Package 'tergmLite'

May 8, 2026

Version 2.6.1

Date 2022-07-20

Title Fast Simulation of Simple Temporal Exponential Random Graph Models

Description Provides functions for the computationally efficient simulation of dynamic networks estimated with the statistical framework of temporal exponential random graph models, implemented in the 'tergm' package.

Depends R (>= 3.5), ergm (>= 4.0), tergm (>= 4.0)

License GPL-3

Imports statnet.common (>= 4.4.0), network (>= 1.17.0), networkDynamic (>= 0.11.0), Rcpp, tibble, methods

Suggests testthat, EpiModel (>= 2.0.5)

LinkingTo Rcpp, ergm

RoxygenNote 7.2.0

Encoding UTF-8

NeedsCompilation yes

Author Samuel M. Jenness [aut, cre], Chad Klumb [aut]

Maintainer Samuel M. Jenness <samuel.m.jenness@emory.edu>

Repository CRAN

Date/Publication 2022-07-20 15:20:02 UTC

Contents

`tergmLite-package` *Fast Simulation of Simple Temporal Exponential Random Graph Models*

Description

Package:	tergmLite
Type:	Package
Version:	2.6.1
Date:	2022-07-20
License:	GPL-3
LazyLoad:	yes

Details

The statistical framework of temporal exponential random graph models (TERGMs) provides a rigorous, flexible approach to estimating generative models for dynamic networks and simulating from them for the purposes of modeling infectious disease transmission dynamics. TERGMs are used within the `EpiModel` software package to do just that. While estimation of these models is relatively fast, the resimulation of them using the tools of the `tergm` package is computationally burdensome, requiring hours to days to iteratively resimulate networks with co-evolving demographic and epidemiological dynamics. The primary reason for the computational burden is the use of the `network` class of object (designed within the package of the same name); these objects have tremendous flexibility in the types of networks they represent but at the expense of object size. Continually reading and writing larger-than-necessary data objects has the effect of slowing the iterative dynamic simulations.

The `tergmLite` package reduces that computational burden by representing networks less flexibly, but much more efficiently. For epidemic models, the only types of networks that we typically estimate and simulate from are undirected, binary edge networks with no missing data (as it is simulated). Furthermore, the network history (edges or node attributes) does not need to be stored for research-level applications in which summary epidemiological statistics (e.g., disease prevalence, incidence, and variations on those) at the population-level are the standard output metrics for epidemic models. Therefore, the network may be stored as a cross-sectional edgelist, which is a two-column matrix of current edges between one node (in column one) and another node (in column two). Attributes of the edges that are called within ERGMs may be stored separately in vector format, as they are in `EpiModel`. With this approach, the simulation time is sped up by a factor of 25-50 fold, depending on the specific research application.

add_vertices *Fast Version of network::add.vertices for Edgelist-formated Network*

Description

This function performs a simple operation of updating the edgelist attribute n that tracks the total network size implicit in an edgelist representation of the network.

Usage

```
add_vertices(el, nv)
```

Arguments

el	A two-column matrix of current edges (edgelist) with an attribute variable n containing the total current network size.
nv	A integer equal to the number of nodes to add to the network size at the given time step.

Details

This function is used in EpiModel modules to add vertices (nodes) to the edgelist object to account for entries into the population (e.g., births and in-migration).

Value

Returns the updated the attribute containing the population size on the edgelist, el, based on the number of new vertices specified to be added in nv.

Examples

```
## Not run:
library("EpiModel")
nw <- network_initialize(100)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
x <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

param <- param.net(inf.prob = 0.3)
init <- init.net(i.num = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, tergmLite = TRUE)

# networkLite representation after initialization
dat <- crosscheck.net(x, param, init, control)
dat <- initialize.net(x, param, init, control)

# Check current network size
attributes(dat$el[[1]])$n
```

```
# Add 10 vertices
dat$el[[1]] <- add_vertices(dat$el[[1]], 10)

# Check new network size
attributes(dat$el[[1]])$n

## End(Not run)
```

`delete_vertices` *Fast Version of network::delete.vertices for Edgelist-formated Network*

Description

Given a current two-column matrix of edges and a vector of IDs to delete from the matrix, this function first removes any rows of the edgelist in which the IDs are present and then permutes downward the index of IDs on the edgelist that were numerically larger than the IDs deleted.

Usage

```
delete_vertices(el, vid)
```

Arguments

`el`	A two-column matrix of current edges (edgelist) with an attribute variable `n` containing the total current network size.
`vid`	A vector of IDs to delete from the edgelist.

Details

This function is used in `EpiModel` modules to remove vertices (nodes) from the edgelist object to account for exits from the population (e.g., deaths and out-migration)

Value

Returns a updated edgelist object, `el`, with the edges of deleted vertices removed from the edgelist and the ID numbers of the remaining edges permuted downward.

