Package ‘OncoSimulR’

October 9, 2015

Type Package

Title Simulation of cancer progresion with order restrictions

Version 1.2.0

Date 2014-07-14

Author Ramon Diaz-Uriarte.

Maintainer Ramon Diaz-Uriarte <`rdiaz02@gmail.com`>

Description Functions for simulating and plotting cancer progression data, including drivers and passengers, and allowing for order restrictions. Simulations use continuous-time models (based on Bozic et al., 2010 and McFarland et al., 2013) and fitness functions account for possible restrictions in the order of accumulation of mutations.

biocViews BiologicalQuestion, SomaticMutation

License GPL (>= 3)

Depends R (>= 3.1.0)

Imports Rcpp (>= 0.11.1), parallel, data.table, graph, Rgraphviz

Suggests BiocStyle, knitr, Oncotree

LinkingTo Rcpp

VignetteBuilder knitr

SystemRequirements C++11

NeedsCompilation yes

R topics documented:

examplePosets *Example posets*

Description

Some example posets. For simplicity, all the posets are in a single list. You can access each poset by accessing each element of the list. The first digit or pair of digits denotes the number of nodes.

Poset 1101 is the same as the one in Gerstung et al., 2009 (figure 2A, poset 2). Poset 701 is the same as the one in Gerstung et al., 2011 (figure 2B, left, the pancreatic cancer poset). Those posets were entered manually at the command line: see `poset`.

Usage

```
data("examplePosets")
```

Format

The format is: List of 13 $ p1101: num [1:10, 1:2] 1 1 3 3 3 7 7 8 9 10 ... $ p1102: num [1:9, 1:2] 1 1 3 3 3 7 7 9 10 2 ... $ p1103: num [1:9, 1:2] 1 1 3 3 3 7 7 8 10 2 ... $ p1104: num [1:9, 1:2] 1 1 3 3 7 7 9 2 10 2 ... $ p901 : num [1:8, 1:2] 1 2 4 5 7 8 5 1 2 3 ... $ p902 : num [1:6, 1:2] 1 2 4 5 7 5 2 3 5 6 ... $ p903 : num [1:6, 1:2] 1 2 5 7 8 1 2 3 6 8 ... $ p904 : num [1:6, 1:2] 1 4 5 5 1 7 2 5 8 6 ... $ p701 : num [1:9, 1:2] 1 1 1 1 2 3 4 4 5 2 ... $ p702 : num [1:6, 1:2] 1 1 1 1 2 4 2 3 4 5 ... $ p703 : num [1:6, 1:2] 1 1 1 1 3 5 2 3 4 5 ... $ p704 : num [1:6, 1:2] 1 1 1 1 4 5 2 3 4 5 ... $ p705 : num [1:6, 1:2] 1 2 1 1 1 2 2 5 4 6 ...

Source

Gerstung et al., 2009. Quantifying cancer progression with conjunctive Bayesian networks. *Bioinformatics*, 21: 2809–2815.

Gerstung et al., 2011. The Temporal Order of Genetic and Pathway Alterations in Tumorigenesis. *PLoS ONE*, 6.

See Also

`poset`

Examples

```
data(examplePosets)

## Plot all of them
par(mfrow = c(3, 5))

invisible(sapply(names(examplePosets),
                 function(x) {plotPoset(examplePosets[[x]],
                     main = x,
                     box = TRUE)}))
```

oncoSimulIndiv *Simulate tumor progression for one or more individuals.*

Description

Simulate tumor progression including possible restrictions in the order of driver mutations. Optionally add passenger mutations. Simulation is done using the BNB algorithm of Mather et al., 2012.

