Package ‘SMITIDvisu’

January 20, 2025

Type Package

Title Visualize Data for Host and Viral Population from 'SMITIDstruct' using 'HTMLwidgets'

Version 0.0.9

Maintainer Jean-Francois Rey <jean-francois.rey@inrae.fr>

Description Visualisation tools for 'SMITIDstruct' package.
Allow to visualize host timeline, transmission tree, index diversities
and variant graph using 'HTMLwidgets'. It mainly using 'D3JS' javascript framework.

Date 2021-02-08

Depends R (>= 3.5.0), utils

LinkingTo Rcpp

NeedsCompilation yes

SystemRequirements C++11

Biarch true

License GPL (>= 3) | file LICENSE

URL https://informatique-mia.inrae.fr/biosp/anr-smitid-project/,
https://gitlab.paca.inrae.fr/SMITID/visu/

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.11.0), htmlwidgets (>= 0.3.2), yaml (>= 2.1.16),
jsonlite (>= 1.5.0), magrittr

Suggests SMITIDstruct, knitr, shiny, testthat (>= 3.0.0)

RoxygenNote 7.1.1

Author Jean-Francois Rey [aut, cre],
Julien Boge [ctb]

Repository CRAN

Date/Publication 2021-02-08 08:50:03 UTC

Contents

`SMITIDvisu-package` *Visualize Data for Host and Viral Population from SMITIDstruct using HTMLwidgets*

Description

Visualisation tools for SMITIDstruct package. Allow to visualize host timeline, transmission tree, index diversities and variant graph using HTMLwidgets. It mainly using D3JS, noUiSlider and FileSaver javascript libraries.

Details

Package:	SMITIDvisu
Type:	Package
Version:	0.0.9
Date:	2021-02-08
License:	GPL (>=3)

Author(s)

Jean-Francois Rey <`jean-francois.rey@inrae.fr`>

Julien Boge <`julien.boge.u@gmail.com`>

Examples

```
## Not run:
 library(SMITIDvisu)
 demo.SMITIDvisu.run()

## End(Not run)
```

`createRainbowColors` *createRainbowColors Create a list of colors for each value v*

Description

createRainbowColors Create a list of colors for each value v

Usage

```
createRainbowColors(v)
```

Arguments

`v`	a vector of characters

Value

a list of value=color

`demo.SMITIDvisu.run` *demo.SMITIDvisu.run*

Description

run a demo to visualize data

Usage

```
demo.SMITIDvisu.run()
```

`df2geojson` *df2geojson*

Description

Transform a data frame into a string formatted in GeoJSON

Usage

```
df2geojson(df, multipleValuesByTime = c())
```

Arguments

`df`	Data frame to convert in GeoJSON. It must contain at least columns 'id', 'time', 'X' and 'Y'. Additionnal columns will be added as features' properties.
`multipleValuesByTime`	Vector of strings indicating the df columns names which can contain several values by time.

Value

a geojson string

Examples

```
library(SMITIDvisu)
data(transmissiontree)
geojson <- df2geojson(tt.events, multipleValuesByTime = c('infectedby', 'probabilities'))
```

hostline *A host infomation over time*

Description

kind of host time line

Usage

```
data("hostline")
```

Format

A data frame with 8 observations on the following 5 variables.

`level` a character vector

`label` a character vector

`ID` a character vector

`timestart` a character vector

`timeend` a character vector

Examples

```
data(hostline)
print(hostline)
```

maptt *maptt*

Description

Display a Transmission Tree over a map.

