

Join GitHub today

Dismiss

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▾

[netpredictor](#) / [R](#) / [netpredictor.R](#)[Find file](#)[Copy path](#) abhik1368 remove prints

ebf022d on Feb 10, 2016

[1 contributor](#)

999 lines (853 sloc) 37.5 KB

```
1 #' @title Perform Random walk on a Unipartite Network
2 #' @description Performs random walk with restart with preferred seed sets. If seed sets are not given then the adjacency
3 #' matrix is taken as the input as the input seed sets. The restart parameter controls the random walk probability. This can be
4 #' changed default is set to 0.8. Normalization of the matrix can be done by row,column,laplacian. For faster computation
5 #' Parallelization is implemented with multicores. Parallelization is done using foreach package.
6 #' @param ig igraph object
7 #' @param normalise normalise method
8 #' @param dataSeed vector or dataframe
9 #' @param restart restart probability parameter
10 #' @param parallel to execute in parallel either TRUE or FALSE
11 #' @param multicores Number of cores to be used when running in parallel
12 #' @param verbose Verbose output
13 #' @name uNetwalk
14 #' @references
15 #' \itemize{
16 #'   \item Kohler S, et al. Walking the Interactome for Prioritization of Candidate Disease Genes. American Journal of Human Genetics. 2008
17 #'   \item Can, T., Camoglu, O., and Singh, A.K. (2005). Analysis of protein-protein interaction networks using random walks. In BIOKDD '05
18 #' }
19 #' @export
20 #' @examples
21 #' \donttest{
22 #' # generate a random graph according to the ER model
23 #' library(igraph)
24 #' library(netpredictor)
25 #' g1 <- upgrade_graph(erdos.renyi.game(100, 1/100))
26 #' V(g1)$name <- seq(1,100,1)
27 #' ## Computing RWR
28 #' pM <- uNetwalk(g1,normalise="laplacian", restart=0.75, parallel=FALSE)
29 #' ## Settin the seed nodes.
30 #' d1 <- c(1,0,1,0,1)
31 #' d2 <- c(0,0,1,0,1)
32 #' dataSeed <- data.frame(d1,d2)
33 #' rownames(dataSeed) <- 1:5
34 #' pM <- uNetwalk(g1, normalise="laplacian", dataSeed=dataSeed, restart=0.8,
35 #'               parallel=FALSE,multicores=NULL, verbose=T)
36 #' }
37
38 uNetwalk <- function(ig, normalise=c("row","column","laplacian","none"), dataSeed=NULL, restart=0.8, parallel=TRUE, multicores=NULL, verbos
39 {
40
41   startT <- Sys.time()
42   stop_delta <- 1e-7
43   if (class(ig) != "igraph"){
44     stop("The function must apply to either 'igraph' or 'matrix' object.\n")
```

```

45     }
46
47
48     if(verbose){
49         now <- Sys.time()
50         message(sprintf("First, get the adjacency matrix of the input graph (%s) ...", as.character(now)), appendLF=T)
51     }
52
53     if(is.null(restart) || is.na(restart) || restart<0 || restart>100){
54         c <- 0.8
55     }
56     else{
57         c <- restart
58     }
59     normalise <- match.arg(normalise)
60
61     if ("weight" %in% list.edge.attributes(ig)){
62         adjM <- get.adjacency(ig, type="both", attr="weight", edges=F, names=T, sparse=getIgraphOpt("sparsematrices"))
63         if(verbose){
64             message(sprintf("\tNotes: using weighted graph!"), appendLF=T)
65         }
66     }else{
67         adjM <- get.adjacency(ig, type="both", attr=NULL, edges=F, names=T, sparse=getIgraphOpt("sparsematrices"))
68         if(verbose){
69             message(sprintf("\tNotes: using unweighted graph!"), appendLF=T)
70         }
71     }
72
73     if(verbose){
74         now <- Sys.time()
75         message(sprintf("Normalising the adjacency matrix using %s normalisation (%s) ...", normalise, as.character(now)), appendLF=T)
76     }
77
78     A <- adjM!=0
79     if(normalise == "row"){
80         D <- Matrix::Diagonal(x=(Matrix::rowSums(A))^-1)
81         nadjM <- adjM %*% D
82     }else if(normalise == "column"){
83         D <- Matrix::Diagonal(x=(Matrix::colSums(A))^-1)
84         nadjM <- D %*% adjM
85     }else if(normalise == "laplacian"){
86         D <- Matrix::Diagonal(x=(Matrix::colSums(A))^-0.5)
87         nadjM <- D %*% adjM %*% D
88     }else{
89         nadjM <- adjM
90     }
91
92     ## A function to make elements in each steady probability vector is one column normalize
93     colNorm<- function(m){
94         #res <- t(t(m)/colSums(m))
95
96         col_sum <- apply(m, 2, sum)
97         col_sum_matrix <- matrix(rep(col_sum, nrow(m)), ncol=ncol(m), nrow=nrow(m), byrow =T)
98         res <- as.matrix(m)/col_sum_matrix
99         res[is.na(res)] <- 0
100         return(res)
101     }
102
103     if(is.null(dataSeed)){
104
105         P0matrix <- Matrix::Matrix(diag(vcount(ig)), sparse=T)
106         rownames(P0matrix) <- V(ig)$name
107         colnames(P0matrix) <- V(ig)$name
108
109     }else{
110         ## check input data
111         if(is.matrix(dataSeed) | is.data.frame(dataSeed)){

