Package 'SynExtend'

October 17, 2020

Type Package

Title Tools for Working With Synteny Objects

Version 1.0.0

biocViews Genetics, Clustering, ComparativeGenomics, DataImport

Description Shared order between genomic sequences provide a great deal of information. Synteny objects produced by the R package DECIPHER provides quantitative information about that shared order. SynExtend provides tools for extracting information from Synteny objects.

Depends R (>= 4.0.0), DECIPHER (>= 2.14.0), igraph (>= 1.2.4.1)

Imports methods, Biostrings, S4Vectors, IRanges, utils, stats

Suggests knitr

License GPL-3

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/SynExtend

git_branch RELEASE_3_11

git_last_commit 19f4e72

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

Author Nicholas Cooley [aut, cre] (<https://orcid.org/0000-0002-6029-304X>),
Adelle Fernando [ctb],
Erik Wright [aut]

Maintainer Nicholas Cooley <npc19@pitt.edu>

R topics documented:

`ExactSelect` *Model for identifying erroneously linked pairs*

Description

Though the function PairSummaries provides an argument allowing users to ask for alignments, given the time consuming nature of that process on large datasets, models are provided which allow for the quick and efficient identification of pairs whose PID would likely fall within a random distribution of PIDs.

Usage

```
data("ExactSelect")
```

Format

The format is: List of 21 $ coefficients : Named num [1:32] 0.662363 46.993855 -2.187871 -0.000554 -4.981938 ... ..- attr(*, "names")= chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ R : num [1:32, 1:32] -63.5 0 0 0 0 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ rank : int 32 $ qr :List of 4 ..$ rank : int 32 ..$ qraux: num [1:32] 1 1 1 1 1 ... ..$ pivot: int [1:32] 1 2 3 4 5 6 7 8 9 10 ... ..$ tol : num 1e-11 ..- attr(*, "class")= chr "qr" $ family :List of 7 ..$ family : chr "binomial" ..$ link : chr "logit" ..$ linkfun :function (mu) ..$ linkinv :function (eta) ..$ mu.eta :function (eta) ..$ initialize: expression(if (NCOL(y) == 1) if (is.factor(y)) y <- y != levels(y)[1L] n <- rep.int(1, nobs) y[weights =| __truncated__ ..$ valideta :function (eta) ..- attr(*, "class")= chr "family" $ deviance : num 31409 $ aic : num 31473 $ null.deviance: num 72189 $ iter : int 9 $ df.residual : int 196744 $ df.null : int 196775 $ converged : logi TRUE $ boundary : logi FALSE $ call : language glm(formula = as.formula(ExactModels[[1]][[31]]), family = binomial(link = "logit"), data = TrainingTable02) $ formula :Class 'formula' language ExactCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop $ terms :Classes 'terms', 'formula' language ExactCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop .. ..- attr(*, "variables")= language list(ExactCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "factors")= int [1:6, 1:31] 0 1 0 0 0 0 0 0 0 1 0 ... .- attr(*, "dimnames")=List of 2 .. ..$: chr [1:6] "ExactCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "term.labels")= chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "order")= int [1:31] 1 1 1 1 1 2 2 2 2 2 ... .- attr(*, "intercept")= int 1 .. ..- attr(*, "response")= int 1 .. ..- attr(*, "predvars")= language list(ExactCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "dataClasses")= Named chr [1:6] "numeric" "numeric" "numeric" "numeric" ... .- attr(*, "names")= chr [1:6] "ExactCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ offset : NULL $ control :List of 3 ..$ epsilon: num 1e-08 ..$ maxit : num 25 ..$ trace : logi FALSE $ method : chr "glm.fit" $ contrasts : NULL $ xlevels : Named list() - attr(*, "class")= chr [1:2] "glm" "lm"

Details

A model for rejecting identified pairs whose link statistics indicate a likely exact PID that would fall within a random distribution in an amino acid alignment.

