Package 'caspr'

January 13, 2016

Type Package

Title Cellular Automata for Spatial Pressure in R

Version 0.2.1

Date 2016-01-13

Author Alain Danet, Alex Genin, Vishwesha Guttal, Sonia Kefi, Sabiha Majumder, Sumithra Sankaran, Florian Schneider (Maintainer)

Maintainer Florian D. Schneider <florian.schneider@univ-montp2.fr>

URL https://github.com/fdschneider/caspr

Description The project package for a study on spatial indicators of catastrophic shifts. Functions to run and analyse a set of cellular automata models that implement a spatially explicit pressure.

License MIT + file LICENSE

Additional_repositories https://github.com/fdschneider/

Imports foreach, moments, SDMTools, Rcpp

Suggests testthat, doSNOW, spatialwarnings, knitr

LinkingTo Rcpp

VignetteBuilder knitr

RoxygenNote 5.0.1

Archs x64

R topics documented:

`as.array.ca_result` *Transfer output of runca() into an array.*

Description

Transfer output of runca() into an array.

Usage

```
## S3 method for class 'ca_result'
as.array(x)
```

Arguments

`x`

Value

an array with dimensions 'width', 'height' and 'snaps'.

`as.landscape` *generic S3 function*

Description

generic S3 function

Usage

```
as.landscape(...)
```

as.landscape.matrix *transfer matrix to landscape.*

Description

transfer matrix to landscape.

Usage

```
## S3 method for class 'matrix'
as.landscape(x, states = NA)
```

Arguments

x	A matrix object with factorial content.
levels	A character vector giving the original sequence of states.

Details

If no vector levels is specified the levels are coerced from the unique states of the matrix, which might cause loss of states (e.g. if not present on this landscape) or a wrong ordering of states. This is not recommended! Always provide a levels vector!

Value

A landscape object.

as.list.ca_result *Transfer output of ca() into a list of matrices.*

Description

Transfer output of ca() into a list of matrices.

Usage

```
## S3 method for class 'ca_result'
as.list(x)
```

Arguments

x

Value

a list of matrices.

as.matrix.landscape *transfer landscape to matrix.*

Description

transfer landscape to matrix.

Usage

```
## S3 method for class 'landscape'
as.matrix(x, as = "character")
```

Arguments

x	A landscape object.

Value

A matrix.

birthdeath *Birth-death model*

Description

Model with two states and a probability of switching between the two.

Usage

```
ca(l, birthdeath, parms = p)
```

Format

An object of class `ca_model` of length 6.

Details

This model implements only two states (dead or alive). Each cell can transition to the other depending only on a probability transition p (dead -> alive) or q (alive -> dead). This model serves as a null model for the others provided for caspr.

Author(s)

None

References

None

ca *Run cellular automata simulation.*

Description

Run cellular automata simulation.

Usage

```
ca(x, model = grazing, parms = "default", t_max = 200, saveeach = 1,
  stopifsteady = FALSE, steady = caspr::steady, steadyparms = list(t_eval
  = 200, accept = 0.001), t_min = 0, plotting = FALSE,
  filename = "modelrun", seed = NULL, ...)
```

Arguments

x	A landscape object.
model	A valid object of class 'ca_model'. Defaults to musselbed. Valid values are: musselbed, grazing.
parms	A list of parameters with one or several parameters containing a vector of parameter values. Those will be combined full-factorially using expand.grid(). Will be checked against template parameters in model. If not provided, template parameters will be used.
t_max	Maximal number of timesteps. Model will be terminated even if still in transient dynamics (i.e. model did not reach steady state)
saveeach	Timespan between timesteps at which a full snapshot of the landscape is saved into the output of the simulation.
stopifsteady	Binary parameter, defaults to FALSE. If TRUE, the function provided in parameter steady will be applied to test in each timestep if steady state is reached.
steady	A function returning TRUE or FALSE, taking exactly the parameters i, result, steadyparms. By default the function returns TRUE if the difference in mean cover of the primary cell state (i.e. the first in the vector provided in model$states) over two subsequent timespans of length steadyparms$t_eval is smaller than steadyparms$accept.
steadyparms	a list of parameters that are required by the function provided in steady.
plotting	A binary variable. If TRUE, simulation is plotted into an animated gif.
filename	A character string. Filename of animated gif (defaults to "modelrun.gif") which will be placed in current working directory. If not provided global seeds of R apply.
seed	An integer number serving as seed for random number generation.
...	Parameters handed over to update function in model$update.
t_eval	Timespan of moving window that is evaluated for the end of transient dynamics.