```

```

112     data <- as.matrix(dataSeed)
113 }else if(is.vector(dataSeed)){
114     data <- as.matrix(dataSeed, ncol=1)
115 }
116
117 if(is.null(rownames(dataSeed))) {
118     stop("The function must require the row names of the input dataSeed.\n")
119 }else if(any(is.na(rownames(data)))){
120     warning("dataSeed with NA as row names will be removed")
121     data <- data[!is.na(rownames(data)),]
122 }
123
124 cnames <- colnames(data)
125 if(is.null(cnames)){
126     cnames <- seq(1,ncol(data))
127 }
128
129 if (class(ig) == "igraph"){
130     ind <- match(rownames(data), V(ig)$name)
131     nodes_mapped <- V(ig)$name[ind[!is.na(ind)]]
132     if(length(nodes_mapped)==0){
133         stop("The row names of input dataSeed do not contain all those in the input graph.\n")
134     }
135     P0matrix <- matrix(0,nrow=nrow(nadjM),ncol=ncol(data))
136     P0matrix[ind[!is.na(ind)],] <- as.matrix(data[!is.na(ind),])
137
138     ## make sure the sum of elements in each steady probability vector is one
139     P0matrix <- colNorm(P0matrix)
140
141     ## Assign row and colnames
142     rownames(P0matrix) <- V(ig)$name
143     colnames(P0matrix) <- cnames
144 }
145 }
146 if(restart==1){
147     ## just seeds themselves
148     PTmatrix <- P0matrix
149 }else{
150     ##### Run in parallel
151     flag_parallel <- F
152     if(parallel==TRUE){
153
154         flag_parallel <- dCheckParallel(multicores=multicores, verbose=verbose)
155         if(flag_parallel){
156             j <- 1
157             PTmatrix <- foreach::`%dopar%` (foreach::foreach(j=1:ncol(P0matrix), .inorder=T, .combine="cbind"), {
158                 P0 <- P0matrix[,j]
159                 ## Initializing variables
160                 delta <- 1
161                 PT <- P0
162                 ## Iterative update till convergence (delta<=1e-10)
163                 while (delta>stop_delta){
164                     PX <- (1-c) * nadjM %*% PT + c * P0
165                     # p-norm of v: sum((abs(v).p)^(1/p))
166                     delta <- sum(abs(PX-PT))
167                     PT <- PX
168                 }
169                 as.matrix(PT)
170             })
171
172             PTmatrix[PTmatrix<1e-6] <- 0
173             #PTmatrix <- Matrix::Matrix(PTmatrix, sparse=T)
174         }
175     }
176     if(flag_parallel==F){

```

```

177     PTmatrix <- Matrix::Matrix(0, nrow=nrow(P0matrix), ncol=ncol(P0matrix), sparse=T)
178     for(j in 1:ncol(P0matrix)){
179         #P0 <- as.matrix(P0matrix[,j],ncol=1)
180         P0 <- P0matrix[,j]
181
182         ## Initializing variables
183         delta <- 1
184
185         PT <- P0
186         ## Iterative update till convergence (delta<=1e-10)
187         while (delta>stop_delta){
188             PX <- (1-c) * nadjM %*% PT + c * P0
189
190             # p-norm of v: sum((abs(v).p)^(1/p))
191             delta <- sum(abs(PX-PT))
192
193             PT <- PX
194             #step <- step+1
195         }
196         #PTmatrix[,j] <- as.matrix(PT, ncol=1)
197         PT[PT<1e-6] <- 0
198         PTmatrix[,j] <- Matrix::Matrix(PT, sparse=T)
199     }
200 }
201 }
202 if(verbose){
203     now <- Sys.time()
204     message(sprintf("Rescaling steady probability vector (%s) ...", as.character(now)), appendLF=T)
205 }
206 PTmatrix <- colNorm(PTmatrix)
207 PTmatrix <- Matrix::Matrix(PTmatrix, sparse=T)
208 rownames(PTmatrix) <- rownames(P0matrix)
209 colnames(PTmatrix) <- colnames(P0matrix)
210
211 endT <- Sys.time()
212 runTime <- as.numeric(difftime(strptime(endT, "%Y-%m-%d %H:%M:%S"), strptime(startT, "%Y-%m-%d %H:%M:%S"), units="secs"))
213 message(paste(c("Runtime in total is: ",runTime," secs\n"), collapse=""), appendLF=T)
214
215 invisible(PTmatrix)
216
217 }
218
219
220 #' Network based inference on Bipartite networks
221 #' @title Network Based Inference
222 #' @description Given a bipartite graph , a two phase resource transfer Information from X(x,y,z) set of nodes gets distributed to Y set o
223 #' This process allows us to define a technique for the calculation of the weight matrix W. if the similarity matrices are not provided it
224 #' bipartite graph to compute netowrk based inference .
225 #' @name nbiNet
226 #' @param A Adjacency Matrix
227 #' @param lamda Tuning parameter (value between 0 and 1) which determines how the distribution of
228 #' resources takes place in thesecond phase
229 #' @param alpha Tuning parameter (value between 0 and 1) to adjust the performance of the algorithm.
230 #' @param s1 Target Similarity matrix
231 #' @param s2 Chemical Similarity Matrix
232 #' @param format type of file as Adjacency file
233 #' @name nbiNet
234 #' @references
235 #' \itemize{
236 #' \item Cheng F, et al. Prediction of drug target interactions and drug repositioning via network-based inference. PLoS Comput. Biol. 2012
237 #' \item Zhou T, et al. Solving the apparent diversity-accuracy dilemma of recommender systems. Proc. Natl Acad. Sci. USA 2010;107:4511-451
238 #' \item Zhou T, et al. Bipartite network projection and personal recommendation. Phys. Rev. E Stat. Nonlin. Soft Matter Phys. 2007;76:0461
239 #' \item Blog post from Abhik Seal \url{http://data2quest.blogspot.com/2015/02/link-prediction-using-network-based.html}
240 #' }
241 #' @examples
242
243 #' \donttest{

