Package ‘dvir’

October 16, 2023

Type Package

Title Disaster Victim Identification

Version 3.2.1

Description Joint DNA-based disaster victim identification (DVI), as described in Vigeland and Egeland (2021) <doi:10.21203/rs.3.rs-296414/v1>. Identification is performed by optimising the joint likelihood of all victim samples and reference individuals. Individual identification probabilities, conditional on all available information, are derived from the joint solution in the form of posterior pairing probabilities. ‘dvir’ is part of the ‘pedsuite’ collection of packages for pedigree analysis.

License GPL-3

URL https://github.com/magnusdv/dvir

BugReports https://github.com/magnusdv/dvir/issues

Depends pedtools (>= 2.4.0), R (>= 4.1.0)

Imports forrel (>= 1.5.2), pedprobr (>= 0.8.0), ribd

Suggests knitr, rmarkdown

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Magnus Dehli Vigeland [aut, cre] (<https://orcid.org/0000-0002-9134-4962>), Thore Egeland [aut] (<https://orcid.org/0000-0002-3465-8885>)

Maintainer Magnus Dehli Vigeland <m.d.vigeland@medisin.uio.no>

Repository CRAN

Date/Publication 2023-10-15 23:40:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>R function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>amDrivenDVI</td>
<td>3</td>
</tr>
<tr>
<td>Bmarginal</td>
<td>4</td>
</tr>
<tr>
<td>combineSummaries</td>
<td>5</td>
</tr>
<tr>
<td>directMatch</td>
<td>5</td>
</tr>
<tr>
<td>dviCompare</td>
<td>6</td>
</tr>
<tr>
<td>dviData</td>
<td>8</td>
</tr>
<tr>
<td>dviSim</td>
<td>9</td>
</tr>
<tr>
<td>dviSolve</td>
<td>10</td>
</tr>
<tr>
<td>example1</td>
<td>11</td>
</tr>
<tr>
<td>example2</td>
<td>11</td>
</tr>
<tr>
<td>excludePairing</td>
<td>12</td>
</tr>
<tr>
<td>exclusionExample</td>
<td>13</td>
</tr>
<tr>
<td>expand.grid.nodup</td>
<td>13</td>
</tr>
<tr>
<td>familias2dvir</td>
<td>14</td>
</tr>
<tr>
<td>findExcluded</td>
<td>16</td>
</tr>
<tr>
<td>findNonidentifiable</td>
<td>17</td>
</tr>
<tr>
<td>findUndisputed</td>
<td>18</td>
</tr>
<tr>
<td>generatePairings</td>
<td>20</td>
</tr>
<tr>
<td>getFamily</td>
<td>21</td>
</tr>
<tr>
<td>getSimpleFams</td>
<td>22</td>
</tr>
<tr>
<td>grave</td>
<td>23</td>
</tr>
<tr>
<td>icmp</td>
<td>23</td>
</tr>
<tr>
<td>jointDVI</td>
<td>24</td>
</tr>
<tr>
<td>KETPch4</td>
<td>26</td>
</tr>
<tr>
<td>KETPex481</td>
<td>27</td>
</tr>
<tr>
<td>KETPex497</td>
<td>27</td>
</tr>
<tr>
<td>KETPex498</td>
<td>28</td>
</tr>
<tr>
<td>mergePM</td>
<td>28</td>
</tr>
<tr>
<td>ncomb</td>
<td>30</td>
</tr>
<tr>
<td>pairwiseLR</td>
<td>31</td>
</tr>
<tr>
<td>plane crash</td>
<td>32</td>
</tr>
<tr>
<td>plotDVI</td>
<td>33</td>
</tr>
<tr>
<td>plotSolution</td>
<td>34</td>
</tr>
<tr>
<td>plotUndisputed</td>
<td>35</td>
</tr>
<tr>
<td>relabelDVI</td>
<td>36</td>
</tr>
<tr>
<td>sequentialDVI</td>
<td>37</td>
</tr>
<tr>
<td>setPairing</td>
<td>38</td>
</tr>
<tr>
<td>subsetDVI</td>
<td>40</td>
</tr>
</tbody>
</table>

Index 41
Description

AM-driven identification, i.e., considering one AM family at a time. Simple families (exactly 1 missing) are handled directly from the LR matrix, while nonsimple families are analysed with jointDVI().

Usage

amDrivenDVI(
  dvi,
  fams = NULL,
  threshold = 10000,
  threshold2 = max(1, threshold/10),
  verbose = TRUE
)

Arguments

dvi A dviData object.
fams A character; the names of families to consider. By default, all families. Special keywords: "simple" (all families with exactly 1 missing) and "nonsimple" (all families with > 1 missing).
threshold LR threshold for 'certain' match.
threshold2 LR threshold for 'probable' match (in simple families).
verbose A logical.

Details

Note: This function assumes that undisputed identifications have been removed. Strange outputs may occur otherwise.

Value

A list of dviReduced and summary.

Examples

w = amDrivenDVI(example2)
w$summary
w$dviReduced

# Bigger example: Undisputed first
u = findUndisputed(planeCrash)
u$summary
# AM-driven analysis of the remaining
amDrivenDVI(u$dviReduced, threshold2 = 500)

## Bmarginal

### Posterior pairing probabilities

#### Description

Compute posterior pairing and non-pairing probabilities, based on a prior and the output from `jointDVI()`.

#### Usage

```r
Bmarginal(jointRes, missing, prior = NULL)
```

#### Arguments

- `jointRes`: Output from `jointDVI()`.
- `missing`: Character vector with names of missing persons.
- `prior`: A numeric vector of length equal the number of rows in `jointRes`. Default is a flat prior.