Details

Runs iterations of the update function model$update() on the initial landscape x until a t_max is reached. The function saves the full timeseries, i.e. a value for each timestep, of the global cover of each state as well as the average local cover of each state. The full landscape object of each timestep is stored in a list result$landscapes of the output object, but frequency of these snapshots can be altered by increasing the parameter saveeach.

Value

The output is returned as a list object of class `ca_result`, containing a full timeseries of global and local cover as well as snapshots of the landscape.

`$model` The entire model object used to generate this simulation run, including the parameters at `$model$parms`

`$time` Vector of timesteps.

`$issteady` Binary vector reporting for each timestep if the criterion for steady state was fulfilled. The criterion can be customized by adjusting parameter `steady`.

`$cover` A list of cover timeseries for each state of the model.

`$local` A list of local cover timeseries for each state of the model.

`$snaps` A vector of indices of saved snapshots.

`$landscapes` A list of landscape objects at each point in `$snaps`

`$issteady` A binary vector of the returned values of function `steady`

Examples

```
# 1. run simulation and save a snapshot each 50 timesteps. plot timeseries and snapshots.

l <- init_landscape(c("+","0","-"), c(0.6,0.2,0.2), width = 100)
p <- list(r = 0.4, d = 0.9, delta = 0.01)   # set parameters
r <- ca(l, model = musselbed, parms = p, t_max = 200)    # run simulation
plot(r)

par(mfrow= c(2,3))
sapply(c(0,25,50,100,150,200)+1, function(i) plot(r$landscapes[[i]]) )

# 2. run simulation and save full landsape at each timestep. create animated gif.

l <- init_landscape(c("1","0"), c(0.6,0.4), 100)
r <- ca(l, model = life, t_max = 400)
animate(r, "life01.gif")
```

`ca_animate` *Create animated gif or video from simulation result.*

Description

Create animated gif or video from simulation result.

Usage

```
ca_animate(x, filename, type = "gif", speed = 1, directory = getwd(), ...)
```

Arguments

x	A ca_result object (generated by ca())
filename	A character value specifying relative path and filename. File endings will be ignored and replaced by value in type.
type	A character value specifying the type of output. Either "gif" (tiny animated gif), "mp4", "wmv", or "avi".
speed	A numeric value specifying the relative speed of the output (defaults to 1).
directory	A character string specifying the absolute or relative target path. Defaults to current working directory.
...	further parameters will be handed to the function saveGIF or saveVIDEO.

Details

This requires R-package [animation] and the additional software ImageMagick (for animated .gif) and FFMPEG (for .mpeg videos).

Value

Returns an animated gif or video.

See Also

animate.ca_model

Examples

```
# 1. run simulation and save full landsape at each timestep. create animated gif.

l <- init_landscape(c("1","0"), c(0.6,0.4), 100)
r <- ca(l, model = life, t_max = 500)
ca_animate(r, "life01.gif")
```

---

ca_array *Run ca() over a parameter array.*

Description

Run ca() over a parameter array.

Usage

```
ca_array(model, init, width = 50, height = width, parms = model$parms,
  save = FALSE, filename = "sim", directory = "", salt = 345678,
  stat_covers_length = 10, ...)
```

Arguments

model	A valid object of class 'ca_model'. Defaults to musselbed. Valid values are: musselbed, grazing.
init	A valid numerical vector of initial cover. Length must be equal to length(model$states).
width	An integer number. Defaults to 50 and is ignored if a landscape object is provided in init.
height	An integer number. Defaults to width if provided and is ignored if a landscape object is provided in init.
parms	A list of parameters with one or several parameters containing a vector of parameter values. Those will be combined full-factorially using expand.grid()
save	Logical. If TRUE, each individual run is saved to a file. They can be loaded into the current working environment using load("filename").
filename	A character vector specifying the root of the filename of the saved files, including a relative path. It will be extended by an individual iteration ID and the fileending ".Rd". Note: If running in parallel on a cluster, output files will be saved in the workers home directory!
salt	An integer number. Used as seed when generating seeds for each single simulation run. Those generated seeds are returned in the output of the function.
...	parameters handed over to function ca().

Details

The function is used to create gradients along one parameter value or an array of parameter values. It runs the simulation for each parameter value or combination of parameter values while making use of a parallel backend provided by foreach.

Make sure to adapt the type of parallel backend to your computer infrastructure (see package vignette of the foreach -package).

