### Arguments

eTerm	an object of class "eTerm"
which_content	the content will be written out. It includes two categories: i) based on "adjp" for adjusted p value, "gadjp" for globally adjusted p value, "pvalue" for p value, "FWER" for family-wise error rate, "FDR" for false discovery rate, "qvalue" for q value; ii) based on "ES" for enrichment score, "nES" for normalised enrichment score. For the former, the content is : first $-1 \times \log_{10}$ -transformed, and then multiplied by -1 if nES is negative.
which_score	which statistics/score will be used for declaring the significance. It can be "adjp" for adjusted p value, "gadjp" for globally adjusted p value, "FWER" for family-wise error rate, "FDR" for false discovery rate, "qvalue" for q value
cutoff	a cutoff to declare the significance. It should be used together with 'which_score'
filename	a character string naming a filename
keep.significance	logical to indicate whether or not to mask those insignificant by NA. By default, it sets to true to mask those insignificant by NA

**Value**

a data frame with following components:

- setID: term ID
- setSize: the number of genes in the set
- name: term name
- namespace: term namespace
- distance: term distance
- sample names: sample names in the next columns

**Note**

If "filename" is not NULL, a tab-delimited text file will be also written out.

**See Also**

[eCal](#)

**Examples**

```
#output <- eWrite(eTerm, which_content="gadjp", which_score="gadjp", cutoff=0.05, filename="eWrite_output.t
```

---

visGraph

---

*Function to visualise a graph object of class "igraph" or "graphNEL"*


---

**Description**

visGraph is supposed to visualise a graph object of class "igraph" or "graphNEL". It also allows the color-coding of vertices by providing the input pattern.

**Usage**

```
visGraph(g, pattern = NULL,
  colormap = c("bwr", "jet", "gbr", "wyr", "br", "yr", "rainbow", "wb"),
  ncolors = 40, zlim = NULL, colorbar = T, newpage = T,
  glayout = layout.fruchterman.reingold,
  vertex.frame.color = NA, vertex.size = NULL,
  vertex.color = NULL, vertex.shape = NULL,
  vertex.label = NULL, vertex.label.cex = NULL,
  vertex.label.dist = NULL, vertex.label.color = "black",
  ...)
```

**Arguments**

g	an object of class "igraph" or "graphNEL"
pattern	a numeric vector used to color-code vertices/nodes. Notably, if the input vector contains names, then these names should include all node names of input graph, i.e. V(g)\$name, since there is a mapping operation. After mapping, the length of the patern vector should be the same as the number of nodes of input graph; otherwise, this input pattern will be ignored. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: colormap, ncolors, zlim and colorbar)

colormap	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in <a href="http://html-color-codes.info/color-names">http://html-color-codes.info/color-names</a>
ncolors	the number of colors specified over the colormap
zlim	the minimum and maximum z/pattern values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted
colorbar	logical to indicate whether to append a colorbar. If pattern is null, it always sets to false
newpage	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
glayout	either a function or a numeric matrix configuring how the vertices will be placed on the plot. If layout is a function, this function will be called with the graph as the single parameter to determine the actual coordinates. This function can be one of "layout.auto", "layout.random", "layout.circle", "layout.sphere", "layout.fruchterman.reingold", "layout.kamada.kawai", "layout.spring", "layout.reingold.tilford", "layout.fruchterman.reingold.grid", "layout.lgl", "layout.graphopt", "layout.svd" and "layout.norm". A full explanation of these layouts can be found in <a href="http://igraph.sourceforge.net/doc/R/layout.html">http://igraph.sourceforge.net/doc/R/layout.html</a>
vertex.frame.color	the color of the frame of the vertices. If it is NA, then there is no frame
vertex.size	the size of each vertex. If it is a vector, each vertex may differ in size
vertex.color	the fill color of the vertices. If it is NA, then there is no fill color. If the pattern is given, this setup will be ignored
vertex.shape	the shape of each vertex. It can be one of "circle", "square", "csquare", "rectangle", "crectangle", "vrectangle", "pie" ( <a href="http://igraph.sourceforge.net/doc/R/vertex.shape.pie.html">http://igraph.sourceforge.net/doc/R/vertex.shape.pie.html</a> ), "sphere", and "none". If it sets to NULL, these vertices with negative will be "csquare" and the rest "circle".
vertex.label	the label of the vertices. If it is NA, then there is no label. The default vertex labels are the name attribute of the nodes
vertex.label.cex	the font size of vertex labels.
vertex.label.dist	the distance of the label from the center of the vertex. If it is 0 then the label is centered on the vertex. If it is 1 then the label is displayed beside the vertex.
vertex.label.color	the color of vertex labels.
...	additional graphic parameters. See <a href="http://igraph.sourceforge.net/doc/R/plot.graph.html">http://igraph.sourceforge.net/doc/R/plot.graph.html</a> for the complete list.

