Package ‘TRAMPR’

June 9, 2016

Title 'TRFLP' Analysis and Matching Package for R
Date 2016-05-20
Version 1.0-8
Author Rich FitzJohn & Ian Dickie
Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>
License GPL-2
Depends R (>= 2.4)
Description Matching terminal restriction fragment length polymorphism ('TRFLP') profiles between unknown samples and a database of known samples. TRAMPR facilitates analysis of many unknown profiles at once, and provides tools for working directly with electrophoresis output through to generating summaries suitable for community analyses with R's rich set of statistical functions. TRAMPR also resolves the issues of multiple 'TRFLP' profiles within a species, and shared 'TRFLP' profiles across species.
NeedsCompilation no
Repository CRAN
Date/Publication 2016-06-09 13:17:58

R topics documented:

TRAMPR-package .......................................................... 2
add.known ............................................................... 3
build.knowns ................................................................ 6
combine ........................................................................... 8
combine.TRAMPsamples ..................................................... 8
create.diffsmatrix .......................................................... 9
demo.knowns ............................................................... 11
demo.samples .............................................................. 11
group.knowns .................................................................. 12
load.abi ........................................................................ 16
Description

This package contains a collection of functions to help analyse terminal restriction fragment length polymorphism (TRFLP) profiles, by matching unknown peaks to known TRFLP profiles in order to identify species.

The TRAMPR package contains a vignette, which includes a worked example; type vignette("TRAMPRdemo") to view it. To see all documented help topics, type library(help=TRAMPR).

Details

Start by reading the TRAMP (and perhaps create.diffsmatrix) help pages, which explain the matching algorithm.

Then read load.abi to learn how to load ABI format data into the program. Alternatively, read TRAMPSamples and read.TRAMPSamples to load already-processed data.

If you already have a collection of knowns, read TRAMPknowns and read.TRAMPknowns to learn how to load them. Otherwise, read build.knowns to learn how to automatically generate a set of known profiles from your data.

Once your data are loaded, reread TRAMP to do the analysis, then read plot.TRAMP and summary.TRAMP to examine the analysis. update.TRAMP may also be useful for modifying your matches. summary.TRAMP is also useful for preparing presence/absence matrices for analysis with other tools (e.g. the vegan package; see the vignette indicated below).

TRAMPR works with database-like objects, and a basic understanding of relational databases and primary/foreign keys will aid in understanding some aspects of the package.

Citation

Please see citation("TRAMPR") for the citation of TRAMPR.
Note

TRAMPR is designed specifically for “database TRFLP” (identifying species based on a database of known TRFLP profiles: see Dicke et al. 2002. It is not designed for direct community analysis of TRFLP profiles as in peak-profile TRFLP.

Author(s)

Rich FitzJohn and Ian Dickie, Landcare Research

References


add.known

Add Knowns To TRAMPknowns Databases

Description

Add a single known or many knowns to a knowns database in a TRAMPknowns object. add.known takes a TRAMPknowns object, and adds the peak profile of a single sample from a TRAMPSamples object. combine.TRAMPknowns combines two TRAMPknowns objects (similar to combine.TRAMPSamples). add.known and combine are generic, so if x argument is a TRAMP object, then the knowns component of that object will be updated.

Usage

add.known(x, ...)
## S3 method for class 'TRAMPknowns'
add.known(x, samples, sample.fk, prompt=TRUE, default.species=NULL, ...)
## S3 method for class 'TRAMP'
add.known(x, sample.fk, rebuild=TRUE, ...)

## S3 method for class 'TRAMPknowns'
combine(x, y, rewrite.knowns.pk=FALSE, ...)
## S3 method for class 'TRAMP'
combine(x, y, rebuild=TRUE, ...)
Arguments

x
A TRAMPknowns or TRAMP object, containing identified TRFLP patterns.
samples
A TRAMPSamples object, containing unidentified samples.
sample.fk
sample.fk of sample in samples to add to the knowns database. If x is a TRAMP object, then sample.fk refers to a sample in the TRAMPSamples object used in the creation of that TRAMP object (stored as x$samples: see labels(x$samples) for codes).

prompt
Logical: Should the function interactively prompt for a new species name?

default.species
Default species name. If NULL (the default), the name chosen will be the value of samples$info$species for the current sample. Set to NA if no name is currently known (see group.knowns - identical non-NA names are considered related).

y
A second TRAMPknowns object, containing knowns to add to x.

rewrite.knowns.pk
Logical: If the new knowns data contain knowns.pk values that conflict with those in the original TRAMPknowns object, should the new knowns be renumbered? If this is TRUE, do not rely on any knowns.pk values staying the same for the newly added knowns. knowns.pk values in the original TRAMPknowns object will never be changed.

rebuild
Logical: should the TRAMP object be rebuilt after adding knowns, by running rebuild.TRAMP on it? This is important to determine if the new known(s) match any of the samples in the TRAMP object. This should be left as TRUE unless you plan on manually rebuilding the object later.

... Additional arguments passed to future methods.

Details

(add.known only): When adding the profile of a single individual via add.known, if more than one peak per enzyme/primer combination is present we select the most likely profile by picking the highest peak (largest height value) for each enzyme/primer combination (a warning will be given). If two peaks are of the same height, then the peak taken is unspecified (similar to build.knowns with min.ratio=0).

(Combine only): rewrite.knowns.pk provides a simple way of merging knowns databases that use the same values of knowns.pk. Because knowns.pk must be unique, if y (the new knowns database) uses knowns.pk values present in x (the original database), then the knowns.pk values in y must be rewritten. This will be done by adding max(labels(x)) to every knowns.pk value in y$info and knowns.pk value in y$data.

If retaining knowns.pk information is important, we suggest saving the value of knowns.pk before running this function, e.g.

info$knowns.pk.old <- info$knowns.pk

If more control over the renaming process is required, manually adjust y$info$knowns.pk yourself before calling this function. However, by default no translation will be done, and an error will occur if x and y share knowns.pk values.

For add.known, only a subset of columns are passed to the knowns object (a future version may be more inclusive):
• From samples$info: sample.pk (as knowns.pk.)
• From samples$data: sample.fk (as knowns.fk), primer, enzyme, size.

For combine, the data and info elements of the resulting TRAMPknowns object will have the union of the columns present in both sets of knowns. If any additional elements exist as part of the second TRAMPknowns object (e.g. passed as ... to TRAMPknowns when creating y), these will be ignored.

Value

An object of the same class as x: if a TRAMP object is supplied, a new TRAMP object with an updated TRAMPknowns component will be returned, and if the object is a TRAMPknowns object an updated TRAMPknowns object will be returned.

Note

If the TRAMPknowns object has a file.pat element (see TRAMPknowns), then the new knowns database will be written to file. This may be confusing when operating on TRAMP objects directly, since both the TRAMPknowns object used in the TRAMP object and the original TRAMPknowns object will share the same file.pat argument, but contain different data as soon as add.known or combine is used. In short - be careful! To avoid this issue, either set file.pat to NULL before using add.known or combine.

See Also

build.knowns, which automatically builds a knowns database, and TRAMPknowns, which documents the object containing the knowns database.

combine.TRAMPSamples, which combines a pair of TRAMPSamples objects.

