# Package ‘micropan’

October 13, 2022

**Type** Package

**Title** Microbial Pan-Genome Analysis

**Version** 2.1

**Date** 2020-07-15

**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 (>= 3.5.0), microseq, dplyr, stringr, igraph

**Imports** tibble, rlang

**LazyData** FALSE

**ZipData** TRUE

**RoxygenNote** 7.1.0

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-07-15 13:30:03 UTC

## R topics documented:

---

`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.tbl, linkage = "complete", threshold = 0.75, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `dist.tbl` | A `tibble` with pairwise distances. |
| `linkage` | A text indicating what type of clustering to perform, either ‘`complete`’ (default), ‘`average`’ or ‘`single`’. |
| `threshold` | Specifies the maximum size of a cluster. Must be a distance, i.e. a number between 0.0 and 1.0. |
| `verbose` | Logical, turns on/off text output during computations. |

## 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 `dist.tbl` where only the relevant distances are listed. The columns ‘`Query`’ and ‘`Hit`’ contain tags identifying pairs of sequences. The column ‘`Distance`’ contains the distances, always a number from 0.0 to 1.0. Typically, this is the output from `bDist`. All pairs of sequences *not* listed are assumed to have distance 1.0, which is considered the 'infinite' distance. All sequences must be listed at least once in ceither column '`Query`' or '`Hit`' of the `dist.tbl`. This should pose no problem, since all sequences must have distance 0.0 to themselves, and should be listed with this distance once ('`Query`' and '`Hit`' containing the same tag).

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 '`Query`' or '`Hit`' columns of the input '`dist.tbl`'. The name of each element indicates the sequence. The numerical values have no meaning as such, they are simply categorical indicators of cluster membership.

## Author(s)

Lars Snipen and Kristian Hovde Liland.

## See Also

`bDist`, `hclust`, `dClust`, `isOrtholog`.

## Examples

```
# Loading example BLAST distances
data(xmpl.bdist)

# Clustering with default settings
clst <- bClust(xmpl.bdist)
# Other settings, and verbose
clst <- bClust(xmpl.bdist, linkage = "average", threshold = 0.5, verbose = TRUE)
```

## bDist *Computes distances between sequences*

### Description

Computes distance between all sequences based on the BLAST bit-scores.

### Usage

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

### Arguments

| | |
|---|---|
| `blast.files` | A text vector of BLAST result filenames. |
| `blast.tbl` | A table with BLAST results. |
| `e.value` | A threshold E-value to immediately discard (very) poor BLAST alignments. |
| `verbose` | Logical, indicating if textual output should be given to monitor the progress. |

### Details

The essential input is either a vector of BLAST result filenames (`blast.files`) or a table of the BLAST results (`blast.tbl`). It is no point in providing both, then `blast.tbl` is ignored.

For normal sized data sets (e.g. less than 100 genomes), you would provide the BLAST filenames as the argument `blast.files` to this function. Then results are read, and distances are computed. Only if you have huge data sets, you may find it more efficient to read the files using `readBlastSelf` and `readBlastPair` separately, and then provide as the argument `blast.tbl]` the table you get from binding these results. In all cases, the BLAST result files must have been produced by `blastpAllAll`.

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 based on alignment bitscores. Assume the alignment of query A against hit B has a bitscore of S(A,B). The distance is D(A,B)=1-2*S(A,B)/(S(A,A)+S(B,B)) where S(A,A) and S(B,B) are the self-alignment bitscores, i.e. the scores of aligning against itself. A distance of 0.0 means A and B are identical. The maximum possible distance is 1.0, meaning there is no BLAST between A and B.

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 table with columns '`Dbase`', '`Query`', '`Bitscore`' and '`Distance`'. Each row corresponds to a pair of sequences (Dbase and Query sequences) 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

blastpAllAll, readBlastSelf, readBlastPair, bClust, isOrtholog.