## Value

invisible

**Note**

none

**See Also**

[eGraphFind](#)

**Examples**

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) visualise the subg with vertices being color-coded by the pattern
pattern <- runif(vcount(subg))
names(pattern) <- V(subg)$name
visGraph(g=subg, pattern=pattern, vertex.shape="sphere")
```

---

visGraphAnimate

*Function to animate the same graph but with multiple graph node colorings according to input data matrix*

---

**Description**

visGraphAnimate is supposed to animate the same graph but with multiple colorings according to input data matrix. The output can be a pdf file containing a list of frames/images or a mp4 video file. To support video output file, the software 'ffmpeg' must be first installed (see Note for HOWTO).

**Usage**

```
visGraphAnimate(g, data, filename = "visGraphAnimate",
  filetype = c("pdf", "mp4"), num.frame = ncol(data),
  sec_per_frame = 1, height.device = 7,
  margin = rep(0.1, 4), border.color = "#EEEEEE",
  colormap = c("bwr", "jet", "gbr", "wyr", "br", "yr", "rainbow", "wb"),
  ncolors = 40, zlim = NULL, colorbar = T,
  colorbar.fraction = 0.25,
  glayout = layout.fruchterman.reingold, mtext.side = 3,
  mtext.adj = 0, mtext.cex = 1, mtext.font = 2,
  mtext.col = "black", ...)
```

**Arguments**

g	an object of class "igraph" or "graphNEL"
data	an input data matrix used to color-code vertices/nodes. One column corresponds to one graph node coloring. The input matrix must have row names, and these names should include all node names of input graph, i.e. V(g)\$name, since there is a mapping operation. After mapping, the length of the pattern vector should be the same as the number of nodes of input graph. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: colormap, ncolors, zlim and colorbar)

filename	the without-extension part of the name of the output file.
filetype	the type of the output file, i.e. the extension of the output file name. It can be one of either 'pdf' for the pdf file or 'mp4' for the mp4 video file.
num.frame	a numeric value specifying the number of frames/images. By default, it sets to the number of columns in the input data matrix
sec_per_frame	a numeric value specifying how long (seconds) it takes to stream a frame/image. This argument only works when producing mp4 video.
height.device	a numeric value specifying the height (or width) of device/frame/image.
margin	margins as units of length 4 or 1
border.color	the border color of each figure
colormap	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in <a href="http://html-color-codes.info/color-names">http://html-color-codes.info/color-names</a>
ncolors	the number of colors specified over the colormap
zlim	the minimum and maximum z/pattern values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted
colorbar	logical to indicate whether to append a colorbar. If pattern is null, it always sets to false
colorbar.fraction	the relative fraction of colorbar block against the figure block
layout	either a function or a numeric matrix configuring how the vertices will be placed on the plot. If layout is a function, this function will be called with the graph as the single parameter to determine the actual coordinates. This function can be one of "layout.auto", "layout.random", "layout.circle", "layout.sphere", "layout.fruchterman.reingold", "layout.kamada.kawai", "layout.spring", "layout.reingold.tilford", "layout.fruchterman.reingold.grid", "layout.lgl", "layout.graphopt", "layout.svd" and "layout.norm". A full explanation of these layouts can be found in <a href="http://igraph.sourceforge.net/doc/R/layout.html">http://igraph.sourceforge.net/doc/R/layout.html</a>
mtext.side	on which side of the mtext plot (1=bottom, 2=left, 3=top, 4=right)
mtext.adj	the adjustment for mtext alignment (0 for left or bottom alignment, 1 for right or top alignment)
mtext.cex	the font size of mtext labels
mtext.font	the font weight of mtext labels
mtext.col	the color of mtext labels
...	additional graphic parameters. See <a href="http://igraph.sourceforge.net/doc/R/plot.graph.html">http://igraph.sourceforge.net/doc/R/plot.graph.html</a> for the complete list.

## Value

invisible

**Note**

Shell command lines for ffmpeg installation in Terminal (for both Linux and Mac):