Examples

```
data(ExactSelect)
```

`gffToDataFrame` *Generate a DataFrame of gene calls from a gff3 file*

Description

Generate a DataFrame of gene calls from a gff3 file

Usage

```
gffToDataFrame(GFF,
               AdditionalAttrs = NULL,
               AdditionalTypes = NULL,
               RawTableOnly = FALSE,
               Verbose = FALSE)
```

Arguments

`GFF`	A url or filepath specifying a gff3 file to import
`AdditionalAttrs`	A vector of character strings to designate the attributes to pull. Default Attributes include: "ID", "Parent", "Name", "gbkey", "gene", "product", "protein_id", "gene_biotype", and "Note".
`AdditionalTypes`	A vector of character strings to query from the the "Types" column. Default types are limited to "Gene" and "Pseudogene", but any possible entry for "Type" in a gff3 format can be added, such as "rRNA", or "CRISPR_REPEAT".
`RawTableOnly`	Logical specifying whether to return the raw imported GFF without complex parsing. Remains as a holdover from function construction and debugging. For simple gff3 import see `rtracklayer::import`.
`Verbose`	Logical specifying whether to print a progress bar and time difference.

Details

Import a gff file into a rectangular parsable object.

Value

A DataFrame with relevant information extracted from a GFF.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

Examples

```
ImportedGFF <- gffToDataFrame(GFF = system.file("extdata",
                                                "GCA_006740685.1_ASM674068v1_genomic.gff.gz",
                                                     package = "SynExtend"),
                              Verbose = TRUE)
```

`GlobalSelect` *Model for identifying erroneously linked pairs*

Description

Though the function PairSummaries provides an argument allowing users to ask for alignments, given the time consuming nature of that process on large datasets, models are provided which allow for the quick and efficient identification of pairs whose PID would likely fall within a random distribution of PIDs.

Usage

```
data("GlobalSelect")
```

Format

The format is: List of 21 $ coefficients : Named num [1:32] 4.67 3.07e+01 -1.14e+01 -2.84e-04 -1.91e+01 ... ..- attr(*, "names")= chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ R : num [1:32, 1:32] -40.7 0 0 0 0 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ rank : int 32 $ qr :List of 4 ..$ rank : int 32 ..$ qraux: num [1:32] 1 1 1 1 1 ... ..$ pivot: int [1:32] 1 2 3 4 5 6 7 8 9 10 ... ..$ tol : num 1e-11 ..- attr(*, "class")= chr "qr" $ family :List of 7 ..$ family : chr "binomial" ..$ link : chr "logit" ..$ linkfun :function (mu) ..$ linkinv :function (eta) ..$ mu.eta :function (eta) ..$ initialize: expression(if (NCOL(y) == 1) if (is.factor(y)) y <- y != levels(y)[1L] n <- rep.int(1, nobs) y[weights =| __truncated__ ..$ valideta :function (eta) ..- attr(*, "class")= chr "family" $ deviance : num 13600 $ aic : num 13664 $ null.deviance: num 56107 $ iter : int 9 $ df.residual : int 196744 $ df.null : int 196775 $ converged : logi TRUE $ boundary : logi FALSE $ call : language glm(formula = as.formula(GlobalModels[[1]][[31]]), family = binomial(link = "logit"), data = TrainingTable02) $ formula :Class 'formula' language GlobalCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop $ terms :Classes 'terms', 'formula' language GlobalCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop .. ..- attr(*, "variables")= language list(GlobalCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "factors")= int [1:6, 1:31] 0 1 0 0 0 0 0 0 1 0 ... .- attr(*, "dimnames")=List of 2 .. ..$: chr [1:6] "GlobalCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "term.labels")= chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "order")= int [1:31] 1 1 1 1 1 2 2 2 2 2 ... .- attr(*, "intercept")= int 1 .. ..- attr(*, "response")= int 1 .. ..- attr(*, "predvars")= language list(GlobalCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "dataClasses")= Named chr [1:6] "numeric" "numeric" "numeric" "numeric" ... .- attr(*, "names")= chr [1:6] "GlobalCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ offset : NULL $ control :List of 3 ..$ epsilon: num 1e-08 ..$ maxit : num 25 ..$ trace : logi FALSE $ method : chr "glm.fit" $ contrasts : NULL $ xlevels : Named list() - attr(*, "class")= chr [1:2] "glm" "lm"