Examples

```
## Not run:
library("EpiModel")
set.seed(12345)
nw <- network_initialize(100)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
```

```
x <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

param <- param.net(inf.prob = 0.3)
init <- init.net(i.num = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, tergmLite = TRUE)

# Set seed for reproducibility
set.seed(123456)

# networkLite representation structure after initialization
dat <- crosscheck.net(x, param, init, control)
dat <- initialize.net(x, param, init, control)

# Current edges
head(dat$el[[1]], 20)

# Remove nodes 1 and 2
nodes.to.delete <- 1:2
dat$el[[1]] <- delete_vertices(dat$el[[1]], nodes.to.delete)

# Newly permuted edges
head(dat$el[[1]], 20)

## End(Not run)
```

`get_vertex_attribute` *Get Vertex Attribute on Network Object*

Description

Gets a vertex attribute from an object of class `network`, wrapping the related function in the `network` package.

Usage

```
get_vertex_attribute(x, attrname)
```

Arguments

`x`	An object of class network.
`attrname`	The name of the attribute to get.

Details

This function is used in `EpiModel` workflow to query vertex attributes on an initialized empty network object (with `network_initialize`.

Value

Returns an object of class network.

Examples

```
## Not run:
nw <- network_initialize(100)
nw <- set_vertex_attribute(nw, "age", runif(100, 15, 65))
get_vertex_attribute(nw, "age")

## End(Not run)
```

---

init_tergmLite *Initializes EpiModel netsim Object for tergmLite Simulation*

Description

Initializes EpiModel netsim Object for tergmLite Simulation

Usage

```
init_tergmLite(dat)
```

Arguments

dat	A list object containing a networkDynamic object and other initialization information passed from netsim.

Details

This function is typically used within the initialization modules of EpiModel to establish the necessary infrastructure needed for tergmLite network resimulation. The example below demonstrates the specific information returned.

Value

Returns the list object dat and adds the element el which is an edgelist representation of the network. Also converts the nw element to a networkLite representation.

Examples

```
## Not run:
library("EpiModel")
nw <- network_initialize(100)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
x <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)
```

```
param <- param.net(inf.prob = 0.3)
init <- init.net(i.num = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, tergmLite = TRUE)

# networkLite representation after initialization
dat <- crosscheck.net(x, param, init, control)
dat <- initialize.net(x, param, init, control)
str(dat, max.level = 1)

# Element added is el (edgelist representation of network)...
dat$el

# ... and nw is now a networkLite
dat$nw[[1]]

## End(Not run)
```

---

networkLite *networkLite Constructor Utilities*

Description

Constructor methods for networkLite objects.

Usage

```
networkLite(x, ...)

## S3 method for class 'numeric'
networkLite(
  x,
  directed = FALSE,
  bipartite = FALSE,
  loops = FALSE,
  hyper = FALSE,
  multiple = FALSE,
  ...
)

## S3 method for class 'edgelist'
networkLite(x, attr = list(), ...)

## S3 method for class 'matrix'
networkLite(x, attr = list(), ...)
```

Arguments

x	either an edgelist class network representation (including network attributes in its attributes list), or a number specifying the network size.
...	additional arguments used by other methods.
directed, bipartite, loops, hyper, multiple	common network attributes that may be set via arguments to the networkLite.numeric method.
attr	a named list of vertex attributes for the network represented by x.

Details

Currently there are two distinct networkLite constructor methods available.

The edgelist method takes an edgelist class object x with network attributes attached in its attributes list, and a named list of vertex attributes attr, and returns a networkLite object, which is a named list with fields el, attr, and gal; the fields el and attr match the arguments x and attr respectively, and the field gal is the list of network attributes (copied from attributes(x)). Missing attributes directed, bipartite, loops, hyper, and multiple are defaulted to FALSE; the network size attribute n must not be missing. Attributes class, dim, and vnames (if present) are not copied from x to the networkLite. (For convenience, a matrix method, identical to the edgelist method, is also defined, to handle cases where the edgelist is, for whatever reason, not classed as an edgelist.)

The numeric method takes a number x as well as the network attributes directed, bipartite, loops, hyper, and multiple (defaulting to FALSE), and returns an empty networkLite with these network attributes and number of nodes x.

Within tergmLite, the networkLite data structure is used in the calls to ergm and tergm simulate functions.

Value

A networkLite object with edge list el, vertex attributes attr, and network attributes gal.