Examples

data(demo.knowns)
data(demo.samples)

## (1) Using add.known(), to add a single known:

## Sample "101" looks like a potential known, add it to our knowns
## database:
plot(demo.samples, 101)

## Add this to a knowns database:
## Because there is more than one peak per enzyme/primer combination, a
## warning will be given. In this case, since there are clear peaks it
## is harmless.
demo.knowns.2 <- add.known(demo.knowns, demo.samples, 101,
    prompt=FALSE)

## The known has been added:
demo.knowns.2[101]
try(demo.knowns[101]) # error - known didn't exist in original knowns

## Same, but adding to an existing TRAMP object.
res <- TRAMP(demo.samples, demo.knowns)
plot(res, 101)
res2 <- add.known(res, 101, prompt=FALSE, default.species="New known")

## Now the new known matches itself.
plot(res2, 101)

## (2) Using combine() to combine knowns databases.

## Let's split the original knowns database in two:
demo.knowns.a <- demo.knowns[head(labels(demo.knowns), 10)]
demo.knowns.b <- demo.knowns[tail(labels(demo.knowns), 10)]

## Combining these is easy:
demo.knowns.c <- combine(demo.knowns.a, demo.knowns.b)

## Knowns from both the small database are present in the new one:
identical(c(labels(demo.knowns.a), labels(demo.knowns.b)),
labels(demo.knowns.c))

## Demonstration of knowns rewriting:
demo.knowns.d <- demo.knowns.a
demo.knowns.a$info$from <- "a"
demo.knowns.d$info$from <- "d"

try(combine(demo.knowns.a, demo.knowns.d)) # error
demo.knowns.e <- combine(demo.knowns.a, demo.knowns.d,
rewrite.knowns.pk=TRUE)

## See that both data sets are here (check the "from" column).
demo.knowns.e$info

## Note that a better approach in might be to manually resolve
## conflicting knowns.pk values before combining.

build.knowns          Automatically Build Knowns Database

Description

This function uses several filters to select likely knowns, and construct a TRAMPknowns object from a TRAMPSamples object. Samples are considered to be “potential knowns” if they have data for an adequate number of enzyme/primer combinations, and if for each combination they have either a single peak, or a peak that is “distinct enough” from any other peaks.

Usage

build.knowns(d, min.ratio=3, min.comb=NA, restrict=FALSE, ...)
Arguments

- **d**
  - A TRAMPsamples object, containing samples from which to build the knowns database.

- **min.ratio**
  - Minimum ratio of maximum to second highest peak to accept known (see Details).

- **min.comb**
  - Minimum number of enzyme/primer combinations required for each known (see Details for behaviour of default).

- **restrict**
  - Logical: Use only cases where d$\text{info}\$species is non-blank? (These are assumed to come from samples of a known species. However, it is not guaranteed that all samples with data for \texttt{species} will become knowns; if they fail either the min.ratio or min.comb checks they will be excluded.)

- **...**
  - Additional arguments passed to TRAMPknowns (e.g. cluster.pars, file.pat and any additional objects).

Details

For all samples and enzyme/primer combinations, the ratio of the largest to the second largest peak is calculated. If it is greater than min.ratio, then that combination is accepted. If the sample has at least min.comb valid enzyme/primer combinations, then that sample is included in the knowns database. If min.comb is NA (the default), then every enzyme/primer combination present in the data is required.

Value

A new TRAMPknowns object. It will generally be necessary to edit this object; see \texttt{read.TRAMPknowns} for details on how to write, edit, and read back a modified object.

Note

If two peaks have the same height, then using min.ratio=1 will not allow the entry as part of the knowns database; use min.ratio=0 instead if this is desired. In this case, the peak chosen is unspecified.

Note that this function is sensitive to data quality. In particular split peaks may cause a sample not to be added. These samples may be manually added using \texttt{add.known}.

Examples

```r
data(demo.samples)
demo.knowns.auto <- build.knowns(demo.samples, min.comb=4)
plot(demo.knowns.auto, cex=.75)
```
combine.TRAMPsamples

Combine Two Objects

Description
This function is used to combine TRAMPsamples together, and to combine TRAMPknowns to TRAMPknowns or TRAMP objects. combine is generic; please see combine.TRAMPsamples and combine.TRAMPknowns for more information.

Usage
combine(x, y, ...)

Arguments
x, y Objects to be combined. See combine.TRAMPsamples and combine.TRAMPknowns for more information.
...
Additional arguments required by methods.

See Also
See combine.TRAMPsamples and combine.TRAMPknowns for more information.

combine.TRAMPsamples
Combine TRAMPsamples Objects

Description
Combines two TRAMPsamples objects into one large TRAMPsamples object containing all the samples for both original objects.

Usage
## S3 method for class 'TRAMPsamples'
combine(x, y, rewrite.sample.pk=FALSE, ...)

Arguments
x, y TRAMPsamples objects, containing TRFLP patterns.
rewrite.sample.pk Logical: If the new sample data (y) contains sample.pk values that conflict with those in the original TRAMPsamples object (x), should the new samples be renumbered? If this is TRUE, do not rely on any sample.pk values staying the same for the newly added samples. sample.pk values in the original TRAMPsamples object will never be changed.
...
Further arguments passed to or from other methods.
create.diffsmatrix

Calculate Matrix of Distances between Peaks

Description

Generate an array of goodness-of-fit (or distance) between samples and knowns based on the sizes (in base pairs) of TRFLP peaks. For each sample/known combination, and for each enzyme/primer combination, this calculates the minimum distance between any peak in the sample and the single peak in the known.
create.diffsmatrix

Usage

create.diffsmatrix(samples, knowns)

Arguments

samples A TRAMPsamples object, containing unidentified samples.
knowns A TRAMPknowns object, containing identified TRFLP patterns.

Details

This function will rarely need to be called directly, but does most of the calculations behind TRAMP, so it is useful to understand how this works.

This function generates a three-dimensional \( s \times k \times n \) matrix of the (smallest, see below) distance in base pairs between peaks in a collection of unknowns (run data) and a database of knowns for several enzyme/primer combinations. \( s \) is the number of different samples in the samples data \((\text{length}(labels(samples)))\), \( k \) is the number of different types in the knowns database \((\text{length}(labels(knowns)))\), and \( n \) is the number of different enzyme/primer combinations. The enzyme/primer combinations used are all combinations present in the knowns database; combinations present only in the samples will be ignored. Not all samples need contain all enzyme/primer combinations present in the knowns.

In the resulting array, \( m_{i,j,k} \) is the difference (in base pairs) between the \( i \)th sample and the \( j \)th known for the \( k \)th enzyme/primer combination. The ordering of the \( n \) enzyme/primer combinations is arbitrary, so a data.frame of combinations is included as the attribute enzyme.primer, where enzyme.primer$enzyme[k] and enzyme.primer$primer[k] correspond to enzyme and primer used for the distances in \( m_{i,j,k} \).

Each case in the knowns database has a single (or no) peak for each enzyme/primer combination, but each sample may contain multiple peaks for an enzyme/primer combination; the difference is always the smallest distance from the sample to the known peak. Where a sample and/or a known lacks an enzyme/primer combination, the value of the difference is \( \text{NA} \). The smallest \textit{absolute} distance is taken between sample and known peaks, but the sign of the difference is preserved (negative where the closest sample peak was less than the known peak, positive where greater; see \texttt{absolute.min}).

Value

A three-dimensional matrix, with an attribute enzyme.primer, described above.

See Also

TRAMP, which uses output from create.diffsmatrix.

Examples

data(demo.samples)
data(demo.knowns)
s <- length(labels(demo.samples))
k <- length(labels(demo.knowns))
demo.knowns

```r
n <- nrow(unique(demo.knowns$data[,c("enzyme", "primer")]))

m <- create.diffsmatrix(demo.samples, demo.knowns)

dim(m)
identical(dim(m), c(s, k, n))

## Maximum error for each sample/known (i.e. across all enzyme/primer
## combinations), similar to how calculated by `tramp`
error <- apply(abs(m), 1:2, max, na.rm=TRUE)
dim(error)

## Euclidian error (see `tramp`)
error.euclid <- sqrt(rowSums(m^2, TRUE, 2)/rowSums(!is.na(m), dims=2)

## Euclidian and maximum error will require different values of
## accept.error in TRAMP:
plot(error, error.euclid, pch=".")
```

demo.knowns  Demonstration Knowns Database

Description

A knowns database, for demonstrating the TRAMP package. This
is a subset of a full knowns database, and not intended to
represent any real data set, and should not be assumed to be accurate.

The data are stored as a `TRAMPknowns` object. Columns in the
info and data components are described on the `TRAMPknowns` page.

Usage

data(demo.knowns)

Licence

This data set is provided under a Creative Commons “Attribution-NonCommercial-NoDerivs 2.5”
licence. Please see http://creativecommons.org/licenses/by-nc-nd/2.5/ for details.

demo.samples  Demonstration Samples Database
Description

A samples database, for demonstrating the TRAMPR package. This is a subset of a full samples database, is not intended to represent any real data set, and should not be assumed to be accurate.