#### Details

The prior assigns a probability to each assignment, each row of `jointRes`. If the prior is not specified, a flat prior is used. The prior needs not sum to 1 since the user may rather choose a flat prior on the *a priori* possible assignments.

#### Value

A matrix. Row *i* gives the posterior probability that victim *i* is one of the missing persons or someone else, denoted '*'.

#### See Also

`jointDVI()`

#### Examples

```r
jointRes = jointDVI(example1)
Bmarginal(jointRes, example1$missing)

# Artificial example: all but optimal solution excluded by prior
Bmarginal(jointRes, example1$missing, prior = c(1, rep(0,26)))
```
**combineSummaries**

*Combine summary tables*

**Description**

Combines summary tables from various functions into a final result table.

**Usage**

```r
combineSummaries(dfs, orderBy = NULL, dvi = NULL)
```

**Arguments**

- **dfs** A list of data frames.
- **orderBy** A character with column names to sort by.
- **dvi** A `dviData` object used for sorting. Note that if given, this must contain all victims and families.

**Value**

A data frame.

**Examples**

```r
u = findUndisputed(planecrash)
a = amDrivenDVI(u$dviReduced, threshold2 = 500)

u$summary
a$summary

combineSummaries(list(u$summary, a$summary),
                 orderBy = c("Family", "Missing"),
                 dvi = planecrash)
```

---

**directMatch**

*Direct match LR*

**Description**

Computes the likelihood ratio comparing if two samples are from the same individual or from unrelated individuals.

**Usage**

```r
directMatch(x, y, geno1 = NULL, geno2 = NULL)
```
Arguments

\( x, y \)          Typed singletons.
\( \text{geno1}, \text{geno2} \)  (Optional) Named character vectors with genotypes for \( x \) and \( y \) respectively.

Value

A nonnegative likelihood ratio.

See Also

\text{mergePM}().

Examples

\begin{verbatim}
pm = singletons(c("V1", "V2", "V3")) |>
  addMarker(V1 = "1/1", V2 = "2/2", V3 = "1/1",
  afreq = c("1" = 0.01, "2" = 0.99), name = "L1")

directMatch(pm[[1]], pm[[2]])
directMatch(pm[[1]], pm[[3]])
\end{verbatim}
Arguments

- `dvi` A `dviData` object, typically created with `dviData()`.
- `true` A character of the same length as `dvi$pm`, with the true solution, e.g., `true = c("M2", "M3", "*")` if the truth is V1 = M2, V2 = M3 and V3 unmatched.
- `refs` Character vector with names of the reference individuals. By default the typed members of `dvi$am`.
- `methods` A subset of the numbers 1,2,3,4,5,6.
- `markers` If `simulate = FALSE`: A vector indicating which markers should be used.
- `threshold` An LR threshold passed on to the sequential methods.
- `simulate` A logical, indicating if simulations should be performed.
- `db` A frequency database used for simulation, e.g., `forrel::NorwegianFrequencies`. By default the frequencies attached to `dvi$am` are used.
- `Nsim` A positive integer; the number of simulations.
- `returnSims` A logical: If TRUE, the simulated data are returned without any DVI comparison.
- `seed` A seed for the random number generator, or NULL.
- `numCores` The number of cores used in parallelisation. Default: 1.
- `verbose` A logical.

Details

The following methods are available for comparison, through the `methods` parameter:

1. Sequential, without LR updates
2. Sequential, with LR updates
3. Sequential (undisputed) + joint (remaining). Always return the most likely solution(s).
4. Joint - brute force. Always return the most likely solution(s).
5. Like 3, but return winner(s) only if LR > `threshold`; otherwise the empty assignment.
6. Like 4, but return winner(s) only if LR > `threshold`; otherwise the empty assignment.

Value

A list of solution frequencies for each method, and a vector of true positive rates for each method.

Examples

```r
refs = "R1"
db = forrel::NorwegianFrequencies[1:3]

# True solution
true = c("M1", "M2", "M3")

# Run comparison
```
# dviCompare(example1, refs, true = true, db = db, Nsim = 2, seed = 123)

# Alternatively, simulations can be done first...

sims = dviCompare(example1, refs, true = true, simulate = TRUE,
                   db = db, Nsim = 2, seed = 123, returnSims = TRUE)

# ... and computations after:

# dviCompare(sims, refs, true = true, simulate = FALSE)

dviData

---

**dviData**

**DVI data**

### Description

DVI data

### Usage

dviData(pm, am, missing, generatePairings = TRUE)

checkDVI(
  dvi,
  pairings = NULL,
  errorIfEmpty = FALSE,
  ignoreSex = FALSE,
  verbose = TRUE
)

### Arguments

- **pm**: A list of singletons: The victim samples.
- **am**: A list of pedigrees: The reference families.
- **missing**: A character vector with names of missing persons.
- **generatePairings**: A logical. If TRUE (default) a list of sex-compatible pairings is included as part of the output.
- **dvi**: A dviData object.
- **pairings**: A list of pairings.
- **errorIfEmpty**: A logical.
- **ignoreSex**: A logical.
- **verbose**: A logical.
dviSim

Value

An object of class dviData, which is basically a list of pm, am, missing and pairings.

Examples

dvi = dviData(pm = singleton("V1"), am = nuclearPed(1), missing = "3")
dvi

checkDVI(dvi)

dviSim

Simulate genotypes in a DVI dataset

Description

Simulates genotypes for the references and missing persons in each AM family, transfers to the PM singletons according to the indicated matching. Remaining victims are simulated as unrelated.

Usage

dviSim(
  dvi,
  refs = typedMembers(dvi$am),
  truth = NULL,
  seed = NULL,
  verbose = FALSE
)

Arguments

dvi          A dviData object.
refs         A character with names of all reference individuals. By default, the typed members of the input.
truth        A named vector of the format c(vic1 = mis1, vic2 = mis2, ...).
seed         An integer seed for the random number generator.
verbose      A logical.