Usage

```
maptt(
  data,
  multipleValuesByTime = c(),
  circleRadius = 6,
  defaultNodeColor = "steelblue",
  nodeColorByState = list(),
  moveEdgeColor = "steelblue",
  color1 = "green",
  color2 = "red",
  nbColors = 10,
  minWeight = 0,
  maxWeight = 1,
  weight1 = 0,
  weight2 = 1,
  autoFocus = TRUE,
  keepOldFeatures = TRUE,
  optionsControl = TRUE,
  gradientControl = TRUE,
  legend = TRUE,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`data`	Either a data frame that will be converted to a GeoJSON collection, or a string describing a valid GeoJSON collection. The data frame must contain at least columns 'id', 'time', 'X' and 'Y'. It can contain columns 'infectedby', 'probabilities'. Additionnal columns will be added as properties, but will do nothing in this implementation of maptt. See the ‘df2geojson‘ function for more informations.
`multipleValuesByTime`	Vector of strings indicating the df columns names which can contain several values by time. Typically, you would use ‘c('infectedby','probabilities')‘ if you have these values.
`circleRadius`	Numeric value specifying the radius of the nodes in pixels.
`defaultNodeColor`	String indicating the default color of nodes, if their status doesn't match whith any color. Colors can be specified in hex.
`nodeColorByState`	List of strings, indicating the color scheme for each node state.
`moveEdgeColor`	String indicating the color of the edges representing the move of a node.
`color1`	String indicating the color corresponding to the minWeight value.
`color2`	String indicating the color corresponding to the maxWeight value.
`nbColors`	Number of colors for the color scheme using a gradient between color1 and color2. These colors will be used to represent the infection edges according to the infection probability. If no probability is used, the edge will use color2. Three intervals are created : color1 will be used for the probabilities between minWeight and weight1. Colors between color1 and color2 will be used for probabilities between weight1 and weight2. color2 will be used for probabilities between weight2 and maxWeight. This setting can be modified directly on the map if 'gradientControl' is activated.
`minWeight`	Minimal weight.
`maxWeight`	Maximal weight.

`weight1`	Lowest weight for the color scheme. This setting can be modified directly on the map if 'gradientControl' is activated.
`weight2`	Greatest weight for the color scheme. This setting can be modified directly on the map if 'gradientControl' is activated.
`autoFocus`	Boolean indicating if the map should focus at the displayed features at each time. This setting can be toggled directly on the map if 'optionsControl' is activated.
`keepOldFeatures`	Boolean indicating if old features should be displayed or not. Features are considered "old" if their last 'time' is prior to the current time displayed. This setting can be toggled directly on the map if 'optionsControl' is activated.
`optionsControl`	Boolean indicating if the options control should be displayed or not
`gradientControl`	Boolean indicating if the gradient control should be displayed or not
`legend`	Boolean indicating if the legend should be displayed or not
`width`	Numeric width for the area in pixels.
`height`	Numeric hieght for the area in pixels.
`elementId`	The element ID where the map is displayed

Examples

```
library(SMITIDvisu)
data(transmissiontree)

maptt(tt.events, multipleValuesByTime = c('infectedby', 'probabilities'))

# In this example:
# - values lower than 20 will be yellow ;
# - values between 20 and 25 will use colors between yellow and red ;
# - values greater than 25 will be red.
maptt(tt.events,
 multipleValuesByTime = c('infectedby', 'probabilities'),
 color1 = 'yellow',
 color2 = 'red',
 nbColors = 10,
 minWeight = 0,
 maxWeight = 30,
 weight1 = 20,
 weight2 = 25
)
```

`mapttProxy` *mapttProxy*

Description

get mapttProxy

Usage

```
mapttProxy(mapttId, session = shiny::getDefaultReactiveDomain())
```

Arguments

`mapttId`	widget instance identifier
`session`	shiny session

Examples

```
## Not run:
library(SMITIDvisu)
## server.R
mapttProxy <- mapttProxyProxy("mapttOutput")

## End(Not run)
```

`mapttSelectHost` *mapttSelectHost*

Description

select a host on a MapTT instance

Usage

```
mapttSelectHost(mapttProxy, hostId)
```

Arguments

`mapttProxy`	mapttProxy instance
`hostId`	the id of the host to select

See Also

`maptt`

Examples

```
## Not run:
library(SMITIDvisu)
data(transmissiontree)
## server.R
mapttProxy("mapttOutput") %>% mapttSelectHost()