Usage

```
oncoSimulIndiv(poset, model = "Bozic", numPassengers = 30, mu = 1e-6,
               detectionSize = 1e8, detectionDrivers = 4,
               sampleEvery = 1, initSize = 500, s = 0.1, sh = -1,
               K = initSize/(exp(1) - 1), keepEvery = sampleEvery,
               finalTime = 0.25 * 25 * 365, onlyCancer = TRUE,
               max.memory = 2000, max.wall.time = 200,
               verbosity = 0)

oncoSimulPop(Nindiv, poset, model = "Bozic", numPassengers = 30, mu = 1e-6,
               detectionSize = 1e8, detectionDrivers = 4,
               sampleEvery = 1, initSize = 500, s = 0.1, sh = -1,
               K = initSize/(exp(1) - 1), keepEvery = sampleEvery,
               finalTime = 0.25 * 25 * 365, onlyCancer = TRUE,
               max.memory = 2000, max.wall.time = 200,
               verbosity = 0, mc.cores = detectCores())
```

Arguments

`Nindiv`	Number of individuals or number of different trajectories to simulate.
`poset`	The poset that specifies the order restrictions. See `poset`.
`model`	One of "Bozic", "Exp", "McFarlandLog" (the last one can be abbreviated to "McFL").
`numPassengers`	The number of passenger genes.The total number of genes (drivers plus passengers) must be smaller than 64. All driver genes should be included in the poset (even if they depend on no one and no one depends on them), and will be numbered from 1 to the total number of driver genes. Thus, passenger genes will be numbered from (number of driver genes + 1):(number of drivers + number of passengers).
`mu`	Mutation rate.
`detectionSize`	What is the minimal number of cells for cancer to be detected.
`detectionDrivers`	The minimal number of drivers present in any clone for cancer to be detected.

`sampleEvery`	How often the whole population is sampled. This is not the same as the interval between successive samples that keep stored (for that, see `keepEvery`). For very fast growing clones, you might need to have a small value here to minimize possible numerical problems (such as huge increase in population size between two successive samples that can then lead to problems for random number generators). Likewise, for models with density dependence (such as McF) this value should be very small.
`initSize`	Initial population size.
`s`	Selection coefficient for drivers.
`sh`	Selection coefficient for drivers with restrictions not satisfied. A value of 0 means there are no penalties for a driver appearing in a clone when its restrictions are not satisfied. To specify "sh=Inf" (in Diaz-Uriarte, 2014) use sh = -1.
`K`	Initial population equilibrium size in the McFarland models.
`keepEvery`	Time interval between successive whole population samples that are actually stored. This must be larger or equal to sampleEvery. If keepEvery is not a multiple integer of sampleEvery, the keepEvery in use will be the smallest multiple integer of keepEvery larger than the specified keepEvery. If you want nice plots, set `sampleEvery` and `keepEvery` to small values (say, 1 or 0.5). Otherwise, you can use a `sampleEvery` of 1 but a `keepEvery` of 15, so that the return objects are not huge.
`finalTime`	What is the maximum number of time units that the simulation can run.
`onlyCancer`	Return only simulations that reach cancer? If set to TRUE, only simulations that satisfy the detectionDrivers or the detectionSize will be returned (i.e., the simulation will be repeated until one which meets that cancer requirements is met). Otherwise, the simulation is returned regardless of final population size or number of drivers in any clone and this includes simulations where the population goes extinct.
`max.memory`	The largest size (in MB) of the matrix of Populations by Time. If it creating it would use more than this amount of memory, it is not created. This prevents you from accidentally passing parameters that will return an enormous object.
`max.wall.time`	Maximum wall time for each individual simulation run. If the simulation is not done in this time, it is aborted.
`verbosity`	If 0, run as silently as possible. Otherwise, increasing values of verbosity provide progressively more information about intermediate steps, possible numerical notes/warnings from the C++ code, etc.
`mc.cores`	Number of cores to use when simulating more than one individual (i.e., when calling oncoSimulPop).

Details

The basic simulation algorithm implemented is the BNB one of Mather et al., 2012, where I have added modifications to fitness based on the restrictions in the order of mutations.

Full details about the algorithm are provided in Mather et al., 2012. The evolutionary models, including references, and the rest of the parameters are explained in Diaz-Uriarte, 2014, especially in the Supplementary Material. The model called "Bozic" is based on Bozic et al., 2010, and the model called "McFarland" in McFarland et al., 2013.

oncoSimulPop simply calls oncoSimulIndiv multiple times. When run on POSIX systems, it can use multiple cores (via mclapply).