Value

A dviData object similar to the input.

See Also

forrel::profileSim().
dviSolve

A complete pipeline for solving a DVI case

Description

This wraps several other functions into a complete pipeline for solving a DVI case.

Usage

dviSolve(
  dvi,
  threshold = 10000,
  threshold2 = max(1, threshold/10),
  maxIncomp = 2,
  ignoreSex = FALSE,
  limit = 0,
  verbose = TRUE,
  debug = FALSE
)

Arguments

dvi A dviData object.
threshold LR threshold for 'significant' match.
threshold2 LR threshold for 'probable' match.
maxIncomp An integer passed onto findExcluded(). A pairing is excluded if the number of incompatible markers exceeds this.
ignoreSex A logical, by default FALSE.
limit A number passed onto findUndisputed(); only pairwise LR values above this are considered.
verbose, debug Logicals.

Value

A data frame.

Examples

dviSolve(example2)
dviSolve(example2, threshold = 5, verbose = FALSE)
**DVI dataset: Generational trio**

**Description**
A proof-of-concept dataset involving three missing members (child, father, grandfather) of a single family. With the given data, stepwise victim identification fails to find the correct solution, while joint identification succeeds.

**Usage**
`example1`

**Format**
A `dviData` object with the following content:
- `pm`: A list of 3 singletons (victims).
- `am`: A pedigree with three missing persons and one typed reference individual.
- `missing`: A vector containing the names of the missing persons.

**Examples**

```
example1
plotDVI(example1, marker = 1)
jointDVI(example1)
```

**DVI dataset: Two reference families**

**Description**
A small DVI example with three victims, and three missing persons from two reference families

**Usage**
`example2`
Format

A dviData object with the following content:

- pm: A list of 3 singletons (victims).
- am: A list of 2 pedigrees with three missing persons and one typed reference individual.
- missing: A vector containing the names of the missing persons.

Examples

```r

# Disallow V1 = M1 in the example2 dataset:
ex = excludePairing(example2, victim = "V1", missing = "M1")
jointDVI(ex, verbose = FALSE)
```

Description

Disallow certain pairings by removing them from the list `dvi$pairings` of candidate pairings for a given victim sample.

Usage

```r

excludePairing(dvi, victim, missing)
```

Arguments

- `dvi`: A dviData object.
- `victim`: The name of a single victim sample.
- `missing`: The name(s) of one or several missing individuals.

Value

A dviData object.

Examples

```r

# Disallow V1 = M1 in the `example2` dataset:
ex = excludePairing(example2, victim = "V1", missing = "M1")
jointDVI(ex, verbose = FALSE)

# Compare with original
jointDVI(example2, verbose = FALSE)
```
exclusionExample

# The only difference is in the 'pairings' slot:
ex$pairings
example2$pairings

### exclusionExample  Dataset: Exclusion example

**Description**

This data is based on a real case, but pedigrees have been changed and marker data simulated to preserve anonymity.

**Usage**

exclusionExample

**Format**

A dviData object with the following content:

- **pm**: A list of 16 singletons (male victims).
- **am**: A list of 15 pedigrees, each with one missing person
- **missing**: A vector containing the names of the 15 missing persons.

**Examples**

exclusionExample

---

**expand.grid.nodup**  Combinations without duplications

**Description**

This is similar to `expand.grid()` except that combinations with repeated elements are not included. The element "*" is treated separately, and is allowed to be repeated.

**Usage**

expand.grid.nodup(lst, max = 1e+05)
Arguments

lst  A list of vectors.
max  A positive integer. If the number of combinations (according to a preliminary lower bound) exceeds this, the function aborts with an informative error message. Default: 1e5.

Value

A data frame.

See Also

expand.grid()

Examples

lst = list(1, 1:2, 3:4)
# Compare
expand.grid.nodup(lst)
expand.grid(lst)

# Typical use case for DVI
lst2 = generatePairings(example1)
expand.grid.nodup(lst2)

familias2dvir  Convert a Familias file to DVI data

Description

This is a wrapper for readFam() that reads Familias files with DVI information.

Usage

familias2dvir(
  famfile,
  victimPrefix = NULL,
  familyPrefix = NULL,
  refPrefix = NULL,
  missingPrefix = NULL,
  missingFormat = NULL,
  othersPrefix = NULL,
  verbose = FALSE,
  missingIdentifier = "Missing"
)
familias2dvir

Arguments

- **famfile**: Path to Familias file.
- **victimPrefix**: Prefix used to label PM individuals.
- **familyPrefix**: Prefix used to label the AM families.
- **refPrefix**: Prefix used to label the reference individuals, i.e., the typed members of the AM families.
- **missingPrefix**: Prefix used to label the missing persons. At most one of `missingPrefix` and `missingFormat` can be given.
- **missingFormat**: A string indicating family-wise labelling of missing persons, using `[FAM]`, `[IDX]`, `[MIS]` as place holders with the following meanings (see Examples):
  - `[FAM]`: family index
  - `[IDX]`: index of missing person within the family
  - `[MIS]`: index within all missing persons
- **othersPrefix**: Prefix used to label other untyped individuals. Use "" for numeric labels (1, 2, ...).
- **verbose**: A logical. Passed on to `readFam()`.
- **missingIdentifier**: A character of length 1 used to identify missing persons in the Familias file. The default chooses everyone whose label begins with "Missing".

Details

The sex of the missing persons need to be checked as this information may not be correctly recorded in the fam file.

Value

A dviData object.

