Package 'micropan'

February 15, 2018

Type Package

Title Microbial Pan-Genome Analysis

Version 1.2

Date 2018-02-15

Author Lars Snipen and Kristian Hovde Liland

Maintainer Lars Snipen <lars.snipen@nmbu.no>

Description A collection of functions for computations and visualizations of microbial pan-genomes.

License GPL-2

Depends R (>= 3.3.0), igraph, microseq

Imports Rcpp (>= 0.12.0)

LinkingTo Rcpp (>= 0.12.0), BH

LazyData FALSE

ZipData TRUE

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-02-15 11:22:41 UTC

R topics documented:

`barrnap` *Finding rRNA genes*

Description

Locating all rRNA genes in genomic DNA using the barrnap software.

Usage

```
barrnap(genome.file, bacteria = TRUE, cpu = 1)
```

Arguments

genome.file	A fasta-formatted file with the genome sequence(s).
bacteria	Logical, the genome is either a bacteria (default) or an archea.
cpu	Number of CPUs to use, default is 1.

Details

The external software barrnap is used to scan through a prokaryotic genome to detect the rRNA genes (5S, 16S, 23S). This free software can be installed from https://github.com/tseemann/barrnap.

Value

A gff.table (see readGFF for details) with one row for each detected rRNA sequence.

Note

The barrnap software must be installed on the system for this function to work, i.e. the command 'system("barrnap --help")' must be recognized as a valid command if you run it in the Console window.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

readGFF, gff2fasta.

Examples

```
## Not run:
# This example requires the external barrnap software
# Using a genome file in this package.
xpth <- file.path(path.package("micropan"),"extdata")
genome.file <- file.path(xpth,"Example_genome.fasta.xz")

# We need to uncompress it first...
tf <- tempfile(fileext=".xz")
s <- file.copy(genome.file,tf)
tf <- xzuncompress(tf)

# Searching for rRNA sequences, and inspecting
gff.table <- barrnap(tf)
print(gff.table)

# Retrieving the sequences
genome <- readFasta(tf)
rRNA.fasta <- gff2fasta(gff.table,genome)

# ...and cleaning...
```

```
file.remove(tf)

## End(Not run)
```

`bClust` *Clustering sequences based on pairwise distances*

Description

Sequences are clustered by hierarchical clustering based on a set of pariwise distances. The distances must take values between 0.0 and 1.0, and all pairs *not* listed are assumed to have distance 1.0.

Usage

```
bClust(dist.table, linkage = "single", threshold = 1)
```

Arguments

`dist.table`	A `data.frame` with pairwise distances. The columns '`Sequence.A`' and '`Sequence.B`' contain tags identifying pairs of sequences. The column '`Distance`' contains the distances, always a number from 0.0 to 1.0.
`linkage`	A text indicating what type of clustering to perform, either '`single`' (default), '`average`' or '`complete`'.
`threshold`	Specifies the maximum size of a cluster. Must be a distance, i.e. a number between 0.0 and 1.0.

Details

Computing clusters (gene families) is an essential step in many comparative studies. `bClust` will assign sequences into gene families by a hierarchical clustering approach. Since the number of sequences may be huge, a full all-against-all distance matrix will be impossible to handle in memory. However, most sequence pairs will have an 'infinite' distance between them, and only the pairs with a finite (smallish) distance need to be considered.

This function takes as input the distances in a `data.frame` where only the interesting distances are listed. Typically, this `data.frame` is the output from `bDist`. All pairs of sequence *not* listed are assumed to have distance 1.0, which is considered the 'infinite' distance. Note that '`dist.table`' must have the columns '`Sequence.A`', '`Sequence.B`' and '`Distance`'. The first two contain texts identifying sequences, the latter contains the distances. All sequences must be listed at least once. This should pose no problem, since all sequences have distance 0.0 to themselves, and should be listed with this distance once.

The '`linkage`' defines the type of clusters produced. The '`threshold`' indicates the size of the clusters. A '`single`' linkage clustering means all members of a cluster have at least one other member of the same cluster within distance '`threshold`' from itself. An '`average`' linkage means all members of a cluster are within the distance '`threshold`' from the center of the cluster. A

'complete' linkage means all members of a cluster are no more than the distance 'threshold' away from any other member of the same cluster.

Typically, 'single' linkage produces big clusters where members may differ a lot, since they are only required to be close to something, which is close to something,...,which is close to some other member. On the other extreme, 'complete' linkage will produce small and tight clusters, since all must be similar to all. The 'average' linkage is between, but closer to 'complete' linkage. If you want the 'threshold' to specify directly the maximum distance tolerated between two members of the same gene family, you must use 'complete' linkage. The 'single' linkage is the fastest alternative to compute. Using the default setting of 'single' linkage and maximum 'threshold' (1.0) will produce the largest and fewest clusters possible.

Value

The function returns a vector of integers, indicating the cluster membership of every unique sequence from the 'Sequence.A' and 'Sequence.B' columns of the input 'dist.table'. The name of each element indicates the sequence. Sequences having the same number are in the same cluster.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

bDist, hclust, dClust, isOrtholog.

Examples

```
# Loading distance data in the micropan package
data(Mpneumoniae.blast.distances,package="micropan")

# Clustering with default settings
clustering.blast.single <- bClust(Mpneumoniae.blast.distances)

# Clustering with complete linkage and a liberal threshold
clustering.blast.complete <- bClust(Mpneumoniae.blast.distances,linkage="complete",threshold=0.75)
```

---

bDist *Computes distances between sequences based on BLAST results*

Description

Reads a complete set of result files from a BLAST search and computes distance between all sequences based on the BLAST bit-score.

Usage

```
bDist(blast.files, e.value = 1, verbose = TRUE)
```

Arguments

blast.files	A text vector of filenames.
e.value	A threshold E-value to immediately discard (very) poor BLAST alignments. Default is 1.0.
verbose	A logical indicating if textual output should be given to monitor the progress.

Details

Each input file must be a BLAST result file where all proteins of one genome have been queried against a database of all proteins from another genome. The result files must all have 12 columns of results, i.e. have been produced by the option '-outfmt 6' in the BLAST+ software. The filenames must have the format 'GID111_vs_GID222.txt' and are typically produced by blastAllAll.

Setting a small 'e.value' threshold can speed up the computation and reduce the size of the output, but you may loose some alignments that could produce smallish distances for short sequences.

The distance computed is a relative score. If an alignment of query A against hit B has a bit-score of S(A;B), we compute an intermediate distance D(A;B)=1-S(A;B)/S(A;A) where S(A;A) is the bit-score of aligning A against itself. Reversing the search, we also get D(B;A)=1-S(B;A)/S(B;B), where B has been used as query and A is the hit. The final distance is D(A,B)=(D(A;B)+D(B;A))/2. A distance of 0.0 means A and B are identical. The maximum possible distance is 1.0, meaning there is no BLAST hit found either way.

This distance should not be interpreted as lack of identity. A distance of 0.0 means 100% identity, but a distance of 0.25 does *not* mean 75% identity. It has some resemblance to an evolutinary (raw) distance, but since it is based on protein alignments, the type of mutations plays a significant role, not only the number of mutations.

Value

The function returns a 'data.frame' with columns 'Sequence.A', 'Sequence.B' and 'Distance'. Each row corresponds to a pair of sequence having at least one BLAST hit between them. All pairs *not* listed in the output have distance 1.0 between them.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

blastAllAll, readBlastTable, bClust, isOrtholog.

Examples

```
# Using BLAST result files in this package...
prefix <- c("GID1_vs_GID1.txt",
            "GID1_vs_GID2.txt",
            "GID1_vs_GID3.txt",
            "GID2_vs_GID1.txt",
            "GID2_vs_GID2.txt",
            "GID2_vs_GID3.txt",
```

```
            "GID3_vs_GID1.txt",
            "GID3_vs_GID2.txt",
            "GID3_vs_GID3.txt")
xpth <- file.path(path.package("micropan"),"extdata")
blast.files <- file.path(xpth,paste(prefix,".xz",sep=""))

# We need to uncompress them first...
tf <- tempfile(pattern=prefix,fileext=".xz")
s <- file.copy(blast.files,tf)
tf <- unlist(lapply(tf,xzuncompress))

# Computing pairwise distances
blast.distances <- bDist(tf)

# ...and cleaning...
s <- file.remove(tf)

# See also example for blastAllAll
```

`binomixEstimate` *Binomial mixture model estimates*

Description

Fits binomial mixture models to the data given as a pan-matrix. From the fitted models both estimates of pan-genome size and core-genome size are available.

Usage

```
binomixEstimate(pan.matrix, K.range = 3:5, core.detect.prob = 1,
  verbose = TRUE)
```

Arguments

`pan.matrix`	A `Panmat` object, see `panMatrix` for details.
`K.range`	The range of model complexities to explore. The vector of integers specify the number of binomial densities to combine in the mixture models.
`core.detect.prob`	The detection probability of core genes. This should almost always be 1.0, since a core gene is by definition always present in all genomes, but can be set fractionally smaller.
`verbose`	Logical indicating if textual output should be given to monitor the progress of the computations.

Details

A binomial mixture model can be used to describe the distribution of gene clusters across genomes in a pan-genome. The idea and the details of the computations are given in Hogg et al (2007), Snipen et al (2009) and Snipen & Ussery (2012).

Central to the concept is the idea that every gene has a detection probability, i.e. a probability of being present in a genome. Genes who are always present in all genomes are called core genes, and these should have a detection probability of 1.0. Other genes are only present in a subset of the genomes, and these have smaller detection probabilities. Some genes are only present in one single genome, denoted ORFan genes, and an unknown number of genes have yet to be observed. If the number of genomes investigated is large these latter must have a very small detection probability.

A binomial mixture model with 'K' components estimates 'K' detection probabilities from the data. The more components you choose, the better you can fit the (present) data, at the cost of less precision in the estimates due to less degrees of freedom. `binomixEstimate` allows you to fit several models, and the input '`K.range`' specifies which values of 'K' to try out. There no real point using 'K' less than 3, and the default is '`K.range=3:5`'. In general, the more genomes you have the larger you can choose 'K' without overfitting. Computations will be slower for larger values of 'K'. In order to choose the optimal value for 'K', `binomixEstimate` computes the BIC-criterion, see below.

As the number of genomes grow, we tend to observe an increasing number of gene clusters. Once a 'K'-component binomial mixture has been fitted, we can estimate the number of gene clusters not yet observed, and thereby the pan-genome size. Also, as the number of genomes grows we tend to observe fewer core genes. The fitted binomial mixture model also gives an estimate of the final number of core gene clusters, i.e. those still left after having observed 'infinite' many genomes.

