Package 'NetSim'

July 2, 2014

Type Package

Title A Social Networks Simulation Tool in R

Version 0.9

Date 2013-06-13

Author Christoph Stadtfeld

Maintainer Christoph Stadtfeld <c.stadtfeld@rug.nl>

Description NetSim allows to combine and simulate a variety of micro-models to research their impact on the macro-features of social networks.

License GPL-2

Depends Rcpp (>= 0.9.15)

LinkingTo Rcpp

Suggests igraph

SystemRequirements GNU make

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-12-15 11:31:27

R topics documented:

`NetSim-package` *NetSim: A Social Networks Simulation Tool in R.*

Description

NetSim allows to combine and simulate a variety of micro-models to research their impact on the macro-features of social networks.

Details

Package:	NetSim
Type:	Package
Version:	0.9
Date:	2013-06-13
License:	GPL-2

Simulation models in **NetSim** are based on model sequences including the following steps
1. Determine the time span until the next change occurs
2. Apply change models based on the time span determined
3. Determine changes based on the updated states
4. Apply these changes

There may be several of such model chains running simultaneously. The comparison of step 1 of the models returns which action chain is started.

Author(s)

Christoph Stadtfeld
Maintainer: Christoph Stadtfeld <c.stadtfeld@rug.nl>

References

If you use NetSim for publications, please cite
Christoph Stadtfeld (2013): **NetSim: A Social Networks Simulation Tool in R** URL: http://www.christoph-stadtfeld.com/netsim
(check the web site for updated references)

change_models *Overview of change models implemented in NetSim*

Description

Overview of change models implemented in NetSim. The first two and the fifth are define stochastic actor-oriented models for network or behavior change. The third defines a Jackson and Rogers node inclusion model, the fourth a Watts and Strogatz 'small world' model.

For the SAOM-related models additional model specifications are necessary.

Usage

```
create_multinomial_choice_network_change_model(
focalActorIndex, networkIndex, effectContainer)
create_multinomial_choice_behavior_change_model(
focalActorIndex, attributeIndex, effectContainer)
create_jackson_rogers_change_model(
networkIndex,
pLinkToParentNode = 1.0, pLinkToNeigborNode = 1.0,
nParentNodes = 1, nNeighborNodes = 1)
create_watts_strogatz_change_model(networkIndex)
create_attribute_multinomial_choice_network_change_model(
networkIndex, poissonAttributeIndex,
updater = create_tie_swap_updater(networkIndex))
```

Arguments

focalActorIndex	Index of focal actor of the multinomial (network or behavior) change model
networkIndex	Index of the network that is subject to change (dependent network)
effectContainer	Container object storing effects as defined in the SIENA manual
attributeIndex	Index of the attribute container that is subject to change (dependent attribute)
pLinkToParentNode	Probability to link to a 'parent node'. See Jackson \& Rogers (2007), p.894
pLinkToNeigborNode	Probability to link to a selected neighbor of a parent. See Jackson \& Rogers (2007), p.894
nParentNodes	Number of 'parent node'. See Jackson \& Rogers (2007), p.894
nNeighborNodes	Number of neighbors considered of parent node. See Jackson \& Rogers (2007), p.894
updater	Defines a change that is considered by the focal actor after choosing a particular tie. By default, this is a tie swap updater.
poissonAttributeIndex	Index of a attribute container containing a number of concurring Poisson parameters

References

Ripley, R. M.; Snijders, T. A. B. & Preciado Lopez, P.: Manual for SIENA 4.0. (2012), Oxford: University of Oxford, Department of Statistics; Nuffield College. URL: http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manua

Jackson, M. O. & Rogers, B. W. Meeting strangers and friends of friends: How random are social networks? American Economic Review, 2007, 97, 890-915

Watts, D. J. & Strogatz, S. H. Collective dynamics of 'small-world' networks Nature, 1998, 393, 440-442

See Also

add_change_model create_tie_swap_updater create_effect_container add_to_effect_container create_effect add_effect