See Also

`jointDVI()`, `dviData()`, `relabelDVI()`

Examples

```r
# Family with three missing
file = system.file("extdata", "dvi-example.fam", package="dvir")

# Read file without relabelling
y = familias2dvir(file)
plotDVI(y)

# With relabelling
z = familias2dvir(file, missingFormat = "M[FAM]-[IDX]",
                   refPrefix = "ref", othersPrefix = "E")
plotDVI(z)
```
findExcluded

Excluded individuals and pairings in a DVI dataset

Description

Analysing exclusions is often an efficient way to reduce large DVI datasets. A pairing V = M is excluded if it implies (too many) genetic inconsistencies. The function `findExcluded()` identifies and removes (i) victim samples with too many inconsistencies against all missing persons, (ii) missing persons with too many inconsistencies against all victim samples, and (iii) inconsistent pairings among the remaining.

Usage

```r
findExcluded(
  dvi,
  maxIncomp = 2,
  pairings = NULL,
  ignoreSex = FALSE,
  verbose = TRUE
)

exclusionMatrix(dvi, pairings = NULL, ignoreSex = FALSE)
```

Arguments

- **dvi**: A `dviData()` object.
- **maxIncomp**: An integer. A pairing is excluded if the number of incompatible markers exceeds this.
- **pairings**: A list of possible pairings for each victim. By default, `dvi$pairings` is used, or, if this is NULL, `generatePairings(dvi, ignoreSex)`.
- **ignoreSex**: A logical, by default: FALSE.
- **verbose**: A logical, by default TRUE.

Details

The main calculation in `findExcluded()` is done by `exclusionMatrix()`, which records number of incompatible markers of each pairwise comparison.
findNonidentifiable

Description

A missing person in a DVI case is *nonidentifiable* if unrelated to all (genotyped) reference individuals and all other missing persons in the reference family. It is often wise to ignore such individuals in `jointDVI()` and other analyses, to relieve the computational burden.

Usage

`findNonidentifiable(dvi)`
Arguments

dvi A dviData object, typically created with dviData().

Details

The implementation uses ribd::kinship() to identify individuals having kinship coefficient 0 with all relevant individuals.

Value

A list with the following entries:

- nonidentifiable: A character vector (possibly empty) with the names of the nonidentifiable missing persons.
- dviReduced: A reduced dviData object, where the nonidentifiable individuals are removed from the list of missing persons. If there are no nonidentifiable, this is just a copy of dvi.
- summary: A data frame summarising the findings.

Examples

# Example 1: No nonidentifiables in dataset 'example1'
findNonidentifiable(example1)

# Example 2: Add nonidentifiable person "A"
amNew = example1$am[[1]] |> addSon(parents = c("NN", "A"))
missNew = c(example1$missing, "A")

dvi = dviData(pm = example1$pm, am = amNew, missing = missNew)
plotDVI(dvi, textAbove = c(A = "nonidentif."))

findNonidentifiable(dvi)

findUndisputed

Undisputed identifications in a DVI problem

Description

This function uses the pairwise LR matrix to find undisputed matches between victims and missing individuals. An identification $V_i = M_j$ is called undisputed, relative to a threshold $T$, if the corresponding likelihood ratio $LR_{i,j} \geq T$ AND $LR_{i,j}$ is at least $T$ times greater than all other pairwise LRs involving $V_i$ or $M_j$. 

findUndisputed
findUndisputed

Usage

findUndisputed(
  dvi,
  pairings = NULL,
  ignoreSex = FALSE,
  threshold = 10000,
  strict = FALSE,
  relax = !strict,
  limit = 0,
  nkeep = NULL,
  check = TRUE,
  numCores = 1,
  verbose = TRUE
)

Arguments

dvi A dviData object, typically created with dviData().
pairings A list of possible pairings for each victim. If NULL, all sex-consistent pairings
  are used.
ignoreSex A logical.
threshold A non-negative number. If no pairwise LR exceed this, the iteration stops.
strict A logical affecting the definition of being undisputed (see Details). Default: FALSE.
relax Deprecated; use strict = FALSE instead.
limit A positive number. Only pairwise LR values above this are considered.
nkeep An integer, or NULL. If given, only the nkeep most likely pairings are kept for
  each victim.
check A logical indicating if the input data should be checked for consistency. Default:
  TRUE.
numCores An integer; the number of cores used in parallelisation. Default: 1.
verbose A logical. Default: TRUE.

Details

If the parameter strict is set to TRUE, the last criterion is replaced with the stronger requirement
that all other pairwise LRs involving \( V_i \) or \( M_j \) must be at most 1.

Value

A list with the following entries:

- dviReduced: A reduced version of dvi, where undisputed victims/missing persons are re-
  moved, and data from undisputed victims inserted into the reference data.
- summary: A data frame summarising the undisputed matches.
- LRmatrix: Output from pairwiseLR() applied to the reduced problem.
generatePairings

Sex-consistent pairings

Description
Generate a list of sex-consistent pairings for each victim in a DVI problem. By default, the empty pairing (denoted *) is included for each victim.

Usage
generatePairings(dvi, includeEmpty = TRUE, ignoreSex = FALSE)

Arguments
dvi A dviData object, typically created with dviData().
includeEmpty A logical. If TRUE (default), the do-nothing symbol (*) is included for each victim.
ignoreSex A logical.

Value
A list of character vectors. Each vector is a subset of missing, plus the character * denoting no pairing.

See Also
jointDVI()
getFamily

Examples

```r
pm = list(singleton("V1", sex = 1),
           singleton("V2", sex = 2))

missing = paste0("M", 1:4)
am = list(nuclearPed(children = missing[1:3]),
           nuclearPed(children = missing[4], sex = 2))

dvi = dviData(pm, am, missing)
generatePairings(dvi)
```

---

**getFamily**  
*Get AM component of selected individuals*

**Description**

Get AM component of selected individuals

**Usage**

```r
getFamily(dvi, ids)
```

**Arguments**

- `dvi`  
  A `dviData()` object.
- `ids`  
  A vector of ID labels of members of `dvi$am`.