The `summary` methods for these classes return some of the return values (see next) as a one-row (for class oncosimul) or multiple row (for class oncosimulpop) data frame. The `print` methods for these classes simply print the summary.

Value

For `oncoSimulIndiv` a list, of class "oncosimul", with the following components:

`pops.by.time`	A matrix of the population sizes of the clones, with clones in columns and time in row. Not all clones are shown here, only those that were present in at least on of the keepEvery samples.
`NumClones`	Total number of clones in the above matrix.
`TotalPopSize`	Total population size at the end.
`Genotypes`	A matrix of genotypes. For each of the clones in the pops.by.time matrix, its genotype, with a 0 if the gene is not mutated and a 1 if it is mutated.
`MaxNumDrivers`	The largest number of mutated driver genes ever seen in the simulation in any clone.
`MaxDriversLast`	The largest number of mutated drivers in any clone at the end of the simulation.
`NumDriversLargestPop`	The number of mutated driver genes in the clone with largest population size.
`LargestClone`	Population size of the clone with largest number of population size.
`PropLargestPopLast`	Ratio of LargestClone/TotalPopSize
`FinalTime`	The time (in time units) at the end of the simulation.
`NumIter`	The number of iterations of the BNB algorithm.
`HittedWallTime`	TRUE if we reached the limit of max.wall.time. FALSE otherwise.
`TotalPresentDrivers`	The total number of mutated driver genes, whether or not in the same clone. The number of elements in `OccurringDrivers`, below.
`CountByDriver`	A vector of length number of drivers, with the count of the number of clones that have that driver mutated.
`OccurringDrivers`	The actual number of drivers mutated.
`PerSampleStats`	A 5 column matrix with a row for each sampling period. The columns are: total population size, population size of the largest clone, the ratio of the two, the largest number of drivers in any clone, and the number of drivers in the clone with the largest population size.
`other`	A list that contains statistics for an estimate of the simulation error when using the McFarland model. The relevant value is errorMF, which is -99 unless in the McFarland model. For the McFarland model it is the largest difference of successive death rates.

For `oncoSimulPop` a list of length `Nindiv`, and of class `"oncosimulpop"`, where each element of the list is itself a list, of class `oncosimul`, with components as described above.

Author(s)

Ramon Diaz-Uriarte

References

Bozic, I., et al., (2010). Accumulation of driver and passenger mutations during tumor progression. *Proceedings of the National Academy of Sciences of the United States of America\/*, **107**, 18545–18550.

Diaz-Uriarte, R. (2014). Inferring restrictions in the temporal order of mutations during tumor progression: effects of passenger mutations, evolutionary models, and sampling. `http://dx.doi.org/10.1101/005587`

McFarland, C.~D. et al. (2013). Impact of deleterious passenger mutations on cancer progression. *Proceedings of the National Academy of Sciences of the United States of America\/*, **110**(8), 2910–5.

Mather, W.~H., Hasty, J., and Tsimring, L.~S. (2012). Fast stochastic algorithm for simulating evolutionary population dynamics. *Bioinformatics (Oxford, England)\/*, **28**(9), 1230–1238.

See Also

`plot.oncosimul`, `examplePosets`, `samplePop`

Examples

```
## use poset p701
data(examplePosets)
p701 <- examplePosets[["p701"]]

## Bozic Model

b1 <- oncoSimulIndiv(p701)
summary(b1)

plot(b1, addtot = TRUE)

## McFarland; need to modify sampleEvery, but use a reasonable
##   keepEvery.
## We also modify mutation rate to values similar to those in the
##   original paper.
## Note that detectionSize will play no role
## finalTime is large, since this is a slower process
## initSize is set to 4000 so the default K is larger and we are likely
## to reach cancer. Alternatively, set K = 2000.

m1 <- oncoSimulIndiv(p701,
                     model = "McFL",
                     mu = 5e-7,
                     initSize = 4000,
                     sampleEvery = 0.025,
                     finalTime = 15000,
                     keepEvery = 5)
plot(m1, addtot = TRUE, log = "")




## Simulating 4 individual trajectories
## (I set mc.cores = 2 to comply with --as-cran checks, but you
##  should either use a reasonable number for your hardware or
##  leave it at its default value).


p1 <- oncoSimulPop(4, p701,
                   keepEvery = 10,
                   mc.cores = 2)
```