```

```

243 #' data(Enzyme)
244 #' A <- t(enzyme_ADJ)
245 #' S1 = as.matrix(enzyme_Csim)
246 #' S2 = as.matrix(enzyme_Gsim)
247 #' g1 = upgrade_graph(graph.incidence(A))
248 #' ## other format available \code{format = c("igraph","matrix","pairs")}
249 #' M2 <- nbiNet(A,alpha=0.5, lamda=0.5, s1=S1, s2=S2,format = "matrix")
250 #' M3 <- nbiNet(A,alpha=0.5,lamda=0.5,format="matrix")
251 #' }
252 #' @export
253
254 ## Edit the code to include HeatS code to only predict if we have adjacency matrix
255 nbiNet <- function (A, alpha=0.5, lamda=0.5, s1=NA, s2=NA,format = c("igraph","matrix","pairs")) {
256
257   startT <- Sys.time()
258   format <- match.arg(format)
259   now <- Sys.time()
260   message(sprintf("Running computation of the input graph (%s) ...", as.character(startT)), appendLF=T)
261   if (format == "igraph"){
262     adjM = get.incidence(A)
263   }
264   else if (format == "matrix"){
265
266     adjM <- as.matrix(A)
267   }
268   else if(format == "pairs") {
269     d<- graph.data.frame(A) ## only accepts pairs file
270     V(d)$type <- V(d)$name %in% A[,1]
271     data <- get.incidence(d)
272     adjM <- transpose(data)
273   }
274   else stop ("Adjacency matrix should be 'igraph','matrix' or 'pairs' file type \n.")
275
276
277   n = nrow(adjM)
278   m = ncol(adjM)
279   if (is.na(s1) && is.na(s2)){
280
281     Ky <- diag(1/colSums(adjM))
282     Ky[is.infinite(Ky) | is.na(Ky)] <- 0
283
284     kx <- rowSums(adjM)
285     kx[is.infinite(kx) | is.na(kx)] <- 0
286     Nx <- 1/(matrix(kx, nrow=n, ncol=n, byrow=TRUE)^(lamda) *
287               matrix(kx, nrow=n, ncol=n, byrow=FALSE)^(1-lamda))
288     Nx[is.infinite(Nx) | is.na(Nx)] <- 0
289     cl <- makeCluster(detectedCores())
290     W <- suppressWarnings(t(snow::parMM(cl,adjM,Ky)))
291     W <- suppressWarnings(snow::parMM(cl, adjM, W))
292     #W <- t(adjM %% Ky)
293     W <- Nx * W
294     rownames(W) <- rownames(adjM)
295     colnames(W) <- rownames(adjM)
296     rM <- suppressWarnings(snow::parMM(cl,W,adjM))
297     endT <- Sys.time()
298     stopCluster(cl)
299     runTime <- as.numeric(difftime(strptime(endT, "%Y-%m-%d %H:%M:%S"), strptime(startT, "%Y-%m-%d %H:%M:%S"), units="secs"))
300     message(sprintf("Done computation of the input graph (%s) ...", as.character(endT)), appendLF=T)
301     message(paste(c("Runtime in total is: ",runTime," secs\n"), collapse=""), appendLF=T)
302     invisible (rM)
303
304   } else {
305
306     if (nrow(s2) != m || ncol(s2) != m) {
307       stop("The matrix s2 should be an m by m matrix with same number of columns as A.")
308     }
309   }
310 }

```

```

309     if (nrow(s1) != n || ncol(s1) != n) {
310         stop("The matrix s1 should be an n by n matrix with same number of rows as A")
311     }
312
313     Ky <- diag(1/colSums(adjM))
314     Ky[is.infinite(Ky) | is.na(Ky)] <- 0
315
316     kx <- rowSums(adjM)
317     kx[is.infinite(kx) | is.na(kx)] <- 0
318     Nx <- 1/(matrix(kx, nrow=n, ncol=n, byrow=TRUE)^(lamda) *
319         matrix(kx, nrow=n, ncol=n, byrow=FALSE)^(1-lamda))
320     Nx[is.infinite(Nx) | is.na(Nx)] <- 0
321     c1 <- makeCluster(detectCores())
322     W <- suppressWarnings(t(snow::parMM(c1,adjM,Ky)))
323     W <- suppressWarnings(snow::parMM(c1, adjM, W))
324     #W <- t(adjM %%% Ky)
325     W <- Nx * W
326     rownames(W) <- rownames(adjM)
327     colnames(W) <- rownames(adjM)
328     X5 <- suppressWarnings(snow::parMM(c1, adjM, s2))
329     X6 <- suppressWarnings(snow::parMM(c1, X5, t(adjM)))
330     X7 <- suppressWarnings(snow::parMM(c1, adjM, matrix(1, nrow=m, ncol=m)))
331     X8 <- suppressWarnings(snow::parMM(c1, X7, t(adjM)))
332     S3 <- X6 / X8
333
334     W <- W * ((alpha * s1) + ((1-alpha) * S3))
335
336     W[is.nan(W)] <- 0
337     rM <- suppressWarnings(snow::parMM(c1,W,adjM))
338
339     endT <- Sys.time()
340     stopCluster(c1)
341     runTime <- as.numeric(difftime(strptime(endT, "%Y-%m-%d %H:%M:%S"), strptime(startT, "%Y-%m-%d %H:%M:%S"), units="secs"))
342     message(sprintf("Done computation of the input graph (%s) ...", as.character(endT)), appendLF=T)
343     message(paste(c("Runtime in total is: ",runTime," secs\n"), collapse=""), appendLF=T)
344     invisible (rM)
345 }
346 }
347
348
349 #' Randommm walk with restart on Bipartite networks
350 #' @title Bipartite Random Walk
351 #' @name biNetwalk
352 #' @param g1 Bipartite graph igraph object.
353 #' @param s1 Accepts a matrix object of similarity scores for targets.
354 #' @param s2 Accepts a matrix object similarity scores for compounds.
355 #' @param normalise Normalisation of matrix using laplacian, Chen, None(the transition matrix will be column normalized)
356 #' @param dataSeed seeds file
357 #' @param restart restart value
358 #' @param weight if we want to use a weighted network . Options are either TRUE or FALSE.
359 #' @references
360 #' \itemize{
361 #' \item {Chen X, et al. Drug target interaction prediction by random walk on the heterogeneous network. Mol. Biosyst 2012;8:1970-1978.}
362 #' \item {Vanunu O, Sharan R. Proceedings of the German Conference on Bioinformatics. Germany: GI; 2008. A propagation-based algorithm for
363 #' }
364 #' @examples
365 #' \dontrun{
366 #' data(Enzyme)
367 #' A <- enzyme_ADJ
368 #' S2 = enzyme_Csim
369 #' S1 = enzyme_Gsim
370 #' g1 = graph.incidence(A)
371 #' M3 <- biNetwalk(g1,s1=S1,s2=S2,normalise="laplace", dataSeed=NULL,restart=0.8,
372
373 #' parallel=FALSE, verbose=T,weight=FALSE)
374 #' dataF<- read.csv("seedFile.csv",header=FALSE)
375 #' Mat <- biNetwalk(g1,s1=S1,s2=S2,normalise="laplace", dataSeed=dataF,restart=0.8,

