Package ‘OCNet’

July 23, 2020

Title Optimal Channel Networks

Version 0.3.2

Description Generate and analyze Optimal Channel Networks (OCNs): oriented spanning trees reproducing all scaling features characteristic of real, natural river networks. As such, they can be used in a variety of numerical experiments in the fields of hydrology, ecology and epidemiology. See Carraro et al. (2020) <doi:10.1002/ece3.6479> for a presentation of the package; Rinaldo et al. (2014) <doi:10.1073/pnas.1322700111> for a theoretical overview on the OCN concept; Furrer and Sain (2010) <doi:10.18637/jss.v036.i10> for the construct used.

Imports fields, spam, rgl, methods, igraph, SSN, rgdal, sp

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.6)

Suggests knitr, rmarkdown, bookdown

VignetteBuilder knitr

NeedsCompilation yes

Author Luca Carraro [aut, cre],
Florian Altermatt [ctb],
Emanuel A. Fronhofer [ctb],
Reinhard Furrer [ctb],
Isabelle Gounand [ctb],
Andrea Rinaldo [ctb],
Enrico Bertuzzo [aut]

Maintainer Luca Carraro `<luca.carraro@hotmail.it>`

Repository CRAN

Date/Publication 2020-07-23 13:20:03 UTC

R topics documented:

`OCNet-package` *Create and analyze Optimal Channel Networks.*

Description

A package that allows the generation and analysis of synthetic river network analogues, called Optimal Channel Networks (OCNs).

References

Rinaldo, A., Rigon, R., Banavar, J. R., Maritan, A., & Rodriguez-Iturbe, I. (2014). Evolution and selection of river networks: Statics, dynamics, and complexity. Proceedings of the National Academy of Sciences of the United States of America, 111(7), 2417-2424. doi:10.1073/pnas.1322700111

Carraro et al.

Author(s)

Luca Carraro (<luca.carraro@hotmail.it>), Florian Altermatt, Emanuel A. Fronhofer, Reinhard Furrer, Isabelle Gounand, Andrea Rinaldo, Enrico Bertuzzo

See Also

`vignette("OCNet")`

`aggregate_OCN` *Aggregate an Optimal Channel Network*

Description

Function that, given an OCN, builds the network at the river network (RN), aggregated (AG), subcatchment (SC), and catchment (CM) levels.

Usage

```
aggregate_OCN(OCN, thrA = 0.002 * OCN$dimX * OCN$dimY *
  OCN$cellsize^2, streamOrderType = "Strahler", maxReachLength = Inf)
```

Arguments

`OCN`	List as produced by `landscape_OCN`.
`thrA`	Threshold value on drainage area used to derive the aggregated network. If `thrA = 0`, no aggregation is performed: every FD node is also a node at the RN and AG levels. In this case, the function `aggregate_OCN` can still be used to compute statistics such as `OCN$AG$streamOrder`.
`streamOrderType`	If `"Strahler"`, Strahler stream order is computed; if `"Shreve"`, Shreve stream order is computed.
`maxReachLength`	Maximum reach length allowed (in planar units). If the path length between a channel head and the downstream confluence is higher than `maxReachLength`, the reach starting from the channel head will have a length up to `maxReachLength`, while the next downstream pixel is considered as a new channel head, from which a new reach departs.

Details

Note that each node (and the corresponding edge exiting from it, in the case of non-outlet nodes) at the AG level corresponds to a subcatchment at the SC level that shares the same index: for instance, `SC$toFD[i]` contains all elements of `AG$toFD[i]` (that is, the indices of pixels at FD level that constitute the edge departing from node `i` are also part of subcatchment `i`).

Value

A list that contains all objects contained in `OCN`, in addition to the objects listed below. New sublists RN, AG, SC, containing variables at the corresponding aggregation levels, are created. Refer to section 4.2 of the vignette for a more detailed explanation on values `OCN$XX$toYY`, where `XX` and `YY` are two random aggregation levels.

FD$toRN
: Vector (of length OCNFDnNodes) whose values are equal to 0 if the FD node is not a node at the RN level. If FD$toRN[i] != 0, then FD$toRN[i] is the index at the RN level of the node whose index at the FD level is i. Thereby, FD$toRN[i] = j implies RN$toFD[j] = i.

FD$toSC
: Vector (of length OCNFDnNodes) of SC indices for all nodes at the FD level. If OCNFDtoSC[i] = j, then i %in% OCNSCtoFD[[j]] = TRUE.

RN$A
: Vector (of length RN$nNodes) containing drainage area values for all RN nodes (in square planar units).

RN$W
: Adjacency matrix (RN$nNodes by RN$nNodes) at the RN level. It is a spam object.

RN$downNode
: Vector (of length RN$nNodes) representing the adjacency matrix at RN level in a vector form: if RN$downNode[i] = j then RN$W[i,j] = 1. If o is the outlet node, then RN$downNode[o] = 0.

RN$drainageDensity
: Drainage density of the river network, calculated as total length of the river network divided by area of the lattice. It is expressed in planar units^(-1).

RN$leng
: Vector (of length RN$nNodes) of lengths of edges departing from nodes at the RN level. Its values are equal to either 0 (if the corresponding node is an outlet), OCN$cellsize (if the corresponding flow direction is horizontal/vertical), or sqrt(2)*OCN$cellsize (diagonal flow).

RN$nNodes
: Number of nodes at the RN level.

RN$nUpstream
: Vector (of length RN$nNodes) providing the number of nodes upstream of each node (the node itself is included).

RN$outlet
: Vector (of length OCNFDnOutlet) indices of nodes at RN level corresponding to outlets.

RN$Slope
: Vector (of length RN$nNodes) of pixel slopes at RN level.

RN$toAG
: Vector (of length RN$nNodes) whose values are equal to 0 if the RN node is not a node at the AG level. If RN$toAG[i] != 0, then RN$toAG[i] is the index at the AG level of the node whose index at the RN level is i. Thereby, RN$toAG[i] = j implies AG$toRN[j] = i.

RN$toAGReach
: Vector (of length RN$nNodes) identifying to which edge (reach) the RN nodes belong. If RN$toAGReach[i] = j, the RN node i belongs to the edge departing from from the AG node j (which implies that it may correspond to the AG node j itself.)

RN$toFD
: Vector (of length RN$nNodes) with indices at FD level of nodes belonging to RN level. RN$toFD[i] = j implies OCN$FD$toRN[j] = i.

RN$toCM
: Vector (of length RN$nNodes) with catchment index values for each RN node. Example: RN$toCM[i] = j if node i drains into the outlet whose location is defined by outletSide[j], outletPos[j].

RN$upstream
: List (of length RN$nNodes) whose object i is a vector (of length RN$nUpstream[i]) containing the indices of nodes upstream of a node i (including i).

RN$X, RN$Y
: Vectors (of length RN$nNodes) of X, Y coordinates of nodes at RN level.

RN$Z
: Vector (of length RN$nNodes) of Z coordinates of nodes at RN level.

`AG$A`	Vector (of length `AG$nNodes`) containing drainage area values for all nodes at AG level. If `i` is a channel head, then `AG$A[RN$toAG[i]] = RN$A[i]`.
`AG$AReach`	Vector (of length `AG$nNodes`) containing drainage area values computed by accounting for the areas drained by edges departing from AG nodes. In other words, `AG$AReach[i]` is equal to the drainage area of the last downstream node belonging to the reach that departs from `i` (namely `AG$AReach[i] = max(RN$A[RN$toAG == i])`).
`AG$W`	Adjacency matrix (`AG$nNodes` by `AG$nNodes`) at the AG level. It is a `spam` object.
`AG$downNode`	Vector (of length `AG$nNodes`) representing the adjacency matrix at AG level in a vector form: if `AG$downNode[i] = j` then `AG$W[i,j] = 1`. If `o` is the outlet node, then `AG$downNode[o] = 0`.
`AG$leng`	Vector (of length `AG$nNodes`) of lengths of edges departing from nodes at AG level. Note that `AG$leng[i] = sum(RN$leng[RN$toAG == i])`. If `o` is an outlet node (i.e. `(o %in% AG$outlet) = TRUE`), then `AG$leng[i] = 0`.
`AG$nNodes`	Number of nodes resulting from the aggregation process.
`AG$nUpstream`	Vector (of length `AG$nNodes`) providing the number of nodes (at the AG level) upstream of each node (the node itself is included).
`AG$outlet`	Vector (of length `OCN$FD$nOutlet`) with indices of outlet nodes, i.e. nodes whose `AG$downNode` value is 0.
`AG$slope`	Vector (of length `AG$nNodes`) of slopes at AG level. It represents the (weighted) average slope of edges departing from nodes. If `i` is an outlet node (i.e. `(i %in% AG$outlet) = TRUE`), then `AG$slope[i] = NaN`.
`AG$streamOrder`	Vector (of length `AG$nNodes`) of stream order values for each node. If `streamOrderType = "Strahler"`, Strahler stream order is computed. If `streamOrderType = "Shreve"`, Shreve stream order is computed.
`AG$upstream`	List (of length `AG$nNodes`) whose object `i` is a vector (of length `AG$nUpstream[i]`) containing the indices of nodes (at the AG level) upstream of a node `i` (including `i`).
`AG$toFD`	Vector of length `AG$nNodes`) with with indices at FD level of nodes belonging to AG level. `AG$toFD[i] = j` implies `OCN$FD$toAG[j] = i`.
`AG$ReachToFD`	List (of length `AG$nNodes`) whose object `i` is a vector of indices of FD nodes constituting the edge departing from node `i`.
`AG$toRN`	Vector of length `AG$nNodes`) with with indices at RN level of nodes belonging to AG level. `AG$toRN[i] = j` implies `OCN$FD$toRN[j] = i`.
`AG$ReachToRN`	List (of length `AG$nNodes`) whose object `i` is a vector of indices of RN nodes constituting the edge departing from node `i`.
`AG$toCM`	Vector (of length `AG$nNodes`) with catchment index values for each AG node. Example: `AG$toCM[i] = j` if node `i` drains into the outlet whose location is defined by `outletSide[j]`, `outletPos[j]`.
`AG$X, AG$Y`	Vectors (of length `AG$nNodes`) of X, Y coordinates (in planar units) of nodes at the AG level. These correspond to the X, Y coordinates of the nodes constituting the upstream tips of the reaches. If `i` and `j` are such that `AG$X[i] == RN$X[j]` and `AG$Y[i] == RN$Y[j]`, then `AG$A[i] = RN$A[j]`.