Details

A model for rejecting identified pairs whose link statistics indicate a likely global PID that would fall within a random distribution in an amino acid alignment.

Examples

```
data(GlobalSelect)
```

`LinkedPairs` *Tables of where syntenic hits link pairs of genes*

Description

Syntenic blocks describe where order is shared between two sequences. These blocks are made up of exact match hits. These hits can be overlayed on the locations of sequence features to clearly illustrate where exact sequence similarity is shared between pairs of sequence features.

Usage

```
## S3 method for class 'LinkedPairs'
print(x,
      quote = FALSE,
      right = TRUE,
      ...)
```

Arguments

`x`	An object of class `LinkedPairs`.
`quote`	Logical indicating whether to print the output surrounded by quotes.
`right`	Logical specifying whether to right align strings.
`...`	Other arguments for `print`.

Details

Objects of class `LinkedPairs` are stored as square matrices of list elements with `dimnames` derived from the `dimnames` of the object of class "Synteny" from which it was created. The diagonal of the matrix is only filled if `OutputFormat` "Comprehensive" is selected in `NucleotideOverlap`, in which case it will be filled with the gene locations supplied to `GeneCalls`. The upper triangle is always filled, and contains location information in nucleotide space for all syntenic hits that link features between sequences in the form of an integer matrix with named columns. "QueryGene" and "SubjectGene" correspond to the integer rownames of the supplied gene calls. "QueryIndex" and "SubjectIndex" correspond to "Index1" and "Index2" columns of the source synteny object position. Remaining columns describe the exact positioning and size of extracted hits. The lower triangle is not filled if `OutputFormat` "Sparse" is selected and contains relative displacement positions for the 'left-most' and 'right-most' hit involved in linking the particular features indicated in the related line up the corresponding position in the upper triangle.

The object serves only as a simple package for input data to the `PairSummaries` function, and as such may not be entirely user friendly. However it has been left exposed to the user should they find this data interesting.

Value

An object of class "LinkedPairs".

Author(s)

Nicholas Cooley <npc19@pitt.edu>

`LocalSelect` *Model for identifying erroneously linked pairs*

Description

Though the function PairSummaries provides an argument allowing users to ask for alignments, given the time consuming nature of that process on large datasets, models are provided which allow for the quick and efficient identification of pairs whose PID would likely fall within a random distribution of PIDs.