Value

Returns a dataframe with global and local cover for each state for each parameter value or combination of parameter values given in parms.

The function returns a dataframe with global and local cover for each state for each parameter value or combination of parameter values given in 'parms', as well as the seed used for the individual simulation run.

Examples

```
## Not run:
  p <- list(
    r = 0.4, # recolonisation of empty sites dependent on local density
    d = seq(0,1,0.1), # wave disturbance
    delta = 0.01, # intrinsic disturbance rate
    replicates = 1:2  # repeat the same parameter set twice
  )

# provides parallel backend
 library(foreach)
 library(doMC)
```

```
registerDoMC(cores=7)

musselgradient <- ca_array(musselbed, parms = p, init = c(0.6,0.2,0.2), t_max = 400, save = TRUE, file = "out

registerDoSEQ()

musselgradient
plot(musselgradient[,"mean_cover.+"] ~ musselgradient$d, pch = 20)

## End(Not run)
```

---

ca_arraySS *Run ca_snapsSS() over a parameter array.*

Description

Run ca_snapsSS() over a parameter array.

Usage

```
ca_arraySS(model, init, width = 50, height = width, parms = model$parms,
  save = FALSE, filename = "sim", directory = "", salt = 345678,
  length.stat = 10, ...)
```

Arguments

model	A valid object of class 'ca_model'. Defaults to musselbed. Valid values are: musselbed, grazing.
init	A valid numerical vector of initial cover. Length must be equal to length(model$states).
width	An integer number. Defaults to 50 and is ignored if a landscape object is provided in init.
height	An integer number. Defaults to width if provided and is ignored if a landscape object is provided in init.
parms	A list of parameters with one or several parameters containing a vector of parameter values. Those will be combined full-factorially using expand.grid()
save	Logical. If TRUE, each individual run is saved to a file. They can be loaded into the current working environment using load("filename").
filename	A character vector specifying the root of the filename of the saved files, including a relative path. It will be extended by an individual iteration ID and the fileending ".Rd". Note: If running in parallel on a cluster, output files will be saved in the workers home directory!
salt	An integer number. Used as seed when generating seeds for each single simulation run. Those generated seeds are returned in the output of the function.
...	parameters handed over to function ca_snapsSS(). Recommended the use of nsnaps in ca_snapsSS() to specify the number of landscape replicate to be saved per combination of parameters in parms.

Details

The function is used to create gradients along one parameter value or an array of parameter values. It runs the simulation for each parameter value or combination of parameter values while making use of a parallel backend provided by foreach.

Make sure to adapt the type of parallel backend to your computer infrastructure (see package vignette of the foreach -package).

Value

Returns a list containing two elements. Firt one is a dataframe with global and local cover for each state for each parameter value or combination of parameter values given in parms. Second one is a list containing the specified number of snapshots in steady state for each parameter values comination given in parms.

The function returns a dataframe with global and local cover for each state for each parameter value or combination of parameter values given in 'parms', as well as the seed used for the individual simulation run.

Examples

```
## Not run:
  p <- list(
    r = 0.4, # recolonisation of empty sites dependent on local density
    d = seq(0,1,0.1), # wave disturbance
    delta = 0.01, # intrinsic disturbance rate
    replicates = 1:2  # repeat the same parameter set twice
  )

# provides parallel backend
 library(foreach)
 library(doMC)

 registerDoMC(cores=7)

 musselgradient <- ca_array(musselbed, parms = p, init = c(0.6,0.2,0.2), t_max = 400, save = TRUE, file = "out

 registerDoSEQ()

 musselgradient
 plot(musselgradient[,"mean_cover.+"] ~ musselgradient$d, pch = 20)

## End(Not run)
```

---

ca_snapsSS *Run cellular automata simulation.*

Description

Run cellular automata simulation.