The detection probability of core genes should be 1.0, but can at times be set fractionally smaller. This means you accept that even core genes are not always detected in every genome, e.g. they may be there, but your gene prediction has missed them. Notice that setting the '`core.detect.prob`' to less than 1.0 may affect the core gene size estimate dramatically.

Value

`binomixEstimate` returns a `Binomix` object, which is a small (S3) extension of a `list` with two components. These two components are named '`BIC.table`' and '`Mix.list`'.

The '`BIC.table`' is a matrix listing, in each row, the results for each number of components used, given by the input '`K.range`'. The column '`Core.size`' is the estimated number of core gene families, the column '`Pan.size`' is the estimated pan-genome size. The column '`BIC`' is the Bayesian Information Criterion (Schwarz, 1978) that should be used to choose the optimal value for 'K'. The number of components where '`BIC`' is minimized is the optimal. If minimum '`BIC`' is reached for the largest 'K' value you should extend the '`K.range`' and re-fit. The function will issue a `warning` to remind you of this.

The '`Mix.list`' is a list with one element for each number of components tested. The content of each '`Mix.list`' element is a matrix describing one particular fitted binomial mixture model. A fitted model is characterized by two vectors (rows) denoted '`Detect.prob`' and '`Mixing.prop`'. '`Detect.prob`' are the estimated detection probabilities, sorted in ascending order. The '`Mixing.prop`' are the corresponding mixing proportions. A mixing proportion is the proportion of the gene clusters having the corresponding detection probability.

The generic functions `plot.Binomix` and `summary.Binomix` are available for `Binomix` objects.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Hogg, J.S., Hu, F.Z, Janto, B., Boissy, R., Hayes, J., Keefe, R., Post, J.C., Ehrlich, G.D. (2007). Characterization and modeling of the Haemophilus influenzae core- and supra-genomes based on the complete genomic sequences of Rd and 12 clinical nontypeable strains. Genome Biology, 8:R103.

Snipen, L., Almoy, T., Ussery, D.W. (2009). Microbial comparative pan-genomics using binomial mixture models. BMC Genomics, 10:385.

Snipen, L., Ussery, D.W. (2012). A domain sequence approach to pangenomics: Applications to Escherichia coli. F1000 Research, 1:19.

Schwarz, G. (1978). Estimating the Dimension of a Model. The Annals of Statistics, 6(2):461-464.

See Also

panMatrix, chao, plot.Binomix, summary.Binomix.

Examples

```
# Loading a Panmat object in the micropan package
data(list="Mpneumoniae.blast.panmat",package="micropan")

# Estimating binomial mixture models
bino <- binomixEstimate(Mpneumoniae.blast.panmat,K.range=3:8)  # using 3,4,...,8 components
print(bino$BIC.table) # minimum BIC at 3 components

# Plotting the optimal model, and printing the summary
plot(bino)
summary(bino)

# Plotting the 8-component model as well
plot(bino,ncomp=8)  # clearly overfitted, we do not need this many sectors

# Plotting the distribution in a single genome
plot(bino,type="single")  # completely dominated by core genes
```

---

blastAllAll *Making BLAST search all against all genomes*

Description

Runs a reciprocal all-against-all BLAST search to look for similarity of proteins within and across genomes. The main job is done by the BLAST+ software.

Usage

```
blastAllAll(prot.files, out.folder, e.value = 1, job = 1, threads = 1,
  verbose = T)
```

Arguments

prot.files	A text vector with the names of the FASTA files where the protein sequences of each genome is found.
out.folder	The name of the folder where the result files should end up.
e.value	The chosen E-value threshold in BLAST. Default is 'e.value=1', a smaller value will speed up the search at the cost of less sensitivity.
job	An integer to separate multiple jobs. You may want to run several jobs in parallell, and each job should have different number here to avoid confusion on databases. Default is 'job=1'.
threads	The number of CPU's to use.
verbose	Logical, if TRUE some text output is produced to monitor the progress.

Details

A basic step in pangenomics and many other comparative studies is to cluster proteins into groups or families. One commonly used approach is based on reciprocal BLASTing. This function uses the BLAST+ software available for free from NCBI (Camacho et al, 2009).

A vector listing FASTA files of protein sequences is given as input in 'prot.files'. These files must have the GID-tag in the first token of every header, and in their filenames as well, i.e. all input files should first be prepared by panPrep to ensure this. Note that only protein sequences are considered here. If your coding genes are stored as DNA, please translate them to protein prior to using this function, see translate.

A BLAST database is made from each genome in turn. Then all genomes are queried against this database, and for every pair of genomes a result file is produced. If two genomes have GID-tags 'GID111', and 'GID222' then both result file 'GID111_vs_GID222.txt' and 'GID222_vs_GID111.txt' will be found in 'out.folder' after the completion of this search. This reciprocal (two-way) search is required because of the heuristics of BLAST.

The 'out.folder' is scanned for already existing result files, and blastAllAll never overwrites an existing result file. If a file with the name 'GID111_vs_GID222.txt' already exists in the 'out.folder', this particular search is skipped. This makes it possible to run multiple jobs in parallell, writing to the same 'out.folder'. It also makes it possible to add new genomes, and only BLAST the new combinations without repeating previous comparisons.

This search can be slow if the genomes contain many proteins and it scales quadratically in the number of input files. It is best suited for the study of a smaller number of genomes (less than say 100). By starting multiple R sessions, you can speed up the search by running blastAllAll from each R session, using the same 'out.folder' but different integers for the job option. If you are using a computing cluster you can also increase the number of CPUs by increasing threads.

The result files are text files, and can be read into R using readBlastTable, but more commonly they are used as input to bDist to compute distances between sequences for subsequent clustering.

Value

The function produces *N*N* result files if 'prot.files' lists *N* sequence files. These result files are located in out.folder. Existing files are never overwritten by blastAllAll, if you want to re-compute something, delete the corresponding result files first.

Note

The BLAST+ software must be installed on the system for this function to work, i.e. the command 'system("makeblastdb -help")' must be recognized as valid commands if you run them in the Console window.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., Madden, T.L. (2009). BLAST+: architecture and applications. BMC Bioinformatics, 10:421.

See Also

panPrep, readBlastTable, bDist.

Examples

```
## Not run:
# This example requires the external BLAST+ software
# Using protein files in this package
xpth <- file.path(path.package("micropan"),"extdata")
prot.files <- file.path(xpth,c("Example_proteins_GID1.fasta.xz",
                               "Example_proteins_GID2.fasta.xz",
                               "Example_proteins_GID3.fasta.xz"))

# We need to uncompress them first...
tf <- tempfile(fileext=c("GID1.fasta.xz","GID2.fasta.xz","GID3.fasta.xz"))
s <- file.copy(prot.files,tf)
tf <- unlist(lapply(tf,xzuncompress))

# Blasting all versus all...(requires BLAST+)
tmp.dir <- tempdir()
blastAllAll(tf,out.folder=tmp.dir)

# Reading results, and computing blast.distances
blast.files <- dir(tmp.dir,pattern="GID[0-9]+_vs_GID[0-9]+.txt")
blast.distances <- bDist(file.path(tmp.dir,blast.files))

# ...and cleaning tmp.dir...
s <- file.remove(tf)
s <- file.remove(file.path(tmp.dir,blast.files))
```

```
## End(Not run)
```

---

chao *The Chao lower bound estimate of pan-genome size*

Description

Computes the Chao lower bound estimated number of gene clusters in a pan-genome.

Usage

```
chao(pan.matrix)
```

Arguments

`pan.matrix`	A `Panmat` object, see `panMatrix` for details.

Details

The size of a pan-genome is the number of gene clusters in it, both those observed and those not yet observed.

The input '`pan.matrix`' is a `Panmat` object, i.e. it is a matrix with one row for each genome and one column for each observed gene cluster in the pan-genome. See `panMatrix` for how to construct such objects.

The number of observed gene clusters is simply the number of columns in '`pan.matrix`'. The number of gene clusters not yet observed is estimated by the Chao lower bound estimator (Chao, 1987). This is based solely on the number of clusters observed in 1 and 2 genomes. It is a very simple and conservative estimator, i.e. it is more likely to be too small than too large.

Value

The function returns an integer, the estimated pan-genome size. This includes both the number of gene clusters observed so far, as well as the estimated number not yet seen.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Chao, A. (1987). Estimating the population size for capture-recapture data with unequal catchability. Biometrics, 43:783-791.

See Also

`panMatrix`, `binomixEstimate`.

Examples

```
# Loading a Panmat object in the micropan package
data(list="Mpneumoniae.blast.panmat",package="micropan")

# Estimating the pan-genome size using the Chao estimator
chao.pansize <- chao(Mpneumoniae.blast.panmat)
```

`dClust` *Clustering sequences based on domain sequence*

Description

Proteins are clustered by their sequence of protein domains. A domain sequence is the ordered sequence of domains in the protein. All proteins having identical domain sequence are assigned to the same cluster.

Usage

```
dClust(hmmer.table)
```

Arguments

`hmmer.table`	A `data.frame` of results from a `hmmerScan` against a domain database.

Details

A domain sequence is simply the ordered list of domains occurring in a protein. Not all proteins contain known domains, but those who do will have from one to several domains, and these can be ordered forming a sequence. Since domains can be more or less conserved, two proteins can be quite different in their amino acid sequence, and still share the same domains. Describing, and grouping, proteins by their domain sequence was proposed by Snipen & Ussery (2012) as an alternative to clusters based on pairwise alignments, see `bClust`. Domain sequence clusters are less influenced by gene prediction errors.

The input is a `data.frame` of the type produced by `readHmmer`. Typically, it is the result of scanning proteins (using `hmmerScan`) against Pfam-A or any other HMMER3 database of protein domains. It is highly reccomended that you remove overlapping hits in '`hmmer.table`' before you pass it as input to `dClust`. Use the function `hmmerCleanOverlap` for this. Overlapping hits are in some cases real hits, but often the poorest of them are artifacts.

Value

The output is a numeric vector with one element for each unique sequence in the '`Query`' column of the input '`hmmer.table`'. Sequences with identical number belong to the same cluster. The name of each element identifies the sequence.

This vector also has an attribute called '`cluster.info`' which is a character vector containing the domain sequences. The first element is the domain sequence for cluster 1, the second for cluster

2, etc. In this way you can, in addition to clustering the sequences, also see which domains the sequences of a particular cluster share.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Snipen, L. Ussery, D.W. (2012). A domain sequence approach to pangenomics: Applications to Escherichia coli. F1000 Research, 1:19.

See Also

panPrep, hmmerScan, readHmmer, hmmerCleanOverlap, bClust.