**Value**

A vector of the same length as `ids`, containing the family names (if `dvi$am` is named) or component indices (otherwise) of the `ids` individuals.

**Examples**

```r
getFamily(example2, ids = example2$missing)
```
getSimpleFams

Find the simple families of a DVI dataset

Description

Extract the names (if present) or indices of the simple reference families, i.e., the families containing exactly 1 missing person.

Usage

gsimpleFams(dvi)

Arguments

dvi A dviData object.

Value

A character (if dvi$am has names) or integer vector.

See Also

ggetFamily()

Examples

# No simple families
simple1 = getSimpleFams(example1)
stopifnot(length(simple1) == 0)

# Second family is simple
simple2 = getSimpleFams(example2)
stopifnot(simple2 == 2)

# With family names
simple3 = example2 |> relabelDVI(familyPrefix = "FAM") |> getSimpleFams()
stopifnot(simple3 == "FAM2")
grave

DVI dataset: Family grave

Description

Family grave data in Kling et al. (2021) "Mass Identifications: Statistical Methods in Forensic Genetics". There are 5 female victims and 3 male victims. There is one reference family with 5 missing females and 3 missing males. There are 23 markers, no mutation model.

Usage

grave

Format

A dviData object with the following content:

- pm: A list of 8 singletons (victims).
- am: A pedigree with 8 missing persons.
- missing: A vector containing the names of the missing persons.

Examples

grave

# plotDVI(grave, marker = 1)

# jointDVI(grave)

icmp

DVI dataset: A large reference pedigree

Description

DVI dataset based loosely on the ICMP workshop material https://www.few.vu.nl/~ksn560/Block-III-PartI-KS-ISFG2017.pdf (page 18). There are 3 female victims, 2 male victims and 6 missing persons of both sexes. We have renamed the individuals and simulated data for 13 CODIS markers (see Details).

Usage

icmp
**Format**

A `dviData` object with the following content:

- **pm**: A list of 5 singletons (victims).
- **am**: A reference pedigree with 6 genotyped members and 12 missing persons.
- **missing**: A vector containing the names of the missing persons.

**Details**

The 13 markers are, in order: CSF1PO, D3S1358, D5S818, D7S820, D8S1179, D13S317, D16S539, D18S51, D21S11, FGA, TH01, TPOX, and vWA.

Source code for the simulation, and a file containing the allele frequencies, can be found in the `data-raw` folder of the GitHub repository: https://github.com/magnusdv/dvir.

**Examples**

```r
icmp

# plotDVI(icmp)

# Markers and allele frequencies
db = pedtools::getFreqDatabase(icmp$pm)

db
```

---

**jointDVI**

*Joint DVI search*

**Description**

Victims are given as a list of singletons, and references as a list of pedigrees. All possible assignments are evaluated and solutions ranked according to the likelihood.

**Usage**

```r
jointDVI(
  dvi,
  pairings = NULL,
  ignoreSex = FALSE,
  assignments = NULL,
  limit = 0,
  nkeep = NULL,
  undisputed = TRUE,
  markers = NULL,
  threshold = 10000,
  strict = FALSE,
  relax = !strict,
)```
disableMutations = NA,
maxAssign = 1e+05,
numCores = 1,
check = TRUE,
verbose = TRUE
)

compactJointRes(jointRes, LRthresh = NULL)

Arguments

dvi A dviData object, typically created with dviData().
pairings A list of possible pairings for each victim. If NULL, all sex-consistent pairings are used.
ignoreSex A logical.
assignments A data frame containing the assignments to be considered in the joint analysis. By default, this is automatically generated by taking all combinations from pairings.
limit A positive number, by default 0. Only pairwise LR values above this are considered.
nkeep An integer, or NULL. If given, only the nkeep most likely pairings are considered for each victim.
undisputed A logical, by default TRUE.
markers A vector indicating which markers should be included in the analysis. By default all markers are included.
threshold A positive number, passed onto findUndisputed(). Default: 1e4.
strict A logical, passed onto findUndisputed(). Default: FALSE.
relax Deprecated.

disableMutations A logical, or NA (default). The default action is to disable mutations in all reference families without Mendelian errors.
maxAssign A positive integer. If the number of assignments going into the joint calculation exceeds this, the function will abort with an informative error message. Default: 1e5.
numCores An integer; the number of cores used in parallelisation. Default: 1.
check A logical, indicating if the input data should be checked for consistency.
verbose A logical.
jointRes A data frame produced by jointDVI().
LRthresh A positive number, used as upper limit for the LR comparing the top result with all others.
Value

A data frame. Each row describes an assignment of victims to missing persons, accompanied with its log likelihood, the LR compared to the null (i.e., no identifications), and the posterior corresponding to a flat prior.

The function `compactJointRes()` removes columns without assignments, and solutions whose LR compared with the top result is below 1/LRthresh.

See Also

`pairwiseLR()`, `findUndisputed()`

Examples

`jointDVI(example2)`

---

Description

Data used in last example of Chapter 4 in Kling et al. (2021) "Mass Identifications: Statistical Methods in Forensic Genetics". There are 2 female victims, 2 male victims. There are four reference families with 2 missing females and 2 missing males. There are 21 markers. An 'equal mutation mode with rate 0.005 is specified.

Usage

KETPch4

Format

A `dviData` object with the following content:

- `pm`: A list of 4 singletons (victims).
- `am`: A list of 3 pedigrees.
- `missing`: A vector containing the names of the missing persons.

Examples

KETPch4

`plotDVI(KETPch4, nrowPM = 4)`
**KETPex481**  
*Data used in the book Kling et al. (2021)*

**Description**

Data used in Example 4.8.1 in Kling et al. (2021) "Mass Identifications: Statistical Methods in Forensic Genetics". The victims are V1 and V2, both females. There is one reference family with 2 missing persons, both females. There are 21 markers, no mutation model.

**Usage**

KETPex481

**Format**

A `dviData` object with the following content:

- `pm`: A list of 2 singletons (victims).
- `am`: A list of 1 pedigree.
- `missing`: A vector containing the names of the missing persons.

**Examples**