Usage

```
ca_snapsSS(x, model = grazing, parms = "default", t_min = 500,
  t_max = 3000, nsnaps = 10, stopifsteady = FALSE,
  steady = caspr::steady, steadyparms = list(t_eval = 200, accept = 0.001),
  plotting = FALSE, filename = "modelrun", seed = NULL,
  length.stat = 100, ...)
```

Arguments

x	A landscape object.
model	A valid object of class 'ca_model'. Defaults to musselbed. Valid values are: musselbed, grazing.
parms	A list of parameters with one or several parameters containing a vector of parameter values. Those will be combined full-factorially using expand.grid(). Will be checked against template parameters in model. If not provided, template parameters will be used. @param t_min Minimum timespan for starting model evaluation of the steady state
t_max	Maximal number of timesteps. Model will be terminated even if still in transient dynamics (i.e. model did not reach steady state)
nsnaps	number of independent snapshots to be saved after steady state is achieved. Minimum is 3. Default is 10.
stopifsteady	Binary parameter, defaults to FALSE. If TRUE, the function provided in parameter steady will be applied to test in each timestep if steady state is reached.
steady	A function returning TRUE or FALSE, taking exactly the parameters i, result, steadyparms. By default the function returns TRUE if the difference in mean cover of the primary cell state (i.e. the first in the vector provided in model$states) over two subsequent timespans of length steadyparms$t_eval is smaller than steadyparms$accept.
steadyparms	a list of parameters that are required by the function provided in steady.
plotting	A binary variable. If TRUE, simulation is plotted into an animated gif.
filename	A character string. Filename of animated gif (defaults to "modelrun.gif") which will be placed in current working directory.
	If not provided global seeds of R apply.
seed	An integer number serving as seed for random number generation.
...	Parameters handed over to update function in model$update.
t_eval	Timespan of moving window that is evaluated for the end of transient dynamics.

Details

Runs iterations of the update function model$update() on the initial landscape x until a t_max is reached. The function saves the full timeseries, i.e. a value for each timestep, of the global cover of each state as well as the average local cover of each state. The full landscape object of each timestep is stored in a list result$landscapes of the output object, but frequency of these snapshots can be altered by increasing the parameter saveeach.

Value

The output is returned as a list object of class `ca_result`, containing a full timeseries of global and local cover as well as snapshots of the landscape.

`$model` The entire model object used to generate this simulation run, including the parameters at `$model$parms`

`$time` Vector of timesteps.

`$issteady` Binary vector reporting for each timestep if the criterion for steady state was fulfilled. The criterion can be customized by adjusting parameter `steady`.

`$cover` A list of cover timeseries for each state of the model.

`$local` A list of local cover timeseries for each state of the model.

`$snaps` A vector of indices of saved snapshots.

`$landscapes` A list of landscape objects at each point in `$snaps`

`$issteady` A binary vector of the returned values of function `steady`

Examples

```
# 1. run simulation and save a snapshot each 50 timesteps. plot timeseries and snapshots.

l <- init_landscape(c("+","0","-"), c(0.6,0.2,0.2), width = 100)
p <- list(r = 0.4, d = 0.9, delta = 0.01)   # set parameters
r <- ca(l, model = musselbed, parms = p, t_max = 200)    # run simulation
plot(r)

par(mfrow= c(2,3))
sapply(c(0,25,50,100,150,200)+1, function(i) plot(r$landscapes[[i]]) )

# 2. run simulation and save full landsape at each timestep. create animated gif.

l <- init_landscape(c("1","0"), c(0.6,0.4), 100)
r <- ca(l, model = life, t_max = 400)
animate(r, "life01.gif")
```

---

count.landscape *Count neighbors.*

Description

Count neighbors.

Usage

```
## S3 method for class 'landscape'
count(x, neighbor)
```

Arguments

`x`	A landscape object.
`neighbor`	A character value. The state to count.

Value

Returns a vector of the counts in the neighborhood specified by the `mapping`, by default the 4-cell neighborhood.

`forestgap` *Forest gap model*

Description

Dynamic model for forest pattern after recurring wind disturbance with two cell states: empty (`"0"`) and vegetated (`"+"`).

Usage

```
ca(l, forestgap, parms = p)
```

Arguments

`alpha`	A numerical value. Reproduction rate per year.
`delta`	A numerical value. Death rate depending on local gap density.
`d`	A numerical value. intrinsic death rate.

Format

An object of class `ca_model` of length 6.

Details

This model can use either an explicit height for trees, in which case states can be anywhere in a range [Smin...Smax] (SolÃ© et al., 1995), or use only two states, vegetated (non-gap) and empty (gap) (Kubo et al., 1996). Here we focus on the version that uses only two states: gap (0) and non-gap (+). Without spatial spreading of disturbance (all cells are independent), a cell transitions from empty to vegetated with a birth probability b and from vegetated to empty with death probability d.

However, gap expansion occurs in nature as trees having empty (non-vegetated) surroundings are more likely to fall due to disturbance (e.g. wind blows). Let p(0) be the proportion of neighbouring sites that are gaps. We can implement this expansion effect by modifying the death rate into `d + delta p(0)`. Since 0 <= p(0) <= 1, delta represents the maximal added death rate due to gap expansion (i.e. the spatial component intensity).