```

```

375 #'           parallel=TRUE,verbose=T,weight=FALSE)
376 #' }
377 #' @export
378
379 biNetwork <- function(g1,s1,s2,normalise=c("laplace","none","chen"), dataSeed=NULL,restart=0.8,verbose=T,weight=FALSE) {
380
381   startT <- Sys.time()
382
383   if (!exists('s1') || !exists('s2')){
384     stop("You must submit s1 and s2 matrices.\n")
385   }
386
387   if (class(g1) != "igraph"){
388     stop("The function applies to 'igraph' object.\n")
389   }
390
391   if (!bipartite.mapping(g1)$res){
392     stop("The function applies to bipartite graphs.\n")
393   }
394
395   if(verbose){
396     now <- Sys.time()
397     message(sprintf("First, get the adjacency matrix of the input graph (%s) ...", as.character(now)), appendLF=T)
398   }
399   if(is.null(restart) || is.na(restart) || restart<0 || restart>100){
400     c <- 0.8
401   }
402   else{
403     c <- restart
404   }
405   normalise <- match.arg(normalise)
406   if (weight){
407     if ("weight" %in% list.edge.attributes(g1)){
408       adjM <- get.incidence(g1, attr="weight", names=T)
409       if(verbose){
410         message(sprintf("Notes: using weighted graph!"), appendLF=T)
411       }
412     }
413   }else{
414     adjM <- get.incidence(g1, attr=NULL, names=T)
415     if(verbose){
416       message(sprintf("Note: using unweighted graph!"), appendLF=T)
417     }
418   }
419   adjM <- as.matrix(adjM)
420   # get the transition matrix
421   W = tMat(adjM,s1,s2,normalise=normalise)
422   message(sprintf("got the transition matrix for RWR"))
423   if(is.null(dataSeed)){
424
425     M<-Matrix(adjM)
426     M2<-0.99*M
427     d<-Matrix(0.01*diag(nrow(s2)))
428     P0matrix<-rBind(M2,d)
429
430   }else{
431
432     # part of the section for input file name
433     drug.names <- as.character(unique(dataSeed$V2))
434     P0matrix <- matrix(0,nrow=nrow(W),ncol=length(drug.names))
435
436     for (i in 1:length(drug.names)){
437       sub.fr <- dataSeed[dataSeed$V2==drug.names[i],]
438
439       proteins <- as.character(sub.fr$V1)
440       ind1 <- match(proteins, rownames(W))
441       ind2 <- match(drug.names[i],rownames(W))

```

```

441     ind <- append(ind1,ind2)
442     nodes_mapped <- rownames(W)[ind[!is.na(ind)]]
443     if(length(nodes_mapped)!=length(ind)){
444         warning("The row names of input dataSeed do not contain all those in the input graph.\n")
445     }
446
447     P0matrix[ind[!is.na(ind)],i] <- 1
448 }
449 P0matrix <- colNorm(P0matrix)
450
451 }
452
453
454 if (exists("W")){
455     rmat <- rwr(W,P0matrix,r=c)
456 } else{
457     stop("Transition matrix couldnt be generated..")
458 }
459
460 if (!exists("rmat")){
461     stop("Couldn't return the RWR matrix. \n")
462 }else{
463     if(verbose){
464         now <- Sys.time()
465         message(sprintf("Rescaling steady probability vector (%s) ...", as.character(now)), appendLF=T)
466     }
467     rmat[rmat < 1e-06] <- 0
468     rmat <- rmat[1:nrow(adjM),]
469
470     rmat <- colNorm(as.matrix(rmat))
471     rownames(rmat)<- rownames(adjM)
472     if(!is.null(dataSeed)){
473         colnames(rmat)<- drug.names
474         invisible(rmat)
475     } else {
476         colnames(rmat)<- colnames(adjM)
477         invisible(rmat)
478     }
479     endT <- Sys.time()
480     runTime <- as.numeric(difftime(strptime(endT, "%Y-%m-%d %H:%M:%S"), strptime(startT, "%Y-%m-%d %H:%M:%S"), units="secs"))
481     message(paste(c("Runtime in total is: ",runTime," secs\n"), collapse=""), appendLF=T)
482     invisible(rmat)
483
484 }
485
486 }
487
488 #' Significance of Bipartite networks
489 #' @title Significant Network
490 #' @param data n x m Adjacency matrix of seeds or dataframe of pairs.
491 #' @param g ighrah object of bipartite incident adjacency matrix.
492 #' @param Amatrix Affinity matrix computed from biNetWalk or uNetWalk.
493 #' @param num.permutation number of permutation of Affinity matrix needed to performed.
494 #' @param adjp.cutoff pvalue cutoff 0.05
495 #' @param p.adjust.method Adjusting the pvalue by diiferent method.It uses method from the stats package.
496 #' @param parallel Using parallization either True or False
497 #' @param multicores If parallisation is set TRUE number of cores to perform parallel computaion.
498 #' @param verbose For verbose output.
499 #' @name sig.net
500 #' @examples
501 #' \donttest{
502 #' data(Enzyme)
503 #' A <- enzyme_ADJ
504 #' S1 = enzyme_Gsim
505 #' S2 = enzyme_Csim
506 #' g1 = graph.incidence(A)
507 #' Q = biNetwalk(g1,s1=S1,s2=S2,normalise="laplace", dataSeed=NULL, file=NULL,restart=0.8,verbose=T)