## Examples

```
# Using BLAST result files in this package...
prefix <- c("GID1_vs_GID1_",
            "GID2_vs_GID1_",
            "GID3_vs_GID1_",
            "GID2_vs_GID2_",
            "GID3_vs_GID2_",
            "GID3_vs_GID3_")
bf <- file.path(path.package("micropan"), "extdata", str_c(prefix, ".txt.xz"))

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

# Computing pairwise distances
blast.dist <- bDist(blast.files)

# Read files separately, then use bDist
self.tbl <- readBlastSelf(blast.files)
pair.tbl <- readBlastPair(blast.files)
blast.dist <- bDist(blast.tbl = bind_rows(self.tbl, pair.tbl))

# ...and cleaning...
ok <- file.remove(blast.files)

# See also example for blastpAl
```

---

# 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 pan-matrix, 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 list with two components, the 'BIC.tbl' and 'Mix.tbl'.

The 'BIC.tbl' is a tibble 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 component number ('K'). The number of components where 'BIC' is minimized is the optimum. If minimum 'BIC' is reached for the largest 'K' value you should extend the 'K.range' to larger values and re-fit. The function will issue a warning to remind you of this.

The 'Mix.tbl' is a tibble with estimates from the mixture models. The column 'Component' indicates the model, i.e. all rows where 'Component' has the same value are from the same model. There will be 3 rows for 3-component model, 4 rows for 4-component, etc. The column 'Detection.prob' contain the estimated detection probabilities for each component of the mixture models. A 'Mixing.proportion' is the proportion of the gene clusters having the corresponding 'Detection.prob', i.e. if core genes have 'Detection.prob' 1.0, the corresponding 'Mixing.proportion' (same row) indicates how large fraction of the gene families are core genes.

### 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.

### Examples

```
# Loading an example pan-matrix
data(xmpl.panmat)

# Estimating binomial mixture models
binmix.lst <- binomixEstimate(xmpl.panmat, K.range = 3:8)
```

```
print(binmix.lst$BIC.tbl) # minimum BIC at 3 components

## Not run:
# The pan-genome gene distribution as a pie-chart
library(ggplot2)
ncomp <- 3
binmix.lst$Mix.tbl %>%
  filter(Components == ncomp) %>%
  ggplot() +
  geom_col(aes(x = "", y = Mixing.proportion, fill = Detection.prob)) +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "Pan-genome gene distribution") +
  scale_fill_gradientn(colors = c("pink", "orange", "green", "cyan", "blue"))

# The distribution in an average genome
binmix.lst$Mix.tbl %>%
  filter(Components == ncomp) %>%
  mutate(Single = Mixing.proportion * Detection.prob) %>%
  ggplot() +
  geom_col(aes(x = "", y = Single, fill = Detection.prob)) +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "Average genome gene distribution") +
  scale_fill_gradientn(colors = c("pink", "orange", "green", "cyan", "blue"))

## End(Not run)
```

## blastpAllAll *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.

### Usage

```
blastpAllAll(
  prot.files,
  out.folder,
  e.value = 1,
  job = 1,
  threads = 1,
  start.at = 1,
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| `prot.files` | A vector with FASTA filenames. |
| `out.folder` | The folder where the result files should end up. |
| `e.value` | The chosen E-value threshold in BLAST. |
| `job` | An integer to separate multiple jobs. |
| `threads` | The number of CPU's to use. |
| `start.at` | An integer to specify where in the file-list to start BLASTing. |
| `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 BLASTing. This function uses the ‘`blast+`’ software available for free from NCBI (Camacho et al, 2009). More precisely, the blastp algorithm with the BLOSUM45 scoring matrix and all composition based statistics turned off.

A vector listing FASTA files of protein sequences is given as input in ‘`prot.files`’. These files must have the genome_id 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`.

In the first version of this package we used reciprocal BLASTing, i.e. we computed both genome A against B and B against A. This may sometimes produce slightly different results, but in reality this is too costly compared to its gain, and we now only make one of the above searches. This basically halves the number of searches. This step is still very time consuming for larger number of genomes. Note that the protein files are sorted by the genome_id (part of filename) inside this function. This is to ensure a consistent ordering irrespective of how they are enterred.

For every pair of genomes a result file is produced. If two genomes have genome_id's ‘`GID111`’, and ‘`GID222`’ then the result file ‘`GID222_vs_GID111.txt`’ will be found in ‘`out.folder`’ after the completion of this search. The last of the two genome_id is always the first in alphabetical order of the two.

The ‘`out.folder`’ is scanned for already existing result files, and `blastpAllAll` 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. By starting multiple R sessions, you can speed up the search by running `blastpAllAll` from each R session, using the same ‘`out.folder`’ but different integers for the `job` option. At the same time you may also want to start the BLASTing at different places in the file-list, by giving larger values to the argument `start.at`. This is 1 by default, i.e. the BLASTing starts at the first protein file. If you are using a multicore computer you can also increase the number of CPUs by increasing `threads`.

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

## Value

The function produces a result file for each pair of files listed in ‘prot.files’. These result files are located in out.folder. Existing files are never overwritten by blastpAllAll, 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, bDist.

## Examples