The data are stored as a TRAMPSamples object. Columns in the info and data components are described on the TRAMPSamples page, but with some additions:

- **info:**
  - soilcore.fk: Key to the soil core from which a sample came. See soilcore, below.

- **data:**
  - sample.file.name: Original .fsa file corresponding to the TRFLP run. This is included in all TRAMPSamples objects created by load.abi.

- **soilcore:** A data.frame with information about the soilcore from which samples came.
  - soilcore.pk: Key, distinguishing soil cores.
  - plot: Plot number (1 to 10).
  - elevation: Height above mean sea level, in metres.
  - east: Easting (New Zealand Map Grid/NZMG).
  - north: Northing (NZMG).
  - vegetation: Vegetation type (Nothofagus solandri or Pinus contorta).

Usage

data(demo.samples)

Format

A TRAMPSamples object.

Licence

This data set is provided under a Creative Commons “Attribution-NonCommercial-NoDerivs 2.5” licence. Please see http://creativecommons.org/licenses/by-nc-nd/2.5/ for details.

---

Description

Group a TRAMPknowns object so that knowns with similar TRFLP patterns and knowns that share the same species name “group” together. In general, this function will be called automatically whenever appropriate (e.g. when loading a data set or adding new knowns). Please see Details to understand why this function is necessary, and how it works.

The main reason for manually calling group.knowns is to change the default values of the arguments; if you call group.knowns on a TRAMPknowns object, then any subsequent automatic call to group.knowns will use any arguments you passed in the manual group.knowns call (e.g. after doing group.knowns(x, cut.height=20), all future groupings will use cut.height=20).
Usage

```r
  group.knowns(x, ...)  # S3 method for class 'TRAMPknowns'
  group.knowns(x, dist.method, hclust.method, cut.height, ...)  # S3 method for class 'TRAMP'
  group.knowns(x, ...)  # S3 method for class 'TRAMP'
```

Arguments

- `x` A `TRAMPknowns` or `TRAMP` object, containing identified TRFLP patterns.
- `dist.method` Distance method used in calculating similarity between different knowns (see `dist`). Valid options include "maximum", "euclidian" and "manhattan".
- `hclust.method` Clustering method used in generating clusters from the similarity matrix (see `hclust`).
- `cut.height` Passed to `cutree`; controls how similar members of each group should be (the larger `cut.height`, the more inclusive knowns groups will be).
- `...` Arguments passed to further methods.

Details

`group.knowns` groups together knowns in a `TRAMPknowns` object based on two criteria: (1) TRFLP profiles that are very similar across shared enzyme/primer combinations (based on clustering) and (2) TRFLP profiles that belong to the same species (i.e. share a common species column in the `info` data.frame of `x`; see `TRAMPknowns` for more information). This is to solve three issues in TRFLP analysis:

1. The TRFLP profile of a single species can have variation in peak sizes due to DNA sequence variation. By including multiple collections of each species, variation in TRFLP profiles can be accounted for. If a `TRAMPknowns` object contains multiple collections of a species, these will be aggregated by `group.knowns`. This aggregation is essential for community analysis, as leaving individual collections will artificially inflate the number of “present species” when running `TRAMP`.
   Some authors have taken an alternative approach by using a larger tolerance in matching peaks between samples and knowns (effectively increasing `accept.error` in `TRAMP`) to account for within-species variation. This is not recommended, as it dramatically increases the risk of incorrect matches.

2. Distinctly different TRFLP profiles may occur within a species (or in some cases within an individual); see Avis et al. (2006). `group.knowns` looks at the species column of the `info` data.frame of `x` and joins any knowns with identical `species` values as a group.
   This can also be used where multiple profiles are present in an individual.

3. Different species may share a similar TRFLP profile and therefore be indistinguishable using TRFLP. If these patterns are not grouped, two species will be recorded as present wherever either is present. `group.knowns` prevents this by joining knowns with “very similar” TRFLP patterns as a group. Ideally, these problematic groups can be resolved by increasing the number of enzyme/primer pairs in the data.
Groups names are generated by concatenating all unique (sorted) species names together, separated by commas.

To determine if knowns are “similar enough” to form a group, we use R’s clustering tools: dist, hclust and cutree. First, we generate a distance matrix of the knowns profiles using dist, and using method dist.method (see Example below; this is very similar to what TRAMP does, and dist.method should be specified accordingly). We then generate clusters using hclust, and using method hclust.method, and “cut” the tree at cut.height using cutree.

Knowns are grouped together iteratively; so that all groups sharing a common cluster are grouped together, and all knowns that share a common species name are grouped together. In certain cases this may chain together seemingly unrelated groups.

Because group.knowns is generic, it can be run on either a TRAMPknowns or a TRAMP object. When run on a TRAMP object, it updates the TRAMPknowns object (stored as x$knowns), so that subsequent calls to plot.TRAMPknowns or summary.TRAMPknowns (for example) will use the new grouping parameters.

Parameters set by group.knowns are retained as part of the object, so that when adding additional knowns (add.known and combine), or when subsetting a knowns database (see [.TRAMPknowns, aka TRAMPindexing), the same grouping parameters will be used.

Value

For group.knowns.TRAMPknowns, a new TRAMPknowns object. The cluster.pars element will have been updated with new parameters, if any were specified.

For group.knowns.TRAMP, a new TRAMP object, with an updated knowns element. Note that the original TRAMPknowns object (i.e. the one from which the TRAMP object was constructed) will not be modified.

Warning

Warning about missing data: where there are NA values in certain combinations, NAs may be present in the final distance matrix, which means we cannot use hclust to generate the clusters! In general, NA values are fine. They just can’t be everywhere.

References


See Also

TRAMPknowns, which describes the TRAMPknowns object.
build.knowns, which attempts to generate a knowns database from a TRAMPSamples data set.
plot.TRAMPknowns, which graphically displays the relationships between knowns.
Examples

data(demo.knowns)
data(demo.samples)

demo.knowns <- group.knowns(demo.knowns, cut.height=2.5)
plot(demo.knowns)

## Increasing cut.height makes groups more inclusive:
plot(group.knowns(demo.knowns, cut.height=100))

res <- TRAMP(demo.samples, demo.knowns)
m1.ungrouped <- summary(res)
m1.grouped <- summary(res, group=TRUE)
ncol(m1.grouped) # 94 groups

res2 <- group.knowns(res, cut.height=100)
m2.ungrouped <- summary(res2)
m2.grouped <- summary(res2, group=TRUE)
ncol(m2.grouped) # Now only 38 groups

## group.knowns results in the same distance matrix as produced by
## TRAMP, therefore using the same method (e.g. method="maximum") is
## important. The example below shows how the matrix produced by
## dist(summary(x)) (as calculated by group.knowns) is the same as that
## produced by TRAMP:

f <- function(x, method="maximum") {
  ## Create a pseudo-samples object from our knowns
  y <- x
  y$data$height <- 1
  names(y$info)[names(y$info) == "knowns.pk"] <- "sample.pk"
  names(y$data)[names(y$data) == "knowns fk"] <- "sample fk"
  class(y) <- "TRAMPSamples"
  
  ## Run TRAMP, clean up and return
  ## (If method != "maximum", rescale the error to match that
  ## generated by dist()).
  z <- TRAMP(y, x, method=method)
  if (method != "maximum") z$error <- z$error * z$n
  names(dimnames(z$error)) <- NULL
  z
}

g <- function(x, method="maximum")
  as.matrix(dist(summary(x), method=method))

all.equal(f(demo.knowns, "maximum")$error, g(demo.knowns, "maximum"))
all.equal(f(demo.knowns, "euclidian")$error, g(demo.knowns, "euclidian"))
all.equal(f(demo.knowns, "manhattan")$error, g(demo.knowns, "manhattan"))

## However, TRAMP is over 100 times slower in this special case.
system.time(f(demo.knowns))
system.time(g(demo.knowns))
Description

These functions help convert data from Applied Biosystems Gene Mapper (ABI) output format into TRAMPsamples objects for analysis. Note that this operates on the summarised output (a text file), rather than the .fsa files containing data for individual runs.