## End(Not run)
```

`mstCompute` *compute the minimum spanning tree*

Description

compute the minimum spanning tree of a matrix representing edges between nodes (of a graph)

Usage

```
mstCompute(mat)
```

Arguments

`mat`	weighted matrix representing nodes connection (edges weight)

Value

a matrix with 1 if nodes are linked, 0 otherwise.

`mstVariant` *mstVariant*

Description

Draw Variants genotypes distances as a graph using Minimum Spanning Tree algorithm.

Usage

```
mstVariant(
  mat,
  prop,
  node.prop = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`mat`	a distance matrix between sequence of variants (interger distance no floating values)
`prop`	a data.frame for variants sequences proportions and count (see details)
`node.prop`	list of variants with proportions and time (default NULL)
`width`	numeric width for the area in pixels.
`height`	numeric hieght for the area in pixels.
`elementId`	the element ID where is draw

Details

mat is a simple distance matrix with interger values, row and lines contain a unique identifier of each variant sequences. **prop** is a data.frame where each row is a variant sequence, it have to contain in columns factor "ID", "proportion" and "count". "ID" is a unique identifier matching matrix value identifier, "proportion" is the proportions of the variant sequence and "count" the number of variant sequence in a varions set. **node.prop** is a list with name that matching **mat** identifier and **prop** "ID". Each list element contains a subvector time (Julian or timestamp) and value (proportions). That allow to draw variants proportions over time.

Examples

```
library(SMITIDvisu)
data(st)
mstV <- mstVariant(st.dist113_all,st.prop113_all, st.listTimeProp113)
## Not run:
## export as standalone html file
htmlwidgets::saveWidget(mstV, "mstVariant.html")
browseURL("mstVariant.html")

## End(Not run)
```

`mstVariantProxy` *mstVariantProxy*

Description

get mstVariantProxy

Usage

```
mstVariantProxy(mstVid, session = shiny::getDefaultReactiveDomain())
```

Arguments

`mstVid`	widget instance identifier
`session`	shiny session

Examples

```
## Not run:
library(SMITIDvisu)
## server.R
mstVariantProxy <- mstVaraintProxy("mstvariantoutput")

## End(Not run)
```

`SMITIDvisu-shiny` *Shiny bindings for visualisation widgets*

Description

Output and render functions for using visualisation widgets within Shiny applications and interactive Rmd documents.

Usage

```
mapttOutput(outputId, width = "100%", height = "400px")

renderMaptt(expr, env = parent.frame(), quoted = FALSE)

mstVariantOutput(outputId, width = "100%", height = "600px")

rendermstVariant(expr, env = parent.frame(), quoted = FALSE)

timeLineOutput(outputId, width = "100%", height = "400px")

renderTimeLine(expr, env = parent.frame(), quoted = FALSE)

transmissionTreeOutput(outputId, width = "100%", height = "500px")

renderTransmissionTree(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`outputId`	output variable to read from
`width, height`	Must be a valid CSS unit (like `"100%"`, `"400px"`, `"auto"`) or a number, which will be coerced to a string and have `"px"` appended.
`expr`	An expression that generates a networkD3 graph
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

`st` *A SMITIDstruct package variable.*

Description

A SMITIDstruct package variable from simul-chain as a list. The list is a set of HostSet, ViralPopset and an index

Usage

```
data("st")
```

Format

The format is: List of 3 $:List of 21 .. ..- attr(*, "class")= chr "HostSet" $:List of 20 .. ..- attr(*, "class")= chr "ViralPopSet" $:'data.frame': 79 obs. of 3 variables: ..$ TIME : chr [1:79] "0" "0" "1.26" "1.35" ... ..$ ID_HOST : chr [1:79] "1" "2" "2" "2" ... ..$ EVENTCODE: chr [1:79] "000011" "000110" "001000" "001000" ...

Examples

```
data(st)
## maybe str(st) ; plot(st) ...
```

`st.dist113_2` *Distance matrix of observed variants sequences of a host 113 at time 2 from simulation.*

Description

Levenshtein Distance matrix with rows and cols label as sequences ID.

Usage