---

plot.oncosimul *Plot simulated tumor progression data.*

Description

Plots data generated from the simulations, either for a single individual or for a population of individuals, with time units in the x axis and nubmer of cells in the y axis. By default, all clones with the same number of drivers are plotted using the same colour (but different line types), and clones with different number of drivers are plotted in different colours.

Usage

```
## S3 method for class 'oncosimul'
plot(x, col = c(8, "orange", 6:1), log = "y",
                    ltyClone = 2:6, lwdClone = 0.2,
                    ltyDrivers = 1, lwdDrivers = 3,
                    xlab = "Time units",
                    ylab = "Number of cells", plotClones = TRUE,
                    plotDrivers = TRUE, addtot = FALSE,
                    addtotlwd = 0.5, yl = NULL, thinData = FALSE,
                    thinData.keep = 0.1, thinData.min = 2, ...)

## S3 method for class 'oncosimulpop'
plot(x, ask = TRUE, col = c(8, "orange", 6:1),
                       log = "y",
                       ltyClone = 2:6, lwdClone = 0.2,
                       ltyDrivers = 1, lwdDrivers = 3,
                       xlab = "Time units",
                       ylab = "Number of cells", plotClones = TRUE,
                       plotDrivers = TRUE, addtot = FALSE,
                       addtotlwd = 0.5, yl = NULL, thinData = FALSE,
                       thinData.keep = 0.1, thinData.min = 2, ...)
```

Arguments

`x`	An object of class `oncosimul` (for `plot.oncosimul`) or `oncosimulpop` (for `plot.oncosimulpop`).
`ask`	Same meaning as in `par`.
`col`	Colour of the lines, where each type of clone (where type is defined by number of drivers) has a different color. If there are many drivers, col is recycled, so you might want to increase the number of possible colours.
`log`	See `log` in `plot.default`. The default, "y", will make the y axis logarithmic.
`ltyClone`	Line type for each clone. Recycled as needed. You probably do not want to use lty=1 for any clone, to differentiate from the clone type, unless you change the setting for `ltyDrivers`.
`lwdClone`	Line width for clones.
`ltyDrivers`	Line type for the driver type.
`lwdDrivers`	Line width for the driver type.
`xlab`	Same as `xlab` in `plot.default`.
`ylab`	Same as `ylab` in `plot.default`.
`plotClones`	Should clones be plotted?
`plotDrivers`	Should clone types (which are defined by number of drivers), be plotted?
`addtot`	If TRUE, add a line with the total populatino size.
`addtotlwd`	Line width for total population size.
`yl`	If non NULL, limits of the y axis. Same as in `plot.default`. If NULL, the limits are calculated automatically.
`thinData`	If TRUE, the data plotted is a subset of the original data. The original data are "thinned" in such a way that the origin of each clone is not among the non-shown data (i.e., so that we can see when each clone/driver originates). Thining is done to reduce the plot size and to speed up plotting..
`thinData.keep`	The fraction of the data to keep (actually, a lower bound on the fraction of data to keep).
`thinData.min`	Any time point for which a clone has a population size < thinData.min will be kept (i.e., will not be removed from) in the data.
`...`	Other arguments passed to `plots`. For instance, `main`.

