Package 'ergmgp'

December 5, 2023

Version 0.1-1

Date 2023-12-04

Title Tools for Modeling ERGM Generating Processes

Depends network (>= 1.15), ergm (>= 3.10.1), networkDynamic (>= 0.10), parallel

Imports statnet.common (>= 4.2.0)

LinkingTo ergm

Description Provides tools for simulating draws from continuous time processes with well-defined exponential family random graph (ERGM) equilibria, i.e. ERGM generating processes (EGPs). A number of EGPs are supported, including the families identified in Butts (2023) <doi:10.1080/0022250X.2023.2180001>, as are functions for hazard calculation and timing calibration.

License GPL-3 + file LICENSE

URL https://statnet.org

BugReports https://github.com/statnet/ergmgp/issues

NeedsCompilation yes

Author Carter T. Butts [aut, cre], Statnet Commons [ctb]

Maintainer Carter T. Butts <buttsc@uci.edu>

Repository CRAN

Date/Publication 2023-12-05 11:50:05 UTC

R topics documented:

`ergmgp-package` *Tools for Modeling ERGM Generating Processes*

Description

Tools for simulation and analysis of continuous time graph processes with equilibria that can be described in exponential family random graph (ERGM) form.

Details

A random graph G on support $\mathcal{G}$ is said to be expressed in exponential family random graph (ERGM) form when its probability mass function (pmf) is written as

$$\Pr(G = g|\theta, X) = \frac{\exp\left(\theta^T w(g, X)\right) h(g)}{\sum_{g' \in \mathcal{G}} \exp\left(\theta^T w(g', X)\right) h(g')}$$

where θ is a parameter vector, w is a vector of sufficient statistics, X is a covariate set, and h is a reference measure. This form is quite general; in particular, any pmf on finite support can be written in ERGM form (albeit not always elegantly), making it a natural language for expressing graph distributions.

Now, consider a continuous time process whose state space is $\mathcal{G}$. A process of this type having an equilibrium distribution that can be written in (known) ERGM form is said to be an *ERGM generating process* or EGP. Although there are many types of EGPs, perhaps the most natural are continuous time Markov chains (CTMCs) whose transitions involve the addition or removal of individual edges from the current graph state. The transition rates of such CTMCs have the topology of the Hamming adjacency on $\mathcal{G}$; this is only sensible when considering graphs on a fixed vertex set, which is the typical use case. We can think of this class of EGPs as continuous time analogs of the Markov chains used in ERGM simulation (see `ergm`), with equilibrium obtained in the limit of time rather than simulation steps. EGPs are potentially useful as dynamic interpretations of empirically obtained ERGMs, or as *a priori* models in their own right. Since many Markovian EGPs are identified by their equilibrium ERGM together with a pacing constant, they are also natural choices when dynamics must be inferred from limited data (e.g., a cross-sectional network observation together with pacing or duration information).

The `ergmgp` package supports a number of different EGPs, all of which are currently Markovian with support on graphs or digraphs of fixed order. The following EGPs are currently supported; all definitions and notation follow Butts (2023). Define $q(g) = \theta^T w(g, X) + \ln h(g)$ to be the *graph potential*; in some cases, separate potentials (q_f and q_d) may be employed for the formation and dissolution of edges. For brevity, we define the normalizing factor of the equilibrium ERGM distribution by $Z = \sum_{g' \in \mathcal{G}} q(g')$, let w_e be the edge count statistic, and let $\mathcal{H}(g)$ be the Hamming neighborhood of g (i.e., the set of graphs reachable by single edge changes, or "toggles"). All transition rates from graph a to b (denoted $a \to b$) zero for $b \notin \mathcal{H}(a)$. Processes not otherwise noted were introduced in Butts (2023).

Longitudinal ERGM (`LERGM`) **Description:** Introduced by Koskinen and Snijders (2007), this process was originally conceived of as a continuum analog to the Gibbs sampler, with transition rates that are increasing with differences in graph potential. Grazioli et al. (2019)

subsequently showed that it can also be derived from a physical model with locally Arrhenius-like kinetics. This process has a maximum change rate (but no minimum), and may thus be plausible in settings for which changes can only be made when (exogenously determined) opportunities arise.