```
data("st")
```

Format

The format is: num [1:23, 1:23] 0 1 1 1 2 1 1 1 1 2 ...

Examples

```
data(st)
```

`st.dist113_all` *Distance matrix of observed variants sequences of a host 113 at time 2, 3 and 4 from simulation.*

Description

Levenshtein Distance matrix with rows and cols label as sequences ID. Unique sequence variants observed on host 113 at time 2, 3 and 4 from a simulation.

Usage

```
data("st")
```

Format

The format is: num [1:51, 1:51] 0 1 1 1 1 1 3 1 1 1 ...

Examples

```
data(st)
```

`st.listTimeProp113` *List of variants ID with subvector for time and value.*

Description

A list indexed by variants sequences ID. Each element contain a time and value vector for time of observation and proportions observed at this time.

Usage

```
data("st")
```

Examples

```
data(st)
```

`st.prop113_2` *Variants proportions and count for host 113 at time 2 from simulation.*

Description

A data.frame with label "ID", "proportion" and "count" for an host 113 at time 2 from simulation. Each row is a sequence.

Usage

```
data("st")
```

Format

A data frame with 23 observations on the following 3 variables.

`ID` a character vector

`proportion` a numeric vector

`count` a numeric vector

Examples

```
data(st)
```

`st.prop113_all` *Variants proportions and count for an host 113 at time 2, 3 and 4 from simulation.*

Description

A data.frame with label "ID", "proportion" and "count" for an host 113 at time 2, 3 and 4 from simulation. Each row is a sequence.

Usage

```
data("st")
```

Format

A data frame with 51 observations on the following 3 variables.

`ID` a character vector

`proportion` a numeric vector

`count` a numeric vector

Examples

```
data(st)
```

timeLine *timeLine*

Description

Draw a host time line. Time use timestamp or Date in ISO format.

Usage

```
timeLine(
  data,
  title,
  color = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`data`	a data.frame that reprensent hosts status in time with ID, status and time in columns
`title`	a tttle as character
`color`	list of color for timeline elements
`width`	numeric width for the area in pixels.
`height`	numeric hieght for the area in pixels.
`elementId`	the element ID where is draw

Examples

```
library(SMITIDvisu)
data(hostline)
tl <- timeLine(hostline,
               title="Example host 113",
               color=list("infected"="red","offspring"="green",
                             "alive"="blue","inf"="orange",
                             "dead"="black","Obs"="purple"))
## Not run:
## export as standalone html file
htmlwidgets::saveWidget(tl, "timeline.html")
browseURL("timeline.html")

## End(Not run)
```

timeLineProxy — *timeLineProxy get an instance of a timeline*

Description

timeLineProxy get an instance of a timeline

Usage

```
timeLineProxy(tlid, session = shiny::getDefaultReactiveDomain())
```

Arguments

`tlid`	a timeline instance id
`session`	shiny session

Value

an object of class timeline_proxy

Examples

```
## Not run:
## server.R
## output server variable
output$timeline <- renderTimeLine({
         timeLine(data.frame(), "")
       })
## ui.R
timeLineOutput("timeline")
## server.R
tlproxy <- timeLineProxy("timeline")

## End(Not run)
```

transmissionTree — *transmissionTree*

Description

Draw a transmission tree over the time. Time use timestamp or Date in ISO format ("

Usage

```
transmissionTree(
  nodes,
  edges,
  nodes.color = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`nodes`	a data.frame that reprensent hosts status in time with ID, status and time in columns
`edges`	a data.frame that reprensent tramsmission link between hosts (pathogens) with ID, source, target and time in columns
`nodes.color`	a list of color for nodes status "status"="color"
`width`	numeric width for the area in pixels.
`height`	numeric hieght for the area in pixels.
`elementId`	the element ID where is draw

Examples