```
wget http://www.ffmpeg.org/releases/ffmpeg-2.1.1.tar.gz
tar xvfz ffmpeg-2.1.1.tar.gz
cd ffmpeg-2.1.1
./configure --disable-yasm
make
make install
```

**See Also**

[visGraphMul](#)

**Examples**

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) visualise the module with vertices being color-coded by scores
nnodes <- vcount(subg)
nsamples <- 10
data <- matrix(runif(nnodes*nsamples), nrow=nnodes, ncol=nsamples)
rownames(data) <- V(subg)$name
# output as a pdf file
visGraphAnimate(g=subg, data=data, filetype="pdf")
# output as a mp4 file
visGraphAnimate(g=subg, data=data, filetype="mp4")
```

---

visGraphArc

---

*Function to visualise an igraph object via arc diagram*


---

**Description**

visGraphArc is supposed to visualise a graph object of class "igraph" via arc diagram in one-dimensional layout. More precisely, it displays vertices (nodes) along an axis, with edges linked by arcs. With proper ordering of vertices (e.g. according to communities and degrees), arc diagram is able to identify clusters and bridges (as effective as two-dimensional layout). One advantage of using arc diagram is to allow for easy annotations along vertices.

**Usage**

```
visGraphArc(g, orientation = c("vertical", "horizontal"),
  newpage = T, ordering = NULL, labels = V(g)$name,
  vertex.label.color = "black", vertex.label.cex = 1,
  vertex.color = "transparent",
  vertex.frame.color = "black",
  vertex.size = log(degree(g)) + 0.1, vertex.pch = 21,
  vertex.lwd = 1, edge.color = "grey", edge.width = 1,
  edge.lty = 1, ...)
```

**Arguments**

<code>g</code>	an object of class "igraph"
<code>orientation</code>	the orientation of the plots. It can be either "vertical" (default) or "horizontal"
<code>newpage</code>	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
<code>ordering</code>	a numeric vector about the ordering of vertices. It is optional. It is highly recommend to order vertices according to communities and degrees
<code>labels</code>	the label of the vertices. The default vertex labels are the name attribute of the nodes
<code>vertex.label.color</code>	the color of vertex labels
<code>vertex.label.cex</code>	the font size of vertex labels
<code>vertex.color</code>	the fill color of the vertices. The default vertex colors are transparent
<code>vertex.frame.color</code>	the color of the frame of the vertices. The default vertex frame colors are black
<code>vertex.size</code>	the size of each vertex. By default, it is decided according to node degrees
<code>vertex.pch</code>	the shape of each vertex. Either an integer specifying a symbol or a single character to be used as the default in plotting points. See <a href="http://www.statmethods.net/advgraphs/parameters.html">http://www.statmethods.net/advgraphs/parameters.html</a>
<code>vertex.lwd</code>	line width for the vertices (default 1)
<code>edge.color</code>	the color of the edges (default "grey")
<code>edge.width</code>	line width for the edges (default 1)
<code>edge.lty</code>	line type for the edges (default 1)
<code>...</code>	additional graphic parameters associated with 'mtext'