Details of the procedure of this function are given below, and a worked example is given in the package vignette; type vignette("TRAMPdemo") to view it.

The function peakscanner.to.genemapper is an experimental function to convert from peakscanner output to abi genemapper output. The peakscanner output is very slightly different in format, and currently load.abi is very fussy about the input file’s structure. Eventually load.abi will be made more tolerant, but as an interim solution, run peakscanner.to.genemapper on your file. By default, running peakscanner.to.genemapper(myfile.csv) will produce a file myfile.txt. This can then be loaded using load.abi as described below, specifying myfile.txt as the file argument.

Usage

load.abi(file, file.template, file.info, primer.translate, ...)
load.abi.create.template(file, file.template)
load.abi.create.info(file, file.template, file.info)

peakscanner.to.genemapper(filename, output)

Arguments

file The name of the file from which the ABI data are to be read from.
file.template The name of the file containing the “template” file (see Details).
file.info (Optional) the name of the file containing extra information associated with each sample (see Details).
primer.translate List used to translate dye codes into primers. The same codes are assumed to apply across the whole file. See Details for format.
... Additional objects to incorporate into a TRAMPsamples object. See TRAMPsamples for details.
filename In peakscanner.to.genemapper, the name of the csv file containing output.
output In peakscanner.to.genemapper, the name of the file to be output in abi format (if omitted, this will be automatically generated).
Details

Some terminology: a “sample” refers to a physical sample (e.g., a root tip), while a “run” refers to an individual TRFLP run (i.e., one enzyme and one primer). Because two primers are run at once, each “runfile” contains information on two “runs”, but each “sample” may contain more than one “runfile”. Runfiles are distinguished by different sample.file.name values in the ABI file, while different samples are distinguished by different sample_fk/sample_pk values.

primer.translate is a list used to translate between the dyes recorded in the ABI file and the primers used. Each element corresponds to a different primer, and is a vector of different colour dyes. The list:

```
list(ITS1F="B", ITS4="G")
```

would translate all dyes with the value "B" to "ITS1F", and all dyes with the value "G" to "ITS4". The list:

```
list(ITS1F="B", ITS4=c("G", "Y"))
```

would do the same, except that both "G" and "Y" dyes would be converted to "ITS4". If a dye is used in the data that is not represented within primer.translate, then it will be excluded (e.g., all rows of data with dye as "R" will be excluded).

The procedure for loading in ABI data is:

1. Create the “template” file. Template files are required to record which enzymes were used for each run, since that is not included in the ABI output, and to group together separate runs (typically different enzymes) that apply to the same individual. The function load.abi.create.template will create a template that contains all the unique file names found in the ABI file (as sample.file.name), and blank columns titled enzyme and sample.index. Running

```
load.abi.create.template(x)
```

where `x` is the name of your ABI file will create a template file in the same directory as the ABI file. The function will print the name and location of the template file to the console.

2. Edit the template file and save. The enzyme and sample.index columns are initially empty and need filling in, which can be done in Excel, or another spreadsheet program. The sample.index column links sample.file.name back to an individual sample; multiple sample.file.names that share sample.index values come from the same individual sample. (If editing with Excel, ignore all the warnings about incompatible file formats when saving.) sample.index should be a positive integer (but see Note below).

3. Optionally create an “info” file, which is useful if you want to associate extra information against your samples. The function load.abi.create.info will create an info file that contains all the unique values of sample.index, and an empty column titled species. The species column can be filled in where the species is known (e.g., from collections of sporocarps). Any additional columns may be added. Running

```
load.abi.create.info(x)
```

where `x` is the name of your ABI file will create an info file in the same directory as the ABI file. The function will print the name and location of the info file to the console. Edit and save this file.

4. Create the TRAMPsamples object by running load.abi. This loads your ABI data, plus the new template file, plus an optional information file. Running

```
my.samples <- load.abi(x, primer.translate=primer.translate)
```

will create an object “my.samples” containing your data.
By default, the filenames of the template and info files will be automatically generated: `<prefix>.<ext>` becomes `<prefix>_template.csv` or `<prefix>_info.csv`. If you choose to specify `file.template` or `file.info` manually when running `load.info.create.template` or `load.info.create.info`, you must use the same values of `file.template` and `file.info` when running `load.abi`.

**Warning**

Do not change the names of any columns produced by `load.abi.create.template` or `load.abi.create.info`.

**Note**

There is no reason that data from other types of output files could not be manually imported using `TRAMP.samples`. We welcome contributions for other major data formats.

When creating `sample.index` values, these should be positive integers. If you enter strings (e.g. `a1, b1`), these will be automatically converted into integers. Once loaded, `sample.pk/sample.fk` is always a positive integer key, but `sample.index` will be retained as your string keys.

**See Also**

`read.abi`, which reads in ABI data with few modifications.

`TRAMP.samples`, which documents the data type produced by `load.abi`.

The package vignette, which includes a worked example of loading data using these functions; to locate the vignette, type `help(library=TRAMP)`, and scroll to the bottom of the page, or type: `system.file("doc/TRAMPR_demo.pdf", package="TRAMP")`.

---

**plot.TRAMP**

*Plot a TRAMP Object*

**Description**

Creates a graphical representation of matches performed by `TRAMP`. The plot displays (1) “matches”, showing how samples match the knowns and (2) “peak profiles”, showing the locations of peaks for individual enzyme/primer combinations.

**Usage**

```r
## S3 method for class 'TRAMP'
plot(x, sample.fk, ...)
TRAMP.plotone(x, sample.fk, grouped=FALSE, ignore=FALSE,
              all.knowns=TRUE, all.samples=FALSE,
              all.samples.global=FALSE, col=1:10,
              pch=if (grouped) 15 else 16, xmax=NULL, horiz.lines=TRUE,
              mar.default=.5, p.top=.5, p.labels=1/3, cex.axis=NULL,
              cex.axis.max=1)
```
### plot.TRAMP

**Arguments**

- **x**: A TRAMP object.
- **sample.fk**: The sample.fk to plot. If omitted, then all samples are plotted, one after the other (this is useful for generating a summary of all fits for printing out: see Example).
- **grouped**: Logical: Should the matched knowns be grouped?
- **ignore**: Logical: Should matches marked as ignored by `remove.TRAMP.match` be excluded?
- **all knowns, all samples, all samples.global**: Controls which enzyme/primer combinations are displayed (see Details)
- **col**: Vector of colours to plot the different enzyme/primer combinations. There must be at least as many colours as there are different combinations.
- **pch**: Plotting symbol to use (see `points` for possible values and their interpretation). By default, this will use filled circles when ungrouped and filled squares when grouped.
- **xmax**: Maximum size (in base pairs) for the plots to cover. NULL (the default) uses the range of all data found in the TRAMPsamples object (rounded up to the nearest 100). NA will use the range of all data in the current sample.
- **horiz.lines**: Logical: Should horizontal grid lines be used for each matched known?
- **mar.default**: Margin size (in lines of text) to surround the plot.
- **p.top**: Proportion of the plotting area to be used for the “matches”. The “peak profiles” will share the bottom 1−p. top of the plot.
- **p.labels**: Proportion of the plotting area to be used for labels to the left of the plots. 1−p.labels will be used for the plots (try increasing this if you have very long species or group names).
- **cex.axis**: Size of the text used for axes. If NULL (the default), then the largest cex that will exactly fit labels is chosen (up to `cex.axis.max`).
- **cex.axis.max**: Maximum size of the text used for axes, if automatically determining the label size (i.e. `cex.axis` is NULL).
- **...**: Additional arguments passed to TRAMP.plotone.

**Details**

This constructs a plot of a TRAMP fit, illustrating where knowns match the sample data, and which sample peaks remain unmatched.

The top portion of the plot displays “matches”, showing how samples match the knowns. Individual species (or groups if grouped is TRUE) are represented by different horizontal lines. Where the sample matches a particular known, a symbol is drawn (Beware: it may look like only one symbol is drawn when several symbols are plotted on top of one another).