```
plotDVI(KETPex481, marker = 1)
```

---

**KETPex497**  
*Data used in the book Kling et al. (2021)*

**Description**

Data used in Exercise 4.9.7 in Kling et al. (2021) "Mass Identifications: Statistical Methods in Forensic Genetics". There are 3 female victims and 3 reference families with 3 missing females. There are 23 markers, equal mutation model, rate 0.001.

**Usage**

KETPex497

**Format**

A `dviData` object with the following content:

- `pm`: A list of 3 singletons (victims).
- `am`: A list of 3 pedigrees.
- `missing`: A vector containing the names of the missing persons.
Examples

plotDVI(KETPex497, nrowPM = 3)

---

**mergePM**  
*Identity and merge matching PM samples*

**Description**

Computes the direct matching LR of each pair of samples, and merges the matching samples.

**Usage**

```r
mergePM(
  pm,
  threshold = 10000,
  method = c("mostcomplete", "first", "combine"),
  verbose = TRUE
)
```

**Examples**

```r
mergePM(
  pm,
  threshold = 10000,
  method = c("mostcomplete", "first", "combine"),
  verbose = TRUE
)
```
mergePM

Arguments

- `pm` A list of typed singletons.
- `threshold` LR threshold for positive identification.
- `method` A keyword indicating how to merging matching samples. See Details.
- `verbose` A logical.

Details

The available methods for merging matched samples are:

- "mostcomplete": Use the sample with the highest number of non-missing genotypes
- "first": Use the first in each group, according to the input order
- "combine": Not implemented yet.

Value

A list with the following entries:

- `groups`: A list containing the groups of matching samples.
- `LRmat`: A symmetric matrix (with 0s on the diagonal) containing the direct matching LR values.
- `nonmissing`: A named vector reporting the number of non-missing genotypes for each sample.
- `pmReduced`: A list of singletons. If `use` is "best" or "first", this is a subset of the input `pm`.

See Also

directMatch()

Examples

```r
pm = singletons(c("V1", "V2", "V3")) |> 
  addMarker(V1 = "1/1", V2 = "2/2", V3 = "1/1", 
             afreq = c("1" = 0.01, "2" = 0.99), name = "L1")
mergePM(pm)
```
The number of assignments for DVI problem

Description

The number of victims and missing persons of each sex is given. The number of possible assignments, i.e., the number of ways the victims can be identified with the missing persons, is calculated.

Usage

```
ncomb(nVfemales, nMPfemales, nVmales, nMPmales)
```

Arguments

- `nVfemales`: Integer. The number of female victims.
- `nMPfemales`: Integer. The number of female missing persons.
- `nVmales`: Integer. The number of male victims.
- `nMPmales`: Integer. The number of male missing persons.

Value

The total number of possible assignments.

Examples

```
# Example
m1 = ncomb(5,5,5) #

# Example: 3 male victims; 2 male missing persons.
# The number of a priori possible assignments is
m1 = ncomb(0,0,3,2) # 13

# Compare with the complete list of assignments
m2 = expand.grid.nodup(list(V1 = c("*", "M1", "M2"),
                           V2 = c("*", "M1", "M2"),
                           V3 = c("*", "M1", "M2")))
stopifnot(m1 == nrow(m2))
```
**pairwiseLR**

**Pairwise LR matrix**

**Description**

For a given DVI problem, compute the matrix consisting of pairwise likelihood ratios \( LR_{i,j} \) comparing \( V_i = M_j \) to the null. The output may be reduced by specifying arguments `limit` or `nkeep`.

**Usage**

```r
pairwiseLR(
  dvi,
  pairings = NULL,
  ignoreSex = FALSE,
  limit = 0,
  nkeep = NULL,
  check = TRUE,
  numCores = 1,
  verbose = FALSE
)
```

**Arguments**

- `dvi`: A `dviData` object, typically created with `dviData()`.
- `pairings`: A list of possible pairings for each victim. If NULL, all sex-consistent pairings are used.
- `ignoreSex`: A logical.
- `limit`: A nonnegative number controlling the `pairings` slot of the output: Only pairings with LR greater or equal to `limit` are kept. If zero (default), pairings with \( LR > 0 \) are kept.
- `nkeep`: An integer, or NULL. If given, only the `nkeep` most likely pairings are kept for each victim.
- `check`: A logical, indicating if the input data should be checked for consistency.
- `numCores`: An integer; the number of cores used in parallelisation. Default: 1.
- `verbose`: A logical.

**Value**

A list with 3 elements:

- `LRmatrix`: A matrix containing the pairwise LR values.
- `LRlist`: A list of numerical vectors, containing the pairwise LR values in list format.
- `pairings`: A reduced version of the input `pairings`, keeping only entries with corresponding \( LR >= \) `limit`. For the default case \( limit = 0 \) a strict inequality is used, i.e., \( LR > 0 \).
Examples

```r
pairwiseLR(example1, verbose = TRUE)
```

---

**DVI dataset: Simulated plane crash**

**Description**

A simulated dataset based on Exercise 3.3 in Egeland et al. "Relationship Inference with Familias and R" (2015).

**Usage**

