Package 'gpyramid'

December 12, 2025

Type Package

Title Identify Efficient Crossing Schemes for Gene Pyramiding

Version 0.0.1

Maintainer Shoji Taniguchi <taniguchi.shoji938@naro.go.jp>

Description Calculates the cost of crossing in terms of the number of individuals and generations, which is theoretically
formulated by Servin et al. (2004) <DOI:10.1534/genetics.103.023358>. This package has been designed for selecting
appropriate parental genotypes and find the most efficient crossing scheme for gene pyramiding, especially for plant breeding.

Imports utils, dplyr, ape

License GPL-2 | GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Shoji Taniguchi [aut, cre],
The National Agriculture and Food Research Organization (NARO) [cph]

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2025-12-12 21:10:02 UTC

Contents

`allCrosses` *Generate all crosses from given parent set.*

Description

Generate all crosses from given parent set.

Usage

```
allCrosses(n, line_id)
```

Arguments

`n`	Number of parents
`line_id`	Character vector of parent names

Value

List of crossing procedure. The output value is defined by "multiPhylo" class.

Note

This function is adapted and modified from ‘allTrees‘, ‘dfactorial‘, and ‘ldfactorial‘ functions in the R package ‘phangorn‘ (GPL-2 | GPL-3). See: https://cran.r-project.org/web/packages/phangorn/index.html

References

K.P. Schliep (2011) phangorn: phylogenetic analysis in R

Examples

```
n <- 3
line_id <- c("x1", "x2", "x7")
rslt <- allCrosses(n, line_id)
plot(rslt[[1]])
plot(rslt[[2]])
plot(rslt[[3]])
```

`calcCost` *Function to calculate N_total by the algorithm proposed by Sevin et al. (2004)*

Description

Function to calculate N_total by the algorithm proposed by Sevin et al. (2004)

Usage

```
calcCost(
  topolo,
  gene_df1_sel,
  gene_df2_sel,
  recom_mat,
  prob_total,
  last_cross = FALSE,
  last_selfing = FALSE
)
```

Arguments

`topolo`	Crossing scheme described by topology of tree
`gene_df1_sel`	Parental set of crossing
`gene_df2_sel`	Parental set of crossing
`recom_mat`	Matrix of recombination rate among genes.
`prob_total`	Probability of success.
`last_cross`	Whether or not to conduct the last cross to a cultivar without target alleles.
`last_selfing`	Whether or not to conduct the last selfing.

Value

'calCost' function returns the list of 'n_plant_df', 'gene_df1_sel' and 'gene_df2_sel'. 'n_plant_df' contains the information about the number of progenies should be produced at each crossing (node in the tree).

`calcCostAll` *Function to calculate the number of necessary individuals and generations as the crossing cost for all the crossing schemes. This is the wrapper function of calCost.*

Description

Function to calculate the number of necessary individuals and generations as the crossing cost for all the crossing schemes. This is the wrapper function of calCost.

Usage

```
calcCostAll(
  line_comb_lis,
  gene_df1,
  gene_df2,
  recom_mat,
  prob_total,
  last_cross = FALSE,
  last_selfing = FALSE
)
```

Arguments

`line_comb_lis`	A list of combinations of parents.
`gene_df1`	Data frame of one set of haplotype. Values take 1 (target allele) or 0 (non-target).
`gene_df2`	Data frame of the other set of haplotype. Values take 1 or 0.
`recom_mat`	Matrix of recombination rate among genes.
`prob_total`	Probability of success.
`last_cross`	Whether or not to conduct the last cross to a cultivar without target alleles.
`last_selfing`	Whether or not to conduct the last selfing.

Value

calcCostAll function returns a ‘gpyramid_all‘ object. This is a named list with the following components: * ‘cost_all‘ (‘data frame‘): Data frame showing the number of necessary generations (n_generation) and the number of individuals (N_total) for each crossing scheme. The crossing schemes are identified by cross_id. * Other components mirror the input parameters for downstream analysis.

`findPset` *Find parent sets from the candidate cultivars*

Description

Find parent sets from the candidate cultivars

Usage

```
findPset(gene_df1, gene_df2, line_id)
```

Arguments

`gene_df1`	Data frame of one set of haplotype. Values take 1 (target allele) or 0 (non-target).
`gene_df2`	Data frame of the other set of haplotype. Values take 1 or 0.
`line_id`	character vector of cultivar names

Value

'findPset()' returns a list of parental cultivar sets. Each element of list contains the parent names.