The bottom portion of the plot displays the “peak profile” of the sample, showing the locations and heights of peaks for various enzyme/primer combinations (the exact combination depends on the values of all knowns, all samples and all samples.global; see below). The height is arbitrary, so units are omitted.
The arguments `all.knowns`, `all.samples` and `all.samples.global` control which enzyme/primer combinations are displayed in the plot. `all.knowns=TRUE` displays all combinations present in the knowns database and `all.samples=TRUE` displays all combinations present in the samples; when `all.samples.global=TRUE` this is combinations across the entire samples data set, otherwise this is samples present in the current sample only. At least one of `all.knowns` and `all.samples` must be `TRUE`.

**Note**

While `TRAMP.plotone` does the actual plot, it should not be called directly; please use `plot(x, sample.fk, ...)`. 

**See Also**

`plot.TRAMPknowns`, for plotting `TRAMPknowns` objects, and `plot.TRAMPsamples`, for plotting `TRAMPsamples` objects.

**Examples**

```r
data(demo.samples)
data(demo.knowns)
res <- TRAMP(demo.samples, demo.knowns)

plot(res, 101)
plot(res, 110)
plot(res, 117)

plot(res, 117, grouped=TRUE)
```

```r
## Not run:
# Create a PDF file with all matches:
pdf("all_matches.pdf")
plot(res)
dev.off()
## End(Not run)
```

---

**plot.TRAMPknowns**

**Summary Plot of Knowns Data**

**Description**

Creates a plot showing the clustering and profiles of a `TRAMPknowns` object (a “knowns database”). The plot has three vertical panels:

- The leftmost contains a dendrogram, showing how similar the profiles of knowns are (see `group.knowns` for details).
- The rightmost displays the TRFLP profile for each individual (with a different colour symbol for each different enzyme/primer combination).
- The middle panel displays information on the species names and groups of the knowns.
Usage

```r
## S3 method for class 'TRAMPknowns'
plot(x, cex=1, name="species", pch=1, peaks.col, p=.02,
     group.clusters=TRUE, groups.col=1:4, grid.by=5, grid.col="gray",
     widths=c(1, 2, 1), ...) 
```

Arguments

- **x**: A TRAMPknowns object.
- **cex**: Character size for the plot. Because knowns databases can be large, this should be small and may need to be adjusted. Most aspects of the plot will scale with this.
- **name**: Column name to use when generating species names; must be one of species or group.name.
- **pch**: Plotting symbol to use for peaks in the peak profiles.
- **peaks.col**: Vector of colours to plot the different enzymes in the peak profiles. These will be used in the order of the columns of `summary(x)`.
- **p**: Scaling factor for the middle plot; this specifies the proportion of the width that elements are spaced horizontally from one another. Columns of text are p apart, brackets grouping knowns are p/2 apart, and cluster groups (if present) are p*2/3 apart.
- **group.clusters**: Logical: Should groups of clusters (determined by `group.strict` - see `group.knowns`) be joined together?
- **groups.col**: Vector of colours to plot different group clusters in. This will be recycled as necessary.
- **grid.by**: Interval between horizontal grid lines. Grid lines start at ceiling(grid.by/2) from the bottom of the plot. A value of NA suppresses grid lines.
- **grid.col**: Colour of the horizontal grid lines.
- **widths**: Relative widths of the three panels of the plot (see `layout`). `widths` must be a vector of 3 elements, corresponding to the three panels from left to right.
- **...**: Additional arguments (ignored).

Note

In general, there will probably be too many knowns to make a legible plot when displayed on the screen. We recommend creating a PDF of the plot and viewing that instead (see Example).

When plotted on the interactive plotting device, if the plot is resized, the plot is likely to look strange.

See Also

- `group.knowns`, which controls the grouping of knowns, and `TRAMPknowns`, which constructs TRAMPknowns objects.
plot.TRAMPsamples

### Arguments

- **x**: A `TRAMPsamples` object, containing profiles to plot.
- **sample.fk**: The `sample.fk` to plot. If omitted, then all samples are plotted, one after the other (this is useful for generating a summary of all fits for printing out: see Example).
- **all.samples.global**: Logical: Should plots be set up for all enzyme/primer combinations present in `x`, even if the combinations are not present for all individual cases? Analogous to the same argument in `plot.TRAMP`. (This is useful for keeping combinations in the same place, and plotted with the same colours.)
- **col**: Vector of colours to plot the different enzyme/primer combinations. There must be at least as many colours as there are different combinations.
- **xmax**: Maximum size (in base pairs) for the plots to cover. `NULL` (the default) uses the range of all data found in the `TRAMPsamples` object (rounded up to the nearest 100). `NA` will use the range of all data in the current sample.
- **mar.default**: Margin size (in lines of text) to surround the plot.
read.abi

mar.labels Number of lines of text to be used for labels to the left of the plots. Increase this if labels are being truncated.

cex Scaling factor for text.
...
Additional arguments (ignored).

See Also

plot.TRAMP, the plotting method for TRAMP objects, and plot.TRAMPknowns, for TRAMPknowns objects.

Examples

data(demo.samples)

plot(demo.samples, 101)
plot(demo.samples, 117)

## Not run:
# Create a PDF file with all profiles:
pdf("all_profiles.pdf")
plot(demo.samples)
dev.off()

## End(Not run)

read.abi Read ABI Output Files

Description

Read an Applied Biosystems Gene Mapper (ABI) output file, and prepare for analysis.

Note that this operates on the summarised output (a text file), rather than the .fsa files containing data for individual runs.

Usage

read.abi(file)

Arguments

file The name of the file from which the data are to be read.
Details

The ABI file format contains a few features that make it difficult to interact with directly, so `read.abi` provides a wrapper around `read.table` to work around these. The three issues are (1) trailing tab characters, (2) mixed case and punctuation in column names, and (3) parsing the “Dye/Sample Peak” column.

Because each line of an ABI file contains a trailing tab character (\t), `read.table` fails to read the file correctly. `read.abi` renames all columns so that non-alphanumeric characters all become periods, and all uppercase letters are converted to lower case.

The column Dye/Sample Peak contains data of the form `<Dye>,<Sample Peak>`, where `<Dye>` is a code for the dye colour used and `<Sample Peak>` is an integer indicating the order of the peaks. Entries where the contents of Dye/Sample Peak terminates in a "*" character (indicating an internal size standard) are automatically excluded from the analysis.

The final column names are:

- `sample.file.name`: Name of the file containing data.
- `size`: Size of the peak (in base pairs).
- `height`: Height of the peak (arbitrary units).
- `dye`: Code for dye used.
- `sample.peak`: Rank of peak within current sample.

In addition, other column names may be retained from ABI output, but not used.

Note

There is no reason that data from other types of output files could not be manually imported using `TRAMPsamples`. We welcome contributions for other major data formats.

See Also

`load.abi`, which attempts to construct a `TRAMPsamples` object from an ABI file (with a bit of user intervention).

---

**read.write**

*Read/Write TRAMPknowns and TRAMPsamples Objects*

**Description**

Saves and loads `TRAMPknowns` and `TRAMPsamples` objects as a series of “csv” (comma separated value) files for external editing.

If you do not want to edit your data, then saving with `save` is preferable; it is faster, creates smaller files, and will save any additional components in the objects (see Examples).
**Usage**

read.TRAMPknowns(file.pat, auto.save=TRUE, overwrite=FALSE)
write.TRAMPknowns(x, file.pat=x$file.pat, warn=TRUE)

read.TRAMPsamples(file.pat)
write.TRAMPsamples(x, file.pat)

**Arguments**

x A TRAMPknowns or TRAMPsamples object.
file.pat Pattern, with the filename prefix: “info” and “data” objects will be read/written as <file.pat>_info.csv and <file.pat>_data.csv, respectively.
auto.save Logical: Should TRAMPknowns object be automatically saved back to the loaded filename as it is modified (e.g. knowns added to the database). If this is TRUE, the original files will be backed up as <file.pat>(info|data)_<YYYYMMDD>.csv, where <YYYYMMDD> is the ISO date.
overwrite Should previous backup files be overwritten when creating new backups?
warn Should the function warn when no filename is given? (Because this function is called automatically when adding new knowns, and because TRAMPknowns objects need not contain a file.pat element, it may not be possible or necessary to save).