Examples

```
# Using HMMER3 result files in the micropan package
xpth <- file.path(path.package("micropan"),"extdata")
hmm.files <- file.path(xpth,c("GID1_vs_Pfam-A.hmm.txt.xz",
                              "GID2_vs_Pfam-A.hmm.txt.xz",
                              "GID3_vs_Pfam-A.hmm.txt.xz"))

# We need to uncompress them first...
tf <- tempfile(fileext=rep(".xz",length(hmm.files)))
s <- file.copy(hmm.files,tf)
tf <- unlist(lapply(tf,xzuncompress))

# Reading the HMMER3 results, cleaning overlaps...
hmmer.table <- NULL
for(i in 1:3){
  htab <- readHmmer(tf[i])
  htab <- hmmerCleanOverlap(htab)
  hmmer.table <- rbind(hmmer.table,htab)
}

# The clustering
clustering.domains <- dClust(hmmer.table)

# ...and cleaning...
s <- file.remove(tf)
```

---

distJaccard *Computing Jaccard distances between genomes*

Description

Computes the Jaccard distances between all pairs of genomes.

Usage

```
distJaccard(pan.matrix)
```

Arguments

pan.matrix	A Panmat object, see panMatrix for details.

Details

The Jaccard index between two sets is defined as the size of the interesection of the sets divided by the size of the union. The Jaccard distance is simply 1 minus the Jaccard index.

The Jaccard distance between two genomes describes their degree of overlap with respect to gene cluster content. If the Jaccard distance is 0.0, the two genomes contain identical gene clusters. If it is 1.0 the two genomes are non-overlapping. The difference between a genomic fluidity (see fluidity) and a Jaccard distance is small, they both measure overlap between genomes, but fluidity is computed for the population by averaging over many pairs, while Jaccard distances are computed for every pair. Note that only presence/absence of gene clusters are considered, not multiple occurrences.

The input 'pan.matrix' is typically constructed by panMatrix.

Value

A dist object (see dist) containing all pairwise Jaccard distances between genomes.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panMatrix, fluidity, dist.

Examples

```
# Loading two Panmat objects in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")

# Jaccard distances based on a BLAST clustering Panmat object
Jdist.blast <- distJaccard(Mpneumoniae.blast.panmat)

# Jaccard distances based on domain sequence clustering Panmat object
Jdist.domains <- distJaccard(Mpneumoniae.domain.panmat)
```

`distManhattan` *Computing Manhattan distances between genomes*

Description

Computes the (weighted) Manhattan distances beween all pairs of genomes.

Usage

```
distManhattan(pan.matrix, scale = 0, weights = rep(1, dim(pan.matrix)[2]))
```

Arguments

`pan.matrix`	A `Panmat` object, see `panMatrix` for details.
`scale`	An optional scale to control how copy numbers should affect the distances.
`weights`	Vector of optional weights of gene clusters.

Details

The Manhattan distance is defined as the sum of absolute elementwise differences between two vectors. Each genome is represented as a vector (row) of integers in '`pan.matrix`'. The Manhattan distance between two genomes is the sum of absolute difference between these rows. If two rows (genomes) of the '`pan.matrix`' are identical, the corresponding Manhattan distance is '`0.0`'.

The '`scale`' can be used to control how copy number differences play a role in the distances computed. Usually we assume that going from 0 to 1 copy of a gene is the big change of the genome, and going from 1 to 2 (or more) copies is less. Prior to computing the Manhattan distance, the '`pan.matrix`' is transformed according to the following affine mapping: If the original value in '`pan.matrix`' is '`x`', and '`x`' is not 0, then the transformed value is '`1 + (x-1)*scale`'. Note that with '`scale=0.0`' (default) this will result in 1 regardless of how large '`x`' was. In this case the Manhattan distance only distinguish between presence and absence of gene clusters. If '`scale=1.0`' the value '`x`' is left untransformed. In this case the difference between 1 copy and 2 copies is just as big as between 1 copy and 0 copies. For any '`scale`' between 0.0 and 1.0 the transformed value is shrunk towards 1, but a certain effect of larger copy numbers is still present. In this way you can decide if the distances between genomes should be affected, and to what degree, by differences in copy numbers beyond 1. Notice that as long as '`scale=0.0`' (and no weighting) the Manhattan distance has a nice interpretation, namely the number of gene clusters that differ in present/absent status between two genomes.

When summing the difference across gene clusters we can also up- or downweight some clusters compared to others. The vector '`weights`' must contain one value for each column in '`pan.matrix`'. The default is to use flat weights, i.e. all clusters count equal. See `geneWeights` for alternative weighting strategies.

Value

A `dist` object (see `dist`) containing all pairwise Manhattan distances between genomes.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panMatrix, distJaccard, geneWeights, panTree.

Examples

```
# Loading two Panmat objects in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")

# Manhattan distances based on a BLAST clustering Panmat object
Mdist.blast <- distManhattan(Mpneumoniae.blast.panmat)

# Manhattan distances based on domain sequence clustering Panmat object
Mdist.domains <- distManhattan(Mpneumoniae.domain.panmat,scale=0.5)
```

---

entrezDownload *Downloading genome data*

Description

Retrieving genomes from NCBI using the Entrez programming utilities.

Usage

```
entrezDownload(accession, out.file, verbose = TRUE)
```

Arguments

accession	A character vector containing a set of valid accession numbers at the NCBI Nucleotide database.
out.file	Name of the file where downloaded sequences should be written in FASTA format.
verbose	Logical indicating if textual output should be given during execution, to monitor the download progress.

Details

The Entrez programming utilities is a toolset for automatic download of data from the NCBI databases, see E-utilities Quick Start for details. entrezDownload can be used to download genomes from the NCBI Nucleotide database through these utilities.

The argument 'accession' must be a set of valid accession numbers at NCBI Nucleotide, typically all accession numbers related to a genome (chromosomes, plasmids, contigs, etc). For completed genomes, where the number of sequences is low, 'accession' is typically a single text listing all

accession numbers separated by commas. In the case of some draft genomes having a large number of contigs, the accession numbers must be split into several comma-separated texts. The reason for this is that Entrez will not accept too many queries in one chunk (less than 500).

The downloaded sequences are saved in '`file`' on your system. This will be a FASTA formatted file, and should by convention have the filename extension '`.fsa`'. Note that all downloaded sequences end up in this file. If you want to download multiple genomes, you call `entrezDownload` multiple times.

Value

The name of the resulting FASTA file is returned (same as `file`), but the real result of this function is the creation of the file itself.

Author(s)

Lars Snipen and Kristian Liland.

See Also

`getAccessions`, `readFasta`.

Examples

```
# Accession numbers for the chromosome and plasmid of Buchnera aphidicola, strain APS
acc <- "BA000003.2,AP001071.1"
tf <- tempfile(pattern="Buchnera_aphidicola",fileext=".fasta")
txt <- entrezDownload(acc,out.file=tf)

# Reading file to inspect
genome <- readFasta(tf)
summary(genome)

# ...cleaning...
s <- file.remove(tf)
```

`findOrfs` *Finding ORFs in genomes*

Description

Finds all ORFs in prokaryotic genome sequences.

Usage

```
findOrfs(genome, circular = F)
```

Arguments

genome	A Fasta object with the genome sequence(s).
circular	Logical indicating if the genome sequences are completed, circular sequences.

Details

A prokaryotic Open Reading Frame (ORF) is defined as a subsequence starting with a start-codon (ATG, GTG or TTG), followed by an integer number of triplets (codons), and ending with a stop-codon (TAA, TGA or TAG). This function will locate all ORFs in a genome.

The argument genome will typically have several sequences (chromosomes/plasmids/scaffolds/contigs). It is vital that the *first token* (characters before first space) of every genome$Header is unique, since this will be used to identify these genome sequences in the output.

Note that for any given stop-codon there are usually multiple start-codons in the same reading frame. This function will return all, i.e. the same stop position may appear multiple times. If you want ORFs with the most upstream start-codon only (LORFs), then filter the output from this function with lorfs.

By default the genome sequences are assumed to be linear, i.e. contigs or other incomplete fragments of a genome. In such cases there will usually be some truncated ORFs at each end, i.e. ORFs where either the start- or the stop-codon is lacking. In the gff.table returned by this function this is marked in the Attributes column. The texts "Truncated=10" or "Truncated=01" indicates truncated at the Start or End, respectively.

If the supplied genome is a completed genome, with circular chromosome/plasmids, set the flag circular=TRUE and no truncated ORFs will be listed. In cases where an ORF runs across the origin of a circular genome sequences, the Stop coordinate will be larger than the length of the genome sequence. This is in line with the specifications of the GFF3 format, where a Start cannot be larger than the corresponding End.

Value

This function returns a gff.table, which is simply a data.frame with columns adhering to the format specified by the GFF3 format, see readGFF. If you want to retrieve the ORF sequences, use gff2fasta.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

readGFF, gff2fasta, lorfs.

Examples

```
# Using a genome file in this package
xpth <- file.path(path.package("micropan"),"extdata")
genome.file <- file.path(xpth,"Example_genome.fasta.xz")

# We need to uncompress them first...
```

```
tf <- tempfile(fileext=".xz")
s <- file.copy(genome.file,tf)
tf <- xzuncompress(tf)

# Reading into R and finding orfs
genome <- readFasta(tf)
orf.table <- findOrfs(genome)

# Computing ORF-lengths
orf.lengths <- orfLength(orf.table)
barplot(table(orf.lengths[orf.lengths>1]))

# Filtering to retrieve the LORFs only
lorf.table <- lorfs(orf.table)
lorf.lengths <- orfLength(lorf.table)
barplot(table(lorf.lengths[lorf.lengths>1]))

# ...and cleaning...
s <- file.remove(tf)
```

`fluidity` *Computing genomic fluidity for a pan-genome*

Description

Computes the genomic fluidity, which is a measure of population diversity.

Usage

```
fluidity(pan.matrix, n.sim = 10)
```

Arguments

`pan.matrix`	A `Panmat` object, see `panMatrix` for details.
`n.sim`	An integer specifying the number of random samples to use in the computations.

Details

The genomic fluidity between two genomes is defined as the number of unique gene families divided by the total number of gene families (Kislyuk et al, 2011). This is averaged over '`n.sim`' random pairs of genomes to obtain a population estimate.

The genomic fluidity between two genomes describes their degree of overlap with respect to gene cluster content. If the fluidity is 0.0, the two genomes contain identical gene clusters. If it is 1.0 the two genomes are non-overlapping. The difference between a Jaccard distance (see `distJaccard`) and genomic fluidity is small, they both measure overlap between genomes, but fluidity is computed for the population by averaging over many pairs, while Jaccard distances are computed for every pair. Note that only presence/absence of gene clusters are considered, not multiple occurrences.

The input '`pan.matrix`' is typically constructed by `panMatrix`.

Value

A list with two elements, the mean fluidity and its sample standard deviation over the 'n.sim' computed values.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Kislyuk, A.O., Haegeman, B., Bergman, N.H., Weitz, J.S. (2011). Genomic fluidity: an integrative view of gene diversity within microbial populations. BMC Genomics, 12:32.