AG$XReach, AG$YReach
: Vector (of length AG$nNodes) of X, Y coordinates (in planar units) of the downstream tips of the reaches. If i and j are such that AG$XReach[i] == RN$X[j] and AG$YReach[i] == RN$Y[j], then AG$AReach[i] = RN$A[j]. If o is an outlet node, then AG$XReach = NaN, AG$YReach = NaN.

AG$Z
: Vector (of length AG$nNodes) of elevation values (in elevational units) of nodes at the AG level. These correspond to the elevations of the nodes constituting the upstream tips of the reaches.

AG$ZReach
: Vector (of length AG$nNodes) of Z coordinates (in elevational units) of the downstream tips of the reaches. If o is an outlet node, then AG$ZReach = NaN.

SC$ALocal
: Vector (of length SC$nNodes) with values of subcatchment area, that is the number of FD pixels (multiplied by OCNFDcellsize^2) that constitutes a subcatchment. If o is an outlet node, then ALocal[o] = 0.

SC$W
: Adjacency matrix (SC$nNodes by SC$nNodes) at the subcatchment level. Two subcatchments are connected if they share a border. Note that this is not a flow connection. Unlike the adjacency matrices at levels FD, RN, AG, this matrix is symmetric. It is a spam object. If o is an outlet node, then SC$W[o,] and SC$W[,o] only contain zeros (i.e., o is unconnected to the other nodes).

SC$nNodes
: Number of subcatchments into which the lattice is partitioned. If nOutlet = 1, then SC$nNodes = AG$nNodes. If multiple outlets are present, SC$nNodes might be greater than AG$nNodes in the case when some catchments have drainage area lower than thrA. In this case, the indices from AG$nNodes + 1 to SC$nNodes identify subcatchment that do not have a corresponding AG node.

SC$toFD
: List (of length SC$nNodes) whose object i is a vector of indices of FD pixels constituting the subcatchment i.

SC$X, SC$Y
: Vectors (of length SC$nNodes) of X, Y coordinates (in planar units) of subcatchment centroids.

SC$Z
: Vector (of length SC$nNodes) of average subcatchment elevation (in elevational units).

Finally, thrA is added to the list.

Examples

```
# 1) aggregate a 20x20 OCN by imposing thrA = 4
OCN <- aggregate_OCN(landscape_OCN(OCN_20), thrA = 4)


# 2) explore the effects of thrA and maxReachLength on a large OCN
OCN <- landscape_OCN(OCN_250_T) # it takes some seconds
OCN_a <- aggregate_OCN(OCN, thrA = 200) # it takes some seconds
OCN_b <- aggregate_OCN(OCN, thrA = 1000) # it takes some seconds
OCN_c <- aggregate_OCN(OCN, thrA = 1000, maxReachLength = 20) # it takes some seconds

old.par <- par(no.readonly = TRUE)
par(mfrow = c(1,3))
draw_subcatchments_OCN(OCN_a)
points(OCN_a$AG$X, OCN_a$AG$Y, pch = 19, col = "#0044bb")
```

```
title(paste("No. AG nodes = ", as.character(OCN_a$AG$nNodes),
sep=""))
draw_subcatchments_OCN(OCN_b)
points(OCN_b$AG$X, OCN_b$AG$Y, pch = 19, col = "#0044bb")
title(paste("No. AG nodes = ", as.character(OCN_b$AG$nNodes),
sep=""))
draw_subcatchments_OCN(OCN_c)
points(OCN_c$AG$X, OCN_c$AG$Y, pch = 19, col = "#0044bb")
title(paste("No. AG nodes = ", as.character(OCN_c$AG$nNodes),
sep=""))
par(old.par)
```

`create_OCN` *Create an Optimal Channel Network*

Description

Function that performs the OCN search algorithm on a rectangular lattice and creates OCN at the flow direction (FD) level.

Usage

```
create_OCN(dimX, dimY, nOutlet = 1, outletSide = "S",
  outletPos = round(dimX/3), periodicBoundaries = FALSE,
  typeInitialState = NULL, flowDirStart = NULL, expEnergy = 0.5,
  cellsize = 1, xllcorner = 0.5 * cellsize, yllcorner = 0.5 *
  cellsize, nIter = 40 * dimX * dimY, nUpdates = 50,
  initialNoCoolingPhase = 0, coolingRate = 1,
  showIntermediatePlots = FALSE, thrADraw = 0.002 * dimX * dimY *
  cellsize^2, easyDraw = NULL, saveEnergy = FALSE, saveExitFlag = FALSE,
  saveN8 = FALSE, saveN4 = FALSE, displayUpdates = 1)
```

Arguments

`dimX`	Longitudinal dimension of the lattice (in number of pixels).
`dimY`	Latitudinal dimension of the lattice (in number of pixels).
`nOutlet`	Number of outlets. If `nOutlet = "All"`, all border pixels are set as outlets.
`outletSide`	Side of the lattice where the outlet(s) is/are placed. It is a vector of characters, whose allowed values are `"N"` (northern side), `"E"`, `"S"`, `"W"`. Its length must be equal to `nOutlet`.
`outletPos`	Vector of positions of outlets within the sides specified by `outletSide`. If `outletSide[i] = "N"` or `"S"`, then `outletPos[i]` must be a natural number in the interval `1:dimX`; if `outletSide[i] = "W"` or `"E"`, then `outletPos[i]` must be a natural number in the interval `1:dimY`. If `nOutlet > 1` is specified by the user and `outletSide`, `outletPos` are not, a number of outlets equal to `nOutlet` is randomly drawn among the border pixels. Its length must be equal to `nOutlet`.

periodicBoundaries
: If TRUE, periodic boundaries are applied. In this case, the lattice is the planar equivalent of a torus.

typeInitialState
: Configuration of the initial state of the network. Possible values: "I" (representing a valley); "T" (T-shaped drainage pattern); "V" (V-shaped drainage pattern); "H" (hip roof). Default value is set to "I", unless when nOutlet = "All", where default is "H". See Details for explanation on initial network state in the multiple outlet case.

flowDirStart
: Matrix (dimY by dimX) with custom initial flow directions. Possible entries to flowDirStart are natural numbers between 1 and 8. Key is as follows (capital letters indicate cardinal directions)

  1 E (+1 column)
  2 SE (-1 row, +1 column)
  3 S (-1 row)
  4 SW (-1 row, -1 column)
  5 W (-1 column)
  6 NW (+1 row, -1 column)
  7 N (+1 row)
  8 NE (+1 row, +1 column)

expEnergy
: Exponent of the functional sum(A^expEnergy) that is minimized during the OCN search algorithm.

cellsize
: Size of a pixel in planar units.

xllcorner
: Longitudinal coordinate of the lower-left pixel.

yllcorner
: Latitudinal coordinate of the lower-left pixel.

nIter
: Number of iterations for the OCN search algorithm.

nUpdates
: Number of updates given during the OCN search process (only effective if any(displayUpdates,showIntermediatePlots)=TRUE.).

initialNoCoolingPhase, coolingRate
: Parameters of the function used to describe the temperature of the simulated annealing algorithm. See details.

showIntermediatePlots
: If TRUE, the OCN plot is updated nUpdates times during the OCN search process. Note that, for large lattices, showIntermediatePlots = TRUE might slow down the search process considerably (especially when easyDraw = FALSE).

thrADraw
: Threshold drainage area value used to display the network (only effective when showIntermediatePlots = TRUE).

easyDraw
: Logical. If TRUE, the whole network is displayed (when showIntermediatePlots = TRUE), and pixels with drainage area lower than thrADraw are displayed in light gray. If FALSE, only pixels with drainage area greater or equal to thrADraw are displayed. Default is FALSE if dimX*dimY <= 40000, and TRUE otherwise. Note that setting easyDraw = FALSE for large networks might slow down the process considerably.

saveEnergy
: If TRUE, energy is saved (see Value for its definition).

saveExitFlag If TRUE, exitFlag is saved (see Value for its definition).

saveN8 If TRUE, the adjacency matrix relative to 8-nearest-neighbours connectivity is saved.

saveN4 If TRUE, the adjacency matrix relative to 4-nearest-neighbours connectivity is saved.

displayUpdates State if updates are printed on the console while the OCN search algorithm runs.