Examples

```
## The following example is taken from section 4.1 and 4.2 in package vignette
## C. Stadtfeld: "Netsim: A Social Networks Simulation tool in R" (2013)
## Examples of the other models can be taken from that source

# construct process state
nActors <- 21
mat <- matrix(0, nActors, nActors)
att <- c(rep(0, nActors/3),rep(1, nActors/3),rep(2, nActors/3))
network <- create_network(mat)
attributeContainer <- create_scale_attribute_container(att,
min=0, max=3, by = 1)
processState <- create_process_state()
processState <- add_network(processState, network, name = "network")
processState <- add_attribute_container(processState, attributeContainer,
name = "attribute")
networkIndex <- get_network_index(processState)
attributeIndex <- get_attribute_container_index(processState)

# construct effect container
# homophily model with basic network effects
effectContainerHomophily <- create_effect_container()
effectContainerHomophily <- add_to_effect_container(
effectContainerHomophily,
create_effect("density", networkIndex),
-2.5)
effectContainerHomophily <- add_to_effect_container(
effectContainerHomophily,
create_effect("recip", networkIndex),
2.5)
effectContainerHomophily <- add_to_effect_container(
effectContainerHomophily,
create_effect("transTrip", networkIndex),
0.5)
effectContainerHomophily <- add_to_effect_container(
effectContainerHomophily,
create_effect("cycle3", networkIndex),
```

```
-0.5)
effectContainerHomophily <- add_to_effect_container(
effectContainerHomophily,
create_effect("simX",
attributeIndex,
networkIndex,
0.5),
1.0)

# effect container of the behavior change model
effectContainerInfluence <- create_effect_container()
effectContainerInfluence <- add_to_effect_container(
effectContainerInfluence,
create_effect("linear",
attributeIndex),
0.0)
effectContainerInfluence <- add_to_effect_container(
effectContainerInfluence,
create_effect("quad",
attributeIndex),
0.0)
effectContainerInfluence <- add_to_effect_container(
effectContainerInfluence,
create_effect("totSim",
attributeIndex,
networkIndex, 10/18),
2.0)

# Definition of SAOM network model manager
modelManager <- create_model_manager()
# assign homophily model to all actors
for (i in c(0 : (nActors - 1) ) ){
# Poisson models
poissonParameterInfluence <- 5
poissonModelInfluence <- create_poisson_model(
poissonParameterInfluence)

poissonParameter <- 40
poissonModel <- create_poisson_model(poissonParameter)

# saom change models
saomHomophilyModel <- create_multinomial_choice_network_change_model(
i,
networkIndex,
effectContainerHomophily)
behaviorSaom <- create_multinomial_choice_behavior_change_model(
i,
attributeIndex,
effectContainerInfluence
)
# updaters
setAttributeUpdater <- create_actor_attribute_set_updater(
attributeIndex, i)
```

```
tieSwapUpdater <- create_tie_swap_updater(networkIndex)

# define model chains
modelManager <<- add_time_model(modelManager,
poissonModel)
modelManager <<- add_change_model(modelManager,
poissonModel,
saomHomophilyModel)
modelManager <<- add_updater(modelManager,
saomHomophilyModel,
tieSwapUpdater)
modelManager <<- add_time_model(modelManager,
        poissonModelInfluence)
modelManager <<- add_change_model(modelManager,
        poissonModelInfluence,
        behaviorSaom)
modelManager <<- add_updater(modelManager,
        behaviorSaom,
        setAttributeUpdater)

} # for loop

simulator <- create_simulator(processState, modelManager, 10)
# commented out due to Windows compilation problems. Further tests needed!
## Not run: simulate(simulator)
```

`create_attribute_container` *Create an attribute container object and maintain it*

Description

Create an attribute container object representing arbitrary values or values from a scale (factor variable) based on a numeric vector, change single values of the container and transform it back into a numeric vector.