Usage

```
data("LocalSelect")
```

Format

The format is: List of 21 $ coefficients : Named num [1:32] 1.2898 65.8948 -22.4148 -0.0031 -6.7246 ... ..- attr(*, "names")= chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ R : num [1:32, 1:32] -61 0 0 0 0 ... ..- attr(*, "dimnames")=List of 2 .. ..$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:32] "(Intercept)" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ rank : int 32 $ qr :List of 4 ..$ rank : int 32 ..$ qraux: num [1:32] 1 1 1 1 1 ... ..$ pivot: int [1:32] 1 2 3 4 5 6 7 8 9 10 ... ..$ tol : num 1e-11 ..- attr(*, "class")= chr "qr" $ family :List of 7 ..$ family : chr "binomial" ..$ link : chr "logit" ..$ linkfun :function (mu) ..$ linkinv :function (eta) ..$ mu.eta :function (eta) ..$ initialize: expression(if (NCOL(y) == 1) if (is.factor(y)) y <- y != levels(y)[1L] n <- rep.int(1, nobs) y[weights =| __truncated__ ..$ valideta :function (eta) ..- attr(*, "class")= chr "family" $ deviance : num 29193 $ aic : num 29257 $ null.deviance: num 73067 $ iter : int 9 $ df.residual : int 196744 $ df.null : int 196775 $ converged : logi TRUE $ boundary : logi FALSE $ call : language glm(formula = as.formula(LocalModels[[1]][[31]]), family = binomial(link = "logit"), data = TrainingTable02) $ formula :Class 'formula' language LocalCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop $ terms :Classes 'terms', 'formula' language LocalCode ~ MaxCoverage * MinCoverage * ExactMatch * NormDeltaStart * NormDeltaStop .. ..- attr(*, "variables")= language list(LocalCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "factors")= int [1:6, 1:31] 0 1 0 0 0 0 0 0 0 1 0 ... .- attr(*, "dimnames")=List of 2 .. ..$: chr [1:6] "LocalCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... .$: chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "term.labels")= chr [1:31] "MaxCoverage" "MinCoverage" "ExactMatch" "NormDeltaStart" ... .- attr(*, "order")= int [1:31] 1 1 1 1 1 2 2 2 2 2 ... .- attr(*, "intercept")= int 1 .. ..- attr(*, "response")= int 1 .. ..- attr(*, "predvars")= language list(LocalCode, MaxCoverage, MinCoverage, ExactMatch, NormDeltaStart, NormDeltaStop) .. ..- attr(*, "dataClasses")= Named chr [1:6] "numeric" "numeric" "numeric" "numeric" ... .- attr(*, "names")= chr [1:6] "LocalCode" "MaxCoverage" "MinCoverage" "ExactMatch" ... $ offset : NULL $ control :List of 3 ..$ epsilon: num 1e-08 ..$ maxit : num 25 ..$ trace : logi FALSE $ method : chr "glm.fit" $ contrasts : NULL $ xlevels : Named list() - attr(*, "class")= chr [1:2] "glm" "lm"

Details

A model for rejecting identified pairs whose link statistics indicate a likely local PID that would fall within a random distribution in an amino acid alignment.

Examples

```
data(LocalSelect)
```

NucleotideOverlap *Tabulating Pairs of Genomic Sequences*

Description

A function for concisely tabulating where genomic features are connected by syntenic hits.

Usage

```
NucleotideOverlap(SyntenyObject,
                  GeneCalls,
                  LimitIndex = FALSE,
                  OutputFormat = "Normal",
                  Verbose = FALSE)
```

Arguments

SyntenyObject	An object of class "Synteny" built from the function "FindSyntey" in the package "DECIPHER".
GeneCalls	A list of DataFrames that describe feature calls for the sequences represented in SyntenyObject. This list must be named, and these names should match the identifiers of the associated sequences in the DECIPHER sqlite database used to build the associated synteny object. Dataframes built by "gffToDataFrame" can be used directly, while "GRanges" objects may also be used with limited functionality. Using a "GRanges" object will force LimitIndex to FALSE. Additionally, genomic features that extend past the terminal position of the origin FASTA sequence are truncated and only features of type "gene" and "pseudogene" will be included in the analysis.
LimitIndex	Logical indicating whether to limit which indices in a synteny object to query. FALSE by default, when TRUE only the first sequence in all selected identifiers will be used. LimitIndex can be used to skip analysis of plasmids, or solely query a single chromosome.
OutputFormat	Character string to designate how much information to return. "Sparse" returns only a filled upper triangle of exactly matched positions. "Normal" returns a matrix with associated match information in both the upper and lower triangle of the returned matrix, while "Comprehensive" will return GeneCalls used in construction in the diagonal.
Verbose	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

Details

Builds a matrix of lists that contain information about linked pairs of genomic features.

Value

An object of class "LinkedPairs".