In their simulations, the authors (Kubo et al. 1996) use a 100x100 torus-type lattice (with random initial covers?).

The authors consider two cases: one in which the recovery of trees is proportional to the global density of vegetated sites, and one where the recovery is proportional to the local density of vegetation. We use only the first case as it the only one producing bistability.

The birth rate b is replaced with alpha rho+ where rho+ represents the global density of non-gap sites and alpha is a positive constant. This can produce alternative stable states over a range of delta values within 0.15-0.2(alpha is fixed to 0.20 and d to 0.01).

The state transition probabilities thus become:

b = alpha rho+

and

d = d_0 + delta p_0

Author(s)

Kubo, T., Y. Iwasa and N. Furumoto.

References

Kubo, T. et al. (1996) Forest spatial dynamics with gap expansion: total gap area and gap size distribution. J. Theor. Biol. 180, 229â€“246

Kizaki, S. and Katori, M. (1999) Analysis of canopy-gap structures of forests by Isingâ€“Gibbs states â€“ Equilibrium and scaling property of real forests. J. Phys. Soc. Jpn 68, 2553â€“2560

Katori, M. (1998) Forest dynamics with canopy gap expansion and stochastic Ising model. Fractals 6, 81â€“86

SolÃ©, R.V. and Manrubia, S.C. (1995) Self-similarity in rain forests: evidence for a critical state. Phys. Rev. E. 51, 6250â€“6253

SolÃ©, R.V. and Manrubia, S.C. (1995) Are rainforests self-organized in a critical state? J. Theor. Biol. 173, 31â€“40

See Also

Other models: grazing, life, livestock, musselbed, predprey

Examples

```
l <- init_landscape(c("+","0"), c(0.6,0.4), width = 100) # create initial landscape
p <- list(alpha = 0.2, delta = 0.17, d = 0.01)   # set parameters
r <- ca(l, model = forestgap, parms = p, t_max = 100)    # run simulation

r
plot(r)
```

`grazing` *Grazing model*

Description

A spatially-explicit model of grazing pressure in drylands.

Usage

```
ca(l, grazing)
```

Arguments

`del`	local seed dispersal
`b`	environmental quality
`c_`	global competition
`m0`	intrinsic mortality
`g`	grazing pressure

r	regeneration rate of degraded cells
f	local facilitation
d	intrinsic degradation rate
p	associational resistance against grazing

Format

An object of class ca_model of length 6.

Details

The model builds upon a published model by Kefi et al. 2007. Spatial models of vegetation cover so far have considered grazing mortality a rather constant pressure, affecting all plants equally, regardless of their position in space. In the known models it usually adds as a constant to the individual plant risk (Kefi et al 2007 TPB). However, grazing has a strong spatial component: Many plants in rangelands invest in protective structures such as thorns or spines, or develop growth forms that reduce their vulnerability to grazing. Therefore, plants growing next to each other benefit from the protection of their neighbors.

Such **associational resistance** is widely acknowledged in vegetation ecology but hardly integrated in models as a cause for spatially heterogenous grazing pressure. It also renders the plant mortality density dependent, which has important impacts on the bistability of the system.

The model investigates how the assumption of spatially heterogeneous pressure alters the bistability properties and the response of spatial indicators of catastrophic shifts.

The model knows three different cell states: occupied by vegetation "+", empty but fertile "0" and degraded "-". Transitions between cell states are only possible between vegetated and empty (by the processes of plant 'death' and 'recolonization') and between empty and degraded (by 'degradation' and 'regeneration').

To account for the spatially heterogeneous impacts of grazing due to associational resistance, we assumed that a plant's vulnerability to grazers decreases with the proportion of occupied neighbors, $q_+|+$. The individual probability of dying is therefore defined as

$$w_{\{+,0\}} = m_0 + g_0 \left(1 - q_{+|+}\right)$$

where the additional mortality due to grazing is maximized to g_0 if a plant has no vegetated neighbor (i.e., $q_{+|+} = 0$) and gradually reduces to 0 with an increasing fraction of occupied neighbors, $q_{+|+}$.