**Details**

file.pat may contain a path. It is best to use forward slashes as directory separators (path/to/file), but on Windows (only), double backslashes will also work (path\to\file).

Paths may be either relative (e.g. path/to/file), or absolute (e.g. /path/to/file, or x:/path/to/file on Windows).

**See Also**

load.abi, for semi-automatic loading of ABI output files.
save and load, for saving and loading of arbitrary R objects.

**Examples**

```r
## Not run:
# Preferred way of saving/loading objects, if editing is not required:
save(demo.knowns, file="my.knowns.Rdata")

# (possibly in a different session, but _after_ loading TRAMP)
load("my.knowns.Rdata") # -> creates `demo.knowns' in global environment
```

## End(Not run)
**rebuild.TRAMP**  
*Rebuild a TRAMP Object*

**Description**

This function rebuilds a TRAMP object. Typically this will be called automatically after adding knowns (see add.known); there should be little need to call this manually. The same parameters that were used in the original call to TRAMP are used again, and these cannot currently be modified during this call.

**Usage**

rebuild.TRAMP(x)

**Arguments**

x A TRAMP object.

**Value**

A new TRAMP object, with all components recalculated.

---

**remove.TRAMP.match**  
*Mark a TRAMP Match as Ignored*

**Description**

Mark a match in a TRAMP object as ignored; when this is set, a match will be ignored when producing presence/absence matrices (see summary.TRAMP) or when plotting (plot.TRAMP) when ignore is TRUE. update.TRAMP provides an interactive interface for doing this, but remove.TRAMP.match may be useful directly.

**Usage**

remove.TRAMP.match(x, sample.fk, knowns.fk)

**Arguments**

x A TRAMP object.

sample.fk, knowns.fk

Key of sample and known, respectively. See TRAMPsamples and TRAMPknowns for more information.

**Value**

A modified TRAMP object.
Warning
This should be regarded as experimental. There is currently no mechanism for restoring ignored matches, aside from recreating the TRAMP object, or through editing x$presence.ign directly (the format of that table is self-explanatory, but is not guaranteed not to change between TRAMP versions). Note that by default, summary.TRAMP and plot.TRAMP will not remove matches; you must specify ignore=TRUE to enable this.

Note
This function returns a modified object - the TRAMP object is not modified in place. You must do:

```r
x <- remove.TRAMP.match(x, sample.fk, knowns.fk)
```
to mark a match as ignored in the object `x`.

---

**summary.TRAMP**

Create Presence/Absence Matrices from TRAMP Objects

**Description**
Generate a summary of a TRAMP object, by producing a presence/absence matrix. This is the preferred way of extracting the presence/absence matrix from a TRAMP object, and allows for grouping, naming knowns, and ignoring matches (specified by remove.TRAMP.match).

**Usage**
```r
## S3 method for class 'TRAMP'
summary(object, name=FALSE, grouped=FALSE, ignore=FALSE, ...)
```

**Arguments**
- `object` A TRAMP object.
- `name` Logical: Should the knowns be named?
- `grouped` Logical: Should the knowns be grouped?
- `ignore` Logical: Should matches marked as ignored be excluded?
- `...` Further arguments passed to or from other methods.

**Value**
A presence/absence matrix, with samples as rows and knowns as columns. If name is TRUE, then names of knowns (or groups of knowns) are used, otherwise the knowns.fk is used (group.strict if grouped). If grouped is TRUE, then the knowns are collapsed by group (using group.strict; see group.knowns). A group is present if any of the knowns belonging to it are present. If ignore is TRUE, then any matches marked by remove.TRAMP.match are excluded.
Examples

```r
data(demo.knowns)
data(demo.samples)
res <- TRAMP(demo.samples, demo.knowns)

head(summary(res))
head(summary(res, name=TRUE))
head(summary(res, name=TRUE, grouped=TRUE))

## Extract the species richness for each sample (i.e. the number of
## knowns present in each sample)
rowSums(summary(res, grouped=TRUE))

## Extract species frequencies and plot a rank abundance diagram:
## (i.e. the number of times each known was recorded)
sp.freq <- colSums(summary(res, name=TRUE, grouped=TRUE))

sp.freq <- sort(sp.freq[sp.freq > 0], decreasing=TRUE)
plot(sp.freq, xlab="Species rank", ylab="Species frequency", log="y")

text(1:2, sp.freq[1:2], names(sp.freq[1:2]), cex=.7, pos=4, font=3)
```

Description

Determine if TRFLP profiles may match those in a database of knowns. The resulting object can be used to produce a presence/absence matrix of known profiles in environmental samples.

The TRAMP package contains a vignette, which includes a worked example; type `vignette("TRAMPdemo")` to view it.

Usage

```r
TRAMP(samples, knowns, accept.error=1.5, min.comb=4, method="maximum")
```

Arguments

- `samples`: A `TRAMPSamples` object, containing unidentified samples.
- `knowns`: A `TRAMPknowns` object, containing identified TRFLP patterns.
- `accept.error`: The largest acceptable difference (in base pairs) between any peak in the sample data and the knowns database (see Details; interpretation will depend on the value of `method`).
- `min.comb`: Minimum number of enzyme/primer combinations required before presence will be tested. The default (4) should be reasonable in most cases. Setting `min.comb` to `NA` will require that all enzyme/primer combinations in the knowns database are present in the samples.
method

Method used in calculating the difference between samples and knowns; may be one of "maximum", "euclidian" or "manhattan" (or any unambiguous abbreviation).

Details

TRAMP attempts to determine which species in the ‘knowns’ database may be present in a collection of samples.

A sample matches a known if it has a peak that is “close enough” to every peak in the known for every enzyme/primer combination that they share. The default is to accept matches where the largest distance between a peak in the knowns database and the sample is less than accept.error base pairs (default 2), and where at least min.comb enzyme/primer combinations are shared between a sample and a known (default 4).

The three-dimensional matrix of match errors is generated by create.diffsmatrix. In the resulting array, \( m[i,j,k] \) is the difference (in base pairs) between the \( i \)th sample and the \( j \)th known for the \( k \)th enzyme/primer combination.

If \( p_k \) and \( q_k \) are the sizes of peaks for the \( k \)th enzyme/primer combination for a sample and known (respectively), then maximum distance is defined as

\[
\max(|p_k - q_k|)
\]

Euclidian distance is defined as

\[
\frac{1}{n} \sqrt{\sum (p_k - q_k)^2}
\]

and Manhattan distance is defined as

\[
\frac{1}{n} \sum |p_k - q_k|
\]

where \( n \) is the number of shared enzyme/primer combinations, since this may vary across sample/known combinations. For Euclidian and Manhattan distances, accept.error then becomes the mean distance, rather than the total distance.

Value

A TRAMP object, with elements:

- **presence**
  Presence/absence matrix. Rows are different samples (with rownames from labels(samples)) and columns are different knowns (with colnames from labels(knowns)). Do not access the presence/absence matrix directly, but use summary.TRAMP, which provides options for labelling knowns, grouping knowns, and excluding “ignored” matches.

- **error**
  Matrix of distances between the samples and known, calculated by one of the methods described above. Rows correspond to different samples, and columns correspond to different knowns. The matrix dimension names are set to the values sample.pk and knowns.pk for the samples and knowns, respectively.

- **n**
  A two-dimensional matrix (same dimensions as error), recording the number of enzyme/primer combinations present for each combination of samples and knowns.
diffsmatrix  Three-dimensional array of output from \texttt{create.diffsmatrix}.

enzyme.primer  Different enzyme/primer combinations present in the data, in the order of the third dimension of \texttt{diffsmatrix} (see \texttt{create.diffsmatrix} for details).

samples, knowns, accept.error, min.comb, method
The input data objects and arguments, unmodified.

In addition, an element \texttt{presence.ign} is included to allow matches to be ignored. However, this interface is experimental and its current format should not be relied on - use \texttt{remove.TRAMP.match} rather than interacting directly with \texttt{presence.ign}.