0 No update is given.

1 An estimate of duration is given (only if dimX*dimY > 1000, otherwise no update is given).

2 Progress updates are given. The number of these is controlled by nUpdates

Details

Simulated annealing temperature. The function that expresses the temperature of the simulated annealing process is as follows:

if i <= initialNoCoolingPhase*nIter**:** Temperature[i] = Energy[1]

if initialNoCoolingPhase*nIter < i <= nIter**:** Temperature[i] = Energy[1]*(-coolingRate*(i -InitialNocoolingPhase*nIter)/nNodes)

where i is the index of the current iteration and Energy[1] = sum(A^expEnergy), with A denoting the vector of drainage areas corresponding to the initial state of the network. According to the simulated annealing principle, a new network configuration obtained at iteration i is accepted with probability equal to exp((Energy[i] -Energy[i-1])/Temperature[i]) if Energy[i] < Energy[i-1]. To ensure convergence, it is recommended to use coolingRate values between 0.5 and 10 and initialNoCoolingPhase <= 0.3. Low coolingRate and high initialNoCoolingPhase values cause the network configuration to depart more significantly from the initial state. If coolingRate < 0.5 and initialNoCoolingPhase > 0.1 are used, it is suggested to increase nIter with respect to the default value in order to guarantee convergence.

Initial network state. If nOutlet > 1, the initial state is applied with regards to the outlet located at outletSide[1], outletPos[1]. Subsequently, for each of the other outlets, the drainage pattern is altered within a region of maximum size 0.5*dimX by 0.25*dimY for outlets located at the eastern and western borders of the lattice, and 0.25*dimX by 0.5*dimY for outlets located at the southern and northern borders of the lattice. The midpoint of the long size of the regions coincides with the outlet at stake. Within these regions, an "I"-type drainage pattern is produced if typeInitialState = "I" or "T"; a "V"-type drainage pattern is produced if typeInitialState = "V"; no action is performed if typeInitialState = "H". Note that typeInitialState = "H" is the recommended choice only for large nOutlet.

Suggestions for creating "fancy" OCNs. In order to generate networks spanning a realistic, non-rectangular catchment domain (in the "real-shape" view provided by draw_contour_OCN), it is convenient to use the option periodicBoundaries = TRUE and impose at least a couple of diagonally adjacent outlets on two opposite sides, for example nOutlet = 2, outletSide = c("S","N"), outletPos = c(1,2). See also OCN_300_4out_PB_hot. Note that, because the OCN search algorithm is a stochastic process, the successful generation of a "fancy" OCN is not guaranteed: indeed, it is possible that the final outcome is a network where most (if not all) pixels drain towards one of the two outlets, and hence such outlet is surrounded (in the "real-shape" view) by the pixels that it drains. Note that, in order to hinder such occurrence, the two pixels along the lattice perimeter next to each outlet are bound to drain towards such outlet.

In order to create a network spanning a "pear-shaped" catchment (namely where the width of the area spanned in the direction orthogonal to the main stem diminishes downstream, until it coincides with the river width at the outlet), it is convenient to use the option `nOutlet = "All"` (here the value of `periodicBoundaries` is irrelevant) and then pick a single catchment (presumably one with rather large catchment area, see value `OCN$CM$A` generated by `landscape_OCN`) among the many generated. Note that it is not possible to predict the area spanned by such catchment *a priori*. To obtain a catchment whose size is rather large compared to the size of the lattice where the OCN was generated, it is convenient to set `typeInitialState = "I"` and then pick the catchment with largest area (`landscape_OCN` must be run).

The default temperature schedule for the simulated annealing process is generally adequate for generating an OCN that does not resemble the initial network state if the size of the lattice is not too large (say, until `dimX*dimY <= 40000`). When `dimX*dimY > 40000`, it might be convenient to make use of a "warmer" temperature schedule (for example, by setting `coolingRate = 0.5` and `initialNoCoolingPhase = 0.1`; see also the package vignette) and/or increase `nIter` with respect to its default value. Note that these suggestions only pertain to the aesthetics of the final OCN; the default temperature schedule and `nIter` are calibrated to ensure convergence of the OCN (i.e. achievement of a local minimum of `Energy`, save for a reasonable threshold) also for lattices larger than `dimX*dimY = 40000`.

Value

A list whose objects are listed below. Variables that define the network at the FD level are wrapped in the sublist `FD`. Adjacency matrices describing 4- or 8- nearest-neighbours connectivity among pixels are contained in lists N4 and N8, respectively.

`FD$A` Vector (of length `dimX*dimY`) containing drainage area values for all FD pixels (in square planar units).

`FD$W` Adjacency matrix (`dimX*dimY` by `dimX*dimY`) at the FD level. It is a `spam` object.

`FD$downNode` Vector (of length `dimX*dimY`) representing the adjacency matrix at FD level in a vector form: if `FD$downNode[i] = j` then `FD$W[i,j] = 1`. If `o` is the outlet pixel, then `FD$downNode[o] = 0`.

`FD$X (FD$Y)` Vector (of length `dimX*dimY`) containing X (Y) coordinate values for all FD pixels.

`FD$nNodes` Number of nodes at FD level (equal to `dimX*dimY`).

`FD$outlet` Vector (of length `nOutlet`) indices of pixels at FD level corresponding to outlets.

`energy` Vector (of length `nIter`) of energy values for each stage of the OCN during the search algorithm (only present if `saveEnergy = TRUE`).

`exitFlag` Vector (of length `nIter`) showing the outcome of the rewiring process (only present if `saveExitFlag = TRUE`). Its entries can assume one of the following values:

- `0` Rewiring is accepted.
- `1` Rewiring is not accepted (because it does not lower `Energy` or according to the acceptance probability of the simulated annealing algorithm).

2 Rewiring is invalid because a loop in the graph was generated, therefore the network is no longer a direct acyclic graph.

3 Rewiring is invalid because of cross-flow. This means that, for example, in a 2x2 cluster of pixel, the southwestern (SW) corner drains into the NE one, and SE drains into NW. Although this circumstance does not imply the presence of a loop in the graph, it has no physical meaning and is thereby forbidden.

N4$W Adjacency matrix (dimX*dimY by dimX*dimY) that describes 4-nearest-neighbours connectivity between pixels: N4$W[i,j] = 1 if pixel j shares an edge with i, and is null otherwise. It is saved only if saveN4 = TRUE.

N8$W Adjacency matrix (dimX*dimY by dimX*dimY) that describes 8-nearest-neighbours connectivity between pixels: N8$W[i,j] = 1 if pixel j shares an edge or a vertex with i, and is null otherwise. It is saved only if saveN8 = TRUE.

Finally, dimX, dimY, cellsize, nOutlet, periodicBoundaries, expEnergy, coolingRate, typeInitialState, nIter are passed to the list as they were included in the input (except nOutlet = "All" which is converted to 2*(dimX + dimY -2)).

Examples

```
# 1) creates and displays a single outlet 20x20 OCN with default options
set.seed(1)
OCN_a <- create_OCN(20, 20)
draw_simple_OCN(OCN_a)


# 2) creates and displays a 2-outlet OCNs with manually set outlet location,
# and a 4-outlet OCNs with random outlet position.
set.seed(1)
old.par <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
OCN_b1 <- create_OCN(30, 30, nOutlet = 2, outletSide = c("N", "W"), outletPos = c(15, 12))
OCN_b2 <- create_OCN(30, 30, nOutlet = 4)
draw_simple_OCN(OCN_b1)
title("2-outlet OCN")
draw_simple_OCN(OCN_b2)
title("4-outlet OCN")
par(old.par)


## Not run:
# 3) generate 3 single-outlet OCNs on the same (100x100) domain starting from different
# initial states, and show 20 intermediate plots and console updates.
set.seed(1)
OCN_V <- create_OCN(100, 100, typeInitialState = "V", showIntermediatePlots = TRUE,
nUpdates = 20, displayUpdates = 2)
OCN_T <- create_OCN(100, 100, typeInitialState = "T", showIntermediatePlots = TRUE,
nUpdates = 20, displayUpdates = 2)
OCN_I <- create_OCN(100, 100, typeInitialState = "I", showIntermediatePlots = TRUE,
nUpdates = 20, displayUpdates = 2)
```

```
## End(Not run)

## Not run:
# 4) generate a 2-outlet OCN and show intermediate plots. Note that different colors are used
# to identify the two networks  (all pixels are colored because thrADraw = 0).
set.seed(1)
OCN <- create_OCN(150, 70, nOutlet = 2, outletPos = c(1, 150), outletSide = c("S", "N"),
typeInitialState = "V", periodicBoundaries = TRUE,
showIntermediatePlots = TRUE, thrADraw = 0)
# The resulting networks have an irregular contour, and their outlets are located on the contour:
draw_contour_OCN(landscape_OCN(OCN))

## End(Not run)
```

`create_peano` *Create Peano network*

Description

Function that creates Peano networks on a square lattice.