See Also

panMatrix, distJaccard.

Examples

```
# Loading two Panmat objects in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")

# Fluidity based on a BLAST clustering Panmat object
fluid.blast <- fluidity(Mpneumoniae.blast.panmat)

# Fluidity based on domain sequence clustering Panmat object
fluid.domains <- fluidity(Mpneumoniae.domain.panmat)
```

---

geneWeights *Gene cluster weighting*

Description

This function computes weights for gene cluster according to their distribution in a pan-genome.

Usage

```
geneWeights(pan.matrix, type = c("shell", "cloud"))
```

Arguments

pan.matrix	A Panmat object, see panMatrix for details.
type	A text indicating the weighting strategy.

Details

When computing distances between genomes or a PCA, it is possible to give weights to the different gene clusters, emphasizing certain aspects.

As proposed by Snipen & Ussery (2010), we have implemented two types of weighting: The default '"shell"' type means gene families occuring frequently in the genomes, denoted shell-genes, are given large weight (close to 1) while those occurring rarely are given small weight (close to 0). The opposite is the '"cloud"' type of weighting. Genes observed in a minority of the genomes are referred to as cloud-genes. Presumeably, the '"shell"' weighting will give distances/PCA reflecting a more long-term evolution, since emphasis is put on genes who have just barely diverged away from the core. The '"cloud"' weighting emphasizes those gene clusters seen rarely. Genomes with similar patterns among these genes may have common recent history. A '"cloud"' weighting typically gives a more erratic or 'noisy' picture than the '"shell"' weighting.

Value

A vector of weights, one for each column in `pan.matrix`.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Snipen, L., Ussery, D.W. (2010). Standard operating procedure for computing pangenome trees. Standards in Genomic Sciences, 2:135-141.

See Also

`panMatrix`, `distManhattan`.

Examples

```
# Loading a Panmat object in the micropan package
data(list="Mpneumoniae.blast.panmat",package="micropan")

# Weighted Manhattan distances based on a BLAST clustering Panmat object
w <- geneWeights(Mpneumoniae.blast.panmat,type="shell")
Mdist.blast <- distManhattan(Mpneumoniae.blast.panmat,weights=w)
```

---

getAccessions *Collecting contig accession numbers*

Description

Retrieving the accession numbers for all contigs from a master record GenBank file.

Usage

```
getAccessions(master.record.accession)
```

Arguments

master.record.accession	The accession number (single text) to a master record GenBank file having the WGS entry specifying the accession numbers to all contigs of the WGS genome.

Details

In order to download a WGS genome (draft genome) using entrezDownload you will need the accession number of every contig. This is found in the master record GenBank file, which is available for every WGS genome. getAccessions will extract these from the GenBank file and return them in the apropriate way to be used by entrezDownload.

Value

A character vector where each element is a text listing the accession numbers separated by commas. Each vector element will contain no more than 500 accession numbers, see entrezDownload for details on this. The vector returned by getAccessions is typically used as input to entrezDownload.

Author(s)

Lars Snipen and Kristian Liland.

See Also

entrezDownload.

Examples

```
# The master record accession for the WGS genome Mycoplasma genitalium, strain G37
acc <- getAccessions("AAGX00000000")
# Then we use this to download all contigs and save them
tf <- tempfile(fileext=".fasta")
txt <- entrezDownload(acc,out.file=tf)

# Reading the file to inspect it
genome <- readFasta(tf)
summary(genome)

# ...cleaning...
s <- file.remove(tf)
```

gff2fasta *Retrieving sequences from genome*

Description

Retrieving the sequences specified in a gff.table.

Usage

```
gff2fasta(gff.table, genome)
```

Arguments

gff.table	A gff.table (data.frame) with genomic features information.
genome	A Fasta object with the genome sequence(s).

Details

Each row in gff.table (see readGFF) describes a genomic feature in the genome. The information in the columns Seqid, Start, End and Strand are used to retrieve the sequences from genome$Sequence. Every Seqid in the gff.table must match the first token in one of the genome$Header texts.

Value

A Fasta object with one row for each row in gff.table. The Header for each sequence is a summary of the information in the corresponding row of gff.table.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

readGFF, findOrfs.

Examples

```
# Using two files in this package
xpth <- file.path(path.package("micropan"),"extdata")
gff.file <- file.path(xpth,"Example.gff.xz")
genome.file <- file.path(xpth,"Example_genome.fasta.xz")

# We need to uncompress them first...
gff.tf <- tempfile(fileext=".xz")
s <- file.copy(gff.file,gff.tf)
gff.tf <- xzuncompress(gff.tf)
genome.tf <- tempfile(fileext=".xz")
s <- file.copy(genome.file,genome.tf)
genome.tf <- xzuncompress(genome.tf)
```

```
# Reading
gff.table <- readGFF(gff.tf)
genome <- readFasta(genome.tf)

# Retrieving sequences
fasta.obj <- gff2fasta(gff.table,genome)
summary(fasta.obj)
plot(fasta.obj)

# ...and cleaning...
s <- file.remove(gff.tf,genome.tf)
```

`gffSignature` *GFF signature text*

Description

Making a signature text from `gff.table` data.

Usage

```
gffSignature(gff.table)
```

Arguments

`gff.table`	A `gff.table` (`data.frame`) with genomic features information.

Details

For each row in `gff.table` a text is created by pasting these data together, adding some explanatory text. This function is used by `link{gff2fasta}` to create the Header-lines for the `Fasta` object when retrieving the sequences.

Value

A vector of texts, one for each row in `gff.table`.

Author(s)

Lars Snipen.

See Also

`findOrfs`, `gff2fasta`.

Examples

```
# See the example in the Help-file for readGFF.
```

heaps *Heaps law estimate*

Description

Estimating if a pan-genome is open or closed based on a Heaps law model.

Usage

```
heaps(pan.matrix, n.perm = 100)
```

Arguments

pan.matrix	A Panmat object, see panMatrix for details.
n.perm	The number of random permutations of genome ordering.

Details

An open pan-genome means there will always be new gene clusters observed as long as new genomes are being sequenced. This may sound controversial, but in a pragmatic view, an open pan-genome indicates that the number of new gene clusters to be observed in future genomes is 'large' (but not literally infinite). Opposite, a closed pan-genome indicates we are approaching the end of new gene clusters.

This function is based on a Heaps law approach suggested by Tettelin et al (2008). The Heaps law model is fitted to the number of new gene clusters observed when genomes are ordered in a random way. The model has two parameters, an intercept and a decay parameter called 'alpha'. If 'alpha>1.0' the pan-genome is closed, if 'alpha<1.0' it is open.

The number of permutations, 'n.perm', should be as large as possible, limited by computation time. The default value of 100 is certainly a minimum.

Word of caution: The Heaps law assumes independent sampling. If some of the genomes in the data set form distinct sub-groups in the population, this may affect the results of this analysis severely.

Value

A vector of two estimated parameters: The 'Intercept' and the decay parameter 'alpha'. If 'alpha<1.0' the pan-genome is open, if 'alpha>1.0' it is closed.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Tettelin, H., Riley, D., Cattuto, C., Medini, D. (2008). Comparative genomics: the bacterial pan-genome. Current Opinions in Microbiology, 12:472-477.

See Also

binomixEstimate, chao, rarefaction.

Examples

```
# Loading a Panmat object in the micropan package
data(list="Mpneumoniae.blast.panmat",package="micropan")

# Estimating population openness
h.est <- heaps(Mpneumoniae.blast.panmat,n.perm=500)
if(h.est[2]>1){
  cat("Population is closed with alpha =",h.est[2], "\n")
} else {
  cat("Population is open with alpha =",h.est[2], "\n")
}
```

---

hmmerCleanOverlap *Removing overlapping hits from HMMER3 scans*

Description

Removing hits to avoid overlapping HMMs on the same protein sequence.

Usage

```
hmmerCleanOverlap(hmmer.table)
```

Arguments

hmmer.table	A data.frame with hmmerScan results, see readHmmer.

Details

When scanning sequences against a profile HMM database using hmmerScan, we often find that several patterns (HMMs) match in the same region of the query sequence, i.e. we have overlapping hits. The function hmmerCleanOverlap will remove the poorest overlapping hit in a recursive way such that all overlaps are eliminated.

The input is a data.frame of the type produced by readHmmer.

Value

A data.frame which is a subset of the input, where some rows have been deleted to avoid overlapping hits.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

hmmerScan, readHmmer, dClust.

Examples

```
# See the example in the Help-file for dClust.
```

---

hmmerScan *Scanning a profile Hidden Markov Model database*

Description

Scanning FASTA formatted protein files against a database of pHMMs using the HMMER3 software.

Usage

```
hmmerScan(in.files, db, out.folder, threads = 0, verbose = TRUE)
```

Arguments

in.files	A character vector of file names.
db	The full name of the database to scan.
out.folder	The name of the folder to put the result files.
threads	Number of CPU's to use.
verbose	Logical indicating if textual output should be given to monitor the progress.

Details

The HMMER3 software is purpose-made for handling profile Hidden Markov Models (pHMM) describing patterns in biological sequences (Eddy, 2008). This function will make calls to the HMMER3 software to scan FASTA files of proteins against a pHMM database.

The files named in 'in.files' must contain FASTA formatted protein sequences. These files should be prepared by panPrep to make certain each sequence, as well as the file name, has a GID-tag identifying their genome. The database named in 'db' must be a HMMER3 formatted database. It is typically the Pfam-A database, but you can also make your own HMMER3 databases, see the HMMER3 documentation for help.

hmmerScan will query every input file against the named database. The database contains profile Hidden Markov Models describing position specific sequence patterns. Each sequence in every input file is scanned to see if some of the patterns can be matched to some degree. Each input file results in an output file with the same GID-tag in the name. The result files give tabular output, and are plain text files. See readHmmer for how to read the results into R.

Scanning large databases like Pfam-A takes time, usually several minutes per genome. The scan is set up to use only 1 cpu per scan by default. By increasing threads you can utilize multiple CPUs,

typically on a computing cluster. Our experience is that from a multi-core laptop it is better to start this function in default mode from mutliple R-sessions. This function will not overwrite an existing result file, and multiple parallel sessions can write results to the same folder.

Value

This function produces files in the folder specified by '`out.folder`'. Existing files are never overwritten by `hmmerScan`, if you want to re-compute something, delete the corresponding result files first.

Note

The HMMER3 software must be installed on the system for this function to work, i.e. the command '`system("hmmscan -h")`' must be recognized as a valid command if you run it in the Console window.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Eddy, S.R. (2008). A Probabilistic Model of Local Sequence Alignment That Simplifies Statistical Significance Estimation. PLoS Computational Biology, 4(5).

