Evaluate fitness/mutator effects of one or all possible genotypes.

Description

Given a fitnessEffects/mutatorEffects description, obtain the fitness/mutator effects of a single or all genotypes.

Usage

```
evalGenotype(genotype, fitnessEffects, verbose = FALSE, echo = FALSE,
model = "")

evalGenotypeMut(genotype, mutatorEffects, verbose = FALSE, echo = FALSE)

evalAllGenotypes(fitnessEffects, order = FALSE, max = 256, addwt = FALSE,
model = "")

evalAllGenotypesMut(mutatorEffects, max = 256, addwt = FALSE)


evalGenotypeFitAndMut(genotype, fitnessEffects,
                      mutatorEffects, verbose = FALSE, echo = FALSE,
                      model = "")

evalAllGenotypesFitAndMut(fitnessEffects, mutatorEffects,
                          order = FALSE, max = 256, addwt = FALSE,
                          model = "")
```

Arguments

`genotype`	(For `evalGenotype`). A genotype, as a character vector, with genes separated by "," or ">", or as a numeric vector. Use the same integers or characters used in the fitnessEffects object. This is a genotype in terms of genes, not modules. Using "," or ">" makes no difference: the sequence is always taken as the order in which mutations occurred. Whether order matters or not is encoded in the `fitnessEffects` object.
`fitnessEffects`	A `fitnessEffects` object, as produced by `allFitnessEffects`.
`mutatorEffects`	A `mutatorEffects` object, as produced by `allMutatorEffects`.
`order`	(For `evalAllGenotypes`). If TRUE, then order matters. If order matters, then generate not only all possible combinations of the genes, but all possible permutations for each combination.
`max`	(For `evalAllGenotypes`). By default, no output is shown if the number of possible genotypes exceeds the max. Increase as needed.
`addwt`	(For `evalAllGenotypes`). Add the wildtype (no mutations) explicitly? In case of `frequencyDependentFitness` = TRUE the fitness of WT is always shown.
`model`	Either nothing (the default) or "Bozic". If "Bozic" then the fitness effects contribute to decreasing the Death rate. Otherwise Birth rate is shown (and labeled as Fitness).
`verbose`	(For `evalGenotype`). If set to TRUE, print out the individual terms that are added to 1 (or subtracted from 1, if `model` is "Bozic").
`echo`	(For `evalGenotype`). If set to TRUE, show the input genotype and print out a message with the death rate or fitness value. Useful for some examples, as shown in the vignette.

Value

For `evalGenotype` either the value of fitness or (if `verbose = TRUE`) the value of fitness and its individual components.

For `evalAllGenotypes` a data frame with two columns, the Genotype and the Fitness (or Death Rate, if Bozic). The notation for the Genotype colum is a follows: when order does not matter, a comma "," separates the identifiers of mutated genes. When order matters, a genotype shown as “x > y _ z” means that a mutation in “x” happened before a mutation in “y”; there is also a mutation in “z” (which could have happened before or after either of “x” or “y”), but “z” is a gene for which order does not matter. In all cases, a "WT" denotes the wild-type (or, actually, the genotype without any mutations).

If you use both `fitnessEffects` and `mutatorEffects` in a call, all the genes specified in `mutatorEffects` MUST be included in the `fitnessEffects` object. If you want to have genes that have no direct effect on fitness, but that affect mutation rate, you MUST specify them in the call to `fitnessEffects`, for instance as `noIntGenes` with an effect of 0.

When you are in afrequency dependent fitness situation only `fitnessEffects` are allowed and it must contains `frequencydependentFitness` = TRUE and `spPopSizes` must not be NULL and its length equal to the number of possible genotypes. Here only `evalGenotype` and `evalAllGenotypes` make sense.

Note

Fitness is used in a slight abuse of the language. Right now, mutations contribute to the birth rate for all models except Bozic, where they modify the death rate. The general expression for fitness is the usual multiplicative one of *(1 + s1) (1 + s2) .. (1 + sn)*, where each *s1,s2* refers to the fitness effect of the given gene. When dealing with death rates, we use *(1 - s1) (1 - s2) .. (1 - sn)*.

Modules are, of course, taken into account if present (i.e., fitness is specified in terms of modules, but the genotype is specified in terms of genes).

About the naming. This is the convention used: "All" means we will go over all possible genotypes. A function that ends as "Genotypes" returns only fitness effects (for backwards compatibility and because mutator effects are not always used). A function that ends as "Genotype(s)Mut" returns only the mutator effects. A function that ends as "FitAndMut" will return both fitness and mutator effects.

Functions that return ONLY fitness or ONLY mutator effects are kept as separate functions because they free you from specifyin mutator/fitness effects if you only want to play with one of them.

Author(s)

Ramon Diaz-Uriarte

See Also

`allFitnessEffects`.