Usage

```
create_peano(nIterPeano, outletPos = "NE", xllcorner = 1,
  yllcorner = 1, cellsize = 1)
```

Arguments

`nIterPeano`	Number of iteration of the Peano scheme. The resulting network will span a domain of size `2^(nIterPeano + 1)` by `2^(nIterPeano + 1)`.
`outletPos`	Corner where the outlet is located, expressed as intercardinal direction. Possible values are `"NE"`, `"SE"`, `"SW"`, `"NW"`.
`xllcorner`	X coordinate of the lower-left pixel (expressed in planar units).
`yllcorner`	Y coordinate of the lower-left pixel (expressed in planar units).
`cellsize`	Size of a pixel (expressed in planar units).

Value

A list that contains the same objects as those produced by `create_OCN`. As such, it can be used as input for all other complementary functions of the package.

Examples

```
# 1) create a peano network in a 32x32 square,
# use landscape_OCN, aggregate_OCN functions,
# and display subcatchment map and map of drainage area
peano <- create_peano(4)
peano <- aggregate_OCN(landscape_OCN(peano), thrA = 4)
```

```
old.par <- par(no.readonly = TRUE)
par(mfrow=c(1,3))
draw_simple_OCN(peano)
title("Peano network")
draw_subcatchments_OCN(peano)
title("Subcatchments")
draw_thematic_OCN(peano$RN$A, peano)
title("Drainage area at RN level")
par(old.par)
```

---

draw_contour_OCN *Draw Optimal Channel Network with catchment contours*

Description

Function that plots real-shaped OCN and catchment contours.

Usage

```
draw_contour_OCN(OCN, thrADraw = 0.002 * OCN$dimX * OCN$dimY *
  OCN$cellsize^2, exactDraw = TRUE, drawContours = TRUE, colPalRiver = NULL,
  colPalCont = "#000000", drawOutlets = 0, pch = 15, colPalOut = "#000000")
```

Arguments

OCN	List as produced by landscape_OCN.
thrADraw	Threshold drainage area value used to display the network.
exactDraw	If TRUE, the real shape of OCNs is plotted. If flow crosses a boundary, the pixel that is not contiguous to its outlet is flipped. It is only effective if OCN$PeriodicBoundaries = TRUE
drawContours	If TRUE, plot catchment(s) contours.
colPalRiver	Color palette used to plot the river network(s). Default is a rearranged version of theme "Dark 3" (see hcl.pals). colPalRiver accepts both functions creating color palettes and vectors of colors (of which the first OCN$nOutlet elements are used). If a single color value is provided and OCN$nOutlet > 1, all river networks are drawn with the same color.
colPalCont	Color palette used to plot the catchment contour(s). Details as in colPalRiver. Additionally, if colPalCont = 0, the palette specified in colPalRiver is copied.
drawOutlets	If equal to 1, black squares are drawn at the outlets' locations behind the river; if 2 they are plotted on top of the river.
pch	Shape of the outlet points (if drawOutlets = TRUE). See points for legend.
colPalOut	Color palette used to plot the outlet points (if drawOutlets = TRUE). Details as in colPalRiver. Additionally, if colPalOut = 0, the palette specified in colPalRiver is copied.

Details

For not too large networks (i.e. if OCNFDnNodes <= 40000, corresponding to a 200x200 lattice), pixels whose drainage area OCNFDA is lower than thrADraw are drawn with a light grey stroke. If OCNFDnNodes > 40000, in order to speed up the execution of this function, only the network constituted by pixels such that OCNFDA > thrADraw is drawn.

Value

No output is returned.

Examples

```
# 1) draw contour of a 20x20 single-outlet OCN
# (for single-outlet OCNs without periodic boundaries, the output
# of draw_contour_OCN is very similar to that of draw_simple_OCN)
draw_contour_OCN(landscape_OCN(OCN_20), thrADraw = 4)

## Not run:
# 2a) plot real shape of multiple-outlet OCN created with periodic boundaries
# add outlets on top of the rivers
OCN <- landscape_OCN(OCN_300_4out_PB_hot, displayUpdates = 2) # it takes around one minute
draw_contour_OCN(OCN, drawOutlets = 2)

# 2b) same as before, but use same color palette for rivers and contours
draw_contour_OCN(OCN, colPalCont = 0)

# 2c) draw contours of catchments obtained from an OCN with nOutlet = "All"
OCN <- landscape_OCN(OCN_400_Allout, displayUpdates = 2) # it takes some minutes
draw_contour_OCN(OCN)

# 2d) same as above, but do not plot contours, and plot outlets
# with same color palette as rivers
draw_contour_OCN(OCN, drawContours = FALSE, drawOutlets = TRUE,
colPalOut = 0)


## End(Not run)
```

---

draw_elev2D_OCN *Plot 2D map of elevation generated by an OCN*

Description

Function that plots the 2D elevation map generated by an OCN.

Usage

```
draw_elev2D_OCN(OCN, colPalette = terrain.colors(1000, alpha = 1),
addLegend=TRUE)
```

Arguments

`OCN`	List as produced by `landscape_OCN`.
`colPalette`	Color palette used for the plot.
`addLegend`	Logical. If `TRUE`, `image.plot` is used to display the legend; as a result, elements (e.g. node coordinates) subsequently plotted of on top of the 2D elevation map might be wrongly positioned.

Value

No output is returned.

Examples

```
# 1) draw 2D map of a 20x20 OCN with default settings
draw_elev2D_OCN(landscape_OCN(OCN_20))
```

`draw_elev3Drgl_OCN` *Plot 3D map of elevation generated by an OCN via rgl rendering*

Description

Function that plots the 3D elevation map generated by an OCN.

Usage

```
draw_elev3Drgl_OCN(OCN, coarseGrain = c(1, 1), chooseCM = FALSE,
  addColorbar = FALSE, drawRiver = FALSE, thrADraw = 0.002 *
  OCN$dimX * OCN$dimY * OCN$cellsize^2, riverColor = "#00CCFF", ...)
```

Arguments

`OCN`	List as produced by `landscape_OCN`.
`coarseGrain`	2x1 vector (only effective if `chooseCM = FALSE`). For aesthetic purposes, the elevation map can be coarse-grained into a `OCN$dimX/coarseGrain[1]`-by-`OCN$dimX/coarseGrain[2]` domain, where each cell's elevation is the average of elevations of the corresponding `coarseGrain[1]`-by-`coarseGrain[2]` cells of the original elevation field. `coarseGrain[1]` and `coarseGrain[2]` must be divisors of `OCN$dimX` and `OCN$dimY`, respectively. `coarseGrain = c(2,2)` is often sufficient to achieve a good graphical results for large (i.e. at least 100x100 nodes) OCNs.
`chooseCM`	Index of catchment to display (only effective if `OCN$nOutlet > 1`). It can be a logical, or a scalar within `1:length(OCN$nOutlet)`. If `TRUE`, the catchment with largest area is displayed. Note that, if the size of the chosen catchment is too small (e.g. `OCN$CM$A[chooseCM] < 5*OCN$cellsize^2`), an error might occur due to failure in triangulation.
`addColorbar`	If `TRUE`, add colorbar to the plot.

drawRiver	If TRUE, draw the OCN on top of the elevation field.
thrADraw	Threshold drainage area value used to display the network.
riverColor	Color used to plot the river.
...	Further parameters passed to function persp3d. The default value for aspect is c(OCN$dimX/sqrt(OCN$dimX*OCN$dimY,OCN$dimY/sqrt(OCN$dimX*OCN$dimY,1)).

Details

This function makes use of the rgl rendering system. To export the figure in raster format, use rgl.snapshot. To export in vectorial format, use rgl.postscript (but note that this might produce rendering issues, see rgl for details). The function will attempt at drawing a contour of the plotted entity (i.e. the lattice or a catchment, depending on chooseCM) at null elevation, and drawing polygons connecting this contour with the lattice/catchment contour at the real elevation. If chooseCM != FALSE, this might result in errors owing to failure of polygon3d in triangulating the polygons.

Value

No output is returned.

Examples

```
## Not run:
draw_elev3Drgl_OCN(landscape_OCN(OCN_20))

## End(Not run)

## Not run:
# 1a) draw the 3D representation of a single catchment within an OCN
# generated with nOutlet = "All" and add draw the river on top of it
OCN <- landscape_OCN(OCN_400_Allout, displayUpdates = 2) # this takes some minutes
draw_elev3Drgl_OCN(OCN, chooseCM = 983, drawRiver = TRUE)

# 1b) draw the 3D representation of the largest catchment within the OCN
# (here polygon3d may fail at plotting the polygon at zero elevation)
draw_elev3Drgl_OCN(OCN, chooseCM = TRUE)

# 1c) draw the 3D representation of the whole OCN
# and enhance the aspect ratio of Z coordinates
# with respect to the default value (the final result will be ugly):
draw_elev3Drgl_OCN(OCN, aspect = c(1, 1, 0.2))

# 1d) same as above, but operate coarse graining for better aesthetics:
draw_elev3Drgl_OCN(OCN, coarseGrain = c(5,5), aspect = c(1, 1, 0.2))

# 2) draw the 3D representation of a single catchment of an OCN generated
# with periodicBoundaries = TRUE
# (note that the real shape of the catchment is drawn)
OCN <- landscape_OCN(OCN_300_4out_PB, displayUpdates = 2) # this takes some minutes
draw_elev3Drgl_OCN(OCN, chooseCM = TRUE)
```

```
## End(Not run)
```

---

draw_elev3D_OCN *Plot 3D map of elevation generated by an OCN*

Description

Function that plots the 3D elevation map generated by an OCN.