**Value**

invisible

**Note**

none

**See Also**[visGraph](#)**Examples**

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/80)

# 2) produce the induced subgraph only based on the nodes in query
g <- eGraphInduce(g, V(g), knn=0)

# 3) color nodes according to communities identified via a spin-glass model and simulated annealing
com <- spinglass.community(g, spins=4)
vgroups <- com$membership
palette.name <- visColormap(colormap="rainbow")
```



```

vcolors <- palette.name(length(com))[vgroups]

# 4) size nodes according to degrees
vdegrees <- igraph::degree(g)

# 5) sort nodes: first by communities and then degrees
tmp <- data.frame(ind=1:vcount(g), vgroups, vdegrees)
ordering <- tmp[order(vgroups,vdegrees),]$ind

# 6) visualise graph using 1-dimensional arc diagram
visGraphArc(g, ordering=ordering, labels=V(g)$name, vertex.label.color=vcolors, vertex.color=vcolors, vertex.frame.color=vcolors)

# 7) as comparison, also visualise graph on 2-dimensional layout
visGraph(g, layout=layout.kamada.kawai(g), vertex.label=V(g)$name, vertex.color=vcolors, vertex.frame.color=vcolors)

```

---

visGraphCircle	<i>Function to visualise an igraph object via circle diagram</i>
----------------	--

---

## Description

visGraphCircle is supposed to visualise a graph object of class "igraph" via circle diagram. For better visualisation, ordering of vertices is determined according to communities and degrees.

## Usage

```

visGraphCircle(g, com, circles = c("single", "multiple"),
  newpage = T, ordering = NULL,
  colormap = c("rainbow", "bwr", "jet", "gbr", "wyr", "br", "yr", "wb"),
  vertex.label = V(g)$name,
  vertex.size = log(igraph::degree(g)) + 2,
  vertex.label.color = "black", vertex.label.cex = 0.6,
  vertex.label.dist = 0.75, vertex.shape = "sphere",
  edge.width = 1, edge.lty = 1,
  edge.color.within = "grey",
  edge.color.crossing = "black", mark.shape = 1,
  mark.expand = 10, ...)

```

## Arguments

g	an object of class "igraph"
com	an object of class "communities" (see <a href="http://igraph.sourceforge.net/doc/R/communities.html">http://igraph.sourceforge.net/doc/R/communities.html</a> )
circles	how circles are drawn in the plot. It can be either "single" for all communities being drawn in a single circle (by default) or "multiple" for communities being drawn in the different circles (i.e. one circle per community)
newpage	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
ordering	a numeric vector about the ordering of vertices. It is optional. It is highly recommend to order vertices according to communities and degrees

<code>colormap</code>	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in <a href="http://html-color-codes.info/color-names">http://html-color-codes.info/color-names</a>
<code>vertex.label</code>	the label of the vertices. The default vertex labels are the name attribute of the nodes
<code>vertex.size</code>	the size of each vertex. By default, it is decided according to node degrees
<code>vertex.label.color</code>	the color of vertex labels
<code>vertex.label.cex</code>	the font size of vertex labels
<code>vertex.label.dist</code>	the distance of the label from the center of the vertex. If it is 0 then the label is centered on the vertex. If it is 1 then the label is displayed beside the vertex.
<code>vertex.shape</code>	the shape of each vertex. It can be one of "circle", "square", "csquare", "rectangle", "crectangle", "vrectangle", "pie" ( <a href="http://igraph.sourceforge.net/doc/R/vertex.shape.pie.html">http://igraph.sourceforge.net/doc/R/vertex.shape.pie.html</a> ), "sphere", and "none". If it sets to NULL, these vertices with negative will be "csquare" and the rest "circle".
<code>edge.width</code>	line width for the edges (default 1)
<code>edge.lty</code>	line type for the edges (default 1)
<code>edge.color.within</code>	the color for edges within a community (default "grey")
<code>edge.color.crossing</code>	the color for edges between communities (default "black")
<code>mark.shape</code>	a numeric scalar or vector controlling the smoothness of the vertex group marking polygons. Its possible values are between -1 (fully polygons) and 1 (fully smoothness)
<code>mark.expand</code>	a numeric scalar or vector, the size of the border around the marked vertex groups
<code>...</code>	additional graphic parameters. See <a href="http://igraph.sourceforge.net/doc/R/plot.graph.html">http://igraph.sourceforge.net/doc/R/plot.graph.html</a> for the complete list.

**Value**

invisible

**Note**

none

**See Also**[visGraph](#)

## Examples

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/80)

# 2) produce the induced subgraph only based on the nodes in query
g <- eGraphInduce(g, V(g), knn=0)

# 3) color nodes according to communities identified via a spin-glass model and simulated annealing
com <- spinglass.community(g, spins=4)
vgroups <- com$membership
palette.name <- visColormap(colormap="rainbow")
mcolors <- palette.name(length(com))
vcolors <- mcolors[vgroups]

# 4) size nodes according to degrees
vdegrees <- igraph::degree(g)

# 5) sort nodes: first by communities and then degrees

tmp<-data.frame(ind=1:vcount(g), vgroups, vdegrees)
ordering <- tmp[order(vgroups,vdegrees),]$ind

# 6) visualise graph using circle diagram
# 6a) drawn into a single circle
visGraphCircle(g=g, com=com, ordering=ordering, vertex.label=V(g)$name)

# 6b) drawn into multiple circles (one circle per community)
visGraphCircle(g=g, com=com, circles="multiple", ordering=ordering, vertex.label=V(g)$name)

# 7) as comparison, also visualise graph on 2-dimensional layout
mark.groups <- communities(com)
mark.col <- visColoralpha(mcolors, alpha=0.2)
mark.border <- visColoralpha(mcolors, alpha=0.2)
edge.color <- c("grey", "black")[crossing(com,g)+1]
visGraph(g, glayout=layout.fruchterman.reingold, vertex.color=vcolors, vertex.frame.color=vcolors, vertex.l
```