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class

Examples

```
DBPATH <- system.file("extdata",
                      "VignetteSeqs.sqlite",
                      package = "SynExtend")

# Alternatively, to build a database using DECIPHER:
# DBPATH <- tempfile()
# FNAs <- c("ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/006/740/685/GCA_006740685.1_ASM674068v1/GCA_0067406
#       "ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/956/175/GCA_000956175.1_ASM95617v1/GCA_000956175.1
#       "ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/875/775/GCA_000875775.1_ASM87577v1/GCA_000875775.1
# for (m1 in seq_along(FNAs)) {
#  X <- readDNAStringSet(filepath = FNAs[m1])
#  X <- X[order(width(X),
#               decreasing = TRUE)]
#
#  Seqs2DB(seqs = X,
#          type = "XStringSet",
#          dbFile = DBPATH,
#          identifier = as.character(m1),
#          verbose = TRUE)
# }

Syn <- FindSynteny(dbFile = DBPATH)

GeneCalls <- vector(mode = "list",
                    length = ncol(Syn))

GeneCalls[[1L]] <- gffToDataFrame(GFF = system.file("extdata",
                                          "GCA_006740685.1_ASM674068v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)
GeneCalls[[2L]] <- gffToDataFrame(GFF = system.file("extdata",
                                           "GCA_000956175.1_ASM95617v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)
GeneCalls[[3L]] <- gffToDataFrame(GFF = system.file("extdata",
                                           "GCA_000875775.1_ASM87577v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)

# Alternatively:
# GeneCalls <- vector(mode = "list",
#                     length = ncol(Syn))
# GeneCalls[[1L]] <- rtracklayer::import(system.file("extdata",
#                                           "GCA_006740685.1_ASM674068v1_genomic.gff.gz",
#                                                   package = "SynExtend"))
# GeneCalls[[2L]] <- rtracklayer::import(system.file("extdata",
#                                           "GCA_000956175.1_ASM95617v1_genomic.gff.gz",
```

```
#                                                    package = "SynExtend"))
# GeneCalls[[3L]] <- rtracklayer::import(system.file("extdata",
#                                          "GCA_000875775.1_ASM87577v1_genomic.gff.gz,
#                                                    package = "SynExtend"))

names(GeneCalls) <- seq(length(GeneCalls))

Links <- NucleotideOverlap(SyntenyObject = Syn,
                           GeneCalls = GeneCalls,
                           LimitIndex = FALSE,
                           Verbose = TRUE)
```

`PairSummaries` *Summarize connected pairs in a LinkedPairs object*

Description

Takes in a LinkedPairs object and gene calls, and returns a pairs list.

Usage

```
PairSummaries(SyntenyLinks,
              GeneCalls,
              DBPATH,
              PIDs = TRUE,
              IgnoreDefaultStringSet = FALSE,
              Verbose = TRUE,
              GapPenalty = TRUE,
              TerminalPenalty = TRUE,
              Model = "Global",
              Correction = "none")
```

Arguments

`SyntenyLinks`	A `PairedLinks` object.
`GeneCalls`	A list of named DataFrames or GRanges objects. Dataframes built by "gffToDataFrame" can be used directly, while "GRanges" objects may also be used with limited functionality. Using a "GRanges" object will force all alignments to nucleotide alignments.
`DBPATH`	A SQLite connection object or a character string specifying the path to the database file. Constructed from DECIPHER's `Seqs2DB` function.
`PIDs`	Logical indicating whether to perform pairwise alignments. If `TRUE` (the default) all pairs will be aligned using DECIPHER's `AlignSeqs`, or `AlignTranslation` function. This step can be time consuming, especially for large numbers of pairs.
`IgnoreDefaultStringSet`	Logical indicating alignment type preferences. If `FALSE` (the default) pairs that can be aligned in amino acid space will be aligned as an `AAStringSet`. If `TRUE` all pairs will be aligned in nucleotide space.
`Verbose`	Logical indicating whether or not to display a progress bar and print the time difference upon completion.