Matching is based only on peak size (in base pairs), and does not consider peak heights.

See Also

See \texttt{create.diffsmatrix} for discussion of how differences between sample and known profiles are generated.

\texttt{plot.TRAMP}, which displays TRAMP fits graphically.

\texttt{summary.TRAMP}, which creates a presence/absence matrix.

\texttt{remove.TRAMP.match}, which marks TRAMP matches as ignored.

Examples

data(demo.knowns)
data(demo.samples)

res <- TRAMP(demo.samples, demo.knowns)

## The resulting object can be interrogated with methods:

## The goodness of fit of the sample with sample.pk=101 (see
## ?\link{plot.TRAMP}).
plot(res, 101)

## Not run:
## To see all plots (this produces many figures), one after another.
op <- par(ask=TRUE)
plot(res)
par(op)

## End(Not run)

## Produce a presence/absence matrix (see ?\link{summary.TRAMP}).
m <- summary(res)
head(m)
Description

This provides very basic support for subsetting TRAMPsamples and TRAMPknowns objects.

Usage

```R
## S3 method for class 'TRAMPknowns'
x[i, na.interp=TRUE, ...]
## S3 method for class 'TRAMPsamples'
x[i, na.interp=TRUE, ...]
```

Arguments

- `x`: A TRAMPsamples or TRAMPknowns object.
- `i`: A vector of sample.fk or knowns.fk values. For valid values, use labels(x). If any index values are not present in x, then an error will be raised. Alternatively, this may be a logical vector, of the same length as the number of samples or knowns in x. See Examples for use of this.
- `na.interp`: Logical: Controls how NA values should be interpreted when i is a logical vector.
- `...`: Further arguments passed to or from other methods.

Details

When indexing by logical vectors, NA values do not make valid indexes, but may be produced when testing columns that contain missing values, so these must be converted to either TRUE or FALSE. If `i` is a logical index that contains missing values (NAs), then `na.interp` controls how they will be interpreted:

- If `na.interp=TRUE`, then TRUE, FALSE, NA becomes TRUE, FALSE, TRUE.
- If `na.interp=FALSE`, then TRUE, FALSE, NA becomes TRUE, FALSE, FALSE.

Warning

For TRAMPknowns objects, if the file.pat element is specified as part of the object (see TRAMPknowns), then the subasetted TRAMPknowns object will be written to a file. This may not be what you want, so it is probably best to disable knowns writing by doing `x$file.pat <- NULL` before doing any subsetting (where x is the name of your TRAMPknowns object).
Examples

data(demo.samples)
data(demo.knowns)

## Subsetting by sample.fk values
labels(demo.samples)
demo.samples[c(101, 102, 110)]
labels(demo.samples[c(101, 102, 110)])

## Take just samples from the first 10 soilcores:
demo.samples[demo.samples$info$soilcore.fk <= 10]

## Indexing also works on TRAMPknowns:
demo.knowns[733]
labels(demo.knowns[733])

---

**TRAMPknowns Objects**

**Description**

These functions create and interact with TRAMPknowns objects (collections of known TRFLP patterns). Knowns contrast with “samples” (see TRAMPsamples) in that knowns contain identified profiles, while samples contain unidentified profiles. Knowns must have at most one peak per enzyme/primer combination (see Details).

**Usage**

TRAMPknowns(data, info, cluster.pars=list(), file.pat=NULL,
warn.factors=TRUE, ...)

## S3 method for class 'TRAMPknowns'
labels(object, ...)
## S3 method for class 'TRAMPknowns'
summary(object, include.info=FALSE, ...)

**Arguments**

data: data.frame containing peak information.
info: data.frame, describing individual samples (see Details for definitions of both data.frames).
cluster.pars: Parameters used when clustering the knowns database. See Details.
file.pat: Optional partial filename in which to store knowns database after modification. Files <file.pat>_info.csv and <file.pat>_data.csv will be created.
warn.factors: Logical: Should a warning be given if any columns in info or data are converted into factors?
TRAMPknowns

- **object**: A TRAMPknowns object.
- **include.info**: Logical: Should the output be augmented with the contents of the `info` component of the TRAMPknowns object?

... TRAMPknowns: Additional objects to incorporate into a TRAMPknowns object. Other methods: Further arguments passed to or from other methods.

**Details**

The object has at least two components, which relate to each other (in the sense of a relational database). `info` holds information about the individual samples, and `data` holds information about individual peaks (many of which may belong to a single sample).

Column definitions:

- **info**:
  - `knowns.pk`: Unique positive integer, used to identify individual knowns (i.e. a "primary key").
  - `species`: Character, giving species name.

- **data**:
  - `knowns.fk`: Positive integer, indicating which sample the peak belongs to (by matching against `info$knowns.pk`) (i.e. a "foreign key").
  - `primer`: Character, giving the name of the primer used.
  - `enzyme`: Character, giving the name of the restriction digest enzyme used.
  - `size`: Numeric, giving size (in base pairs) of the peak.

In addition, TRAMPknowns will create additional columns holding clustering information (see `group.knowns`). Additional columns are allowed (and retained, but ignored) in both data.frames. Additional objects are allowed as part of the TRAMPknowns object, but these will not be written by `write.TRAMPknowns`; any extra objects passed (via ...) will be included in the final TRAMPknowns object.

The `cluster.pars` argument controls how knowns will be clustered (this will happen automatically as needed). Elements of the list `cluster.pars` may be any of the three arguments to `group.knowns`, and will be used as defaults in subsequent calls to `group.knowns`. If not provided, default values are: `dist.method="maximum", hclust.method="complete", cut.height=2.5` (if only some elements of `cluster.pars` are provided, the remaining elements default to the values above). To change values of clustering parameters in an existing TRAMPknowns object, use `group.knowns`.

A known contains at most one peak per enzyme/primer combination. Where a species is known to have multiple TRFLP profiles, these should be treated as separate knowns with different, unique, `knowns.pk` values, but with identical `species` values. A sample containing either pattern will then be recorded as having that species present (see `group.knowns`).

**Value**

- **TRAMPknowns**: A new TRAMPknowns object: a list with components `info`, `data` (the provided data.frames, with clustering information added to `info`), `cluster.pars` and `file.pat`, plus any extra objects passed as ....
- **labels.TRAMPknowns**: A sorted vector of the unique samples present in `x` (from `info$knowns.pk`).
summary.TRAMPknowns

A data.frame, with the size of the peak (if present) for each enzyme/primer combination, with each known (indicated by knowns.pk) as rows and each combination (in the format <primer>_<enzyme>) as columns.

Note

Across a TRAMPknowns object, primer and enzyme names must be exactly the same (including case and whitespace) to be considered the same. For example "ITS4", "Its4", "ITS 4" and "ITS4 " would be considered to be four different primers.

Factors will not merge correctly (with combine.TRAMPknowns or add.known). TRAMPknowns will attempt to catch factor columns and convert them into characters for the info and data data.frames. Other objects (passed as part of ...) will not be altered.

See Also

TRAMPsamples, which constructs an analagous object to hold "samples" data.
plot.TRAMPknowns, which creates a graphical representation of the knowns data.
TRAMP, for matching unknown TRFLP patterns to TRAMPknowns objects.
group.knowns, which groups similar knowns (generally called automatically).
add.known and combine.TRAMPknowns, which provide tools for adding knowns from a sample data set and merging knowns databases.

Examples

```r
## This example builds a TRAMPknowns object from completely artificial data:

## The info data.frame:
knowns.info <-
data.frame(knowns.pk=1:8, 
species=rep(paste("Species", letters[1:5]), length=8))
knowns.info

## The data data.frame:
knowns.data <- expand.grid(knowns.fk=1:8,
   primer=c("ITS1F", "ITS4"),
   enzyme=c("BsuRI", "HpyCH4IV"))
knowns.data$size <- runif(nrow(knowns.data), min=40, max=800)

## Construct the TRAMPknowns object:
demo.knowns <- TRAMPknowns(knowns.data, knowns.info, warn.factors=FALSE)

## A plot of the pretend knowns:
plot(demo.knowns, cex=1, group.clusters=TRUE)
```
Description

These functions create and interact with TRAMPSamples objects (collections of TRFLP patterns). Samples contrast with “knowns” (see TRAMPKnowns) in that samples contain primarily unidentified profiles. In contrast with knowns, samples may have many peaks per enzyme/primer combination.