visGraphMul

*Function to visualise the same graph but with multiple graph node colorings according to input data matrix*

## Description

visGraphMul is supposed to visualise the same graph but with multiple colorings according to input data matrix

## Usage

```
visGraphMul(g, data, height = 7, margin = rep(0.1, 4),
  border.color = "#EEEEEE",
  colormap = c("bwr", "jet", "gbr", "wyr", "br", "yr", "rainbow", "wb"),
  ncolors = 40, zlim = NULL, colorbar = T,
  colorbar.fraction = 0.25, newpage = T,
  glayout = layout.fruchterman.reingold, mtext.side = 3,
```

```
mtext.adj = 0, mtext.cex = 1, mtext.font = 2,
mtext.col = "black", ...)
```

## Arguments

<code>g</code>	an object of class "igraph" or "graphNEL"
<code>data</code>	an input data matrix used to color-code vertices/nodes. One column corresponds to one graph node coloring. The input matrix must have row names, and these names should include all node names of input graph, i.e. <code>V(g)\$name</code> , since there is a mapping operation. After mapping, the length of the pattern vector should be the same as the number of nodes of input graph. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: <code>colormap</code> , <code>ncolors</code> , <code>zlim</code> and <code>colorbar</code> )
<code>height</code>	a numeric value specifying the height of device
<code>margin</code>	margins as units of length 4 or 1
<code>border.color</code>	the border color of each figure
<code>colormap</code>	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in <a href="http://html-color-codes.info/color-names">http://html-color-codes.info/color-names</a>
<code>ncolors</code>	the number of colors specified over the colormap
<code>zlim</code>	the minimum and maximum <code>z/pattern</code> values for which colors should be plotted, defaulting to the range of the finite values of <code>z</code> . Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted
<code>colorbar</code>	logical to indicate whether to append a colorbar. If <code>pattern</code> is null, it always sets to false
<code>colorbar.fraction</code>	the relative fraction of colorbar block against the figure block
<code>newpage</code>	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
<code>glayout</code>	either a function or a numeric matrix configuring how the vertices will be placed on the plot. If <code>layout</code> is a function, this function will be called with the graph as the single parameter to determine the actual coordinates. This function can be one of "layout.auto", "layout.random", "layout.circle", "layout.sphere", "layout.fruchterman.reingold", "layout.kamada.kawai", "layout.spring", "layout.reingold.tilford", "layout.fruchterman.reingold.grid", "layout.lgl", "layout.graphopt", "layout.svd" and "layout.norm". A full explanation of these layouts can be found in <a href="http://igraph.sourceforge.net/doc/R/layout.html">http://igraph.sourceforge.net/doc/R/layout.html</a>
<code>mtext.side</code>	on which side of the mtext plot (1=bottom, 2=left, 3=top, 4=right)
<code>mtext.adj</code>	the adjustment for mtext alignment (0 for left or bottom alignment, 1 for right or top alignment)
<code>mtext.cex</code>	the font size of mtext labels
<code>mtext.font</code>	the font weight of mtext labels