A scale attribute container has to be chosen for individual *dependent* variables in stochastic actor-oriented models (SAOMs) for behavior change. A normal attribute container can be chosen to represent *independent* individual variables in SAOMs.

Usage

```
create_attribute_container(numericVector)
create_scale_attribute_container(numericVector, min = 0, max = 1, by = 1)
set_value(attributeContainer, i, value)
attribute_container_as_list(attributeContainer)
## S3 method for class 'NetSimAttributeContainer'
as.numeric(x, ...)
```

```
## S3 method for class 'NetSimAttributeContainer'
as.double(x, ...)
## S3 method for class 'NetSimAttributeContainer'
print(x, ...)
```

Arguments

numericVector	A numeric vector
min	Minimum value of the factor variable
max	Maximum value of the factor variable
by	Difference between the factor variables
i	Index of the attribute to be changed. Starts counting from 0, not 1.
value	Set attribute to this value
attributeContainer	A **NetSim** attribute container object
x	A **NetSim** attribute container object
...	Potential additional arguments

See Also

create_process_state, create_network

Examples

```
nActors <- 5
numericVector <- c(rep(0, nActors%/%2), rep(1, nActors - nActors%/%2))
attributeContainer1 <- create_attribute_container(numericVector)
attributeContainer2 <-
create_scale_attribute_container(numericVector, min = 0, max = 2, by = 1)

attributeContainer1 <- set_value(attributeContainer1, i = 0, value = 5)
# does not change anything as 5 is out of range
attributeContainer2 <- set_value(attributeContainer2, i = 0, value = 5)

as.numeric(attributeContainer1)
as.numeric(attributeContainer2)
```

---

create_network *Create an network object and maintain it*

Description

Create a network object in **NetSim** from a matrix, set particular ties of the network and transform it back into a matrix

Usage

```
create_network(matrix, directed = TRUE, reflexive = FALSE)
set_tie(network, i, j, value)
network_as_matrix(network)
## S3 method for class 'NetSimNetwork'
as.matrix(x, ...)
## S3 method for class 'NetSimNetwork'
print(x, ...)
```

Arguments

matrix	A (squared) matrix object
directed	Indicates whether the network is directed (TRUE) or undirected (FALSE)
reflexive	Indicates whether the matrix is reflexive (TRUE) or not (FALSE)
network	A **NetSim** network object
i,j	index of one particular tie. The index starts counting from 0.
value	A value a tie is set to (1 or 0 in binary networks)
x	A **NetSim** network object
...	Additional arguments

See Also

create_process_state create_attribute_container add_random_ties_to_network add_ring_lattice_to_networ

Examples

```
nActors <- 5
network <- create_network(matrix(1, nActors, nActors))

# set ties
set_tie(network, i = 0, j = 1, value = 0)
set_tie(network, i = 0, j = 2, value = 0)
# returns FALSE and keeps the network unchanged as the network is not reflexive
set_tie(network, i = 0, j = 0, value = 1)

# there are two equivalent ways to re-transform the network into a matrix
as.matrix(network)
network_as_matrix(network)
```

`effect_container` *Overview of SAOM-related functions*

Description

Overview of functions to specify stochastic actor-oriented models. The naming of effects is in line with the RSiena manual (see reference below).

The following effects are implemented in SAOMs for NetSim so far (to be extended, see RSiena manual for detailed explanations):
OneModeNetworkEffect
- density
- recip
- transTrip
- cycle3
- nbrDist2
- inPop
- outPop

AttributeOneModeNetworkEffect
- altX
- egoX

SimilarityAttributeOneModeNetworkEffect
- simX
- totSim

AttributeEffect
- linear
- quad

MultiplexNetworkEffect - crprod

Usage

```
create_effect_container()
add_to_effect_container(effectContainer, effect, parameter)
create_effect(name, ...)
## S3 method for class 'OneModeNetworkEffect'
create_effect(name, networkIndex, ...)
## S3 method for class 'AttributeOneModeNetworkEffect'
create_effect(name, attributeIndex, networkIndex, ...)
## S3 method for class 'SimilarityAttributeOneModeNetworkEffect'
create_effect(name, attributeIndex, networkIndex, meanSimilarityScore, ...)
## S3 method for class 'AttributeEffect'
create_effect(name, attributeIndex, ...)
## S3 method for class 'MultiplexNetworkEffect'
create_effect(name, networkIndex1, networkIndex2, ...)
## S3 method for class 'AttributeMultinomialChoiceNetworkChangeModel'
add_effect(changeModel, effect, attributeIndex, ...)
get_effect_type(name)
```