Author(s)

Florian D. Schneider and Sonia Kefi (2015, in revision)

See Also

project on GitHub

Other models: forestgap, life, livestock, musselbed, predprey

Examples

```
l <- init_landscape(c("+","0","-"), c(0.6,0.2,0.2), width = 50) # create initial landscape
p <- list(del = 0.9, b = 0.4, c_ = 0.2, m0 = 0.05, g = 0.25, r = 0.01, f = 0.9, d = 0.1, p = 1)  # set parameter
r <- ca(l, model = grazing, parms = p, t_max = 100)    # run simulation
```

init_landscape — *Create a new landscape object, i.e. a grid for cellular automata processing.*

Description

Create a new landscape object, i.e. a grid for cellular automata processing.

Usage

```
init_landscape(states, cover, width = 50, height = width)
```

Arguments

states	A character vector containing the potential cell states.
cover	A number vector containing the cover.
width	An integer number.
height	An integer number.

Value

A landscape object of dimensions width x height with random distribution of states, in the relative ratio given in cover.

Examples

```
init_landscape(c("+","0","-"), c(0.5,0.25,0.25))
```

life — *Conway's Game of Life*

Description

Model implementation for Conway's Game of Life.

Usage

```
l <- init_landscape(c("1","0"), c(0.35,0.65), 150,150)
run <- ca(l, life)
```

Format

An object of class ca_model of length 7.

Details

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.

The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves or, for advanced players, by creating patterns with particular properties.

Rules

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if caused by under-population.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overcrowding.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed - births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.

Source: Wikipedia

Author(s)

John H. Conway

References

Gardner, Martin (October 1970). Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". Scientific American 223. pp. 120â€“123. ISBN 0-89454-001-7.

See Also

Other models: `forestgap`, `grazing`, `livestock`, `musselbed`, `predprey`

`livestock` *Livestock resilience model*

Description

A spatially-explicit model of resilience in arid rangelands.

Usage

```
ca(l, livestock, parms = p)
```

Arguments

b	environmental quality
m	intrinsic plant mortality, i.e. inverse of av. lifespan
r	max. regeneration rate of plants
K	carrying capacity of the landscape (i.e. global competition)
f	local facilitation
c	local competition
L	livestock density
a	attack rate of livestock
h	handling time of livestock
q	hill-exponent of livestock
p	associational resistance against livestock grazing
v	attractant-decoy effect of plants to livestock

Format

An object of class `ca_model` of length 6.

Details

An unpublished, generalized model of positive and negative local feedbacks in arid rangelands, including mechanisms such as local facilitation, competition, associational resistance and attractant-decoy.

Author(s)

F.D. Schneider

See Also

Other models: `forestgap`, `grazing`, `life`, `musselbed`, `predprey`

`mapping` *Create vector maps for cellular automata updating.*

Description

Create vector maps for cellular automata updating.

Usage

```
mapping(width, height = width, boundary = "periodic",
  i_matrix = matrix(c(0, 1, 0, 1, NA, 1, 0, 1, 0), ncol = 3, byrow = TRUE))
```

Arguments

width	An integer number giving the width of the landscape or a landscape object.
height	An integer number. The height of the landscape. Defaults to value given in width.
boundary	A character value representing the type of boundary conditions. Defaults to "periodic". No other boundaries implemented, yet.
i_matrix	A matrix defining the interaction matrix to be evaluated. Defaults to 4-cell neighborhood.

Details

This function and the objects generated by it are necessary for the function neighbors to vectorise the calculation of neighbors. The interaction matrix can be flexibly defined as a matrix that has precisely one cell with value NA, marking the focal cell. Any number larger than 0 marks a cell that is to be taken into account for counting.

Value

Creates a map of the landscape object to translate it into a vector with boundaries (x_with_border) and another one to back-translate it to a vector without boundaries (x_to_evaluate) into the global environment. This will automatically be called within ca().

---

musselbed *Musselbed Disturbance Model*

Description

Model for spatial pattern of intertidal mussel beds with wave disturbance. With three cell states: occupied by mussels ("+"), empty sites ("0" and disturbed sites ("-").

Usage

```
ca(l, musselbed, parms = p)
```

Arguments

r	A numerical value. recolonisation of empty sites dependent on local density.
d	A numerical value. probability of disturbance of occupied sites if at least one disturbed site is in the direct 4-cell neighborhood.
delta	A numerical value. intrinsic disturbance rate.

Format

An object of class ca_model of length 7.

Details

The model represents the spatial dynamics in mussel cover of rock substrate in intertidal systems. The stochastic wave disturbances will most likely remove mussels that are located next to a gap because of the losened byssal threads in their proximity. This causes a dynamic gap growth.