Examples

```
gene_df1 <-
 data.frame(x1 = c(1, 1, 1, 1, 1, 0, 0),
            x2 = c(1, 1, 1, 0, 1, 1, 0),
            x3 = c(1, 1, 1, 0, 1, 1, 0),
            x4 = c(1, 1, 0, 0, 0, 0, 0),
            x5 = c(0, 0, 1, 0, 1, 1, 0),
            x6 = c(0, 0, 1, 1, 0, 0, 0),
            x7 = c(0, 1, 1, 0, 0, 0, 1))
gene_df2 <-
 data.frame(x1 = c(0, 0, 0, 0, 0, 0, 0),
            x2 = c(0, 0, 0, 0, 0, 1, 0),
            x3 = c(1, 0, 0, 0, 0, 0, 0),
            x4 = c(1, 1, 0, 0, 0, 0, 0),
            x5 = c(0, 0, 1, 0, 1, 1, 0),
            x6 = c(0, 0, 1, 1, 0, 0, 0),
            x7 = c(0, 1, 1, 0, 0, 0, 1))

line_id <- c("x1", "x2", "x3", "x4", "x5", "x6", "x7")

findPset(gene_df1, gene_df2, line_id)
```

getFromAll *Function to get one crossing scheme from all the crossing schemes.*

Description

Function to get one crossing scheme from all the crossing schemes.

Usage

```
getFromAll(gpyramid_all_obj, cross_id)
```

Arguments

gpyramid_all_obj	gpyramid_all object.
cross_id	cross_id in cost_all data frame.

Value

getFromAll function returns a 'getFromAll' object. This is a named list with the following components: * 'topolo' ('phylo'): crossing scheme described as the phylo object, which is defined by ape package. * 'last_cross' ('boolean'): Whether or not to conduct the last cross to a cultivar without target alleles. * 'last_selfint' ('boolean'): Whether or not to conduct the last selfing. * 'cost_onecrossing' ('list'): List of output information about one gene pyramiding scheme. The object has class 'getFromAll' and can be summarized using 'summary()'.

util_haplo *Util fuunction to generate haplotype dataframe from raw data.*

Description

Util fuunction to generate haplotype dataframe from raw data.

Usage

```
util_haplo(in_df, target, non_target, hetero, line_cul)
```

Arguments

in_df	Data frame of raw data.
target	Character of genotype which is the target of gene pyramiding.
non_target	Character of genotype which is not the target of gene pyramiding.
hetero	Character of genotype of heterozygote.
line_cul	Column name containing line identifiers.

Value

A list of genotype and line names for downstream analysis. This is a list with the following components: * ‘gene_df1‘ (‘data frame‘): One set of haplotype. Values take 1 (target allele) or 0 (non-target). * ‘gene_df2‘ (‘data frame‘): The other set of haplotype. Values take 1 or 0. * ‘line_id‘ (‘vector‘): Line names.

Examples

```
in_df <- data.frame(line = c("CV1", "CV2", "CV3", "CV4", "CV5"),
                    gene1 = c("A", "A", "B", "B", "A"),
                    gene2 = c("B", "A", "A", "B", "H"),
                    gene3 = c("A", "A", "H", "H", "A"),
                    gene4 = c("A", "B", "H", "A", "B"))

util_haplo(in_df, target = "A", non_target = "B", hetero = "H", line_cul = "line")
```

`util_recom_mat` *Util function to generate recom_mat from raw data. It returens the recombination probability based on Halden's map function*

Description

Util function to generate recom_mat from raw data. It returens the recombination probability based on Halden's map function

Usage

```
util_recom_mat(in_df, unit = "cM")
```

Arguments

`in_df`	Data frame of raw data. The column names should be "Gene", "Chr" and "cM".
`unit`	Unit of the gene positions. In the current version, it should be "cM".

Value

util_recom_mat function returns a matrix of recombination probability between each combination of genes.

References

Haldane (1919) The combination of linkage values, and the calculation of distances between the loci of linked factors. Journal of Genetics 8: 299-309.

Examples

```
in_df <- data.frame(Gene = c("g1", "g2", "g3", "g4"),
                    Chr = c("1A", "1B", "2A", "2A"),
                    cM = c(30, 50, 35, 50))

util_recom_mat(in_df)
```

Index