```
## Not run:
# This example requires the external BLAST+ software
# Using protein files in this package
pf <- file.path(path.package("micropan"), "extdata",
                str_c("xmpl_GID", 1:3, ".faa.xz"))

# We need to uncompress them first...
prot.files <- tempfile(fileext = c("_GID1.faa.xz","_GID2.faa.xz","_GID3.faa.xz"))
ok <- file.copy(from = pf, to = prot.files)
prot.files <- unlist(lapply(prot.files, xzuncompress))

# Blasting all versus all
out.dir <- "."
blastpAllAll(prot.files, out.folder = out.dir)

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

# ...and cleaning...
ok <- file.remove(prot.files)
ok <- file.remove(file.path(out.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 pan-matrix, 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 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 this.

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 pan-matrix in this package
data(xmpl.panmat)

# Estimating the pan-genome size using the Chao estimator
chao.pansize <- chao(xmpl.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.tbl)
```

## Arguments

| | |
|---|---|
| hmmer.tbl | A tibble 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 tibble 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.tbl’ 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.tbl’. 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

```
# HMMER3 result files in this package
hf <- file.path(path.package("micropan"), "extdata",
                str_c("GID", 1:3, "_vs_microfam.hmm.txt.xz"))

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

# Reading the HMMER3 results, cleaning overlaps...
hmmer.tbl <- NULL
for(i in 1:3){
  readHmmer(hmm.files[i]) %>%
    hmmerCleanOverlap() %>%
    bind_rows(hmmer.tbl) -> hmmer.tbl
}

# The clustering
clst <- dClust(hmmer.tbl)

# ...and cleaning...
ok <- file.remove(hmm.files)
```

## distJaccard *Computing Jaccard distances between genomes*

### Description

Computes the Jaccard distances between all pairs of genomes.

### Usage

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

### Arguments

`pan.matrix` A pan-matrix, see `panMatrix` for details.

### Details

The Jaccard index between two sets is defined as the size of the intersection 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 a pan-matrix in this package
data(xmpl.panmat)

# Jaccard distances
Jdist <- distJaccard(xmpl.panmat)

# Making a dendrogram based on the distances,
# see example for distManhattan
```

---

## `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, ncol(pan.matrix)))
```

## Arguments

| | |
|---|---|
| `pan.matrix` | A pan-matrix, 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`.

## Examples

```
# Loading a pan-matrix in this package
data(xmpl.panmat)

# Manhattan distances between genomes
Mdist <- distManhattan(xmpl.panmat)

## Not run:
# Making a dendrogram based on shell-weighted distances
library(ggdendro)
weights <- geneWeights(xmpl.panmat, type = "shell")
Mdist <- distManhattan(xmpl.panmat, weights = weights)
ggdendrogram(dendro_data(hclust(Mdist, method = "average")),
  rotate = TRUE, theme_dendro = FALSE) +
 labs(x = "Genomes", y = "Shell-weighted Manhattan distance", title = "Pan-genome dendrogram")

## End(Not run)
```

# 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.

The downloaded sequences are saved in ‘out.file’ on your system. This will be a FASTA formatted file. Note that all downloaded sequences end up in this file. If you want to download multiple genomes, you call entrezDownload multiple times and store in multiple files.

## Value

The name of the resulting FASTA file is returned (same as out.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

