dcRWRpipeline

October 24, 2014

dcRWRpipeline	*Function to setup a pipeine to estimate RWR-based contact strength between samples from an input domain-sample data matrix and an input graph*

Description

dcRWRpipeline is supposed to estimate sample relationships (ie. contact strength between samples) from an input domain-sample matrix and an input graph (such as a domain-domain semantic network). The pipeline includes: 1) random walk restart (RWR) of the input graph using the input matrix as seeds; 2) calculation of contact strength (inner products of RWR-smoothed columns of input matrix); 3) estimation of the contact signficance by a randomalisation procedure. It supports two methods how to use RWR: 'direct' for directly applying RWR in the given seeds; 'indirectly' for first pre-computing affinity matrix of the input graph, and then deriving the affinity score. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
dcRWRpipeline(data, g, method = c("indirect", "direct"),
normalise = c("laplacian", "row", "column", "none"), restart = 0.75,
normalise.affinity.matrix = c("none", "quantile"),
permutation = c("random", "degree"), num.permutation = 100,
p.adjust.method = c("BH", "BY", "bonferroni", "holm", "hochberg",
"hommel"),
adjp.cutoff = 0.05, parallel = TRUE, multicores = NULL, verbose = T)
```

Arguments

data	an input domain-sample data matrix used for seeds. Each value in input domain-sample matrix does not necessarily have to be binary (non-zeros will be used as a weight, but should be non-negative for easy interpretation).
g	an object of class "igraph" or Dnetwork
method	the method used to calculate RWR. It can be 'direct' for directly applying RWR, 'indirect' for indirectly applying RWR (first pre-compute affinity matrix and then derive the affinity score)
normalise	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'

`restart` the restart probability used for RWR. The restart probability takes the value from 0 to 1, controlling the range from the starting nodes/seeds that the walker will explore. The higher the value, the more likely the walker is to visit the nodes centered on the starting nodes. At the extreme when the restart probability is zero, the walker moves freely to the neighbors at each step without restarting from seeds, i.e., following a random walk (RW)

`normalise.affinity.matrix` the way to normalise the output affinity matrix. It can be 'none' for no normalisation, 'quantile' for quantile normalisation to ensure that columns (if multiple) of the output affinity matrix have the same quantiles

`permutation` how to do permutation. It can be 'degree' for degree-preserving permutation, 'random' for permutation in random

`num.permutation` the number of permutations used to for generating the distribution of contact strength under randomalisation

`p.adjust.method` the method used to adjust p-values. It can be one of "BH", "BY", "bonferroni", "holm", "hochberg" and "hommel". The first two methods "BH" (widely used) and "BY" control the false discovery rate (FDR: the expected proportion of false discoveries amongst the rejected hypotheses); the last four methods "bonferroni", "holm", "hochberg" and "hommel" are designed to give strong control of the family-wise error rate (FWER). Notes: FDR is a less stringent condition than FWER

`adjp.cutoff` the cutoff of adjusted pvalue to construct the contact graph

`parallel` logical to indicate whether parallel computation with multicores is used. By default, it sets to true, but not necessarily does so. Partly because parallel backends available will be system-specific (now only Linux or Mac OS). Also, it will depend on whether these two packages "foreach" and "doMC" have been installed. It can be installed via: `source("http://bioconductor.org/biocLite.R"); biocLite(c("foreach","doMC"))`. If not yet installed, this option will be disabled

`multicores` an integer to specify how many cores will be registered as the multicore parallel backend to the 'foreach' package. If NULL, it will use a half of cores available in a user's computer. This option only works when parallel computation is enabled

`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 "iContact", a list with following components:

- `ratio`: a symmetric matrix storing ratio (the observed against the expected) between pairwise samples
- `zscore`: a symmetric matrix storing zscore between pairwise samples
- `pval`: a symmetric matrix storing pvalue between pairwise samples
- `adjpval`: a symmetric matrix storing adjusted pvalue between pairwise samples
- `icontact`: the constructed contact graph (as an 'igraph' object) under the cutoff of adjusted value
- `Amatrix`: a pre-computated affinity matrix when using 'inderect' method; NULL otherwise
- `call`: the call that produced this result

Note

The choice of which method to use RWR depends on the number of seed sets and the number of permutations for statistical test. If the total product of both numbers are huge, it is better to use 'indrect' method (for a single run).

See Also

dcRDataLoader, dcDAGannotate, dcDAGdomainSim, dcConverter

Examples

```
# 1) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)

# 2) load SCOP superfamilies annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(SCOP.sf2GOMF)

# 3) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=TRUE)

# 4) calculate pair-wise semantic similarity between 10 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,10)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork

# 5) estimate RWR dating based sample/term relationships
# define sets of seeds as data
# each seed with equal weight (i.e. all non-zero entries are 1)
data <- data.frame(aSeeds=c(1,0,1,0,1), bSeeds=c(0,0,1,0,1))
rownames(data) <- id(dnetwork)[1:5]
# calcualte their two contact graph
coutput <- dcRWRpipeline(data=data, g=dnetwork, parallel=FALSE)
coutput
```