```



```

508 #' Z = sig.net(data=A,g=g1,Amatrix=Q,num.permutation=100,adjp.cutoff=0.01,p.adjust.method="BH",parallel=FALSE,verbose=T)
509 #' }
510 #' @export
511
512 sig.net <- function(data, g, Amatrix, num.permutation=10, adjp.cutoff=0.05, p.adjust.method=c("holm", "hochberg", "hommel", "bonferroni", "
513 {
514   library(igraph)
515   startT <- Sys.time()
516
517   #####
518   permutation <- "random"
519   p.adjust.method <- match.arg(p.adjust.method)
520   ## check input data
521   if(is.matrix(data) | is.data.frame(data)){
522
523     data <- as.matrix(data)
524     if (ncol(data)==2){
525       g.data <- graph.data.frame(data) ## only accepts pairs file
526       V(g.data)$type <- V(g.data)$name %in% data[,1]
527     } else if (ncol(data) > 2){ ## Needs check seeds can be adjacency matrix
528       data <- t(data)
529       g.data <- graph.incidence(data)
530     } else if(ncol(data) < 2){
531       stop("The input data must be matrix or dataframe with at least two columns.\n")
532     }
533
534
535
536   }else if(is.null(data)){
537     stop("The input data must be matrix.\n")
538   }
539
540   ## check input graph
541   if (class(g) != "igraph"){
542     stop("The function must apply to 'igraph' object.\n")
543   }
544
545   if (!bipartite.mapping(g.data)$res){
546     stop("The function applies to igrah bipartite graphs.\n")
547   }
548
549   ## check bipartite mapping between input seed data and graph
550   p <- bipartite.projection(g.data)
551   rnames <- V(p[[1]])$name # rownames of the seed matrix
552   cnames <- V(p[[2]])$name # colnames of the seed matrix
553
554   # Get the adjacency matrix from igraph object
555   g.incident <- get.incidence(g)
556
557   ind <- match(cnames, rownames(g.incident))
558   nodes_mapped <- rownames(Amatrix)[ind[!is.na(ind)]]
559   P0matrix <- matrix(0, nrow=length(rownames(Amatrix)),ncol=length(rnames))
560   P0matrix[ind[!is.na(ind)],] <- 1
561   rownames(P0matrix) <- rownames(Amatrix)
562   colnames(P0matrix) <- rnames
563
564   ## check mapping between input Affinity matrix and graph
565   ind1 <- match(rownames(Amatrix), rownames(g.incident))
566   ind2 <- match(colnames(Amatrix), rnames)
567   if(length(ind1[!is.na(ind1)])!=length(rownames(g.incident)) && length(ind2[!is.na(ind2)])!=length(colnames(g.incident))) {
568     stop("The function must require input Affinity matrix (Amatrix) has the same names (both columns and rows) as the input graph.\n")
569   }
570
571
572   PTmatrix <- Amatrix[ind1[!is.na(ind1)],ind2]
573   PTmatrix <- colNorm(as.matrix(PTmatrix))

```

```

574
575
576 #####
577
578 obs <- as.matrix(t(PTmatrix) %>% PTmatrix)
579 B <- num.permutation
580 if(verbose){
581   message(sprintf("Third, generate the distribution of association scores based on %d permutations on nodes respecting %s (%s)...", B
582 }
583
584
585 f <- function(){
586   pb <- txtProgressBar(min=1, max=num.permutation-1, style=3)
587   count <- 0
588   function(...) {
589     count <- count + length(list(...)) - 1
590     setTxtProgressBar(pb, count)
591     Sys.sleep(0.01)
592     flush.console()
593     c(...)
594   }
595 }
596 ##### parallel computing
597 flag_parallel <- F
598 if(parallel==TRUE){
599
600   flag_parallel <- dCheckParallel(multicores=multicores, verbose=verbose)
601   if(flag_parallel){
602     b <- 1
603     exp_b <- foreach::`%dopar%` (foreach::foreach(b=1:B, .inorder=T, .combine=f()), {
604       PT_random <- PTmatrix[sample(nrow(PTmatrix)), sample(ncol(PTmatrix))]
605       ## make sure the sum of elements in each steady probability vector is one
606       PT_random <- colNorm(as.matrix(PT_random))
607       as.matrix(t(as.matrix(PT_random)) %>% PT_random)
608     })
609   }
610 }
611
612 ## non-parallel computing
613 if(flag_parallel==F){
614   exp_b <- lapply(1:B, function(b){
615     PT_random <- PTmatrix[sample(nrow(PTmatrix)), sample(ncol(PTmatrix))]
616     ## make sure the sum of elements in each steady probability vector is one
617     PT_random <- colNorm(as.matrix(PT_random))
618     as.matrix(t(as.matrix(PT_random)) %>% PT_random)
619   })
620 }
621
622 n <- ncol(obs)
623 ## for zscore
624 exp_mean <- matrix(0, ncol=n, nrow=n)
625 exp_square <- matrix(0, ncol=n, nrow=n)
626 for(b in 1:B){
627   exp <- exp_b[[b]]
628   exp_mean <- exp_mean + exp
629   exp_square <- exp_square + exp^2
630 }
631 exp_mean <- exp_mean/B
632 exp_square <- exp_square/B
633 exp_std <- sqrt(exp_square-exp_mean^2)
634 zscore <- (obs-exp_mean)/exp_std
635 zscore[is.na(zscore)] <- 0
636 zscore[is.infinite(zscore)] <- 0
637
638 ## for pvalue
639 num <- matrix(0, ncol=n, nrow=n)