Arguments

effectContainer	An effect container object
effect	An effect object created with create_effects function
parameter	Parameter of the effect specified
name	String name of the effect according to the SIENA manual
changeModel	A SAOM change model
...	Additional parameters depending on the type of the effect (see description above)
attributeIndex	Index of an attribute containing the (changeable) parameter
meanSimilarityScore	mean similarity score according to effect definitions in the SIENA manual
networkIndex	Index of a network
networkIndex1	Index of a network
networkIndex2	Index of a network

References

Ripley, R. M.; Snijders, T. A. B. & Preciado Lopez, P.: Manual for SIENA 4.0. (2012), Oxford: University of Oxford, Department of Statistics; Nuffield College. URL: http://www.stats.ox.ac.uk/~snijders/siena/RSiena_Manua

See Also

create_multinomial_choice_network_change_model create_multinomial_choice_behavior_change_model
create_attribute_multinomial_choice_network_change_model create_tie_swap_updater

Examples

```
## See sections 4.1, 4.2 and 4.5 in the package vignette for detailed examples
## Small example:
```

---

model_manager *Manage models using a model manager object*

Description

A set of functions to create a model manager and to add 1. time models, 2. time updater, 3. change models and 4. updaters to it.

In general, model managers consist of a set of model chains that follow the generic four-step scheme sketched above.

Usage

```
create_model_manager()
add_time_model(modelManager, timeModel)
add_time_updater(modelManager, timeUpdater)
add_change_model(modelManager, timeModel, changeModel)
add_updater(modelManager, changeModel, updater)
```

Arguments

modelManager	A model manager object
timeModel	A time model object. See time model overview for examples.
timeUpdater	A time updater object. See time updater overview for examples.
changeModel	A change model object. See change model overview for examples.
updater	An updater. See updater overview for examples.

See Also

create_network create_attribute_container

Examples

```
# minimal code example model manager
modelManager <- create_model_manager()

processState <- create_process_state()
processState <- add_global_attribute(processState, 0.0, name = "timer")
timerIndex <- get_global_attribute_index(processState, name = "timer")

myTimeModel <- create_round_based_time_model(timerIndex)

modelManager <- add_time_model(modelManager, myTimeModel)

simulator <- create_simulator(processState, modelManager, 20)
simulate(simulator)
```

---

network_utils *Utility functions for NetSim network objects*

Description

Utility functions to quickly change macro features of NetSim network objects. The first function adds random ties to a network, the second function creates a ring lattice structure to a network.

Usage

```
add_random_ties_to_network(network, probability = 0.5)
add_ring_lattice_to_network(network, nReciprocalTies)
```

Arguments

network	A NetSim network object
probability	Bernoulli probability of a non-existing tie to be set to 1.
nReciprocalTies	The number of reciprocal ties one node has in the ring lattice structure.

See Also

create_network

Examples

```
nActors <- 5
network <- create_network(matrix(0, nActors, nActors))

add_random_ties_to_network(network, probability = 0.5)
add_random_ties_to_network(network, probability = 1)

# reset network to an empty network
network <- create_network(matrix(0, nActors, nActors))
add_ring_lattice_to_network(network, nReciprocalTies = 2)
## Not run:
add_ring_lattice_to_network(network, nReciprocalTies = 0) # throws an error message

## End(Not run)
```

---

package-integration *Functions to transform NetSim objects to objects of other network classes*

Description

Functions to transform NetSim objects to objects of other network classes. So far, a transformation function to igraph objects is implemented