```
## Not run:
# Accession numbers for the chromosome and plasmid of Buchnera aphidicola, strain APS
acc <- "BA000003.2,AP001071.1"
genome.file <- tempfile(pattern = "Buchnera_aphidicola", fileext = ".fna")
txt <- entrezDownload(acc, out.file = genome.file)

# ...cleaning...
ok <- file.remove(genome.file)

## End(Not run)
```

# 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 pan-matrix, 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 vector 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 a pan-matrix in this package
data(xmpl.panmat)

# Fluidity based on this pan-matrix
fluid <- fluidity(xmpl.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 pan-matrix, 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

```
# See examples for distManhattan
```

## getAccessions *Collecting contig accession numbers*

### Description

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

### Usage

```
getAccessions(master.record.accession, chunk.size = 99)
```

### 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.

`chunk.size`
: The maximum number of accession numbers returned in one text.

### 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`.

The download API at NCBI will not tolerate too many accessions per query, and for this reason you need to split the accessions for many contigs into several texts using `chunk.size`.

### Value

A character vector where each element is a text listing the accession numbers separated by comma. Each vector element will contain no more than `chunk.size` 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

```
## Not run:
# 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
genome.file <- tempfile(fileext = ".fna")
```

```
txt <- entrezDownload(acc, out.file = genome.file)

# ...cleaning...
ok <- file.remove(genome.file)

## End(Not run)
```

---

## 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 pan-matrix, 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 pangenome. Current Opinions in Microbiology, 12:472-477.

### See Also

binomixEstimate, chao, rarefaction.

### Examples

```
# Loading a pan-matrix in this package
data(xmpl.panmat)

# Estimating population openness
h.est <- heaps(xmpl.panmat, n.perm = 500)
print(h.est)
# If alpha < 1 it indicates an open pan-genome
```

## hmmerCleanOverlap *Removing overlapping hits from HMMER3 scans*

### Description

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

### Usage

```
hmmerCleanOverlap(hmmer.tbl)
```

### Arguments

| | |
|---|---|
| hmmer.tbl | A table (tibble) 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 tibble of the type produced by readHmmer.

### Value

A tibble which is a subset of the input, where some rows may 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, dbase, out.folder, threads = 0, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `in.files` | A character vector of file names. |
| `dbase` | The full path-name of the database to scan (text). |
| `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 require the external software HMMER
# Using example files in this package
pf <- file.path(path.package("micropan"), "extdata", "xmpl_GID1.faa.xz")
dbf <- file.path(path.package("micropan"), "extdata",
                 str_c("microfam.hmm", c(".h3f.xz",".h3i.xz",".h3m.xz",".h3p.xz")))

# We need to uncompress them first...
prot.file <- tempfile(pattern = "GID1.faa", fileext=".xz")
ok <- file.copy(from = pf, to = prot.file)
prot.file <- xzuncompress(prot.file)
db.files <- str_c(tempfile(), c(".h3f.xz",".h3i.xz",".h3m.xz",".h3p.xz"))
ok <- file.copy(from = dbf, to = db.files)
db.files <- unlist(lapply(db.files, xzuncompress))
db.name <- str_remove(db.files[1], "\\.[a-z0-9]+$")
```

```
# Scanning the FASTA file against microfam.hmm...
hmmerScan(in.files = prot.file, dbase = db.name, out.folder = ".")

# Reading results
hmm.file <- file.path(".", str_c("GID1_vs_", basename(db.name), ".txt"))
hmm.tbl <- readHmmer(hmm.file)

# ...and cleaning...
ok <- file.remove(prot.file)
ok <- file.remove(str_remove(db.files, ".xz"))

## 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.tbl)
```

### 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.tbl` A `tibble` with pairwise distances. The columns '`Query`' and '`Hit`' 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.tbl` 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 and their clustering results
data(list = c("xmpl.bdist","xmpl.bclst"))