Author(s)

Ramon Diaz-Uriarte

See Also

`oncoSimulIndiv`

Examples

```
data(examplePosets)
p701 <- examplePosets[["p701"]]

## simulate and plot a single individual
b1 <- oncoSimulIndiv(p701)
plot(b1, addtot = TRUE)


## simulate and plot 2 individuals
## (I set mc.cores = 2 to comply with --as-cran checks, but you
##  should either use a reasonable number for your hardware or
##  leave it at its default value).

p1 <- oncoSimulPop(2, p701, mc.cores = 2)

par(mfrow = c(1, 2))
plot(p1, ask = FALSE)
```

`plotPoset` *Plot a poset.*

Description

Plot a poset. Optionally add a root and change names of nodes.

Usage

```
plotPoset(x, names = NULL, addroot = FALSE, box = FALSE, ...)
```

Arguments

`x`	A poset. A matrix with two columns where, in each row, the first column is the ancestor and the second the descendant. Note that there might be multiple rows with the same ancestor, and multiple rows with the same descendant. See `poset`.
`names`	If not NULL, a vector of names for the nodes, with the same length as the total number of nodes in a poset (which need not be the same as the number of rows; see `poset`). If `addroot = TRUE`, then 1 + the number of nodes in the poset.

addroot	Add a "Root" node to the graph?
box	Should the graph be placed inside a box?
...	Additional arguments to plot (actually, plot.graphNEL in the Rgraphviz package).

Details

The poset is converted to a graphNEL object.

Value

A plot is produced.

Author(s)

Ramon Diaz-Uriarte

See Also

examplePosets, poset

Examples

```
data(examplePosets)
plotPoset(examplePosets[["p1101"]])

## If you will be using that poset a lot, maybe simpler if

poset701 <- examplePosets[["p701"]]
plotPoset(poset701, addroot = TRUE)

## Compare to Pancreatic cancer figure in Gerstung et al., 2011

plotPoset(poset701,
          names = c("KRAS", "SMAD4", "CDNK2A", "TP53",
                     "MLL3","PXDN", "TGFBR2"))

## If you want to show Root explicitly do

plotPoset(poset701, addroot = TRUE,
          names = c("Root", "KRAS", "SMAD4", "CDNK2A", "TP53",
                     "MLL3","PXDN", "TGFBR2"))


## Of course, names are in the order of nodes, so KRAS is for node 1,
##   etc, but the order of entries in the poset does not matter:

poset701b <- poset701[nrow(poset701):1, ]

plotPoset(poset701b,
          names = c("KRAS", "SMAD4", "CDNK2A", "TP53",
                     "MLL3","PXDN", "TGFBR2"))
```

poset *Poset*

Description

Poset: explanation.

Arguments

`x`	The poset. See details.

Details

A poset is a two column matrix. In each row, the first column is the ancestor (or the restriction) and the second column the descendant (or the node that depends on the restriction). Each node is identified by a positive integer. The graph includes all nodes with integers between 1 and the largest integer in the poset.

Each node can be necessary for several nodes: in this case, the same node would appear in the first column in several rows.

A node can depend on two or more nodes (conjunctions): in this case, the same node would appear in the second column in several rows.

There can be nodes that do not depend on anything (except the Root node) and on which no other nodes depend. The simplest and safest way to deal with all possible cases, including these cases, is to have all nodes with at least one entry in the poset, and nodes that depend on no one, and on which no one depends should be placed on the second column (with a 0 on the first column).

Alternatively, any node not named explicitly in the poset, but with a number smaller than the largest number in the poset, is taken to be a node that depends on no one and on which no one depends. See examples below.

Author(s)

Ramon Diaz-Uriarte

References

Posets and similar structures appear in several places. The following two papers use them extensively.

Gerstung et al., 2009. Quantifying cancer progression with conjunctive Bayesian networks. *Bioinformatics*, 21: 2809–2815.

Gerstung et al., 2011. The Temporal Order of Genetic and Pathway Alterations in Tumorigenesis. *PLoS ONE*, 6.