Usage

```
## S3 method for class 'NetSimNetwork'
as.igraph(x, mode=c("directed", "undirected"), ...)
```

Arguments

x	A Netsim network object
mode	Character scalar, specifies whether igraph should interpret the graph as directed or undirected. More options are not supported, yet.
...	Additional arguments

References

Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal, Complex Systems 1695. 2006. http://igraph.sf.net

Examples

```
nActors <- 4
network <- create_network(matrix(0, nActors, nActors))

# set ties
set_tie(network, i = 0, j = 1, value = 1)
set_tie(network, i = 0, j = 2, value = 1)

## Not run: library(igraph)
## Not run: myIGraph <- as.igraph(network, mode = "directed")
## Not run: plot(myIgraph) # now using the igraph package
```

`process_state` *Manage the process state*

Description

A set of functions to manage the process state of the simulation Markov process. The process state can be created and networks, attribute containers and global attributes be added. The IDs of these objects can be requested and are necessary for some sub-sequent functions.

Usage

```
create_process_state(name = "default")
add_network(processState, network, name = "defaultNetwork")
get_network_index(processState, name = "defaultNetwork")
get_network(processState, name = "defaultNetwork")
add_attribute_container(processState, attributeContainer, name = "defaultAttribute")
get_attribute_container_index(processState, name = "defaultAttribute")
get_attribute_container(processState, name = "defaultAttribute")
add_global_attribute(processState, value = 0.0, name = "defaultGlobalAttribute")
get_global_attribute_index(processState, name = "defaultGlobalAttribute")
get_global_attribute(processState, name = "defaultGlobalAttribute")
get_process_state_name(processState)
```

Arguments

`processState`	A process state object
`network`	A network object
`attributeContainer`	An attribute container object
`name`	A string name of an object
`value`	A value to initialize the global attribute

See Also

create_network, create_attribute_container

Examples

```
# create an empty process state
processState <- create_process_state()

nActors <- 5

# create network object and add to process state
network <- create_network(matrix(1, nActors, nActors))
processState <- add_network(processState, network, name = "friendship")
get_network_index(processState, name = "friendship")

# create attribute container and add to process state
attribute1 <- create_attribute_container(c(rep(0, nActors%/%2), rep(1, nActors - nActors%/%2)))
attribute2 <- create_attribute_container(c(rep(0, nActors%/%2), rep(1, nActors - nActors%/%2)))
processState <- add_attribute_container(processState, attribute1, name = "gender")
processState <- add_attribute_container(processState, attribute2, name = "age")
as.numeric(attribute1)
get_attribute_container_index(processState, name="gender")
get_attribute_container_index(processState, name="age")

# add a global variable to the process state
processState <- add_global_attribute(processState, value = 1, name = "timer")
get_global_attribute(processState, name="timer")
get_global_attribute_index(processState, name="timer")
```

---

simulator *Functions related to run a simulation*

Description

Functions related to run a simulation based on a process state and a model manager. A simulator object needs to be created, then the simulator can be run. After the simulation the number of simulation steps can be returned.

Usage

```
create_simulator(processState, modelManager, periodLength = 1,
verbose = TRUE, debug = FALSE)
simulate(simulator)
get_iteration_steps(simulator)
```

Arguments

processState	A process state object
modelManager	A model manager object
periodLength	A time span to simulate. The default is 1.
verbose	If TRUE it generates some basic information output including a progress bar
debug	If TRUE it generate a very detailed output for some models. Slows down the simulation process significantly.
simulator	A simulator object

See Also

create_process_state create_model_manager

Examples

```
## Detailed examples of process states and model managers can be found in
## section 4 of the package vignette.
processState <- create_process_state()
modelManager <- create_model_manager()

processState <- create_process_state()
processState <- add_global_attribute(processState, 0.0, name = "timer")
timerIndex <- get_global_attribute_index(processState, name = "timer")
myTimeModel <- create_round_based_time_model(timerIndex)
modelManager <- add_time_model(modelManager, myTimeModel)

timeSpan <- 20
simulator <- create_simulator(processState, modelManager, timeSpan,
verbose = TRUE, debug = FALSE)
simulate(simulator)
```

---

time_models *Overview of time models implemented in NetSim*

Description

Overview of different time models implemented in NetSim. The first model is a deterministic round-based time model, the second is a Poisson model based on a fixed (unchangeable) parameter, the third is a Poisson model for which parameters can be stored in an attribute container (changeable).