```

```

640     for(b in 1:B){
641         num <- num + (obs < exp_b[[b]])
642     }
643     pval <- num/B
644     colnames(pval) <- colnames(obs)
645     rownames(pval) <- rownames(obs)
646
647     ## for adjusted pvalue
648     adjpval <- pval
649     ## lower part
650     flag_lower <- lower.tri(pval, diag=F)
651     adjpval[flag_lower] <- stats::p.adjust(pval[flag_lower], method=p.adjust.method)
652     ## upper part
653     flag_upper <- upper.tri(pval, diag=F)
654     adjpval[flag_upper] <- stats::p.adjust(pval[flag_upper], method=p.adjust.method)
655
656     if(verbose){
657         message(sprintf("Also, construct the association graph under the cutoff %1.1e of adjusted-pvalue (%s)...", adjp.cutoff, as.character(
658     )
659     flag <- adjpval < adjp.cutoff
660     adjmatrix <- flag
661     adjmatrix[flag] <- zscore[flag]
662     cgraph <- igraph::graph.adjacency(adjmatrix, mode="undirected", weighted=T, diag=F, add.colnames=NULL, add.rownames=NA)
663
664     #####
665     endT <- Sys.time()
666     runTime <- as.numeric(difftime(strptime(endT, "%Y-%m-%d %H:%M:%S"), strptime(startT, "%Y-%m-%d %H:%M:%S"), units="secs"))
667     message(paste(c("Runtime in total is: ",runTime," secs\n"), collapse=""), appendLF=T)
668
669     result <- list(pval = pval,
670                   adjpval = adjpval,
671                   cgraph = cgraph)
672     invisible(result)
673 }
674
675 #' NetCombo
676 #' @title NetCombo
677 #' @description Performs computation three different algorithms like random walk, network based inference and heterogenous based inference a
678 #' @param g1 igraph object
679 #' @param s1 Accepts a matrix object of similarity scores for targets.
680 #' @param s2 Accepts a matrix object similarity scores for compounds.
681 #' @param nbi.alpha alpha value for network based inference.
682 #' @param nbi.lamda lamda value for network based inference.
683 #' @param norm normalization of matrices options are "laplace" or "none".
684 #' @param restart restart parameter for RWR
685 #' @param par parallel execution for RWR.
686 #' @return Matrix object with sum score values.
687 #' @name netCombo
688 #' @examples
689 #' \donttest{
690 #' data(Enzyme)
691 #' A = enzyme_ADJ
692 #' S1 = as.matrix(enzyme_Gsim)
693 #' S2 = as.matrix(enzyme_Csim)
694 #' g1 = graph.incidence(A)
695 #' P <- netCombo(g1,s1=S1,s2=S2,nbi.alpha=0.5,nbi.lamda=0.5,par=TRUE)
696 #' ## With a different restart
697 #' P <- netCombo(g1,s1=S1,s2=S2,nbi.alpha=0.5,nbi.lamda=0.5,restart=0.7,par=TRUE)
698 #' }
699 #' @export
700
701
702 netCombo <- function(g1,s1,s2,nbi.alpha=0.4,nbi.lamda=0.5,norm="laplace",restart=0.8,par=TRUE) {
703
704     startT <- Sys.time()
705     now <- Sys.time()