```r
planecrash
```

**Format**

A dviData object with the following content:

- **pm**: A list of 8 female singletons (victims).
- **am**: A list of 5 pedigrees, each with one missing member and one genotyped member.
- **missing**: A vector containing the names of the missing persons.

**Details**

The 15 markers are CSF1PO, D13S317, D16S539, D18S51, D21S11, D3S1358, D5S818, D7S820, D8S1179, FGA, PENTA,D, PENTA,E, TH01, TPOX, and VWA.

Source code for the simulation, and a file containing the allele frequencies, can be found in the data-raw folder of the GitHub repository: https://github.com/magnusdv/dvir.

**Examples**

```r
planecrash

# plotDVI(planecrash)

# Markers and allele frequencies
db = pedtools::getFreqDatabase(planecrash$pm)
db
```
Description

Plot a DVI problem

Usage

plotDVI(
  dvi,
  pm = TRUE,
  am = TRUE,
  hatched = typedMembers,
  col = list(red = dvi$missing),
  frames = TRUE,
  titles = c("PM", "AM"),
  widths = NULL,
  nrowPM = NA,
  dev.height = NULL,
  dev.width = NULL,
  newdev = !is.null(c(dev.height, dev.width)),
  ...
)

Arguments

dvi A dviData object, typically created with dviData().

pm Either a logical indicating if the PM data should be plotted (as a set of singletons), or a vector of indices selecting a subset of the PM samples. Default: TRUE.

am Either a logical indicating if the AM families data should be plotted, or a vector of indices selecting a subset of the families. Default: TRUE.

hatched A character vector of ID labels, or the name of a function. By default, typed individuals are hatched.

col A list of colour vectors (see pedtools::plot.ped()). By default, missing members of dvi$am are shown in red.

frames A logical, by default TRUE.

titles A character of length 2.

widths A numeric with relative plot widths.

nrowPM The number of rows in the array of PM singletons.

dev.height, dev.width Plot height and widths in inches. These are optional, and only relevant if newdev = TRUE.
newdev  A logical indicating if a new plot window should be opened.

...  Further parameters to be passed on to pedtools::plot.ped(), e.g., marker, cex, cex.main, symbolsize.

Examples

plotDVI(example2)

# Override default layout of PM singletons
plotDVI(example2, nrowPM = 3)

# Subset
plotDVI(example2, pm = 1:2, am = 1, titles = c("PM (1-2)", "AM (1)"))

# AM only
plotDVI(example2, pm = FALSE, titles = "AM families")

# Further options
# plotDVI(example2, new = T, frames = FALSE, marker = 1, cex = 1.2, nrowPM = 1)

plotSolution    Plot DVI solution

Description

A version of plotDVI() tailor-made to visualise identified individuals, for example as reported by jointDVI().

Usage

plotSolution(dvi, assignment, k = 1, format = "[S]=[M]", ...)

Arguments

dvi  A dviData object.

assignment  A named character of the format c(victim = missing, ...), or a data frame produced by jointDVI().

k  An integer; the row number when assignment is a data frame.

format  A string indicating how identified individuals should be labelled, using [M] and [S] as place holders for the missing person and the matching sample, respectively. (See Examples.)

...  Further arguments passed on to plotDVI().

Value

NULL.
plotUndisputed

Examples

```r
res = jointDVI(example2, verbose = FALSE)
plotSolution(example2, res)

# With line break in labels
plotSolution(example2, res, format = "[M]=\n[S]"

# With genotypes for marker 1
plotSolution(example2, res, marker = 1)

# Non-optimal solutions
plotSolution(example2, res, k = 2, pm = FALSE)
plotSolution(example2, res, k = 2, cex = 1.3)
```

plotUndisputed

Plot undisputed identifications

Description

Plot undisputed identifications

Usage

`plotUndisputed(dvi, undisputed, ...)`

Arguments

- `dvi` A dviData object.
- `undisputed` A data frame containing the undisputed matches, typically the entry undisputed in output from `findUndisputed()` (only three first columns used).
- `...` Further arguments passed on to `plotSolution()`.

See Also

`findUndisputed()`, `plotSolution()`

Examples

```r
# Example
res = findUndisputed(example2, threshold = 2, verbose = FALSE)
u = res$summary
plotUndisputed(example2, u, marker = 1)
```
Description

Relabel the individuals and families in a DVI dataset.

Usage

relabelDVI(  
dvi,  
victimPrefix = NULL,  
familyPrefix = NULL,  
refPrefix = NULL,  
missingPrefix = NULL,  
missingFormat = NULL,  
othersPrefix = NULL  
)

Arguments

dvi A dviData object, typically created with dviData().
victimPrefix Prefix used to label PM individuals.
familyPrefix Prefix used to label the AM families.
refPrefix Prefix used to label the reference individuals, i.e., the typed members of the AM families.
missingPrefix Prefix used to label the missing persons. At most one of missingPrefix and missingFormat can be given.
missingFormat A string indicating family-wise labelling of missing persons, using [FAM], [IDX], [MIS] as place holders with the following meanings (see Examples):
  • [FAM]: family index
  • [IDX]: index of missing person within the family
  • [MIS]: index within all missing persons
othersPrefix Prefix used to label other untyped individuals. Use "" for numeric labels (1, 2,...).

Value

A dviData() object.
sequentialDVI

**Examples**

```r
# Builtin dataset `example2`
relabelDVI(example2,
    victimPrefix = "vic",
    familyPrefix = "fam",
    refPrefix = "ref",
    missingPrefix = "mp")

# Family-wise labelling of missing persons
relabelDVI(example2, missingFormat = "M[FAM]-[IDX]")
relabelDVI(example2, missingFormat = "M[IDX] (F[FAM])")
relabelDVI(example2, missingFormat = "fam[FAM].m[IDX]")
```

**sequentialDVI**  
**Sequential DVI search**

**Description**

Performs a sequential matching procedure based on the pairwise LR matrix. In each step the pairing corresponding to the highest LR is selected and included as a match if the LR exceeds the given threshold. By default, (updateLR = TRUE) the pairwise LRs are recomputed in each step after including the data from the identified sample.

**Usage**