The model describes the process by simplifying the system into three potential cell states: occupied by mussel ("+"), empty but undisturbed ("0"), and disturbed, bare rock with loose byssal threads ("-").

Mussel growth on empty cells is defined by parameter `r` multiplied by the local density of mussels in the direct 4-cell neighborhood.

Any cell occupied by mussels has an intrinsic chance of `delta` to become disturbed from intrinsic cause, e.g. natural death or predation. Additionally, wave disturbance will remove mussels and leave only bare rock, i.e. disturbed sites, with probability `d` if at least one disturbed cell is in the direct 4-cell neighborhood. This causes disturbances to cascade through colonies of mussels.

Disturbed sites will recover into empty sites with a constant rate of 1 per year, i.e. on average a disturbed site becomes recolonisable within one year after the disturbance happened.

Author(s)

Guichard, Halpin, et al. (2003)

References

Guichard, F., Halpin, P.M., Allison, G.W., Lubchenco, J. & Menge, B.A. (2003). Mussel disturbance dynamics: signatures of oceanographic forcing from local interactions. The American Naturalist, 161, 889â€“904.

See Also

Other models: `forestgap`, `grazing`, `life`, `livestock`, `predprey`

Examples

```
l <- init_landscape(c("+","0","-"), c(0.6,0.2,0.2), width = 50) # create initial landscape
p <- list(delta = 0.01, d = 09, r = 0.4)   # set parameters
r <- ca(l, musselbed, p, t_max = 100)    # run simulation
```

---

neighbors *Count neighbors.*

Description

Count neighbors.

Usage

```
neighbors(x, state = levels(x$cells)[1])
```

Arguments

x	A landscape object.
state	A character value. The state to count. Defaults to the primary cell state, i.e. the first state given in levels(x$cells).

Details

This function is vectorised by using the mapping objects defined by a previous call of mapping.

Value

Returns a vector of the counts in the neighborhood of each cell in the landscape. Neighborhood is specified by the global mapping objects, by default the 4-cell neighborhood.

See Also

For counting the neighborhood of *one* target file, e.g. when doing sequential updating of randomly drawn cells, the function neighborsij is slightly faster. However, for such updating processes we recommend using C++ via Rcpp (see implementation of the predator prey model, predprey.R, for an example).

---

neighborsij *Count neighbors of one cell.*

Description

Count neighbors of one cell.

Usage

```
neighborsij(x, state = levels(x$cells)[1], i = NULL, j = NULL,
  select = 1:prod(x$dim))
```

---

parms_timeseries *Create sequence of parameters over time.*

Description

Create sequence of parameters over time.

Usage

```
parms_timeseries(parms, t_max)
```

Arguments

parms	A list of parameters. Each list entry can be either a single value or a vector representing the sequence of parameter values over time (in years). This is inflated by the requested time in t_max.
t_max	An integer number. The number of years to evaluate the timeseries.

Examples

```
p <- list(
  r = 1.0,  # max. regeneration rate of plants
  b = seq(0.4,0.1, length = 5),  # environmental quality
  sigma_b = 0.1, # random annual variation of environmental quality
  f = 0.8,  # local facilitation
  alpha = 0, # water runoff
  K = 0.9, # carrying capacity of the system
  c = 0.2, # local competition
  m = 0.05, # intrinsic mortality of plants (inverse of av. lifespan)
  v = 0.8, # attractant-decoy
  p = 0.99, # associational resistance
  L = rep(c(0,10), each = 1), # Livestock density
  q = 0, # hill exponent of functional response
  h = 50, # handling time
  a = 0.2 # attack rate of livestock
)


plist <- parms_timeseries(p, 5)

l <- init_landscape(c("1","0"), cover = c(0.5,0.5), width = 100)

ca(l, livestock, plist, t_max = 5, saveeach = 5)
```

`plot.ca_result` *plot method for 'ca_result' objects*

Description

plot method for 'ca_result' objects

Usage

```
## S3 method for class 'ca_result'
plot(x, plotstates = c(TRUE, rep(FALSE,
  length(x$model$states) - 1)), snapshots = FALSE, cols = x$model$cols,
  lwd = 1, ...)
```

Arguments

`x`	An output object obtained from running function `ca()`.
`plotstates`	A logical vector of the same length as `x$model$states`. TRUE activates plotting of other cell state populations.
`cols`	A vector of valid colors to replace the default color vector.
`lwd`	Line width for timeseries plot.
`...`	Parameters handed to standard plot function, e.g. 'bty', 'xlab'. See par.