Examples

```
## Not run:
library("EpiModel")
nw <- network_initialize(100)
formation <- ~edges
target.stats <- 50
coef.diss <- dissolution_coefs(dissolution = ~offset(edges), duration = 20)
x <- netest(nw, formation, target.stats, coef.diss, verbose = FALSE)

param <- param.net(inf.prob = 0.3)
init <- init.net(i.num = 10)
control <- control.net(type = "SI", nsteps = 100, nsims = 5, tergmLite = TRUE)

# networkLite representation after initialization
dat <- crosscheck.net(x, param, init, control)
dat <- initialize.net(x, param, init, control)
```

```
# Conversion to networkLite class format
nwl <- networkLite(dat$el[[1]], dat$attr)
nwl

## End(Not run)
```

---

networkLitemethods *networkLite Methods*

Description

S3 methods for networkLite class, for generics defined in network package.

Usage

```
## S3 method for class 'networkLite'
get.vertex.attribute(x, attrname, ...)

## S3 method for class 'networkLite'
set.vertex.attribute(x, attrname, value, v = seq_len(network.size(x)), ...)

## S3 method for class 'networkLite'
list.vertex.attributes(x, ...)

## S3 method for class 'networkLite'
get.network.attribute(x, attrname, ...)

## S3 method for class 'networkLite'
set.network.attribute(x, attrname, value, ...)

## S3 method for class 'networkLite'
list.network.attributes(x, ...)

## S3 method for class 'networkLite'
network.edgecount(x, ...)

## S3 method for class 'networkLite'
as.edgelist(x, output = c("matrix", "tibble"), ...)

## S3 method for class 'networkLite'
mixingmatrix(object, attr, ...)

## S3 replacement method for class 'networkLite'
x[i, j] <- value

## S3 method for class 'networkLite'
```

```
print(x, ...)

## S3 method for class 'networkLite'
network.naedgecount(x, ...)

## S3 method for class 'networkLite'
add.edges(
  x,
  tail,
  head,
  names.eval = NULL,
  vals.eval = NULL,
  ...,
  check.unique = FALSE
)

as.networkLite(x, ...)

## S3 method for class 'network'
as.networkLite(x, ...)

## S3 method for class 'networkLite'
as.networkLite(x, ...)
```

Arguments

x	a networkLite object.
attrname	the name of an attribute in x.
...	any additional arguments.
value	Value to set edges to (must be FALSE for networkLite method)
v	indices at which to set vertex attribute values.
output	Type of edgelist to output.
object	a networkLite object
attr	specification of a vertex attribute in object as described in nodal_attributes
i, j	Nodal indices (must be missing for networkLite method)
tail	vector of tails of edges to add to the networkLite
head	vector of heads of edges to add to the networkLite
names.eval	currently unsupported by add.edges.networkLite
vals.eval	currently unsupported by add.edges.networkLite
check.unique	should a check to ensure uniqueness of edges in the final edgelist be performed?

Details

Allows use of networkLite objects in ergm_model.

`network_initialize` *Initialize Network Object*

Description

Initialize an undirected network object for use in EpiModel workflows.

Usage

```
network_initialize(
  n,
  directed = FALSE,
  hyper = FALSE,
  loops = FALSE,
  multiple = FALSE,
  bipartite = FALSE
)
```

Arguments

`n`	Network size.
`directed`	logical; should edges be interpreted as directed?
`hyper`	logical; are hyperedges allowed?
`loops`	logical; should loops be allowed?
`multiple`	logical; are multiplex edges allowed?
`bipartite`	count; should the network be interpreted as bipartite? If present (i.e., non-NULL) it is the count of the number of actors in the first mode of the bipartite network. In this case, the overall number of vertices is equal to the number of 'actors' (first mode) plus the number of 'events' (second mode), with the vertex.ids of all actors preceeding all events. The edges are then interpreted as nondirected.

Details

This function is used in `EpiModel` workflows to initialize an empty network object with the directed network attribute hard set to `FALSE`.

Value

Returns an object of class `network`.

Examples

```
## Not run:
nw <- network_initialize(100)
nw

## End(Not run)
```

`set_vertex_attribute` *Set Vertex Attribute on Network Object*

Description

Set a vertex attribute on an object of class `network`, wrapping the related function in the `network` package.

Usage

```
set_vertex_attribute(x, attrname, value, v)
```

Arguments

`x`	An object of class network.
`attrname`	The name of the attribute to set.
`value`	A vector of values of the attribute to be set.
`v`	IDs for the vertices whose attributes are to be altered.

Details

This function is used in `EpiModel` workflows to set vertex attributes on an initialized empty network object (with `network_initialize`.

Value

Returns an object of class `network`.

Examples

```
## Not run:
nw <- network_initialize(100)
nw <- set_vertex_attribute(nw, "age", runif(100, 15, 65))
nw

## End(Not run)
```

Index