```r
sequentialDVI(
    dvi,
    updateLR = TRUE,
    threshold = 1,
    check = TRUE,
    verbose = TRUE,
    debug = FALSE
)
```

**Arguments**

- `dvi`: A `dviData` object, typically created with `dviData()`.
- `updateLR`: A logical. If TRUE, the LR matrix is updated in each iteration.
- `threshold`: A non-negative number. If no pairwise LR values exceed this, the iteration stops.
- `check`: A logical, indicating if the input data should be checked for consistency.
- `verbose`: A logical, by default TRUE.
- `debug`: A logical, by default FALSE. If TRUE, the LR matrix is printed in each step.
Details

If, at any point, the highest LR is obtained by more than one pairing, the process branches off and produces multiple solutions. (See Value.)

Value

A list with two elements:

- matches: A single assignment vector, or (if multiple branches) a data frame where each row is an assignment vector.
- details: A data frame (or a list of data frames, if multiple branches) including the LR of each identification in the order they were made.

Examples

```r
# Without LR updates
sequentialDVI(example1, updateLR = FALSE)

# With LR updates (default). Note two branches!
r = sequentialDVI(example1)

# Plot the two solutions
plotSolution(example1, r$matches, k = 1)
plotSolution(example1, r$matches, k = 2)

# Add `debug = T` to see the LR matrix in each step
sequentialDVI(example1, debug = TRUE)

# The output of can be fed into `jointDVI()`:
jointDVI(example1, assignments = r$matches)
```

setPairing

Set identifications manually

Description

Manually set one or several identifications in a DVI dataset. Typically, these are obtained by external means, e.g., fingerprints, dental records etc.

Usage

```r
setPairing(
  dvi,
  match = NULL,
  victim = NULL,
  missing = NULL,
  Conclusion = "Provided",
```


setPairing

```r
Comment = "",
verbose = TRUE
)
```

Arguments

dvi A DVI dataset.
match A named vector of the format c(vic1 = miss2, vic2 = miss2, ...).
victim A vector of victim sample names. If NULL, defaulting to names(match).
missing A vector of missing person names, of the same length as victim. If NULL, defaulting to as.character(match).

Details

The command `setPairing(dvi, c("V" = "M"))` does the following:

- Transfer the data of victim "V" to the individual "M" in the appropriate reference family
- Remove "M" from the list of missing persons
- Remove "V" from the list of victim samples
- Update the list of pairings

Value

A list with the following entries:

- dviReduced: The new dviData object, as described in Details
- summary: A data frame summarising the identifications

Examples

```r
x = setPairing(example2, match = c("V3" = "M2"))
x$dviReduced
x$summary

# Alternative syntax, using 'victim' and 'missing'
y = setPairing(planecrash, victim = c("V4", "V5"), missing = c("M4", "M5"),
               Conclusion = "External evidence", Comment = "Dental")
y$dviReduced
y$summary
```
subsetDVI  

Extract a subset of a DVI dataset

Description

Extract a subset of a DVI dataset

Usage

subsetDVI(dvi, pm = NULL, am = NULL, missing = NULL, verbose = TRUE)

Arguments

dvi  A dviData() object
pm   A vector with names or indices of victim samples. By default, all are included.
am   A vector with names or indices of AM components. By default, components without remaining missing individuals are dropped.
missing A vector with names or indices of missing persons. By default, all missing persons in the remaining AM families are included.
verbose A logical.

Value

A dviData object.

Examples

subsetDVI(example2, pm = 1:2) |> plotDVI()
subsetDVI(example2, pm = "V1", am = 1) |> plotDVI()
subsetDVI(example2, missing = "M3") |> plotDVI()
Index

* datasets
  example1, 11
  example2, 11
  exclusionExample, 13
  grave, 23
  icmp, 23
  KETPch4, 26
  KETPex481, 27
  KETPex497, 27
  KETPex498, 28
  planecrash, 32

amDrivenDVI, 3
Bmarginal, 4

checkDVI (dviData), 8
combineSummaries, 5
compactJointRes (jointDVI), 24

directMatch, 5
directMatch(), 29
dviCompare, 6
dviData, 8
dviData(), 7, 15, 16, 18–21, 25, 31, 33, 36, 37, 40
dviSim, 9
dviSolve, 10

example1, 11
example2, 11
excludePairing, 12
exclusionExample, 13
exclusionMatrix (findExcluded), 16
expand.grid(), 13, 14
expand.grid.nodup, 13

familias2dvir, 14
findExcluded, 16
findExcluded(), 10, 20
findNonIdentifiable, 17

findUndisputed, 18
findUndisputed(), 10, 17, 25, 26, 35
forrel::findExclusions(), 17
forrel::profileSim(), 9

generatePairings, 20
getFamily, 21
getFamily(), 22
getSimpleFams, 22
grave, 23
icmp, 23

jointDVI, 24
jointDVI(), 3, 4, 15, 17, 20, 34
KETPch4, 26
KETPex481, 27
KETPex497, 27
KETPex498, 28

mergePM, 28
mergePM(), 6

ncomb, 30

pairwiseLR, 31
pairwiseLR(), 20, 26
pedtools::plot.ped(), 33, 34
planecrash, 32
plotDVI, 33
plotDVI(), 34
plotSolution, 34
plotSolution(), 35
plotUndisputed, 35

readFam(), 14, 15
relabelDVI, 36
relabelDVI(), 15

sequentialDVI, 37
setPairing, 38
subsetDVI, 40