```

```

706   if (!exists('s1') || !exists('s2')){
707     stop("You must submit s1 and s2 matrices.\n")
708   }
709
710   if (class(g1) != "igraph"){
711     stop("The function must apply to either 'igraph' object.\n")
712   }
713   if (!bipartite.mapping(g1)$res){
714     stop("The function applies to bipartite graphs only.\n")
715   }
716
717   A <- as.matrix(get.incidence(g1))
718   message(sprintf("Running computation of the input graph (%s) ...", as.character(startT)), appendLF=T)
719   message(sprintf("Running computation for RWR..\n"))
720   Q1 = biNetwalk(g1,s1=s1,s2=s2,normalise="laplace",verbose=T,restart = restart)
721
722
723   message(sprintf("Running computation for network based inference..\n"))
724   Q2 = nbiNet(A,lambda=nbi.lambda,alpha=nbi.alpha,s1=as.matrix(s1),s2=as.matrix(s2),format = "matrix")
725
726   if (exists("Q1") && exists("Q2")){
727     M <- (Q1+Q2)/2
728     return (M)
729   }
730
731 }
732
733 #' get the performance of the link Prediction algorithms.
734 #' @title Link Prediction Performance
735 #' @description This function samples links and removes links from the adjacency matrix and predicts them and calculates area under accum
736 #' @param S1 Sequence similarity matrix object
737 #' @param A Drug target association matrix
738 #' @param S2 Accepts a matrix object similarity scores for compounds.
739 #' @param relinks Number of links to remove randomly from the input matrix.
740 #' @param numT Frequency of the number of targets.
741 #' @param restart restart value if using rwr or netcombo
742 #' @param alpha alpha value if using nbi or netcombo
743 #' @param lambda lambda value if using nbi or netcombo
744 #' @param Calgo Algorithm to use for Bipartite link prediction options are "rwr","nbi" & "netcombo".
745 #' @param norm normalization of matrices options are "laplace" or "none".
746 #' @name net.perf
747 #' @return it returns a list of aucc, auc, bedorc, enrichment factor and auc (top 10%)
748 #' \itemize{
749 #'   \item {Truchon et al. Evaluating Virtual Screening Methods: Good and Bad Metrics for the "Early Recognition" Problem. J. Chem. Inf. Mo
750 #'   \item {Sheridan RP et al. Protocols for bridging the peptide to nonpeptide gap in topological similarity searches. J. Chem. Inf. Compu
751 #' }
752 #' @examples
753 #' \dontrun{
754 #' data(Enzyme)
755 #' A = enzyme_ADJ
756 #' S1 = enzyme_Gsim
757 #' S2= enzyme_Csim
758 #' m = net.perf(A,S1,S2,alpha=0.5,lambda=0.5,relinks = 50,numT=2,norm="laplace",Calgo="nbi")
759 #' }
760 #' @export
761
762 net.perf<- function(A,S1,S2,restart=0.8,alpha=0.5,lambda=0.5,relinks=100,numT=2,norm="laplace",Calgo = c("rwr","nbi","netcombo","all")){
763
764   auctop = numeric()
765   aucc = numeric()
766   bdr = numeric()
767   efc = numeric()
768   ranks = numeric()
769   totallinks = sum(A)
770
771   m = dim(A)[1] ## rows for targets

```

```

772     n = dim(A)[2] ## columns for drugs
773
774     if (!exists('S1') || !exists('S2')){
775         stop("You must submit s1 and s2 matrices.\n")
776     }
777
778     if (nrow(S1)!=m | ncol(S1) != m){
779         stop("Your number of targets does not match with target similarity matrix.\n")
780     }
781
782     if (nrow(S2)!=n | ncol(S2) != n){
783         stop("Your number of targets does not match with target similarity matrix.\n")
784     }
785
786
787     ## Get the name of the algorithm.
788     algo <- match.arg(Calgo)
789     g1 <- graph.incidence(A)
790     eg <- get.edgelist(g1)
791     c <- data.frame(table(eg[,2]))
792     c <- c[c$Freq>numT,]
793
794     drugnames <- unique(as.character(c$Var1))
795
796     ids <- which(eg[,2] %in% drugnames)
797     re <- eg[sample(ids,size = relinks,replace=FALSE),]
798
799
800     if (totallinks <= relinks){
801         stop("Total links removed is less than equal given links to be removes. Give a sensible value.")
802     }
803
804     SampledGraph <- g1
805     for (i in 1:dim(re)[1])
806     {
807         if (are.connected(SampledGraph, re[i,1], re[i,2]))
808             SampledGraph <- delete.edges(SampledGraph, E(SampledGraph, P=c(re[i,1], re[i,2])))
809     }
810     g1 = SampledGraph
811     Sg_t <- get.incidence(SampledGraph)
812
813     #Sg_t <- randomizeMatrix(Sg_t,null.model = "frequency",iterations = 1000)
814
815     #mat<-tMat(Sg_t,as.matrix(S1),as.matrix(S2),normalise="laplace")
816
817     drugs <- re[,2]
818
819     message(sprintf("Detected (%s) drugs & (%s) proteins with (%s) interactions...",n,m,totallinks))
820     message(sprintf("Running prediction for (%s) links removed using (%s) .. ",as.character(relinks),as.character(algo)))
821
822     performances <- function(predictR,m,re){
823
824         s1<-predictR[1:m,]
825         s1<- scale(s1, center=FALSE, scale=colSums(s1,na.rm=TRUE))
826         s1[is.na(s1)] <- 0
827         test <- data.frame(re)
828         for (dis in 1:dim(s1)[2]) {
829
830             drugname = colnames(s1)[dis]
831             subfr <- test[test$X2==drugname,]
832
833             pname<-as.character(subfr$X1)
834             id = which(rownames(s1) %in% pname)
835             clabel <- rep(0,m)
836             clabel[id] <- 1
837             res = cbind(s1[,dis],clabel)
838             colnames(res)[1] <- "score"