Usage

```
create_round_based_time_model(timerIndex, intervalLength = 1.0, startTime = 0.0)
create_poisson_model(param = 1)
create_attribute_poisson_model(attributeIndex)
```

Arguments

timerIndex	A pointer to a global attribute in the process state representing the timer
intervalLength	interval length between two subsequent events triggered
startTime	First point in time when the timer index is triggered
param	Fixed Poisson parameter
attributeIndex	A pointer to an attribute container giving a vector of concurring Poisson parameters

See Also

create_model_manager

Examples

```
processState <- create_process_state()

# add attribute container to process state
attribute <- create_attribute_container(rep(2.0, 5))
processState <- add_attribute_container(processState, attribute, name = "poissonAttribute")
attributeIndex <- get_attribute_container_index(processState, name = "poissonAttribute")

# add timer variable to process state
processState <- add_global_attribute(processState, name = "timer", value = 0)
timerIndex <- get_global_attribute_index(processState, name = "timer")


# create time models based on process state pointers
timeModel1 <- create_round_based_time_model(timerIndex)
timeModel2 <- create_poisson_model(3.0)
timeModel3 <- create_attribute_poisson_model(attributeIndex)

# a change of the attribute container changes time model 3
set_value(attribute, 0, 0.5)
```

---

time_updater *Overview of time updaters implemented in NetSim*

Description

Overview of time updaters implemented in NetSim. So far, there is one time updater that keeps a global timer up-to-date by adding time spans determined by time models.

Usage

```
create_timer_updater(timerIndex)
```

Arguments

timerIndex	A pointer to a global attribute in the process state representing the timer

See Also

create_model_manager

Examples

```
processState <- create_process_state()

# add timer variable to process state
processState <- add_global_attribute(processState, name = "timer", value = 0)
timerIndex <- get_global_attribute_index(processState, name = "timer")

# create time updater based on the timer index
timeUpdater <- create_timer_updater(timerIndex)

modelManager <- create_model_manager()

myTimeModel <- create_round_based_time_model(timerIndex)
modelManager <- add_time_model(modelManager, myTimeModel)

timeSpan <- 20
simulator <- create_simulator(processState, modelManager, timeSpan,
verbose = TRUE, debug = FALSE)
simulate(simulator)
```

---

updater *Overview of updaters implemented in NetSim*

Description

Overview of updaters implemented in NetSim. When linking an updateer to a change model one has to make sure that the results of the change model can be interpreted by the updater.

Usage

```
create_tie_swap_updater(networkIndex)
create_actor_attribute_set_updater(attributeIndex, actorIndex)
create_add_actor_updater()
create_add_ties_from_newborn_actor_updater(networkIndex)
create_set_attribute_of_newborn_actor_updater(attributeIndex, value)
create_rewire_tie_updater(networkIndex)
```

Arguments

networkIndex	Index of the focal network
attributeIndex	Index of the focal attribute container
actorIndex	Index of the focal actor
value	Value of the attribute

See Also

create_model_manager

Examples

```
## Extensive examples can be found in section 4 of the package vignette
## A more detailed description is given in section 3.3 of the package vignette
```

---

utils *General utility functions*

Description

General utility functions to read and manipulate the random number generator by setting and reading the random seed. This assures reproducibility of simulation results.

Usage

```
reset_random_seed(seed)
get_random_seed()
```

Arguments

seed	Random number seed, a positive integer

Examples

```
reset_random_seed(12)
get_random_seed()
```

Index