Examples

```
# A three-gene epistasis example
sa <- 0.1
sb <- 0.15
sc <- 0.2
sab <- 0.3
sbc <- -0.25
sabc <- 0.4

sac <- (1 + sa) * (1 + sc) - 1

E3A <- allFitnessEffects(epistasis =
                            c("A:-B:-C" = sa,
                              "-A:B:-C" = sb,
                              "-A:-B:C" = sc,
                              "A:B:-C" = sab,
                              "-A:B:C" = sbc,
                              "A:-B:C" = sac,
                              "A : B : C" = sabc)
                                                )

evalAllGenotypes(E3A, order = FALSE, addwt = FALSE)
evalAllGenotypes(E3A, order = FALSE, addwt = TRUE,  model = "Bozic")

evalGenotype("B, C", E3A, verbose = TRUE)

## Order effects and modules
ofe2 <- allFitnessEffects(orderEffects = c("F > D" = -0.3, "D > F" = 0.4),
                          geneToModule =
                              c("Root" = "Root",
                                "F" = "f1, f2, f3",
                                "D" = "d1, d2") )

evalAllGenotypes(ofe2, order = TRUE, max = 325)[1:15, ]

## Next two are identical
evalGenotype("d1 > d2 > f3", ofe2, verbose = TRUE)
evalGenotype("d1 , d2 , f3", ofe2, verbose = TRUE)

## This is different
evalGenotype("f3 , d1 , d2", ofe2, verbose = TRUE)
## but identical to this one
evalGenotype("f3 > d1 > d2", ofe2, verbose = TRUE)


## Restrictions in mutations as a graph. Modules present.

p4 <- data.frame(parent = c(rep("Root", 4), "A", "B", "D", "E", "C", "F"),
                  child = c("A", "B", "D", "E", "C", "C", "F", "F", "G", "G"),
                  s = c(0.01, 0.02, 0.03, 0.04, 0.1, 0.1, 0.2, 0.2, 0.3, 0.3),
                  sh = c(rep(0, 4), c(-.9, -.9), c(-.95, -.95), c(-.99, -.99)),
                  typeDep = c(rep("--", 4),
                      "XMPN", "XMPN", "MN", "MN", "SM", "SM"))
fp4m <- allFitnessEffects(p4,
                          geneToModule = c("Root" = "Root", "A" = "a1",
                              "B" = "b1, b2", "C" = "c1",
                              "D" = "d1, d2", "E" = "e1",
                              "F" = "f1, f2", "G" = "g1"))

evalAllGenotypes(fp4m, order = FALSE, max = 1024, addwt = TRUE)[1:15, ]

evalGenotype("b1, b2, e1, f2, a1", fp4m, verbose = TRUE)

## Of course, this is identical; b1 and b2 are same module
## and order is not present here

evalGenotype("a1, b2, e1, f2", fp4m, verbose = TRUE)

evalGenotype("a1 > b2 > e1 > f2", fp4m, verbose = TRUE)

## We can use the exact same integer numeric id codes as in the
##   fitnessEffects geneModule component:

evalGenotype(c(1L, 3L, 7L, 9L), fp4m, verbose = TRUE)
```

```
## Epistasis for fitness and simple mutator effects

fe <- allFitnessEffects(epistasis = c("a : b" = 0.3,
                                          "b : c" = 0.5),
                            noIntGenes = c("e" = 0.1))

fm <- allMutatorEffects(noIntGenes = c("a" = 10,
                                       "c" = 5))

evalAllGenotypesFitAndMut(fe, fm, order = "FALSE")


## Simple fitness effects (noIntGenes) and modules
## for mutators

fe2 <- allFitnessEffects(noIntGenes =
                         c(a1 = 0.1, a2 = 0.2,
                           b1 = 0.01, b2 = 0.3, b3 = 0.2,
                           c1 = 0.3, c2 = -0.2))

fm2 <- allMutatorEffects(epistasis = c("A" = 5,
                                       "B" = 10,
                                       "C" = 3),
                         geneToModule = c("A" = "a1, a2",
                                          "B" = "b1, b2, b3",
                                          "C" = "c1, c2"))

## Show only all the fitness effects
evalAllGenotypes(fe2, order = FALSE)

## Show only all mutator effects
evalAllGenotypesMut(fm2)

## Show all fitness and mutator
evalAllGenotypesFitAndMut(fe2, fm2, order = FALSE)

## This is probably not what you want
try(evalAllGenotypesMut(fe2))
## ... nor this
try(evalAllGenotypes(fm2))

## Show the fitness effect of a specific genotype
evalGenotype("a1, c2", fe2, verbose = TRUE)

## Show the mutator effect of a specific genotype
evalGenotypeMut("a1, c2", fm2, verbose = TRUE)

## Fitness and mutator of a specific genotype
evalGenotypeFitAndMut("a1, c2", fe2, fm2, verbose = TRUE)

## This is probably not what you want
try(evalGenotype("a1, c2", fm2, verbose = TRUE))

## Not what you want either
try(evalGenotypeMut("a1, c2", fe2, verbose = TRUE))

##Frequency dependent fitness example
r <- data.frame(Genotype = c("WT", "A", "B", "A, B"),
                Fitness = c("1 + 1.5*f_",
                            "5 + 3*(f_A + f_B + f_A_B)",
                            "5 + 3*(f_A + f_B + f_A_B)",
                            "7 + 5*(f_A + f_B + f_A_B)"),
                stringsAsFactors = FALSE)

afe <- allFitnessEffects(genotFitness = r,
                         frequencyDependentFitness = TRUE,
                         spPopSizes = c(5000, 2500, 2500, 500))

evalAllGenotypes(afe)
```