GapPenalty	Argument passed to AlignTranslation
TerminalPenalty	Argument passed to AlignTranslation
Model	A character string specifying a model to use to identify pairs that are unlikely to be good orthologs. By default this is "Global", but two other models are included; "Local" and "Exact", which have minor differences in performance. Alternatively, a user generated model can be used.
Correction	Argument to be passed to DistanceMatrix, currently only "none" and "Jukes-Cantor" are supported options. Will only be applied to nucleotide alignments.

Details

The LinkedPairs object generated by NucleotideOverlap is a container for raw data that describes possible orthologous relationships, however ultimate assignment of orthology is up to user discretion. PairSummaries generates a clear table with relevant statistics for a user to work with as they choose. The option to align all pairs, though onerous can allow users to apply a hard threshold to predictions by PID, while built in models can allow a more succinct and expedient thresholding.

Value

A data.frame with rownames indicating orthologous pairs.

Author(s)

Nicholas Cooley <npc19@pitt.edu>

See Also

FindSynteny, Synteny-class

Examples

```
DBPATH <- system.file("extdata",
                      "VignetteSeqs.sqlite",
                      package = "SynExtend")

# Alternatively, to build a database using DECIPHER:
# DBPATH <- tempfile()
# FNAs <- c("ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/006/740/685/GCA_006740685.1_ASM674068v1/GCA_0067406
#       "ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/956/175/GCA_000956175.1_ASM95617v1/GCA_000956175.1
#       "ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/875/775/GCA_000875775.1_ASM87577v1/GCA_000875775.1
# for (m1 in seq_along(FNAs)) {
#  X <- readDNAStringSet(filepath = FNAs[m1])
#  X <- X[order(width(X),
#               decreasing = TRUE)]
#
#  Seqs2DB(seqs = X,
#          type = "XStringSet",
#          dbFile = DBPATH,
#          identifier = as.character(m1),
#          verbose = TRUE)
#}

Syn <- FindSynteny(dbFile = DBPATH)
```

```
GeneCalls <- vector(mode = "list",
                    length = ncol(Syn))

GeneCalls[[1L]] <- gffToDataFrame(GFF = system.file("extdata",
                                          "GCA_006740685.1_ASM674068v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)
GeneCalls[[2L]] <- gffToDataFrame(GFF = system.file("extdata",
                                           "GCA_000956175.1_ASM95617v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)
GeneCalls[[3L]] <- gffToDataFrame(GFF = system.file("extdata",
                                           "GCA_000875775.1_ASM87577v1_genomic.gff.gz",
                                                  package = "SynExtend"),
                                  Verbose = TRUE)

# Alternatively:
# GeneCalls <- vector(mode = "list",
#                     length = ncol(Syn))
# GeneCalls[[1L]] <- rtracklayer::import(system.file("extdata",
#                                          "GCA_006740685.1_ASM674068v1_genomic.gff.gz",
#                                                   package = "SynExtend"))
# GeneCalls[[2L]] <- rtracklayer::import(system.file("extdata",
#                                           "GCA_000956175.1_ASM95617v1_genomic.gff.gz",
#                                                   package = "SynExtend"))
# GeneCalls[[3L]] <- rtracklayer::import(system.file("extdata",
#                                            "GCA_000875775.1_ASM87577v1_genomic.gff.gz,
#                                                   package = "SynExtend"))

names(GeneCalls) <- seq(length(GeneCalls))

Links <- NucleotideOverlap(SyntenyObject = Syn,
                           GeneCalls = GeneCalls,
                           LimitIndex = FALSE,
                           Verbose = TRUE)

PredictedPairs <- PairSummaries(SyntenyLinks = Links,
                                GeneCalls = GeneCalls,
                                DBPATH = DBPATH,
                                PIDs = FALSE,
                                Verbose = TRUE,
                                Model = "Global",
                                Correction = "none")
```

Index