See Also

`panPrep`, `readHmmer`.

Examples

```
## Not run:
# This example requires the external HMMER software
# Using two files in the micropan package
xpth <- file.path(path.package("micropan"),"extdata")
prot.file <- file.path(xpth,"Example_proteins_GID1.fasta.xz")
db <- "microfam.hmm"
db.files <- file.path(xpth,paste(db,c(".h3f.xz",".h3i.xz",".h3m.xz",".h3p.xz"),sep=""))

# We need to uncompress them first...
prot.tf <- tempfile(pattern="GID1.fasta",fileext=".xz")
s <- file.copy(prot.file,prot.tf)
prot.tf <- xzuncompress(prot.tf)
db.tf <- paste(tempfile(),c(".h3f.xz",".h3i.xz",".h3m.xz",".h3p.xz"),sep="")
s <- file.copy(db.files,db.tf)
db.tf <- unlist(lapply(db.tf,xzuncompress))
db.name <- gsub("\\",.Platform$file.sep,sub(".h3f$","",db.tf[1]),fixed=T)

# Scanning the FASTA-file against microfam0...
tmp.dir <- tempdir()
hmmerScan(in.files=prot.tf,db=db.name,out.folder=tmp.dir)
```

```
# Reading results
db.nm <- rev(unlist(strsplit(db.name,split=.Platform$file.sep)))[1]
hmm.file <- file.path(tmp.dir,paste("GID1_vs_",db.nm,".txt",sep=""))
hmm.tab <- readHmmer(hmm.file)

# ...and cleaning...
s <- file.remove(prot.tf)
s <- file.remove(sub(".xz","",db.tf))
s <- file.remove(hmm.file)

## End(Not run)
```

`isOrtholog` *Identifies orthologs in gene clusters*

Description

Finds the ortholog sequences in every cluster based on pairwise distances.

Usage

```
isOrtholog(clustering, dist.table)
```

Arguments

`clustering`	A vector of integers indicating the cluster for every sequence. Sequences with the same number belong to the same cluster. The name of each element is the tag identifying the sequence.
`dist.table`	A `data.frame` with pairwise distances. The columns '`Sequence.A`' and '`Sequence.B`' contain tags identifying pairs of sequences. The column '`Distance`' contains the distances, always a number from 0.0 to 1.0.

Details

The input `clustering` is typically produced by `bClust`. The input `dist.table` is typically produced by `bDist`.

The concept of orthologs is difficult for prokaryotes, and this function finds orthologs in a simplistic way. For a given cluster, with members from many genomes, there is one ortholog from every genome. In cases where a genome has two or more members in the same cluster, only one of these is an ortholog, the rest are paralogs.

Consider all sequences from the same genome belonging to the same cluster. The ortholog is defined as the one having the smallest sum of distances to all other members of the same cluster, i.e. the one closest to the 'center' of the cluster.

Note that the status as ortholog or paralog depends greatly on how clusters are defined in the first place. If you allow large and diverse (and few) clusters, many sequences will be paralogs. If you define tight and homogenous (and many) clusters, almost all sequences will be orthologs.

Value

A vector of logicals with the same number of elements as the input 'clustering', indicating if the corresponding sequence is an ortholog (TRUE) or not (FALSE). The name of each element is copied from 'clustering'.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

bDist, bClust.

Examples

```
## Not run:
# Loading distance data in the micropan package
data(list=c("Mpneumoniae.blast.distances","Mpneumoniae.blast.clustering"),package="micropan")

# Finding orthologs
is.ortholog <- isOrtholog(Mpneumoniae.blast.clustering,Mpneumoniae.blast.distances)

## End(Not run)
```

---

lorfs *Longest ORF*

Description

Filtering a gff.table with ORF information to keep only the LORFs.

Usage

```
lorfs(gff.table)
```

Arguments

gff.table	A gff.table (data.frame) with genomic features information.

Details

For every stop-codon there are usually mutliple possible start-codons in the same reading frame (nested ORFs). The LORF (Longest ORF) is defined as the longest of these nested ORFs, i.e. the ORF starting at the most upstream start-codon matching the stop-codon.

Value

A `gff.table` with a subset of the rows of the argument `gff.table`. After this filtering the Type variable in `gff.table` is changed to `"LORF"`. If you want to retirve the LORF sequences, use `gff2fasta`.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

`readGFF`, `findOrfs`, `gff2fasta`.

Examples

```
# See the example in the Help-file for findOrfs.
```

`micropan` *Microbial Pan-Genome Analysis*

Description

A collection of functions for computations and visualizations of microbial pan-genomes. Some of the functions make use of external software that needs to be installed on the system, see the package vignette for more details on this.

It is highly reccomended that you look at the supplied casestudy.pdf document for a guide on how to use this package for pan-genome analyses.

Author(s)

Lars Snipen and Kristian Hovde Liland.

Maintainer: Lars Snipen <lars.snipen@nmbu.no>

References

Snipen, L., Liland, KH. (2015). micropan: an R-package for microbial pan-genomics. BMC Bioinformatics, 16:79.

mpneumoniae *Data sets for the* Mycoplasma pneumoniae *casestudy*

Description

This data set contains several files with various objects related to the casestudy example used for illustration purposes in the micropan package.

Usage

```
data(Mpneumoniae.table)
data(Mpneumoniae.blast.distances)
data(Mpneumoniae.blast.clustering)
data(Mpneumoniae.blast.panmat)
data(Mpneumoniae.domain.clustering)
data(Mpneumoniae.domain.panmat)
```

Details

'Mpneumoniae.table' is a data.frame with 7 rows holding some information about the 7 genomes in the casestudy.

'Mpneumoniae.blast.distances' is a data.frame with 3 columns holding all computed BLAST distances between pairs of sequences in the 7 genomes. This data.frame has 139 543 rows.

'Mpneumoniae.blast.clustering' is a clustering vector of all the 9573 sequences in the genomes based on 'Mpneumoniae.blast.distances'.

'Mpneumoniae.blast.panmat' is a Panmat object containing a pan-matrix with 7 rows and 1210 columns based on 'Mpneumoniae.blast.clustering'.

'Mpneumoniae.domain.clustering' is a clustering vector of 5265 sequences in the genomes based on domain sequences. Notice that only sequences having at least one protein domain is considered here (5265 out of the total 9573).

'Mpneumoniae.domain.panmat' is a Panmat object containing a pan-matrix with 7 rows and 445 columns based on 'Mpneumoniae.domain.clustering'.

Author(s)

Lars Snipen and Kristian Hovde Liland.

Examples

```
# Genome overview table
data(Mpneumoniae.table) #loads the Mpneumoniae.table
if(interactive()){
   View(Mpneumoniae.table)
} else {
   str(Mpneumoniae.table)
}
```

```
# BLAST distances, only the first 20 are displayed
data(Mpneumoniae.blast.distances) #loads the Mpneumoniae.blast.distances
if(interactive()){
   View(Mpneumoniae.blast.distances[1:20,])
} else {
   str(Mpneumoniae.blast.distances[1:20,])
}

# BLAST clustering vector
data(Mpneumoniae.blast.clustering) #loads the Mpneumoniae.blast.clustering
print(Mpneumoniae.blast.clustering[1:30])

# BLAST pan-matrix
data(Mpneumoniae.blast.panmat) #loads the Mpneumoniae.blast.panmat
summary(Mpneumoniae.blast.panmat)

# Domain sequence clustering vector
data(Mpneumoniae.domain.clustering) #loads the Mpneumoniae.domain.clustering
print(Mpneumoniae.domain.clustering[1:30])

# Domain sequence pan-matrix
data(Mpneumoniae.domain.panmat) #loads the Mpneumoniae.domain.panmat
summary(Mpneumoniae.domain.panmat)
```

`orfLength` *ORF lengths*

Description

Computes the lengths of all ORFs in a `gff.table`.

Usage

```
orfLength(gff.table)
```

Arguments

`gff.table`	A `gff.table` (`data.frame`) with genomic features information.

Details

See `findOrfs` for more on `gff.tables`.

Value

A vector of ORF lengths, measured as the number of amino acids after translation.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

findOrfs, lorfs.

Examples

```
# See the example in the Help-file for findOrfs.
```

---

panMatrix *Computing the pan-matrix for a set of gene clusters*

Description

A pan-matrix has one row for each genome and one column for each gene cluster, and cell '[i,j]' indicates how many members genome 'i' has in gene family 'j'.

Usage

```
panMatrix(clustering)
```

Arguments

clustering	A vector of integers indicating the gene cluster for every sequence. Sequences with the same number belong to the same cluster. The name of each element is the tag identifying the sequence.

Details

The pan-matrix is a central data structure for pan-genomic analysis. It is a matrix with one row for each genome in the study, and one column for each gene cluster. Cell '[i,j]' contains an integer indicating how many members genome 'i' has in cluster 'j'.

The input clustering must be an integer vector with one element for each sequence in the study, typically produced by either bClust or dClust. The name of each element is a text identifying every sequence. The value of each element indicates the cluster, i.e. those sequences with identical values are in the same cluster. IMPORTANT: The name of each sequence must contain the GID-tag for each genome, i.e. they must of the form 'GID111_seq1', 'GID111_seq2',... where the 'GIDxxx' part indicates which genome the sequence belongs to. See panPrep for details.

The rows of the pan-matrix is named by the GID-tag for every genome. The columns are just named 'Cluster_x' where 'x' is an integer copied from 'clustering'.

Value

The returned object belongs to the class `Panmat`, which is a small (S3) extension to a matrix. It can be treated as a matrix, but the generic functions `plot.Panmat` and `summary.Panmat` are defined for a `Panmat` object. The input vector '`clustering`' is attached as the attribute '`clustering`' to the `Panmat` object.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

`bClust`, `dClust`, `distManhattan`, `distJaccard`, `fluidity`, `chao`, `binomixEstimate`, `heaps`, `rarefaction`.

Examples

```
# Loading clustering data in the micropan package
data(list=c("Mpneumoniae.blast.clustering","Mpneumoniae.domain.clustering"),package="micropan")

# Pan-matrix based on BLAST clustering
panmat.blast <- panMatrix(Mpneumoniae.blast.clustering)

# Pan-matrix based on domain sequence clustering
panmat.domains <- panMatrix(Mpneumoniae.domain.clustering)

# Plotting the first pan-matrix, and then printing its summary
plot(panmat.blast)
summary(panmat.blast)
```

---

panpca *Principal component analysis of a pan-matrix*

Description

Computes a principal component decomposition of a pan-matrix, with possible scaling and weightings.

Usage

```
panpca(pan.matrix, scale = 0, weights = rep(1, dim(pan.matrix)[2]))
```

Arguments

`pan.matrix`	A `Panmat` object, see `panMatrix` for details.
`scale`	An optional scale to control how copy numbers should affect the distances.
`weights`	Vector of optional weights of gene clusters.