Details

This prompts a (series of) summary plot for the simulation run. By default this is a timeseries of the primary cell state (i.e. the first state given in `x$model$states`).

`plot.landscape` *Plotting an objects of class "landscape"*

Description

Plotting an objects of class "landscape"

Usage

```
## S3 method for class 'landscape'
plot(x, cols = "auto", grid = FALSE, axis = FALSE,
  add = FALSE, ani = FALSE, ...)
```

Arguments

`x`	A landscape object.
`cols`	A color vector. If `"auto"`, then a grayscale vector will be used.
`grid`	If TRUE, plot a grid over the cells. Defaults to FALSE.
`axis`	If TRUE, plot x and y axis with coordinates. Defaults to FALSE.
`add`	If TRUE, no primary plot will be called. The change in cells will be plotted over an existing plot. Used for animated plotting in the screen plotting device.
`ani`	If TRUE, an adjustment constant is added when producing a pixel accurate png or gif file. Required when the function is used to plot animated figures.

Value

A landscape object of dimensions `width` x `height` with random distribution of `states`, in the relative ratio given in `cover`.

Examples

```
obj <- init_landscape(c("+","0","-"), c(0.5,0.25,0.25))
plot(obj)
```

predprey *Predator-prey model*

Description

A spatially-explicit predator prey model.

Usage

```
ca(l, predprey, parms = p)
```

Arguments

betaf	A numerical value. The reproduction rate of prey.
betas	A numerical value. reproduction rate of predators.
delta	A numerical value. Probability of starvation for predators that do not find prey.

Format

An object of class ca_model of length 6.

Details

The particular neighbourhood we consider in the simulations consists of the four nearest sites. Prey growth occurs as a contact process: a prey chooses a neighbouring site at random and gives birth onto it only if this site is empty at rate betas. Predators hunt for prey by inspecting their neighbourhood for the presence of prey at rate 1. If prey are present, the predator selects one at random and eats it, moving to this neighbouring site. Only predators that find a prey can reproduce, and do so with a specified probability, betas. The offspring is placed in the original site of the predator. Predators that do not find prey are susceptible to starvation and die with a probability delta. Random movement occurs through mixing: neighbouring sites exchange state at a constant rate nu.

Author(s)

Mercedes Pascual, M. Roy, F. Guichard & G. Flierl

References

Pascual, M., Roy, M., Guichard, F., & Flierl, G. (2002). Cluster size distributions: signatures of self-organization in spatial ecologies. Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences, 357(1421), 657â€“666. doi: 10.1098/rstb.2001.0983

See Also

Other models: forestgap, grazing, life, livestock, musselbed

print.ca_model *print function for model objects*

Description

print function for model objects

Usage

```
## S3 method for class 'ca_model'
print(x)
```

Arguments

x

steady *Test for steady state dynamics of timeseries.*

Description

Test for steady state dynamics of timeseries.

Usage

```
steady(i, result, steadyparms = list(t_eval = 400, accept = 0.001),
  returnvalue = FALSE)
```

Arguments

returnvalue

Value

a binary value. TRUE, if steady state is reached. FALSE, if still in transient dynamics.

summary.ca_result *summary method for 'ca_result'*

Description

summary method for 'ca_result'

Usage

```
## S3 method for class 'ca_result'
summary(x, length.eval = 10)
```

Arguments

x	
length.eval	Number of time steps at the end of the simulation to consider when computing mean and sd of covers

Value

Returns a list out containing the model name, the final time, the mean cover and standard deviation in the last 10 timesteps.

Note

The mean and sd are supposed to depend on the model being in steady state dynamics, i.e. beyond transitory. Lacking a universal criterion for that, we report the final 10 timesteps.

summary.landscape *Summary of landscape object.*

Description

Summary of landscape object.

Usage

```
## S3 method for class 'landscape'
summary(x)
```

Arguments

x	A landscape object

Value

A list containing the object dimensions, the number of cells in the different cell states, the global cover of the different cell states, the average local cover of the different cell states.

Examples

```
# obj <- init_landscape(c("+","0","-"), c(0.5,0.25,0.25))
# summary(obj)
```

`template` *Template for model objects*

Description

One sentence description. Potential cell states.

Usage

```
ca(l, template, parms = p)
```

Format

An object of class `ca_model` of length 6.

Details

Wordy description about the model details, transition probabilities, biological assumptions.

Author(s)

Author of original model

References

Original reference and further references

Index