Usage

```
draw_elev3D_OCN(OCN, coarseGrain = c(1,1), colPalette = terrain.colors(1000, alpha = 1),
  addColorbar = TRUE, drawRiver = TRUE, thrADraw = 0.002 *
  OCN$dimX * OCN$dimY * OCN$cellsize^2, riverColor = "#00CCFF",
  theta = -20, phi = 30, expand = 0.05, shade = 0.5)
```

Arguments

`OCN`	List as produced by `landscape_OCN`.
`coarseGrain`	2x1 vector (only effective if `chooseCM = FALSE`). For aesthetic purposes, the elevation map can be coarse-grained into a `OCN$dimX/coarseGrain[1]`-by-`OCN$dimX/coarseGrain[2]` domain, where each cell's elevation is the average of elevations of the corresponding `coarseGrain[1]`-by-`coarseGrain[2]` cells of the original elevation field. `coarseGrain[1]` and `coarseGrain[2]` must be divisors of `OCN$dimX` and `OCN$dimY`, respectively. `coarseGrain = c(2,2)` is often sufficient to achieve a good graphical results for large (i.e. at least 100x100 nodes) OCNs.
`colPalette`	Color palette used for the plot.
`addColorbar`	If `TRUE`, add colorbar to the plot.
`drawRiver`	If `TRUE`, draw the OCN on top of the elevation field.
`thrADraw`	Threshold drainage area value used to display the network.
`riverColor`	Color used to plot the river.
`theta, phi, expand, shade`	Additional parameters passed to the perspective plotting function `persp`. `theta` expresses azimuthal direction; `phi` gives colatitude; `expand` is the expansion factor for the Z coordinates; `shade` controls the shade at a surface facet.

Value

No output is returned.

Examples

```
# draw 3D representation of a 20x20 OCN with default options
draw_elev3D_OCN(landscape_OCN(OCN_20))

## Not run:
# 1a) draw the 3D representation of the OCN (without displaying the river
# and the colorbar) and enhance the aspect ratio of Z coordinates
# with respect to the default value (the final result will be ugly):
OCN <- landscape_OCN(OCN_400_Allout, displayUpdates = 2) # this takes some minutes
draw_elev3D_OCN(OCN, expand = 0.2, addColorbar = FALSE, drawRiver = FALSE)

# 1b) same as above, but operate coarse graining  and modify shade for better aesthetics:
draw_elev3D_OCN(OCN, coarseGrain = c(5,5), expand = 0.2,
shade = 0.25, addColorbar = FALSE, drawRiver = FALSE)

## End(Not run)
```

`draw_simple_OCN` *Draw an Optimal Channel Network*

Description

Function that plots the non-aggregated OCN as calculated by `create_OCN`.

Usage

```
draw_simple_OCN(OCN, thrADraw = 0.002 * OCN$dimX * OCN$dimY *
  OCN$cellsize^2, riverColor = "#0066FF", easyDraw = NULL)
```

Arguments

`OCN`	List as produced by `create_OCN`.
`thrADraw`	Threshold drainage area value used to display the network.
`riverColor`	Color used to plot the river.
`easyDraw`	Logical. If `TRUE`, the whole network is displayed, and pixels with drainage area lower than `thrADraw` are displayed in light gray. If `FALSE`, only pixels with drainage area greater or equal to `thrADraw` are displayed. Default is `FALSE` if `OCN$nNodes <= 40000`, and `TRUE` otherwise. Note that setting `easyDraw = FALSE` for large networks might slow down the process considerably.

Value

No output is returned.

Examples

```
# 1a) draw OCN with default settings
draw_simple_OCN(OCN_250_T)
# 1b) same as above, but with decreased thrADraw
draw_simple_OCN(OCN_250_T, thrADraw = 0.001 * OCN_250_T$dimX * OCN_250_T$dimY)

# 1c) same as the first example, but include the portion of network
# with drainage area lower than thrADraw
draw_simple_OCN(OCN_250_T, easyDraw = FALSE) # this will take some seconds
```

`draw_subcatchments_OCN` *Draw subcatchment map from an Optimal Channel Network*

Description

Function that draws a map of subcatchments generated by the aggregation process on the OCN.

Usage

```
draw_subcatchments_OCN(OCN, drawRiver = TRUE, colPalette = NULL)
```

Arguments

`OCN`	List as produced by `aggregate_OCN`.
`drawRiver`	if `TRUE`, draw the OCN on top of the subcatchment map.
`colPalette`	Color palette used. Default is `c("#009900","#FFFF00","#FF9900","#FF0000","#FF00FF","#9900CC` Only the first `n` colors are used, where `n` is the number of different colors needed (calculated via a greedy coloring algorithm). `colPalette` accepts both functions creating color palettes and vectors of colors (see examples); in the latter case, the length of the vector cannot be lower than `n` (`n` cannot be predicted a priori, but generally 6 colors should suffice).

Value

No output is returned.

Examples

```
# 1a) aggregate a 20x20 OCN , use thrA = 5 pixels
# and draw subcatchments with default color palette
OCN <- aggregate_OCN(landscape_OCN(OCN_20), thrA = 5)
draw_subcatchments_OCN(OCN, drawRiver = TRUE)

# 1b) same as above, but define color palette with a function
draw_subcatchments_OCN(OCN, drawRiver = TRUE, colPalette = rainbow)
```

```
# 1c) same as above, but define color palette with a vector of colors
draw_subcatchments_OCN(OCN, drawRiver = TRUE, colPalette = hcl.colors(6, "Dark 3"))
```

draw_thematic_OCN *Draw thematic map on an Optimal Channel Network*

Description

Function that draws OCNs with color of RN or AG nodes depending on an arbitrary theme.

Usage

```
draw_thematic_OCN(theme, OCN,
  chooseAggregation = NULL,
  discreteLevels = FALSE,
  colLevels = NULL, cutoff = FALSE,
  colPalette = colorRampPalette(c("yellow","red","black")),
  exactDraw = FALSE, chooseCM = FALSE, drawNodes = FALSE,
  nodeType = "upstream", cex = 2, pch = 21, nanColor = "#0099FF",
  riverColor = "#0099FF", backgroundColor = "#999999",
  addLegend = TRUE)
```

Arguments

`theme`	Vector (of length `OCN$AG$Nnodes` or `OCN$RN$Nnodes`) expressing the spatial field of interest. The vector can contain `NA` and `NaN` values to identify RN or AG nodes where the theme is not defined.
`OCN`	List as produced by `aggregate_OCN`.
`chooseAggregation`	Only effective if `OCN$RN$nNodes == OCN$AG$nNodes`. In such case, it must be equal to either `"RN"` or `"AG"`; as a result, `theme` will be interpreted as a spatial field in the corresponding aggregation level.
`discreteLevels`	Logical. If `FALSE`, a continuous color scheme is used. If `TRUE`, discrete color levels are applied. See also `colLevels` and examples.
`colLevels`	Number of colors in the palette. If `discreteLevels == FALSE`, `colLevels` must be a vector of the form `c(minval,maxval,N_levels)`. The vector of breakpoints used to attribute `theme` values to a given color is then defined as `seq(minval,maxval,N_levels)`. Default is `minval = min(theme[!(is.nan(theme))])`, `maxval = max(theme[!(is.nan(theme))])`, `N_levels = 1000`. If `discreteLevels == TRUE` and `is.null(colLevels) == TRUE`, each unique value of `theme` is attributed a different color. If `discreteLevels == TRUE` and `colLevels` is a vector, `colLevels` is used as vector of breakpoints. In this case, the number of discrete colors is equal to `length(colLevels) -1`.