Details

A principal component analysis (PCA) can be computed for any matrix, also a pan-matrix. The principal components will in this case be linear combinations of the gene clusters. One major idea behind PCA is to truncate the space, e.g. instead of considering the genomes as points in a high-dimensional space spanned by all gene clusters, we look for a few 'smart' combinations of the gene clusters, and visualize the genomes in a low-dimensional space spanned by these directions.

The 'scale' can be used to control how copy number differences play a role in the PCA. Usually we assume that going from 0 to 1 copy of a gene is the big change of the genome, and going from 1 to 2 (or more) copies is less. Prior to computing the PCA, the 'pan.matrix' is transformed according to the following affine mapping: If the original value in 'pan.matrix' is 'x', and 'x' is not 0, then the transformed value is '1 + (x-1)*scale'. Note that with 'scale=0.0' (default) this will result in 1 regardless of how large 'x' was. In this case the PCA only distinguish between presence and absence of gene clusters. If 'scale=1.0' the value 'x' is left untransformed. In this case the difference between 1 copy and 2 copies is just as big as between 1 copy and 0 copies. For any 'scale' between 0.0 and 1.0 the transformed value is shrunk towards 1, but a certain effect of larger copy numbers is still present. In this way you can decide if the PCA should be affected, and to what degree, by differences in copy numbers beyond 1.

The PCA can also up- or downweight some clusters compared to others. The vector 'weights' must contain one value for each column in 'pan.matrix'. The default is to use flat weights, i.e. all clusters count equal. See geneWeights for alternative weighting strategies.

The functions plotScores and plotLoadings can be used to visualize the results of panpca.

Value

A Panpca object is returned from this function. This is a small (S3) extension of a list with elements 'Evar', 'Scores', 'Loadings', 'Scale' and 'Weights'.

'Evar' is a vector with one number for each principal component. It contains the relative explained variance for each component, and it always sums to 1.0. This value indicates the importance of each component, and it is always in descending order, the first component being the most important. The 'Evar' is typically the first result you look at after a PCA has been computed, as it indicates how many components (directions) you need to capture the bulk of the total variation in the data.

'Scores' is a matrix with one column for each principal component and one row for each genome. The columns are ordered corresponding to the elements in 'Evar'. The scores are the coordinates of each genome in the principal component space. See plotScores for how to visualize genomes in the score-space.

'Loadings' is a matrix with one column for each principal component and one row for each gene cluster. The columns are ordered corresponding to the elements in 'Evar'. The loadings are the contribution from each original gene cluster to the principal component directions. NOTE: Only gene clusters having a non-zero variance is used in a PCA. Gene clusters with the same value for every genome have no impact and are discarded from the 'Loadings'. See plotLoadings for how to visualize gene clusters in the loading space.

'Scale' and 'Weights' are copies of the corresponding input arguments.

The generic functions plot.Panpca and summary.Panpca are available for Panpca objects.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

plotScores, plotLoadings, panTree, distManhattan, geneWeights.

Examples

```
# Loading two Panmat objects in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")

# Panpca based on a BLAST clustering Panmat object
ppca.blast <- panpca(Mpneumoniae.blast.panmat)
plot(ppca.blast) # The generic plot function
plotScores(ppca.blast) # A score-plot

# Panpca based on domain sequence clustering Panmat object
w <- geneWeights(Mpneumoniae.domain.panmat,type="shell")
ppca.domains <- panpca(Mpneumoniae.domain.panmat,scale=0.5,weights=w)
summary(ppca.domains)
plotLoadings(ppca.domains)
```

---

panPrep *Preparing FASTA files for pan-genomics*

Description

Preparing a FASTA file before starting comparisons of sequences in a pan-genome study.

Usage

```
panPrep(in.file, GID.tag, out.file, protein = TRUE, discard = NA)
```

Arguments

in.file	The name of a FASTA formatted file with protein or nucleotide sequences for coding genes in a genome.
GID.tag	The Genome IDentifier tag, see below.
out.file	Name of file where the prepared sequences will be written.
protein	Logical, indicating if the 'in.file' contains protein (TRUE) or nucleotide (FALSE) sequences.
discard	A text, a regular expression, and sequences having a match against this in their header text will be discarded.

Details

This function will read a FASTA file and produce another, slightly modified, FASTA file which is prepared for genome-wise comparisons using blastAllAll, hmmerScan or any other method.

The main purpose of panPrep is to make certain every sequence is labeled with a tag called a 'GID.tag' (Genome IDentifier tag) identifying the genome. This text contains the text "GID" followed by an integer. This integer can be any integer as long as it is unique to every genome in the study. It can typically be the BioProject number or any other integer that is uniquely related to a specific genome. If a genome has the text "GID12345" as identifier, then the sequences in the file produced by panPrep will have headerlines starting with "GID12345_seq1", "GID12345_seq2", "GID12345_seq3"...etc. This makes it possible to quickly identify which genome every sequence belongs to.

The 'GID.tag' is also added to the file name specified in 'out.file'. For this reason the 'out.file' must have a file extension containing letters only. By convention, we expect FASTA files to have one of the extensions '.fsa', '.faa', '.fa' or '.fasta'.

panPrep will also remove very short sequences (< 10 amino acids), removing stop codon symbols ('*'), replacing alien characters with 'X' and converting all sequences to upper-case. If the input 'discard' contains a regular expression, any sequences having a match to this in their headerline are also removed. Example: If we use prodigal to find proteins in a genome, partially predicted genes will have the text 'partial=10' or 'partial=01' in their headerline. Using 'discard="partial=01|partial=10"' will remove these from the data set.

Value

This function produces a FASTA formatted sequence file, and returns the name of this file.

Author(s)

Lars Snipen and Kristian Liland.

See Also

hmmerScan, blastAllAll.

Examples

```
# Using a protein file in the micropan package
xpth <- file.path(path.package("micropan"),"extdata")
prot.file <- file.path(xpth,"Example_proteins.fasta.xz")

# We need to uncompress it first...
tf <- tempfile(fileext=".xz")
s <- file.copy(prot.file,tf)
tf <- xzuncompress(tf)

# Prepping it, using the GID.tag "GID123"
out.file <- tempfile(fileext=".fasta")
prepped.file <- panPrep(tf,GID.tag="GID123",out.file)

# Reading the prepped file
```

```
prepped <- readFasta(prepped.file)
print(prepped$Header[1:5])

# ...and cleaning...
s <- file.remove(tf,prepped.file)
```

---

panTree *Constructing pan-genome trees*

Description

Creates a pan-genome tree based on a pan-matrix and a distance function.

Usage

```
panTree(pan.matrix, dist.FUN = distManhattan, nboot = 0,
  linkage = "average", ...)
```

Arguments

pan.matrix	A Panmat object, see panMatrix.
dist.FUN	A valid distance function, see below.
nboot	Number of bootstrap samples.
linkage	The linkage function, see below.
...	Additional parameters passed on to the specified distance function, see Details below.

Details

A pan-genome tree is a graphical display of the genomes in a pan-genome study, based on some pan-matrix (Snipen & Ussery, 2010). panTree is a constructor that computes a Pantree object, use plot.Pantree to actually plot the tree.

The parameter 'dist.FUN' must be a function that takes as input a numerical matrix (Panmat object) and returns a dist object. See distManhattan or distJaccard for examples of such functions. Any additional arguments ('...') are passed on to this function.

If you want to have bootstrap-values in the tree, set 'nboot' to some appropriate number (e.g. 'nboot=100').

The tree is created by hclust (hierarchical clustering) using the 'average' linkage function, which is according to Snipen & Ussery, 2010. You may specify alternatives by the parameter 'linkage', see hclust for details.

Value

This function returns a Pantree object, which is a small (S3) extension to a list with 4 components. These components are named 'Htree', 'Nboot', 'Nbranch' and 'Dist.FUN'.

'Htree' is a hclust object. This is the actual tree. 'Nboot' is the number of bootstrap samples. 'Nbranch' is a vector listing the number of times each split/clade in the tree was observed in the bootstrap procedure. 'Dist.FUN' is the name of the distance function used to construct the tree.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Snipen, L., Ussery, D.W. (2010). Standard operating procedure for computing pangenome trees. Standards in Genomic Sciences, 2:135-141.

See Also

panMatrix, distManhattan, distJaccard, plot.Pantree.

Examples

```
# Loading a Panmat object, constructing a tree and plotting it
data(list="Mpneumoniae.blast.panmat",package="micropan")
my.tree <- panTree(Mpneumoniae.blast.panmat)
plot(my.tree)

# Computing some weights to be used in the distManhattan
# function below...
w <- geneWeights(Mpneumoniae.blast.panmat,type="shell")
# Creating another tree with scaled and weighted distances and bootstrap values
my.tree <- panTree(Mpneumoniae.blast.panmat, scale=0.1, weights=w)

# ...and plotting with alternative labels and colors from Mpneumoniae.table
data(list="Mpneumoniae.table",package="micropan")
labels <- Mpneumoniae.table$Strain
names(labels) <- Mpneumoniae.table$GID.tag
cols <- Mpneumoniae.table$Color
names(cols) <- Mpneumoniae.table$GID.tag
plot(my.tree, leaf.lab=labels, col=cols,cex=0.8, xlab="Shell-weighted Manhattan distances")
```

---

plot.Binomix *Plot and summary of* Binomix *objects*

Description

Generic functions for Binomix objects.

Usage

```
## S3 method for class 'Binomix'
plot(x, type = "pan", cex = 2, ncomp = NA,
  show.bar = TRUE, ...)

## S3 method for class 'Binomix'
summary(object, ...)
```

Arguments

x	A Binomix object, see below.
type	Type of plot, default is 'type="pan"' which means the pie chart shows distribution over the entire pan-genome. The alternative is 'type="single"' which means the pie chart will show the distribution within a single (average) genome.
cex	Plot symbol scaling.
ncomp	Which model to display. You can override the display of the optimal (minimum BIC) model by specifying the number of components here, e.g. 'ncomp=5' will always display the model with '5' components regardless of its BIC value.
show.bar	Logical indicating if a colorbar should be displayed next to the pie.
...	Optional graphical arguments.
object	A Binomix object, see below.

Details

A Binomix object contains a series of fitted binomial mixture models. It is a small (S3) extension to a list, having two components. These are named 'BIC.table' and 'Mix.list', see binomixEstimate for more details.

The plot.Binomix function will display a Binomix object as a pie chart. Only the model with the smallest BIC-criterion value is displayed. The BIC-criterion is used to rank the various fitted models, and minimum BIC is an objective criterion for finding the best model complexity. Each sector of the pie chart is a component, the color of the sector indicates its detection probability and the size of the sector its mixing proportion. This pie chart illustrates how gene clusters are distributed within the pan-genome. Sectors of (dark) blue color are highly conserved gene clusters (core genes), sectors of greenish colors are medium conserved clusters (shell genes) and sectors of orange/pink colors are non-conserved clusters (cloud genes).