Usage

TRAMPSamples(data, info=NULL, warn.factors=TRUE, ...)

## S3 method for class 'TRAMPSamples'
labels(object, ...)

## S3 method for class 'TRAMPSamples'
summary(object, include.info=FALSE, ...)

Arguments

data: data.frame containing peak information.
info: (Optional) data.frame, describing individual samples (see Details for definitions of both data.frames). If this is omitted, a basic data.frame will be generated.
warn.factors: Logical: Should a warning be given if any columns in info or data are converted into factors?
object: A TRAMPSamples object.
include.info: Logical: Should the output be augmented with the contents of the info component of the TRAMPSamples object?
...

Details

The object has at least two components, which relate to each other (in the sense of a relational database). info holds information about the individual samples, and data holds information about individual peaks (many of which belong to a single sample).

Column definitions:

• info:
  sample.pk: Unique positive integer, used to identify individual samples (i.e. a “primary key”).
  species: Character, giving species name if samples were collected from an identified species.
  If this column is missing, it will be initialised as NA.

• data:
  sample_fk: Positive integer, indicating which sample the peak belongs to (by matching against info$sample.pk) (i.e. a “foreign key”).
primer: Character, giving the name of the primer used.
enzyme: Character, giving the name of the restriction digest enzyme used.
size Numeric, giving size (in base pairs) of the peak.
height Numeric, giving the height (arbitrary units) of the peak.

Additional columns are allowed (and ignored) in both data.frames, and will be retained. This allows notes on data quality and treatments to be easily included. Additional objects are allowed as part of the TRAMPsamples object; any extra objects passed (via ...) will be included in the final TRAMPsamples object.

If info is omitted, then a basic data.frame will be generated, containing just the unique values of sample.fk, and NA for species.

Value

TRAMPsamples A new TRAMPsamples object, as described above.
labels.TRAMPsamples A sorted vector of the unique samples present in x (from info$sample.pk).
summary.TRAMPsamples A data.frame, with the number of peaks per enzyme/primer combination, with each sample (indicated by sample.pk) as rows and each combination (in the format <primer>_<enzyme>) as columns.

Note

Across a TRAMPsamples object, primer and enzyme names must be exactly the same (including case and whitespace) to be considered the same. For example "ITS4", "Its4", "ITS4 " and "ITS 4" would be considered to be four different primers.

Factors will not merge correctly (with combine.TRAMPsamples). TRAMPsamples will attempt to catch factor columns and convert them into characters for the info and data data.frames. Other objects (passed as part of ...) will not be altered.

See Also

plot.TRAMPsamples and summary.TRAMPsamples, for plotting and summarising TRAMPsamples objects.
TRAMPknowns, which constructs an analagous object to hold “knowns” data.
TRAMP, for analysing TRAMPsamples objects.
load.abi, which creates a TRAMPsamples object from Gene Mapper (Applied Biosystems) output.
update.TRAMP

Interactively Alter a TRAMP Object

Description

This function allows some manual checking and correction of a TRAMP object. By default, it steps through each sample, and offers to (1) add a new known to the TRAMP\_knowns database within the TRAMP object (see add\_known for details), (2) mark matches to be ignored in future calls to plot\_TRAMP (see remove\_TRAMP\_match), (3) save the current plot as a PDF.

Usage

```r
## S3 method for class 'TRAMP'
update(object, sample.fk=labels(object$samples), grouped=FALSE,
       ignore=TRUE, delay.rebuild=FALSE, default.species=NULL,
       filename.fmt="TRAMP_%d.pdf", ...)
```

Arguments

- **object**: A TRAMP object.
- **sample.fk**: A vector of sample.fk to cycle through. If omitted, this will default to all samples present in the TRAMP\_samples component of the TRAMP object.
- **grouped, ignore**: Plotting parameters, as in plot\_TRAMP. Currently these cannot be altered from their default values.
- **delay.rebuild**: Logical: Should the rebuild of the TRAMP object be delayed until the function returns? If this is FALSE (the default), then the TRAMP object will rebuild every time a new known is added. This may take a while for large objects, so if set to TRUE, then the TRAMP object will not be rebuilt until all sample.fks have been displayed. This means that any new samples added as knowns will not be included in plots.
- **default.species**: Default species name for newly added knowns. Passed to add\_known.
- **filename.fmt**: Format used to generate filenames when saving PDFs. Include a "%d" to stand in for the sample.fk (so "TRAMP_%d.pdf" becomes "TRAMP_12.pdf" for sample.fk 12).
- **...**: Further arguments passed to the plotting function plot\_TRAMP.

Warning

If an error occurs while running update, all modifications will be lost.
update.TRAMP

Note

update.TRAMP returns a modified TRAMP object, and does not modify the original TRAMP object in place. You must use it like:

```r
x <- update(x)
```
or

```r
x2 <- update(x)
```
to modify the original object or create a new, modified object in place. Note that if creating multiple objects, if the `TRAMPknowns` object has a `file.pat` element, then any changes to either of `x` or `x2` will be written back to file, but the `knowns` contained in `x` and `x2` may be different. See the note in `add.known`.

The action “Quit” will always exit the `update` function and save the object.

Be careful when using a TRAMP object whose `TRAMPknowns` element has a `file.pat` element; new `knowns` added will be immediately written to file.

Examples

```r
## Since this function runs interactively, there can be no sample.
```
Index

*Topic array
  TRAMPindexing, 31
*Topic classes
  TRAMPknowns, 32
  TRAMPsamples, 35
*Topic cluster
  build.knowns, 6
  create.diffsmatrix, 9
  group.knowns, 12
  rebuild.TRAMP, 26
  TRAMP, 28
  update.TRAMP, 37
*Topic connection
  read.abi, 23
*Topic datagen
  build.knowns, 6
*Topic datasets
  demo.knowns, 11
  demo.samples, 11
*Topic file
  load.abi, 16
  read.abi, 23
  read.write, 24
*Topic hplot
  plot.TRAMP, 18
  plot.TRAMPknowns, 20
  plot.TRAMPsamples, 22
*Topic manip
  add.known, 3
  combine.TRAMPsamples, 8
  remove.TRAMP.match, 26
*Topic methods
  combine, 8
  summary.TRAMP, 27
  TRAMPindexing, 31
*Topic multivariate
  build.knowns, 6
  create.diffsmatrix, 9
  group.knowns, 12
  rebuild.TRAMP, 26
  TRAMP, 28
  update.TRAMP, 37
*Topic package
  TRAMP-package, 2
  [.TRAMPknowns, 14
  [.TRAMPknowns (TRAMPindexing), 31
  [.TRAMPsamples (TRAMPindexing), 31
  absolute.min, 10
  add.known, 3, 7, 14, 26, 34, 37, 38
  build.knowns, 2, 4, 5, 6, 14
  combine, 8
  combine.TRAMP (add.known), 3
  combine.TRAMPknowns, 8, 9, 34
  combine.TRAMPknowns (add.known), 3
  combine.TRAMPsamples, 3, 5, 8, 8, 36
  create.diffsmatrix, 2, 9, 29, 30
  cutree, 13, 14
  demo.knowns, 11
  demo.samples, 11
  dist, 13, 14
  group.knowns, 4, 12, 20, 21, 27, 33, 34
  hclust, 13, 14
  labels.TRAMPknowns (TRAMPknowns), 32
  labels.TRAMPsamples (TRAMPsamples), 35
  layout, 21
  load, 25
  load.abi, 2, 12, 16, 24, 25, 36
  peakscanner.to.genemapper (load.abi), 16
  plot.TRAMP, 2, 18, 22, 23, 26, 27, 30, 37
  plot.TRAMPknowns, 14, 20, 20, 23, 34
  plot.TRAMPsamples, 20, 22, 36
  points, 19