```

```

838         d <- res[order(-res[,1]),]
839         ac <- auac(d[,1], d[,2])
840         au <- auc(d[,1], d[,2])
841         at <- auc(d[,1], d[,2],top=0.1)
842         bd <- bedroc(d[,1], d[,2])
843         ef <- enrichment_factor(d[,1], d[,2],top=0.1)
844         aucc <- c(aucc, ac)
845         bdr <- c(bdr,bd)
846         efc <- c(efc,ef)
847         auctop <- c(auctop,at)
848     }
849
850 }
851
852 scores = c(list(auac = mean(aucc),auc= mean(au),auctop = mean(auctop),bdr = mean(bdr),efc = mean(efc)))
853 return (scores)
854 }
855
856 if (algo == "rwr"){
857     #par="True"
858     message(sprintf("Running RWR Algorithm"))
859     mat = biNetwalk(g1,s1=S1,s2=S2,restart=restart,normalise=norm,verbose=T)
860     predictR <- mat[,colnames(mat) %in% drugs]
861     scores <- performances(predictR,m,re)
862     return (scores)
863 }
864 else if (algo == "nbi"){
865     message(sprintf("Running NBI Algorithm"))
866     #S1 = S1[rownames(S1) %in% rownames(N_M),colnames(S1) %in% rownames(N_M)]
867     #S2 = S2[rownames(S2) %in% colnames(N_M),colnames(S2) %in% colnames(N_M)]
868     mat <- nbiNet(Sg_t, lamda=lamda, alpha=alpha, s1=as.matrix(S1), s2=as.matrix(S2),format = "matrix")
869     predictR <- mat[,colnames(mat) %in% drugs]
870     scores <- performances(predictR,m,re)
871     return (scores)
872 }
873
874 else if(algo == "netcombo"){
875     message(sprintf("Running NetCombo Algorithm"))
876     #par="True"
877     mat1 = biNetwalk(g1,s1=S1,s2=S2,normalise=norm,verbose=T,restart=restart)
878     mat2 <- nbiNet(Sg_t, lamda=lamda, alpha=alpha, s1=as.matrix(S1), s2=as.matrix(S2),format = "matrix")
879     mat = (mat1+mat2)/2
880     predictR <- mat[,colnames(mat) %in% drugs]
881     scores <- performances(predictR,m,re)
882     return (scores)
883 } else if (algo == "all"){
884
885     message(sprintf("Running all the algorithms ..."))
886     #par="True"
887     mat1 <- biNetwalk(g1,s1=S1,s2=S2,normalise=norm,verbose=T)
888     mat2 <- nbiNet(Sg_t, lamda=0.5, alpha=0.5, s1=as.matrix(S1), s2=as.matrix(S2),format = "matrix")
889     mat3 <- (mat1+mat2)/2
890     predictR1 <- mat1[,colnames(mat1) %in% drugs]
891     predictR2 <- mat2[,colnames(mat2) %in% drugs]
892     predictR3 <- mat3[,colnames(mat3) %in% drugs]
893
894     scores1 <- performances(predictR1,m,re)
895     scores2 <- performances(predictR2,m,re)
896     scores3 <- performances(predictR3,m,re)
897
898     list1 = list(type = 'rwr',score=scores1)
899     list2 = list(type = 'nbi',score=scores2)
900     list3 = list(type = 'netcombo',score=scores3)
901     scoreList = list(list1,list2,list3)
902     return (scoreList)
903 }

```

```

904     }
905 }
906
907 #' Get top predicted results.
908 #' @title Get Top Results
909 #' @description The function returns the given top number of predicted results along with true interactions.
910 #' @param A Drug target association matrix.
911 #' @param P Drug target predicted matrix.
912 #' @param top top number of predicted targets.
913 #' @param druglist It accepts a vector of drugnames for which results will return
914 #' @name getTopresults
915 #' @return it returns a list of aucc, auc, bedorc, enrichment factor and auc (top 10%)
916 #' @examples
917 #' \donttest{
918 #' data(Enzyme)
919 #' A = enzyme_ADJ
920 #' S1 = enzyme_Gsim
921 #' S2= enzyme_Csim
922 #' ## Running the netcombo algorithm.
923 #' P <- netCombo(g1,s1=S1,s2=S2,nbi.alpha=0.5,nbi.lamda=0.5,par=TRUE)
924 #' result = getTopresults(A,P,top=10,druglist=NULL)
925 #' ## Getting result from a drug list.
926 #' drugs = c("D00014", "D00018", "D00029", "D00036", "D00045", "D00049")
927 #' result = getTopresults(A,P,top=10,druglist=drugs)
928 #' }
929 #' @export
930
931 getTopresults <- function(A,P,top=10,druglist=NULL){
932
933   startT <- Sys.time()
934   now <- Sys.time()
935
936   `%not in%` <- function (x, table) is.na(match(x, table, nomatch=NA_integer_))
937
938   A <- A[,colnames(A) %in% colnames(P)]
939
940   if (length(rownames(A)) <= 0 ){
941     stop("Drugs names doesnt match for Predicted matrix and Original Matrix")
942   }
943
944   g1 <- graph.incidence(A)
945
946   e1 = data.frame(get.edgelist(g1))
947
948   if (is.null(druglist)){
949     drugnames = colnames(P)
950     fr <- data.frame()
951     for (i in 1:length(drugnames)){
952       lt = e1[e1$X2==drugnames[i],]
953       tproteins = as.character(lt$X1)
954       if (length(tproteins) > 0 ){
955         d <- P[order(-P[,i]),]
956         pnames = rownames(d)
957         score <- as.numeric(d[,i])
958         drug <- drugnames[i]
959         result <- data.frame(cbind(drug,pnames,score))
960         tp <- result[result$pnames %in% tproteins,]
961         tp$type <- "True Interactions"
962         pi = result[result$pnames %not in% tproteins,]
963
964         pi = pi[1:top,]
965         pi$type = "Predicted Interactions"
966         r <- rbind(tp,pi)
967         fr <- rbind(fr,r)
968       }
969     }
970   }

```

```
970 }
971
972
973 else {
974     drugnames = druglist
975     fr <- data.frame()
976     for (i in 1:length(drugnames)){
977
978         lt = e1[e1$X2==drugnames[i],]
979         tproteins = as.character(lt$X1)
980         if (length(tproteins) > 0 ){
981             d <- P[order(-P[,colnames(P) %in% drugnames[i]]),]
982             pnames = rownames(d)
983             score <- as.numeric(d[,colnames(P) %in% drugnames[i]])
984             drug <- drugnames[i]
985             result <- data.frame(cbind(drug,pnames,score))
986             tp <- result[result$pnames %in% tproteins,]
987             tp$type <- "True Interactions"
988             pi = result[result$pnames %not in% tproteins,]
989             pi = pi[1:top,]
990             pi$type = "Predicted Interactions"
991             r <- rbind(tp,pi)
992             fr <- rbind(fr,r)
993         }
994     }
995 }
996
997 invisible(fr)
998 }
```