cutoff	Logical. If FALSE, nodes whose theme value is beyond the range established by the vector of breakpoints are attributed the color corresponding to the lowest (or highest) value in the color scheme. If TRUE, such nodes are attributed the color NaNcolor.
colPalette	Color palette used to display theme values. colPalette accepts both functions creating color palettes and vectors of colors. In the latter case, length(colPalette) must be greater than the number of color levels. See examples below and hcl.colors.
chooseCM	Index of catchment to display (only effective if OCN$nOutlet > 1). It can be a logical or a numeric vector. If FALSE, all catchments are displayed. If TRUE, the catchment with largest area is displayed. If chooseCM is a subset of vector 1:length(OCN$nOutlet), only the catchment(s) identified by the indices in chooseCM are displayed.
exactDraw	Logical. If TRUE, the real shape of OCNs is plotted. If flow crosses a boundary, the pixel that is not contiguous to its outlet is flipped.
drawNodes	Logical. If FALSE, the theme is directly displayed on the river network. In this case, the edge departing from a given node is displayed with the color attributed to the node. If TRUE, the theme is displayed via markers at the locations of the nodes at the RN or AG level (depending on the length of theme). In this case, nanColor can be used to define the color of the river network.
nodeType	Only effective if drawNodes == TRUE and length(theme) == OCNRNnNodes. Can assume values "upstream" or "downstream". If "upstream", nodes are drawn at the upstream ends of the corresponding edges (i.e. at the coordinates defined by OCNAGX, OCNAGY). If "downstream", nodes are drawn at the downstream ends of the corresponding edges (i.e. at the coordinates defined by OCNAGXReach, OCNAGYReach).
cex	Only effective if drawNodes == TRUE. It sets the dimension of the markers (equivalent to parameter cex of function points). It can be a scalar or a vector of length length(theme).
pch	Only effective if drawNodes == TRUE. It sets the type of the markers (equivalent to parameter pch of function points). It can be a scalar or a vector of length length(theme).
nanColor	Color attributed to RN or AG nodes whose theme value is NA or NaN.
riverColor	Only effective if drawNodes == TRUE. Color used to display the OCN below the nodes.
backgroundColor	Color used in the background of the figure. It can be either a single value, or a vector with number of components equal to length(chooseCM). If length(backgroundColor) == length(chooseCM), each color is used to identify a different catchment selected in chooseCM (corresponding to the respective outlet). If instead length(chooseCM) > 1 and length(backgroundColor) == 1, all catchments are colored with the same backgroundColor.
addLegend	Logical. If TRUE, add legend to the plot. If also discreteLevels = FALSE, image.plot is used to display the legend, which appears as a colorbar; as a result, elements (e.g. node coordinates) subsequently plotted of on top of the 2D elevation map might be wrongly positioned.

Details

This function can be used to show how a certain spatial field varies along the river network.

Value

No output is returned.

Examples

```
# 1a) Six different ways to display contributing area at the AG level
OCN <- aggregate_OCN(landscape_OCN(OCN_20), thrA = 4)
old.par <- par(no.readonly = TRUE)
par(mfrow=c(2,3), oma = c(0, 0, 3, 0))
draw_thematic_OCN(OCN$AG$A, OCN, colPalette = hcl.colors)
title("Continuous levels \n Colors on edges")
draw_thematic_OCN(OCN$AG$A, OCN, discreteLevels = TRUE,
colPalette = hcl.colors)
title("Discrete, unique levels \n Colors on edges")
draw_thematic_OCN(OCN$AG$A, OCN, discreteLevels = TRUE,
colLevels = c(1, 10, 50, 100, 500),
colPalette = hcl.colors)
title("Discrete, user-defined levels \n Colors on edges")
draw_thematic_OCN(OCN$AG$A, OCN, drawNodes = TRUE,
colPalette = hcl.colors)
title("Continuous levels \n Colors on edges")
draw_thematic_OCN(OCN$AG$A, OCN,  discreteLevels = TRUE,
drawNodes = TRUE, colPalette = hcl.colors)
title("Discrete, unique levels \n Colors on nodes")
draw_thematic_OCN(OCN$AG$A, OCN, discreteLevels = TRUE,
drawNodes = TRUE, colLevels = c(1, 10, 50, 100, 500),
colPalette = hcl.colors)
title("Discrete, user-defined levels \n Colors on nodes")
mtext("Six different ways to display contributing area [no. pixels]", outer = TRUE, cex = 1.5)
par(old.par)

# 1b) Same as above, but use different colLevels, cutoff combinations
# with DiscreteLevels = FALSE
old.par <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
draw_thematic_OCN(OCN$AG$A, OCN, drawNodes = TRUE,
 colLevels = c(0, 200, 1000), colPalette = hcl.colors)
title("All nodes with A > 200 pixels \n are displayed in yellow")
draw_thematic_OCN(OCN$AG$A, OCN, drawNodes = TRUE,
nanColor = "#00000000", colLevels = c(0, 200, 1000),
cutoff = TRUE, colPalette = hcl.colors)
title("All nodes with A > 200 pixels \n are treated as NaN")
par(old.par)

## Not run:
# 2) Display distance to outlet (at the RN level) along the main stem
# of an OCN
OCN <- aggregate_OCN(landscape_OCN(OCN_250_T)) # this takes some seconds
```

```
OCN <- paths_OCN(OCN, pathsRN = TRUE) # this takes some seconds

distanceToOutlet <- OCN$RN$downstreamPathLength[,OCN$RN$outlet]
farthestNode <- which(distanceToOutlet == max(distanceToOutlet))
mainStem <- OCN$RN$downstreamPath[[farthestNode]][[OCN$RN$outlet]]
theme <- rep(NaN, OCN$RN$nNodes)
theme[mainStem] <- distanceToOutlet[mainStem]

draw_thematic_OCN(theme, OCN)
title("Distance to outlet along the main stem [pixel units]")

## End(Not run)
```

`find_area_threshold_OCN` *Find relationship between number of nodes and threshold area in an OCN*

Description

Function that calculates relationship between threshold area and number of nodes at RN and AG level for a given OCN. It can be used prior to application of `aggregate_OCN` in order to derive the drainage area threshold that corresponds to the desired number of nodes of the aggregated network. It is intended for use with single outlet OCNs, although its use with multiple outlet OCNs is allowed (provided that `max(thrValues) <= min(OCN$CM$A)`).

Usage

```
find_area_threshold_OCN(OCN, thrValues = seq(OCN$cellsize^2,
  min(OCN$CM$A), OCN$cellsize^2), maxReachLength = Inf,
  streamOrderType = "Strahler", displayUpdates = 0)
```

Arguments

`OCN`	List as produced by `landscape_OCN`
`thrValues`	Vector of values of threshold drainage area (in squared planar units) for which the respective number of nodes at the RN and AG levels are computed. Note that it must be `max(thrValues) <= min(OCN$CM$A)`, otherwise the catchment(s) with area lower than `max(thrValues)` degenerate to a network with zero nodes at the RN/AG level.
`maxReachLength`	Maximum reach length allowed (in planar units). If the path length between a channel head and the downstream confluence is higher than `maxReachLength`, the reach starting from the channel head will have a length up to `maxReachLength`, while the next downstream pixel is considered as a new channel head, from which a new reach departs.
`streamOrderType`	If `"Strahler"`, Strahler stream order is computed; if `"Shreve"`, Shreve stream order is computed.

displayUpdates If 1, progress updates are printed in the console while the function is running. If 0, no updates are printed.

Value

A list whose objects are listed below.

thrValues Copy of the input vector with the same name.

nNodesRN Vector (of the same length as thrValues) of number of nodes at the RN level resulting from the aggregation process with a threshold area values specified by thrValues.

nNodesAG Vector (of the same length as thrValues) of number of nodes at the AG level resulting from the aggregation process with a threshold area values specified by thrValues.

drainageDensity
Vector (of the same length as thrValues) of values of drainage density of the river network resulting from the aggregation process with a threshold area values specified by thrValues. Drainage density is calculated as total length of the river network divided by area of the lattice. It is expressed in planar units^(-1).

streamOrder Vector (of the same length as thrValues) of values of maximum stream order attained by the river network, resulting from the aggregation process with a threshold area values specified by thrValues.

Examples

```
# 1) derive relationship between threshold area and number of nodes
OCN <- landscape_OCN(OCN_20)
thr <- find_area_threshold_OCN(OCN)
# log-log plot of number of nodes at the AG level versus
# relative threshold area (as fraction of total drainage area)
old.par <- par(no.readonly = TRUE)
par(mai = c(1,1,1,1))
plot(thr$thrValues[thr$nNodesAG > 0]/OCN$CM$A,
thr$nNodesAG[thr$nNodesAG > 0], log = "xy",
xlab = "Relative area threshold", ylab = "Number of AG nodes")
par(old.par)
```

---

landscape_OCN *Generate 3D landscape from an Optimal Channel Network*

Description

Function that calculates the elevation field generated by the OCN and the partition of the domain into different catchments.

Usage

```
landscape_OCN(OCN, slope0 = 1, zMin = 0, optimizeDZ = FALSE,
  optimMethod = "BFGS", optimControl = list(maxit = 100 *
  length(OCN$FD$outlet), trace = 1), displayUpdates = 0)
```

Arguments

OCN	List as produced by create_OCN.
slope0	slope of the outlet pixel (in elevation units/planar units).
zMin	Elevation of the lowest pixel (in elevation units).
optimizeDZ	If TRUE, when there are multiple catchments, minimize differences in elevation at the catchment borders by lifting catchments, while respecting zMin. If FALSE, all outlet pixels have elevation equal to zMin.
optimMethod	Optimization method used by function optim (only used if optimizeDZ = TRUE).
optimControl	List of control parameters used by function optim (only used if optimizeDZ = TRUE).
displayUpdates	State if updates are printed on the console while landscape_OCN runs. 0 No update is given. 1 Concise updates are given. 2 More extensive updates are given (this might slow down the total function runtime). Note that the display of updates during optimization of elevations (when optimizeDZ = TRUE) is controlled by parameter optimControl$trace.