mtext.col      the color of mtext labels  
 ...            additional graphic parameters. See <http://igraph.sourceforge.net/doc/R/plot.graph.html> for the complete list.

### Value

invisible

### Note

none

### See Also

[visGraph](#)

### Examples

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/80)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) visualise the module with vertices being color-coded by scores
nnodes <- vcount(subg)
nsamples <- 10
data <- matrix(runif(nnodes*nsamples), nrow=nnodes, ncol=nsamples)
rownames(data) <- V(subg)$name
visGraphMul(g=subg, data=data, glayout=layout.fruchterman.reingold)
```

---

visGraphReorder	<i>Function to visualise the multiple graph colorings reorded within a sheet-shape rectangle grid</i>
-----------------	---

---

### Description

visGraphReorder is supposed to visualise the multiple graph colorings reorded within a sheet-shape rectangle grid

### Usage

```
visGraphReorder(g, data, sReorder, height = 7,
  margin = rep(0.1, 4), border.color = "#EEEEEE",
  colormap = c("bwr", "jet", "gbr", "wyr", "br", "yr", "rainbow", "wb"),
  ncolors = 40, zlim = NULL, colorbar = T,
  colorbar.fraction = 0.5, newpage = T,
  glayout = layout.fruchterman.reingold, mtext.side = 3,
  mtext.adj = 0, mtext.cex = 1, mtext.font = 2,
  mtext.col = "black", ...)
```

**Arguments**

<code>g</code>	an object of class "igraph" or "graphNEL"
<code>data</code>	an input data matrix used to color-code vertices/nodes. One column corresponds to one graph node coloring. The input matrix must have row names, and these names should include all node names of input graph, i.e. $V(g)$name, since there is a mapping operation. After mapping, the length of the pattern vector should be the same as the number of nodes of input graph. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: colormap, ncolors, zlim and colorbar)$
<code>height</code>	a numeric value specifying the height of device
<code>sReorder</code>	an object of class "sReorder"
<code>margin</code>	margins as units of length 4 or 1
<code>border.color</code>	the border color of each figure
<code>colormap</code>	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in <a href="http://html-color-codes.info/color-names">http://html-color-codes.info/color-names</a>
<code>ncolors</code>	the number of colors specified over the colormap
<code>zlim</code>	the minimum and maximum z/pattern values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted
<code>colorbar</code>	logical to indicate whether to append a colorbar. If pattern is null, it always sets to false
<code>colorbar.fraction</code>	the relative fraction of colorbar block against the figure block
<code>newpage</code>	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
<code>layout</code>	either a function or a numeric matrix configuring how the vertices will be placed on the plot. If layout is a function, this function will be called with the graph as the single parameter to determine the actual coordinates. This function can be one of "layout.auto", "layout.random", "layout.circle", "layout.sphere", "layout.fruchterman.reingold", "layout.kamada.kawai", "layout.spring", "layout.reingold.tilford", "layout.fruchterman.reingold.grid", "layout.lgl", "layout.graphopt", "layout.svd" and "layout.norm". A full explanation of these layouts can be found in <a href="http://igraph.sourceforge.net/doc/R/layout.html">http://igraph.sourceforge.net/doc/R/layout.html</a>
<code>mtext.side</code>	on which side of the mtext plot (1=bottom, 2=left, 3=top, 4=right)
<code>mtext.adj</code>	the adjustment for mtext alignment (0 for left or bottom alignment, 1 for right or top alignment)
<code>mtext.cex</code>	the font size of mtext labels
<code>mtext.font</code>	the font weight of mtext labels
<code>mtext.col</code>	the color of mtext labels
<code>...</code>	additional graphic parameters. See <a href="http://igraph.sourceforge.net/doc/R/plot.graph.html">http://igraph.sourceforge.net/doc/R/plot.graph.html</a> for the complete list.

**Value**

invisible

**Note**

none

**See Also**[visGraph](#), [eGraphReorder](#)**Examples**