The summary.Binomix function will print the estimated core size and pan-genome size for the optimal component model.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

binomixEstimate.

Examples

```
# See examples in the Help-file for binomixEstimate.
```

`plot.Panmat` *Plot and summary of* `Panmat` *objects*

Description

Generic functions for plotting and printing the content of a `Panmat` object.

Usage

```
## S3 method for class 'Panmat'
plot(x, col = "black", xlab = "Number of genomes",
  ylab = "Number of clusters", ...)

## S3 method for class 'Panmat'
summary(object, ...)
```

Arguments

`x`	A `Panmat` object, see below.
`col`	The color, default is '`"black"`', of interior and borders of the bars in the barplot.
`xlab`	The label of the X axis.
`ylab`	The label of the Y axis.
`...`	Optional (graphical) arguments.
`object`	A `Panmat` object, see below.

Details

A `Panmat` object contains a pan-matrix, which is the fundamental data structure for pan-genome analyses. It is a small (S3) extension to a `matrix`. It has one row for each genome in the study, and one column for each gene cluster. The number in cell '`[i,j]`' is the number of sequences in genome '`i`' that belongs to cluster '`j`'. A `Panmat` object is typically created by the function `panMatrix`.

The `plot.Panmat` function will display the content of the `Panmat` object as a bar chart showing the number of clusters found in 1,2,...,G genomes, where G is the total number of genomes in the study (rows in '`Panmat`').

The `summary.Panmat` function will display a text giving the same information as `plot.Panmat`.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panMatrix.

Examples

```
# See examples in the Help-file for panMatrix.
```

---

plot.Panpca *Plot and summary of* Panpca *objects*

Description

Generic functions for Panpca objects.

Usage

```
## S3 method for class 'Panpca'
plot(x, cum = FALSE, col = "black", ...)

## S3 method for class 'Panpca'
summary(object, ...)
```

Arguments

x	A Panpca object, see below.
cum	Logical, default is 'FALSE', indicating if explained variance should be plotted per component or cumulative.
col	Color, default is '"black"', of interior and border of bars in the barplot.
...	Optional graphical arguments.
object	A Panpca object, see below.

Details

A Panpca object contains the results from a principal component analysis (PCA) on a pan-matrix, and is the output from the function panpca. It is a small (S3) extension of a list, and contains the elements 'Evar', 'Scores', 'Loadings', 'Scale' and 'Weights'.

The basic idea of a PCA is to find alternative directions in the space spanned by the pan-matrix columns, in order to be able to visualize or in other ways extract the most relevant information in a small number of dimensions. The variable 'Evar' contains the explained variance for each principal component, scaled such that summed over all components it is 1.0. This quantity indicates the importance of each component, larger values of 'Evar' indicates directions (components) with more information.

The plot.Panpca function shows the 'Evar' values in a barplot. You can either plot the 'Evar' value of each component separately ('cum=FALSE') or the cumulative value ('cum=TRUE'). This is

the basic plot to follow any principal component decomposition, since it tells you how many components you need to capture the bulk of the information in the data. If e.g. component 1, 2 and 3 have 'Evar' values of 0.4, 0.3 and 0.2, respectively, it means these three direction capture 90% (0.4+0.3+0.2=0.9) of all the variation in the data. For some pan-matrices almost all variation can be found in the very few first directions, but more often it is scattered between many. See plotScores and plotLoadings for other informative graphical displays of a Panpca object.

The summary.Panpca function will print the same information as plotted by plot.Panpca.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panpca, plotScores, plotLoadings.

Examples

```
# See examples in the Help-file for panpca.
```

---

plot.Pantree *Plot and summary of* Pantree *objects*

Description

Generic functions for Pantree objects.

Usage

```
## S3 method for class 'Pantree'
plot(x, leaf.lab = NULL, col = "black", xlab = "",
  main = "", cex = 1, show.boot = TRUE, ...)

## S3 method for class 'Pantree'
summary(object, ...)
```

Arguments

x	A Pantree object, see below.
leaf.lab	Alternative labels for the leaves, see below.
col	Color of the leaf labels, see below.
xlab	Text for the x-axis (distance-axis) of the plotted tree.
main	Title above the plotted tree.
cex	Scaling of the leaf labels of the plotted tree.
show.boot	Logical to turn off plotting of bootstrap values.
...	Additional arguments, see below.
object	A Pantree object, see below.

Details

A Pantree object is created by panTree and contains information to display a pan-genome tree. The plot.Pantree function will display the tree as a dendrogram object.

The argument 'leaf.lab' can be used to give alternative labels, the GID-tags are used by default. 'leaf.lab' must be a vector of labels, one for each genome in the Pantree. The labels may be in any order, but the vector must be named by the GID-tags, i.e. each element in 'leaf.lab' must have a name which is a valid GID-tag for some genome. This is necessary to ensure the alternative labels are placed correctly in the tree.

The argument 'col' specifies the color(s) of the leaf labels in the tree. It can either be a single color or a vector of colors, one for each leaf label (genome). Again, the colors may be in any order, but the vector must be named by the GID-tags, i.e. each element in 'col' must have a name which is a valid GID-tag for some genome.

The argument 'cex' scales the leaf label font size.

The argument 'show.boot' can be used to turn off the display of bootstrap values. Note that if the tree was constructed without bootstrapping, no bootstrap values are available, and this argument has no effect.

Any additional arguments are passed on to the plot.dendrogram function.

summary.Pantree prints a short text describing the Pantree object.

Note

Using 'nodePar' to manipulate the dendrogram object will have no effect on the leaf nodes here since these are set by the dendrapply function. The tree is always displayed horizontal, to align the labels in the right margin for easy reading.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panTree.

Examples

```
# See examples in the Help-file for panTree.
```

---

plot.Rarefac *Plot and summary of* Rarefac *objects*

Description

Generic functions for Rarefac object.

Usage

```
## S3 method for class 'Rarefac'
plot(x, type = "b", pch = 16, xlab = "Genomes",
  ylab = "Number of unique gene clusters", ...)

## S3 method for class 'Rarefac'
summary(object, ...)
```

Arguments

x	A Rarefac object, see below.
type	Type of plot, default is '"b"', giving markers with lines between.
pch	Marker type, default is '16', a filled circle.
xlab	Text for horizontal axis.
ylab	Text for vertical axis.
...	Optional graphical arguments.
object	A Rarefac object, see below.

Details

A Rarefac object is a small (S3) extension to a matrix. The first column contains the cumulative number of unique gene clusters found when considering 1,2,...,G genomes in a pan-matrix. Thus, the Rarefac object is a matrix with G rows. Any additional columns will hold similar numbers, but for random shufflings of the genome's ordering. A Rarefac object is typically created by the function rarefaction.

The plot.Rarefac function will display the content of the Rarefac object as a plot of the mean value in rows 1,2,...,G, where G is the total number of genomes in the study.

The summary.Rarefac function will display a text giving the same information as plot.Rarefac.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

rarefaction, heaps.

Examples

```
# See examples in the Help-file for rarefaction.
```

plotScores *Plotting scores and loadings in a* Panpca *object*

Description

Creates informative plots for a principal component analysis of a pan-matrix.

Usage

```
plotScores(pan.pca, x = 1, y = 2, show.labels = TRUE, labels = NULL,
  col = "black", pch = 16, ...)

plotLoadings(pan.pca, x = 1, y = 2, show.labels = TRUE, col = "black",
  pch = 16, ...)
```

Arguments

pan.pca	A Panpca object, see panpca for details.
x	The component to display along the horizontal axis.
y	The component to display along the vertical axis.
show.labels	Logical indicating if labels should be displayed.
labels	Alternative labels to use in the score-plot, see below.
col	Colors for the points/labels, see below.
pch	Marker type, see points.
...	Additional arguments passed on to points or text (if labels are specified).

Details

A Panpca object contains the results of a principal component analysis on a pan-matrix, see panpca for details.

The plotScores gives a visual overview of how the genomes are positioned relative to each other in the pan-genome space. The score-matrix of a Panpca has one row for each genome. The original pan-matrix also has one row for each genome. Two genomes can be compared by their corresponding rows in the pan-matrix, but can also be compared by their rows in the score-matrix, and the latter matrix has (much) fewer columns designed to contain maximum of the original data variation. A plot of the scores will give an approximate overview of how the genomes are located relative to each other.

The plotLoadings gives a visual overview of how the gene clusters affect the principal components. The loadings is a matrix with one row for each of the original non-core gene clusters (core gene clusters have no variation across genomes). Clusters located close to the origin have little impact. Clusters far from the origin has high impact, indicating they separate groups of genomes.

These two plots together can reveal information about the pan-genome: The score-plot shows if genomes are grouped/separated, and the loading-plot can then tell you which gene clusters have high impact on this grouping/separation.

The arguments 'x' and 'y' can be used to plot other components than component 1 and 2 (which is always the most informative). In some cases more components are needed to establish a good picture, i.e. the explained variance is low for component 1 and 2 (see plot.Panpca for more on explained variance). It is quite common to plot component 1 versus 2, then 1 versus 3 and finally 2 versus 3.

The argument 'show.labels' can be used to turn off the display of labels, only markers (dots) will appear.

In plotScores you can specify alternative labels in 'labels'. By default, the GID-tag is used for each genome. You can supply a vector of alternative labels. The labels may be in any order, but the vector must be named by the GID-tags, i.e. each element in 'labels' must have a name which is a valid GID-tag for some genome. This is necessary to ensure the alternative labels are placed correctly in the score-space.

There is no alternative labelling of loading-plots, since the gene clusters lack a GID-tag-like system. You can, however, change the gene cluster names by editing the column names of the pan-matrix directly before you do the panpca.

You may color each label/marker individually. In plotScores you can again supply a vector of colors, and name every element with a GID-tag to make certain they are used correctly. In plotLoadings you can supply a vector of colors, but you must arrange them in proper order yourself.

Additional arguments are passed on to text if 'show.labels=TRUE' and to points if 'show.labels=FALSE'.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

panpca, plot.Panpca.

Examples

```
# Loading a Panmat object in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")
ppca.blast <- panpca(Mpneumoniae.blast.panmat)

# Plotting scores and loadings
plotScores(ppca.blast) # A score-plot
plotLoadings(ppca.blast) # A loading plot

# Plotting score with alternative labels and colors
data(list="Mpneumoniae.table",package="micropan")
labels <- Mpneumoniae.table$Strain
names(labels) <- Mpneumoniae.table$GID.tag
cols <- Mpneumoniae.table$Color
names(cols) <- Mpneumoniae.table$GID.tag
plotScores(ppca.blast,labels=labels,col=cols)
```

prodigal *Gene predictions using Prodigal*

Description

Finds coding genes in a genome using the Prodigal software.