See Also

`examplePosets`, `plotPoset`, `oncoSimulIndiv`

Examples

```
## Node 2 and 3 depend on 1, and 4 depends on no one
p1 <- cbind(c(1, 1, 0), c(2, 3, 4))
plotPoset(p1, addroot = TRUE)

## Node 2 and 3 depend on 1, and 4 to 7 depend on no one.
## We do not have nodes 4 to 6 explicitly in the poset.
p2 <- cbind(c(1, 1, 0), c(2, 3, 7))
plotPoset(p2, addroot = TRUE)

## But this is arguably cleaner
p3 <- cbind(c(1, 1, rep(0, 4)), c(2, 3, 4:7 ))
plotPoset(p3, addroot = TRUE)

## A simple way to create a poset where no gene (in a set of 15) depends
## on any other.

p4 <- cbind(0, 15)
plotPoset(p4, addroot = TRUE)


## Specifying the pancreatic cancer poset in Gerstung et al., 2011
##   (their figure 2B, left). We use numbers, but for nicer plotting we
##   will use names: KRAS is 1, SMAD4 is 2, etc.

pancreaticCancerPoset <- cbind(c(1, 1, 1, 1, 2, 3, 4, 4, 5),
                               c(2, 3, 4, 5, 6, 6, 6, 7, 7))
plotPoset(pancreaticCancerPoset,
          names = c("KRAS", "SMAD4", "CDNK2A", "TP53",
                    "MLL3","PXDN", "TGFBR2"))


## Specifying poset 2 in Figure 2A of Gerstung et al., 2009:

poset2 <- cbind(c(1, 1, 3, 3, 3, 7, 7, 8, 9, 10),
                c(2, 3, 4, 5, 6, 8, 9, 10, 10, 11))

plotPoset(poset2)
```

samplePop *Obtain a sample from a population of simulations.*

Description

Obtain a sample (a matrix of individuals by genes or, equivalently, a vector of genotypes) from an oncosimulpop object (i.e., a simulation of multiple individuals) or a single oncosimul object. Sampling schemes include whole tumor and single cell sampling, and sampling at the end of the tumor progression or during the progression of the disease.

Usage

```
samplePop(x, timeSample = "last", typeSample = "whole",
          thresholdWhole = 0.5)
```

Arguments

`x`	An object of class `oncosimulpop`.
`timeSample`	"last" means to sample each individual in the very last time period of the simulation. "unif" (or "uniform") means sampling each individual at a time choosen uniformly from all the times recorded in the simulation between the time when the first driver appeared and the final time period. "unif" means that it is almost sure that different individuals will be sampled at different times. "last" does not guarantee that different individuals will be sample at the same time unit, only that all will be sampled in the last time unit of their simulation.
`typeSample`	"singleCell" (or "single") for single cell sampling, where the probability of sampling a cell (a clone) is directly proportional to its population size. "wholeTumor" (or "whole") for whole tumor sampling (i.e., this is similar to a biopsy being the entire tumor).
`thresholdWhole`	In whole tumor sampling, whether a gene is detected as mutated depends on thresholdWhole: a gene is considered mutated if it is altered in at least thresholdWhole proportion of the cells in that individual.

Details

samplePop simply repeats the sampling process in each individual of the oncosimulpop object.

Value

A matrix. Each row is a genotype, where 0 denotes no alteration and 1 alteration. To make it more clear that genes/laterations are in columns, columns are named starting by "G." (for "gene").

Author(s)

Ramon Diaz-Uriarte

References

Diaz-Uriarte, R. (2014). Inferring restrictions in the temporal order of mutations during tumor progression: effects of passenger mutations, evolutionary models, and sampling. `http://dx.doi.org/10.1101/005587`

See Also

`oncoSimulPop`

Examples

```
data(examplePosets)
p705 <- examplePosets[["p705"]]

## (I set mc.cores = 2 to comply with --as-cran checks, but you
##  should either use a reasonable number for your hardware or
##  leave it at its default value).

p1 <- oncoSimulPop(4, p705, mc.cores = 2)
samplePop(p1)


## Now single cell sampling

r1 <- oncoSimulIndiv(p705)
samplePop(r1, typeSample = "single")
```

Index