# Finding orthologs
is.ortholog <- isOrtholog(xmpl.bclst, xmpl.bdist)
# The orthologs are
which(is.ortholog)

## End(Not run)
```

# 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.

## 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.

## 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 named vector of integers. |

### 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 a named 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 '`genome_id`' 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 '`genome_id`' for every genome. The columns are just named '`Cluster_x`' where '`x`' is an integer copied from '`clustering`'.

### Value

An integer matrix with a row for each genome and a column for each sequence cluster. The input vector '`clustering`' is attached as the attribute '`clustering`'.

### Author(s)

Lars Snipen and Kristian Hovde Liland.

### See Also

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

## Examples

```
# Loading clustering data in this package
data(xmpl.bclst)

# Pan-matrix based on the clustering
panmat <- panMatrix(xmpl.bclst)

## Not run:
# Plotting cluster distribution
library(ggplot2)
tibble(Clusters = as.integer(table(factor(colSums(panmat > 0), levels = 1:nrow(panmat)))),
       Genomes = 1:nrow(panmat)) %>%
ggplot(aes(x = Genomes, y = Clusters)) +
geom_col()

## End(Not run)
```

---

# 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, ncol(pan.matrix)))
```

## Arguments

| | |
|---|---|
| pan.matrix | A pan-matrix, 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 may 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 list with three tables:

‘Evar.tbl’ has two columns, one listing the component number and one listing the relative explained variance for each component. The relative explained variance always sums to 1.0 over all components. This value indicates the importance of each component, and it is always in descending order, the first component being the most important. This 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.tbl’ has a column listing the ‘GID.tag’ for each genome, and then one column for each principal component. The columns are ordered corresponding to the elements in ‘Evar’. The scores are the coordinates of each genome in the principal component space.

‘Loadings.tbl’ is similar to ‘Scores.tbl’ but contain values for each gene cluster instead of each genome. The columns are ordered corresponding to the elements in ‘Evar’. The loadings are the contributions from each 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’.

## Author(s)

Lars Snipen and Kristian Hovde Liland.

## See Also

distManhattan, geneWeights.

## Examples

```
# Loading a pan-matrix in this package
data(xmpl.panmat)

# Computing panPca
ppca <- panPca(xmpl.panmat)

## Not run:
# Plotting explained variance
library(ggplot2)
ggplot(ppca$Evar.tbl) +
  geom_col(aes(x = Component, y = Explained.variance))
# Plotting scores
ggplot(ppca$Scores.tbl) +
  geom_text(aes(x = PC1, y = PC2, label = GID.tag))
# Plotting loadings
ggplot(ppca$Loadings.tbl) +
  geom_text(aes(x = PC1, y = PC2, label = Cluster))

## End(Not run)
```

## panPrep *Preparing FASTA files for pan-genomics*

### Description

Preparing a FASTA file before starting comparisons of sequences.

### Usage

```
panPrep(in.file, genome_id, out.file, protein = TRUE, min.length = 10, discard = "")
```

### Arguments

| | |
|---|---|
| in.file | The name of a FASTA formatted file with protein or nucleotide sequences for coding genes in a genome. |
| genome_id | The Genome Identifier, 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. |
| min.length | Minimum sequence length |
| discard | A text, a regular expression, and sequences having a match against this in their headerline will be discarded. |

### Details

This function will read the in.file and produce another, slightly modified, FASTA file which is prepared for the comparisons using blastpAllAll, hmmerScan or any other method.

The main purpose of panPrep is to make certain every sequence is labeled with a tag called a 'genome_id' identifying the genome from which it comes. 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. 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 ‘genome_id’ 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 sequences shorter than min.length, 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 the prodigal software (see findGenes) 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, blastpAllAll.

### Examples

```
# Using a protein file in this package
# We need to uncompress it first...
pf <- file.path(path.package("micropan"),"extdata","xmpl.faa.xz")
prot.file <- tempfile(fileext = ".xz")
ok <- file.copy(from = pf, to = prot.file)
prot.file <- xzuncompress(prot.file)

# Prepping it, using the genome_id "GID123"
prepped.file <- panPrep(prot.file, genome_id = "GID123", out.file = tempfile(fileext = ".faa"))

# Reading the prepped file
prepped <- readFasta(prepped.file)
head(prepped)

# ...and cleaning...
ok <- file.remove(prot.file, prepped.file)
```

## 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 pan-matrix, 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.