Usage

```
prodigal(genome.file, prot.file = NULL, nuc.file = NULL,
  closed.ends = TRUE, motif.scan = FALSE)
```

Arguments

genome.file	Name of a FASTA formatted file with all the DNA sequences for a genome (chromosomes, plasmids, contigs etc.).
prot.file	If specified, amino acid sequence of each protein is written to this FASTA file.
nuc.file	If specified, nucleotide sequence of each protein is written to this FASTA file.
closed.ends	Logical, if TRUE genes are not allowed to run off edges (default TRUE).
motif.scan	Logical, if TRUE forces motif scan instead of Shine-Dalgarno trainer (default FALSE).

Details

This function sets up a call to the software Prodigal (Hyatt et al, 2009). This software is designed to find coding genes in prokaryote genomes. It runs fast and has obtained very good results in tests among the automated gene finders. The options used as default here are believed to be the best for pan-genomic analyses.

Value

A gff.table with the metadata for all predicted genes (see readGFF). If prot.file is specified, a FASTA formatted file with predicted protein sequences are also produced. If nuc.file is specified, a similar file with nucleotide sequences is also produced.

Note

The Prodigal software must be installed on the system for this function to work, i.e. the command 'system("prodigal -h")' (no version numbers!) must be recognized as a valid command if you run it in the Console window.

Author(s)

Lars Snipen and Kristian Hovde Liland.

References

Hyatt, D., Chen, G., LoCascio, P.F., Land, M.L., Larimer, F.W., Hauser, L.J. (2010). Prodigal: prokaryotic gene recognition and translation initiation site identification, BMC Bioinformatics, 11:119.

See Also

readGFF.

Examples

```
## Not run:
# This example requires the external Prodigal software
# Using a genome file in this package
xpth <- file.path(path.package("micropan"),"extdata")
genome.file <- file.path(xpth,"Example_genome.fasta.xz")

# We need to uncompress it first...
tf <- tempfile(fileext=".xz")
s <- file.copy(genome.file,tf)
tf <- xzuncompress(tf)

# Calling Prodigal, and writing all predicted proteins to a file as well
prot.file <- tempfile(fileext=".fasta")
gff.table <- prodigal(tf,prot.file)

# Reading protein file as well
proteins <- readFasta(prot.file)

# ...and cleaning...
s <- file.remove(tf,prot.file)

## End(Not run)
```

---

rarefaction *Rarefaction curves for a pan-genome*

Description

Computes rarefaction curves for a number of random permutations of genomes.

Usage

```
rarefaction(pan.matrix, n.perm = 1)
```

Arguments

pan.matrix	A Panmat object, see panMatrix for details.
n.perm	The number of random genome orderings to use. If 'n.perm=1' the fixed order of the genomes in 'pan.matrix' is used.

Details

A rarefaction curve is simply the cumulative number of unique gene clusters we observe as more and more genomes are being considered. The shape of this curve will depend on the order of the genomes. This function will typically compute rarefaction curves for a number of ('n.perm') orderings. By using a large number of permutations, and then averaging over the results, the effect of any particular ordering is smoothed away.

The averaged curve illustrates how many new gene clusters we observe for each new genome. If this levels out and becomes flat, it means we expect few, if any, new gene clusters by sequencing more genomes. The function heaps can be used to estimate population openness based on this principle.

Value

This function returns a Rarefac object, which is a small extension to a matrix. The generic functions plot.Rarefac and summary.Rarefac are available for such objects.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

heaps, panMatrix, plot.Rarefac, summary.Rarefac.

Examples

```
# Loading two Panmat objects in the micropan package
data(list=c("Mpneumoniae.blast.panmat","Mpneumoniae.domain.panmat"),package="micropan")

# Rarefaction based on a BLAST clustering Panmat object
rarefac.blast <- rarefaction(Mpneumoniae.blast.panmat,n.perm=100)
plot(rarefac.blast)

# Rarefaction based on domain sequence clustering Panmat object
rarefac.domains <- rarefaction(Mpneumoniae.domain.panmat,n.perm=1000)
summary(rarefac.domains)
```

readBlastTable *Reading BLAST result file*

Description

Reading a file produced by the BLAST+ software set up to produce tabular output.

Usage

```
readBlastTable(blast.file)
```

Arguments

blast.file	Name of file to read.

Details

This function will read files produced by the BLAST+ software where the option '`-outfmt 6`' has been invoked during its call. This option forces BLAST to produce a short tabular text output for each BLAST search. The function `blastAllAll` produces such files.

Value

The content of the file is returned as a '`data.frame`' with 12 columns and one row for each BLAST result. The columns have self-explanatory names.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

`blastAllAll`, `bDist`.

Examples

```
# Using a BLAST result file in this package
xpth <- file.path(path.package("micropan"),"extdata")
blast.file <- file.path( xpth, "GID1_vs_GID2.txt.xz" )

# We need to uncompress it first...
tf <- tempfile(fileext=".xz")
s <- file.copy(blast.file,tf)
tf <- xzuncompress(tf)

#...then we can read it...
blast.table <- readBlastTable(tf)

# ...and deleting temporary file
```

```
s <- file.remove(tf)
```

`readGFF` *Reading and writing GFF-tables*

Description

Reading or writing a `gff.table` from/to file.

Usage

```
readGFF(in.file)
writeGFF(gff.table, out.file)
```

Arguments

`in.file`	Name of file with a GFF-table.
`gff.table`	A `gff.table` (`data.frame`) with genomic features information.
`out.file`	Name of file.

Details

A `gff.table` is simply a `data.frame` with columns adhering to the format specified by the GFF3 format, see https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md for details. There is one row for each feature.

The following columns should always be in a full `gff.table` of the GFF3 format:

- Seqid. A unique identifier of the genomic sequence on which the feature resides.
- Source. A description of the procedure that generated the feature, e.g. `"R-package micropan::findOrfs"`.
- Type The type of feature, e.g. `"ORF"`, `"16S"` etc.
- Start. The leftmost coordinate. This is the start if the feature is on the Sense strand, but the end if it is on the Antisense strand.
- End. The rightmost coordinate. This is the end if the feature is on the Sense strand, but the start if it is on the Antisense strand.
- Score. A numeric score (E-value, P-value) from the `Source`.
- Strand. A `"+"` indicates Sense strand, a `"-"` Antisense.
- Phase. Only relevant for coding genes. the values 0, 1 or 2 indicates the reading frame, i.e. the number of bases to offset the `Start` in order to be in the reading frame.
- Attributes. A single string with semicolon-separated tokens prociding additional information.

Missing values are described by `"."` in the GFF3 format. This is also done here, except for the numerical columns Start, End, Score and Phase. Here `NA` is used, but this is replaced by `"."` when writing to file.

Value

readGFF returns a gff.table with the columns described above.

writeGFF writes the supplied gff.table to a text-file.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

findOrfs, lorfs.

Examples

```
# Using a GFF file in this package
xpth <- file.path(path.package("micropan"),"extdata")
gff.file <- file.path(xpth,"Example.gff.xz")

# We need to uncompress it first...
tf <- tempfile(fileext=".xz")
s <- file.copy(gff.file,tf)
tf <- xzuncompress(tf)

# Reading, finding signature, and writing...
gff.table <- readGFF(tf)
print(gffSignature(gff.table))
out.tf <- tempfile(fileext=".gff")
writeGFF(gff.table[1:3,], out.tf)

# ...and cleaning...
s <- file.remove(tf,out.tf)
```

---

readHmmer *Reading results from a HMMER3 scan*

Description

Reading a text file produced by hmmerScan.

Usage

```
readHmmer(hmmer.file, e.value = 1, use.acc = TRUE)
```

Arguments

hmmer.file	The name of a hmmerScan result file.
e.value	Numeric threshold, hits with E-value above this are ignored (default is 1.0).
use.acc	Logical indicating if accession numbers should be used to identify the hits.

Details

The function reads a text file produced by hmmerScan. By specifying a smaller 'e.value' you filter out poorer hits, and fewer results are returned. The option 'use.acc' should be turned off (FALSE) if you scan against your own database where accession numbers are lacking.

Value

The results are returned in a 'data.frame' with columns 'Query', 'Hit', 'Evalue', 'Score', 'Start', 'Stop', 'Description'. 'Query' is the tag identifying each sequence in each genome, typically 'GID111_seq1', 'GID121_seq3', etc. 'Hit' is the name or accession number for a pHMM in the database describing patterns. The 'Evalue' is the 'ievalue' in the HMMER3 terminology. The 'Score' is the HMMER3 score for the match between 'Query' and 'Hit'. The 'Start' and 'Stop' are the positions within the 'Query' where the 'Hit' (pattern) starts and stops. 'Description' is the description of the 'Hit'.

There is one line for each hit.

Author(s)

Lars Snipen and Kristian Hovde Liland.

See Also

hmmerScan, hmmerCleanOverlap, dClust.

Examples

```
# See the examples in the Help-files for dClust and hmmerScan.
```

---

xzcompress *Compressing and uncompressing text files*

Description

These functions are adapted from the R.utils package from gzip to xz. Internally xzfile() (see connections) is used to read (write) chunks to (from) the xz file. If the process is interrupted before completed, the partially written output file is automatically removed.

Usage

```
xzcompress(filename, destname = sprintf("%s.xz", filename),
  temporary = FALSE, skip = FALSE, overwrite = FALSE, remove = TRUE,
  BFR.SIZE = 1e+07, compression = 6, ...)

xzuncompress(filename, destname = gsub("[.]xz$", "", filename, ignore.case =
  TRUE), temporary = FALSE, skip = FALSE, overwrite = FALSE,
  remove = TRUE, BFR.SIZE = 1e+07, ...)
```

Arguments

filename	Path name of input file.
destname	Pathname of output file.
temporary	If TRUE, the output file is created in a temporary directory.
skip	If TRUE and the output file already exists, the output file is returned as is.
overwrite	If TRUE and the output file already exists, the file is silently overwritting, otherwise an exception is thrown (unless skip is TRUE).
remove	If TRUE, the input file is removed afterward, otherwise not.
BFR.SIZE	The number of bytes read in each chunk.
compression	The compression level used (1-9).
...	Not used.

Value

Returns the pathname of the output file. The number of bytes processed is returned as an attribute.

Author(s)

Kristian Hovde Liland.

Examples

```
# Creating small file
tf <- tempfile()
cat(file=tf, "Hello world!")

# Compressing
tf.xz <- xzcompress(tf)
print(file.info(tf.xz))

# Uncompressing
tf <- xzuncompress(tf.xz)
print(file.info(tf))
file.remove(tf)
```

Index