```
# 1) generate a random graph according to the ER model
g <- erdos.renyi.game(100, 1/100)

# 2) produce the induced subgraph only based on the nodes in query
subg <- eGraphInduce(g, V(g), knn=0)

# 3) reorder the module with vertices being color-coded by input data
nnodes <- vcount(subg)
nsamples <- 10
data <- matrix(runif(nnodes*nsamples), nrow=nnodes, ncol=nsamples)
rownames(data) <- V(subg)$name
sReorder <- eGraphReorder(g=subg, data, feature="node", node.normalise="none")

# 4) visualise the module with vertices being color-coded by input data
visGraphReorder(g=subg, data=data, sReorder)
```

visRunES

*Function to visualise running enrichment score for a given sample and a gene set*

**Description**

eView is supposed to visualise running enrichment score for a given sample and a gene set. To help understand the underlying running enrichment score, the input gene scores are also displayed. Positions for members in the given gene set are color-coded in both displays (red line for the positive gene scores, and green line for the negative).

**Usage**

```
visRunES(eTerm, which_sample = 1,
         which_term = "GO:0006281", weight = 1,
         orientation = c("vertical", "horizontal"), newpage = T)
```

**Arguments**

eTerm	an object of class "eTerm"
which_sample	which sample will be used. It can be index or sample names
which_term	which term will be used. It can be index or term ID or term names

weight	type of score weighth. It can be "0" for unweighted (an equivalent to Kolmogorov-Smirnov, only considering the rank), "1" for weighted by input gene score (by default), and "2" for over-weighted, and so on
orientation	the orientation of the plots. It can be either "vertical" (default) or "horizontal"
newpage	logical to indicate whether to open a new page. By default, it sets to true for opening a new page

**Value**

invisible

**Note**

none

**See Also**

[eCal](#), [eView](#)

**Examples**

```
#visRunES(eTerm, which_sample=1, which_term=1)
```



# Index

eBUMfit, [2](#), [5](#), [12](#), [13](#)  
eBUMscore, [3](#), [4](#), [12](#), [13](#)  
eCal, [5](#), [18](#), [19](#), [32](#)  
eGraphConfidence, [8](#)  
eGraphFind, [9](#), [10](#), [12](#), [13](#), [21](#)  
eGraphInduce, [10](#), [11](#), [16](#)  
eGraphPipeline, [12](#)  
eGraphReorder, [13](#), [31](#)  
ePvalAggregate, [15](#), [16](#)  
eRWR, [16](#)  
eView, [7](#), [17](#), [32](#)  
eWrite, [7](#), [18](#)  
  
visGraph, [8](#), [19](#), [24](#), [26](#), [29](#), [31](#)  
visGraphAnimate, [21](#)  
visGraphArc, [23](#)  
visGraphCircle, [25](#)  
visGraphMul, [23](#), [27](#)  
visGraphReorder, [15](#), [29](#)  
visRunES, [31](#)