```
library(SMITIDvisu)
data(transmissiontree)
tt <- transmissionTree(tt.nodes,tt.edges, nodes.color = list("default"="black","Inf"="red"))
## Not run:
## export as standalone html file
htmlwidgets::saveWidget(tt, "transTree.html")
browseURL("transTree.html")

## End(Not run)
```

`transmissionTreeProxy` *transmissionTreeProxy*

Description

get transmissionTreeProxy

Usage

```
transmissionTreeProxy(ttid, session = shiny::getDefaultReactiveDomain())
```

Arguments

`ttid`	widget instance identifier
`session`	shiny session

Examples

```
## Not run:
library(SMITIDvisu)
## server.R
transmissionTreeProxy <- transmissionTreeProxyProxy("transmissionTreeoutput")

## End(Not run)
```

`tt.edges` *Pathogen link over the time*

Description

A data.frame of all transmission links between hosts (pathogens). Five columns ID, source, target, time and weight.

Usage

```
data("transmissiontree")
```

Format

A data frame with 13 observations on the following 5 variables.

`ID` a numeric vector

`source` a character vector

`target` a factor with levels `113` `104` `116` `115` `111` `109` `105` `108` `106` `112`

`time` a character vector

`weight` a character vector

Examples

```
data(transmissiontree)
print(tt.edges)
```

tt.events — *Data.frame of hosts events information by time. Fake data.*

Description

Fake simulated data of hosts events over the time.

Usage

```
data("transmissiontree")
```

Format

A data frame with 63 observations on the following 7 variables.

`id` a character vector

`time` a character vector

`status` a character vector

`infectedby` a character vector

`probabilities` a character vector

`X` a numeric vector

`Y` a numeric vector

Examples

```
data(transmissiontree)
print(tt.events)
```

tt.nodes — *Host list with there status over the time.*

Description

a data.frame of all the hosts identify by there ID. Three colums is use ID, status and time

Usage

```
data("transmissiontree")
```

Format

A data frame with 47 observations on the following 3 variables.

`ID` a character vector

`status` a character vector

`time` a character vector

Examples

```
data(transmissiontree)
print(tt.nodes)
```

---

updatemstVariant *updatemstVariant*

Description

update (redraw) an instance on mstVariant

Usage

```
updatemstVariant(mstVProxy, mat, prop, propTime = NULL)
```

Arguments

`mstVProxy`	mstVaraintProxy instance
`mat`	distance matrix
`prop`	proportions data.frame
`propTime`	list of each variant by time and proportions

See Also

`mstVariant`

Examples

```
## Not run:
library(SMITIDvisu)
data(mstVariant)
## server.R
mstVaraintProxy("mstvariantoutput") %>% updatemstVariant(st.dist,st.prop)

## End(Not run)
```

updateTimeLine *updateTimeLine*

Description

updateTimeLine

Usage

```
updateTimeLine(tlProxy, data, title)
```

Arguments

`tlProxy`	a timeline proxy instance
`data`	new data
`title`	new title

See Also

`timeLine`

Examples

```
## Not run:
## server.R
## output server variable
output$timeline <- renderTimeLine({
         timeLine(data.frame(), "")
       })
## ui.R
timeLineOutput("timeline")
## server.R
timeLineProxy("timeline") %>% updateTimeLine(newtimeline, "newId")

## End(Not run)
```

updateTransmissionTree *updateTransmissionTree*

Description

update (redraw) an instance of a transmissionTree

Usage

```
updateTransmissionTree(TTProxy, nodes, edges, options = NULL)
```

Arguments

`TTProxy`	transmissionTreeProxy instance
`nodes`	a data.frame that represent hosts status in time with ID, status and time in columns
`edges`	a data.frame that represent tramsmission link between hosts (pathogens) with ID, source, weight, target and time in columns
`options`	transmissionTree new options

See Also

`transmissionTree`

Examples

```
## Not run:
library(SMITIDvisu)
data(transmissionTree)
## server.R
transmissionTreeProxy("transmissionTreeoutput") %>% updatetransmissionTree(tt.nodes,tt.edges)

## End(Not run)
```

Index