Details

The function features an algorithm (which can be activated via the optional input optimizeDZ) that, given the network configuration and a slope0 value, finds the elevation of OCN$nOutlet -1 outlets relative to the elevation of the first outlet in vectors outletSide, outletPos such that the sum of the absolute differences elevation of neighboring pixels belonging to different catchments is minimized. Such objective function is minimized by means of function optim. The absolute elevation of the outlet pixels (and, consequently, of the whole lattice) is finally attributed by imposing OCNFDZ >= zMin. Note that, due to the high dimensionality of the problem, convergence of the optimization algorithm is not guaranteed for large OCN$nOutlet (say, OCN$nOutlet > 10).

Value

A list that contains all objects contained in OCN, in addition to the objects listed below. A new sublist CM, containing variables at the catchment aggregation levels, is created.

FD$slope	Vector (of length OCNFDnNodes) of slope values (in elevation units/planar units) for each FD pixel, as derived by the slope/area relationship.
FD$leng	Vector (of length OCNFDnNodes) of pixel lengths. OCNFDleng[i] = OCNFDcellsize if flow direction in i is horizontal or vertical; OCNFDleng[i] = OCNFDcellsize*sqrt(2) if flow direction in i is diagonal.

FD$toCM — Vector (of length OCNFDnNodes) with catchment index values for each FD pixel. Example: OCNFDtoCM[i] = j if pixel i drains into the outlet whose location is defined by outletSide[j], outletPos[j].

FD$XDraw — When periodicBoundaries = TRUE, it is a vector (of length OCNFDnNodes) with real X coordinate of FD pixels. If periodicBoundaries = FALSE, it is equal to OCNFDX.

FD$YDraw — When periodicBoundaries = TRUE, it is a vector (of length OCNFDnNodes) with real Y coordinate of FD pixels. If periodicBoundaries = FALSE, it is equal to OCNFDY.

FD$Z — Vector (of length OCNFDnNodes) of elevation values for each FD pixel. Values are calculated by consecutive implementation of the slope/area relationship along upstream paths.

CM$A — Vector (of length OCN$nOutlet) with values of drainage area (in square planar units) for each of the catchments identified by the corresponding OCNFDoutlet.

CM$W — Adjacency matrix (OCN$nOutlet by OCN$nOutlet) at the catchment level. Two catchments are connected if they share a border. Note that this is not a flow connection. Unlike the adjacency matrices at levels FD, RN, AG, this matrix is symmetric. It is a spam object.

CM$XContour (CM$Y_contour)
List with number of objects equal to OCNFDnOutlet. Each object i is a list with X (Y) coordinates of the contour of catchment i for use in plots with exactDraw = FALSE (see functions draw_contour_OCN, draw_thematic_OCN). If catchment i is constituted by regions that are only connected through a diagonal flow direction, CM$XContour[[i]] (CM$Y_contour[[i]]) contains as many objects as the number of regions into which catchment i is split.

CM$XContourDraw (CM$YContourDraw)
List with number of objects equal to OCNFDnOutlet. Each object i is a list with X (Y) coordinates of the contour of catchment i for use in plots with exactDraw = TRUE (see functions draw_contour_OCN, draw_thematic_OCN). If catchment i is constituted by regions that are only connected through a diagonal flow direction, CM$XContourDraw[[i]] (CM$YContourDraw[[i]]) contains as many objects as the number of regions into which catchment i is split.

OptList — List of output parameters produced by the optimization function optim (only present if optimizeDZ = TRUE).

Finally, slope0 and zMin are passed to the list as they were included in the input.

Examples

```
# 1) draw 2D elevation map of a 20x20 OCN with default options
OCN2 <- landscape_OCN(OCN_20)

## Not run:
# 2) generate a 100x50 OCN; assume that the pixel resolution is 200 m
# (the total catchment area is 20 km2)
set.seed(1)
OCN <- create_OCN(100, 50, cellsize = 200,
displayUpdates = 0) # this takes about 40 s
```

```
# use landscape_OCN to derive the 3D landscape subsumed by the OCN
# by assuming that the elevation and slope at the outlet are 200 m
# and 0.0075, respectively
OCN <- landscape_OCN(OCN, zMin = 200, slope0 = 0.0075)
# draw 2D and 3D representations of the landscape
draw_elev2D_OCN(OCN)
draw_elev3D_OCN(OCN)
draw_elev3Drgl_OCN(OCN)

## End(Not run)

## Not run:
# 3) generate a 100x50 OCN with 4 outlets
set.seed(1)
OCN <- create_OCN(100, 50, cellsize = 200,
nOutlet = 4, displayUpdates = 0) # this takes about 40 s
# use landscape_OCN and optimize elevation of outlets
OCN <- landscape_OCN(OCN, slope0 = 0.0075,
optimizeDZ = TRUE)
# display elevation of outlets and 2D elevation map
OCN$FD$Z[OCN$FD$outlet]
draw_elev2D_OCN(OCN)

## End(Not run)
```

OCN_20 *Example of small OCN*

Description

A network built on a 20x20 lattice obtained by executing `set.seed(1); create_OCN(20,20)`.

Usage

```
data(OCN_20)
```

Format

A list. See `create_OCN` documentation for details.

OCN_250_PB *Example of single-outlet OCN with periodic boundaries*

Description

A network built on a 250x250 lattice obtained by executing `set.seed(2); create_OCN(250,250,periodicBoundaries = TRUE)`.

Usage

```
data(OCN_250_PB)
```

Format

A list. See `create_OCN` documentation for details.

OCN_250_T *Example of single-outlet OCN*

Description

A network built on a 250x250 lattice obtained by executing `set.seed(2); create_OCN(250,250,typeInitialState = "T")`.

Usage

```
data(OCN_250_T)
```

Format

A list. See `create_OCN` documentation for details.

OCN_300_4out *Example of multiple-outlet OCN*

Description

A network built on a 300x300 lattice obtained by executing `set.seed(5); create_OCN(300,300,nOutlet = 4,outletSide = c("S","N","W","E"),outletPos = c(1,300,149,150),typeInitialState = "V",cellsize = 50)`.

Usage

```
data(OCN_300_4out)
```

Format

A list. See `create_OCN` documentation for details.

OCN_300_4out_PB_hot — *Example of multiple-outlet OCN with periodic boundaries*

Description

A network built on a 300x300 lattice obtained by executing `set.seed(5); create_OCN(300,300,nOutlet = 4,outletSide = c("S","N","W","E"),outletPos = c(1,300,149,150),typeInitialState = "V",periodicBoundaries = TRUE,cellsize = 50,coolingRate = 0.5,initialNoCoolingPhase = 0.1)`.

Usage

```
data(OCN_300_4out_PB_hot)
```

Format

A list. See `create_OCN` documentation for details.

OCN_4 — *Example of small OCN*

Description

A network built on a 4x4 lattice for illustrative purposes.

Usage

```
data(OCN_4)
```

Format

A list. See `create_OCN` documentation for details.

Details

Despite its name, this network is not an OCN: indeed, it has been generated manually and not via `create_OCN`.

OCN_400_Allout *Example of OCN with all perimetric pixels as outlets*

Description

A network built on a 400x400 lattice obtained by executing `set.seed(8); create_OCN(400,400,nOutlet = "All",cellsize = 50)`.

Usage

```
data(OCN_400_Allout)
```

Format

A list. See `create_OCN` documentation for details.

OCN_to_igraph *Transform OCN into igraph object*

Description

Function that transforms an OCN into an igraph object.

Usage

```
OCN_to_igraph(OCN, level)
```

Arguments

`OCN`	List as produced by `aggregate_OCN`.
`level`	Aggregation level at which the OCN is converted into an igraph object. It must be equal to either `"FD"`, `"RN"` or `"AG"`.

Value

An `igraph` object.

Examples

```
# 1) transform a 20x20 OCN, at the AG level, into a graph object
OCN <- aggregate_OCN(landscape_OCN(OCN_20), thrA = 4)
g <- OCN_to_igraph(OCN, level = "AG")
plot(g, layout = matrix(c(OCN$AG$X,OCN$AG$Y), ncol = 2, nrow = OCN$AG$nNodes))
```

OCN_to_SSN *Transform OCN into SSN object*

Description

Function that transforms an OCN into a SpatialStreamNetwork object. It is analogous to function `createSSN` from package `SSN`.

Usage

```
OCN_to_SSN(OCN, level, obsDesign,
  predDesign = noPoints, path, importToR = FALSE)
```

Arguments

`OCN`	List as produced by `aggregate_OCN`.
`level`	Aggregation level at which the OCN is converted into a SpatialStreamNetwork object. It must be equal to either `"FD"`, `"RN"` or `"AG"`.
`obsDesign`	Same as the argument of the same name in `createSSN`. Note that the length of the argument of the design function must be equal to `OCN$N_outlet`.
`predDesign`	Same as the argument of the same name in `createSSN`. Note that, if a design function is specified, the length of its argument must be equal to `OCN$N_outlet`.
`path`	Same as the argument of the same name in `createSSN`.
`importToR`	Same as the argument of the same name in `createSSN`.