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

A table with the curves in the columns. The first column is the number of genomes, while all other columns are the cumulative number of clusters, one column for each permutation.

## Author(s)

Lars Snipen and Kristian Hovde Liland.

## See Also

`heaps`, `panMatrix`.

## Examples

```
# Loading a pan-matrix in this package
data(xmpl.panmat)

# Rarefaction
rar.tbl <- rarefaction(xmpl.panmat, n.perm = 1000)

## Not run:
# Plotting
library(ggplot2)
library(tidyr)
rar.tbl %>%
  gather(key = "Permutation", value = "Clusters", -Genome) %>%
  ggplot(aes(x = Genome, y = Clusters, group = Permutation)) +
    geom_line()
```

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

## `readBlastSelf` *Reads BLAST result files*

### Description

Reads files from a search with blastpAllAll

### Usage

```
readBlastSelf(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. |
| `verbose` | Logical, indicating if textual output should be given to monitor the progress. |

### Details

The filenames given as input must refer to BLAST result files produced by `blastpAllAll`.

With `readBlastSelf` you only read the self-alignment results, i.e. blasting a genome against itself. With `readBlastPair` you read all the other files, i.e. different genomes compared. You may use all blast file names as input to both, they will select the proper files based on their names, e.g. GID1_vs_GID1.txt is read by `readBlastSelf` while GID2_vs_GID1.txt is read by `readBlastPair`.

Setting a small '`e.value`' threshold will filter the alignment, and may speed up this and later processing, but you may also loose some important alignments for short sequences.

Both these functions are used by `bDist`. The reason we provide them separately is to allow the user to complete this file reading before calling `bDist`. If you have a huge number of files, a skilled user may utilize parallell processing to speed up the reading. For normal size data sets (e.g. less than 100 genomes) you should probably use `bDist` directly.

### Value

The functions returns a table with columns '`Dbase`', '`Query`', '`Bitscore`' and '`Distance`'. Each row corresponds to a pair of sequences (a Dbase and a Query sequence) having at least one BLAST hit between them. All pairs *not* listed have distance 1.0 between them. You should normally bind the output from `readBlastSelf` to the ouptut from `readBlastPair` and use the result as input to `bDist`.

### Author(s)

Lars Snipen.

## See Also

bDist, blastpAllAll.

## Examples

```
# Using BLAST result files in this package...
prefix <- c("GID1_vs_GID1_",
            "GID2_vs_GID1_",
            "GID3_vs_GID1_",
            "GID2_vs_GID2_",
            "GID3_vs_GID2_",
            "GID3_vs_GID3_")
bf <- file.path(path.package("micropan"), "extdata", str_c(prefix, ".txt.xz"))

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

# Reading self-alignment files, then the other files
self.tbl <- readBlastSelf(blast.files)
pair.tbl <- readBlastPair(blast.files)

# ...and cleaning...
ok <- file.remove(blast.files)

# See also examples for bDist
```

# 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 'tibble' with columns 'Query', 'Hit', 'Evalue', 'Score', 'Start', 'Stop' and 'Description'. 'Query' is the tag identifying each query sequence. '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.
```

# xmpl *Data sets for use in examples*

## Description

This data set contains several files with various objects used in examples in some of the functions in the micropan package.

## Usage

```
data(xmpl.bdist)
data(xmpl.bclst)
data(xmpl.panmat)
```

## Details

'xmpl.bdist' is a tibble with 4 columns holding all BLAST distances between pairs of proteins in an example with 10 small genomes.

'xmpl.bclst' is a clustering vector of all proteins in the genomes from 'xmpl.bdist'.

'xmpl.panmat' is a pan-matrix with 10 rows and 1210 columns computed from 'xmpl.bclst'.

### Author(s)

Lars Snipen and Kristian Hovde Liland.

### Examples

```
# BLAST distances, only the first 20 are displayed
data(xmpl.bdist)
head(xmpl.bdist)

# Clustering vector
data(xmpl.bclst)
print(xmpl.bclst[1:30])

# Pan-matrix
data(xmpl.panmat)
head(xmpl.panmat)
```

## 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