Details

The generated SpatialStreamNetwork object consists of `OCN$N_outlet` networks. Note that an error is thrown if, at the selected aggregation `level`, at least one of these networks is degenerate (i.e. it has less than two nodes). This is typically the case for OCNs generated with option `N_outlet = "All"`.

If `OCN$PeriodicBoundaries == FALSE`, nodes' locations in the SpatialStreamNetwork object are given by the lattice coordinates (i.e. `OCN$level$X`, `OCN$level$Y`); if `OCN$PeriodicBoundaries == TRUE`, real coordinates are used (i.e. those defined by `OCN$FD$X_draw`, `OCN$FD$Y_draw`, see `landscape_OCN`).

Value

A `SpatialStreamNetwork` object if `importToR` is `TRUE`, otherwise `NULL`.

Examples

```
# transform a 20x20 single-outlet OCN (aggregated at the AG level)
# into a SSN object and plot it
OCN <- aggregate_OCN(landscape_OCN(OCN_20), thrA = 4)
ssn1 <- OCN_to_SSN(OCN, "AG", obsDesign = SSN::poissonDesign(10),
path=paste(tempdir(),"/OCN.ssn", sep = ""), importToR = TRUE)
plot(ssn1)


# 1) create a 50x50 OCN with two outlets and periodic boundaries;
set.seed(1)
OCN <- create_OCN(50, 50, nOutlet = 2, outletSide = c("S", "N"),
outletPos = c(1, 50), periodicBoundaries = TRUE)
# aggregate the OCN;
OCN <- aggregate_OCN(landscape_OCN(OCN))
# transform it into a SSN object aggregated at the RN level;
ssn2 <- OCN_to_SSN(OCN, "RN", obsDesign = SSN::binomialDesign(c(10, 10)),
path = paste(tempdir(),"/OCN2.ssn", sep = ""), importToR = TRUE)
# and plot the SSN object; it is constituted by two networks,
# and nodes' coordinates are the "real" ones
old.par <- par(no.readonly = TRUE)
par(mai = c(1, 1, 1, 1))
plot(ssn2, xlab = "X", ylab = "Y")
par(old.par)
```

---

paths_OCN *Calculate paths between nodes in an Optimal Channel Network*

Description

Function that determines upstream and downstream paths and path lengths between any nodes of the network at the aggregated level.

Usage

```
paths_OCN(OCN, pathsRN = FALSE, includeDownstreamNode = FALSE)
```

Arguments

OCN	List as produced by aggregate_OCN.
pathsRN	Logical. If TRUE paths and path lengths at the RN level are copied to the output list. If FALSE, only path lengths at the RN level are copied to the output list.
includeDownstreamNode	Logical. If TRUE, path lengths include the length of the edge departing from the last downstream node of the path.

Value

A list that contains all objects contained in `OCN`, in addition to the objects listed below.

`RN$downstreamPath`
: (only present if `pathsRN = TRUE`) List (of length `OCN$RN$nNodes`) whose object `i` is a list (of length `OCN$RN$nNodes`). If nodes `i` and `j` are connected by a downstream path, then `RN$downstreamPath[[i]][[j]]` is a vector containing the indices of the nodes constituting such path (`i` and `j` are included). If `i` and `j` are not connected by a downstream path, then `RN$downstreamPath[[i]][[j]] = 0`.

`RN$downstreamPathLength`
: Sparse matrix (`OCN$RN$nNodes` by `OCN$RN$nNodes`) containing length of paths between nodes that are connected by a downstream path; if `i` and `j` are not connected by a downstream path, then `RN$downstreamPathLength[i,j] = 0`. Note that `RN$downstreamPathLength[i,i] = 0` if `includeDownstreamNode = FALSE`; alternatively, it is `RN$downstreamPathLength[i,i] = OCN$RN$leng[i]`. It is a `spam` object.

`AG$downstreamPath`
: List (of length `OCN$AG$nNodes`) whose object `i` is a list (of length `OCN$AG$nNodes`). If nodes `i` and `j` are connected by a downstream path, then `AG$downstreamPath[[i]][[j]]` is a vector containing the indices of the nodes constituting such path (`i` and `j` are included). If `i` and `j` are not connected by a downstream path, then `AG$downstreamPath[[i]][[j]] = 0`.

`AG$downstreamPathLength`
: Sparse matrix (`OCN$AG$nNodes` by `OCN$AG$nNodes`) containing length of paths between nodes that are connected by a downstream path; if `i` and `j` are not connected by a downstream path, then `AG$downstreamPathLength[i,j] = 0`. Note that `AG$downstreamPathLength[i,i] = 0` if `includeDownstreamNode = FALSE`; alternatively, it is `AG$downstreamPathLength[i,i] = OCN$AG$leng[i]`. It is a `spam` object.

`AG$downstreamLengthUnconnected`
: Sparse matrix (`OCN$AG$nNodes` by `OCN$AG$nNodes`). `AG$downstreamLengthUnconnected[i,j]` is the length of the downstream portion of a path joining node `i` to `j` if `i` and `j` are not connected by a downstream path. Specifically, `AG$downstreamLengthUnconnected[i,j] = AG$downstreamPathLength[i,k]`, where `k` is a node such that there exist a downstream path from `i` to `k` and from `j` to `k`, and these paths are the shortest possible. Note that the length of the upstream portion of the path joining `i` to `j` is given by `AG$downstreamLengthUnconnected[j,i]`. If instead `i` and `j` are joined by a downstream path, then `AG$downstreamLengthUnconnected[i,j] = 0`. It is a `spam` object.

Examples

```
# 1) Calculate paths between nodes of an OCN
OCN <- paths_OCN(aggregate_OCN(landscape_OCN(OCN_20), thrA = 4))
## Not run:
# 2) Display distance to outlet (at the RN level) along the main stem
# of an OCN
OCN <- aggregate_OCN(landscape_OCN(OCN_250_T)) # this takes some seconds
```

```
OCN <- paths_OCN(OCN, pathsRN = TRUE) # this takes some seconds

distanceToOutlet <- OCN$RN$downstreamPathLength[,OCN$RN$outlet]
farthestNode <- which(distanceToOutlet == max(distanceToOutlet))
mainStem <- OCN$RN$downstreamPath[[farthestNode]][[OCN$RN$outlet]]
theme <- rep(NaN, OCN$RN$nNodes)
theme[mainStem] <- distanceToOutlet[mainStem]

draw_thematic_OCN(theme, OCN)
title("Distance to outlet along the main stem [pixel units]")

## End(Not run)
```

`rivergeometry_OCN` *River geometry of an Optimal Channel Network*

Description

Function that calculates river width, depth and water velocity by applying Leopold's scaling relationships to nodes at the RN and AG levels.

Usage

```
rivergeometry_OCN(OCN, widthMax = 1, depthMax = 1,
  velocityMax = 1, expWidth = NaN, expDepth = NaN,
  expVelocity = NaN)
```

Arguments

`OCN`	List as produced by `aggregate_OCN`.
`widthMax`	Maximum river width allowed. If `nOutlet = 1`, it corresponds to the width at the outlet node.
`depthMax`	Maximum river depth allowed. If `nOutlet = 1`, it corresponds to the depth at the outlet node.
`velocityMax`	Maximum water velocity allowed. If `nOutlet = 1`, it corresponds to the water velocity at the outlet node.
`expWidth, expDepth, expVelocity`	Exponents for the power law relationship between river width, depth, water velocity and contributing area. If none of `expWidth`, `expDepth`, `expVelocity` is specified by the user, the values `expWidth = 0.5`, `expDepth = 0.4`, `expDepth = 0.1` proposed by Leopold and Maddock [1953] are used. It is possible to specify two out of these three exponents, provided that each of them lies in the range (0; 1) and their sum is lower than one. In this case, the missing exponent is calculated as the complement to one of the sum of the two values provided. If all three exponents are specified by the user, their sum must be equal to one.

Details

The values of contributing area used to evaluate river geometry at the AG level are equal to `0.5*(OCN$AG$A + OCN$AG$AReach)`. See also `aggregate_OCN`.

See also Leopold, L. B., & Maddock, T. (1953). *The hydraulic geometry of stream channels and some physiographic implications* (Vol. 252). US Government Printing Office.

Value

A list that contains all objects contained in `OCN`, in addition to the objects listed below.

`RN$width`	Vector (of length `OCN$RN$nNodes`) of river width values for every RN node.
`RN$depth`	Vector (of length `OCN$RN$nNodes`) of river depth values for every RN node.
`RN$velocity`	Vector (of length `OCN$RN$nNodes`) of water velocity values for every RN node.
`AG$width`	Vector (of length `OCN$AG$nNodes`) of river width values for every AG node.
`AG$depth`	Vector (of length `OCN$AG$nNodes`) of river depth values for every AG node.
`AG$velocity`	Vector (of length `OCN$AG$nNodes`) of water velocity values for every AG node.

Finally, `widthMax`, `depthMax`, `velocityMax`, `expWidth`, `expDepth`, `expVelocity` are added to the list.

Examples

```
# 1) Compute river geometry of a 20x20 OCN with default options
# and display river width at the RN level
OCN <- rivergeometry_OCN(aggregate_OCN(landscape_OCN(OCN_20)))
draw_thematic_OCN(OCN$RN$width,OCN)
```

Index