Event Rate ($a \to b$): $[1 + \exp[q(a) - q(b)]]^{-1}$

Equilibrium: $\exp[q(a)]/Z$

Competing Rate Stochastic Actor-Oriented Model (`CRSAOM`) Description: Introduced as an EGP by Snijders (2001), this model was originally proposed as a behavioral process, where vertices represent actors controlling their outgoing edges, the rate at which actors make tie changes is a function of the attractiveness of the networks reachable by making such changes, and (given opportunity to act) edge changes are chosen by a multinomial logit with utility function q. Dynamics in this model are distinctive in being driven solely by the attractiveness of the target state, which can sometimes lead to rapid state switching when multiple high-potential states are Hamming-adjacent.

Event Rate ($a \to b$): $\exp[q(b)]$

Equilibrium: $\exp[q(a)]/Z$

Change Inhibition (`CI`) Description: In the same sense that the LERGM is analogous to a continuum Gibbs sampler, this process is loosely analogous to continuum Metropolis algorithm. Downhill transitions with respect to the graph potential occur with a rate that is decreasing with the potential difference; uphill transitions, however, occur at a fixed rate (irrespective of the potential difference). The process thus works by selectively inhibiting downhill moves, rather than by preferentially moving to graphs of highest local potential.

Event Rate ($a \to b$): $\min(1, \exp[q(b) - q(a)])$

Equilibrium: $\exp[q(a)]/Z$

Differential Stability (`DS`) Description: Analogous to a "win-stay, lose-shift" process, transition targets in this EGP are chosen uniformly at random, with structure arising entirely from transition *times*. The time to exit a state g is proportional to $\exp[q(g)]$, making high-potential states exponentially more persistent than low-potential states. Note that this process is in a sense the inverse of the `CRSAOM`, being dependent only on the potential of the source state (while the `CRSAOM` depends only on the potential of the target state). Since the transitions themselves from a random walk, it should be noted that this process can generate very large numbers of transition events involving low-potential states that, while taking little phenomenological time, nevertheless are expensive to compute.

Event Rate ($a \to b$): $|\mathcal{H}(a)|^{-1} \exp[-q(a)]$

Equilibrium: $\exp[q(a)]/Z$

Constant Dissolution Continuum STERGM (`CDCSTERGM`) Description: This process is a continuum version of the discrete time constant dissolution separable temporal ERGM (STERGM) introduced by Carnegie et al. (2015); here, edges are lost randomly at a fixed rate, with a formation potential q_f that governs edge addition. This is a special case of the continuum STERGMs (below), and is particularly easy to identify from limited information.

Event Rate ($a \to b$): If b is formed by adding an edge to a, then $\exp[q_f(b) - q_f(a)]$; otherwise $\exp[\theta_d]$

Equilibrium: $\exp[q_f(a) - \theta_d w_e(a)]/Z$

Constant Formation Continuum STERGM (`CFCSTERGM`) Description: This process is analogous to the `CDCSTERGM`, except that in this case edge *formation* occurs randomly at a fixed rate, with a dissolution potential q_d governing edge loss. It is a simple model for settings in

which edges arise from essentially idiosyncratic events, with the resulting network structure subsequently stabilizing or destabilizing particular edges.

Event Rate ($a \to b$): If b is formed by adding an edge to a, then $\exp[\theta_f]$; otherwise $\exp[q_d(b) - q_d(a)]$

Equilibrium: $\exp[q_d(a) + \theta_f w_e(a)]/Z$

Continuum STERGM (`CSTERGM`) **Description:** This process represents a continuum limit of the discrete time separable temporal ERGMs (STERGMs) introduced by Krivitsky and Handcock (2014). Edge formation is here governed by one potential (q_f), while dissolution is governed by another (q_d), allowing these processes to be governed by different effects. The resulting equilibrium pmf is based on the sum of both potentials.

Event Rate ($a \to b$): If b is formed by adding an edge to a, then $\exp[q_f(b) - q_f(a)]$; otherwise $\exp[q_d(b) - q_d(a)]$

Equilibrium: $\exp[q_d(a) + q_f(a)]/Z$

Continuum TERGM (`CTERGM`) **Description:** This process is a continuum limit of the discrete time temporal ERGMs (TERGMs) introduced in Robins and Pattison (2001). The transition rates for this class are particularly natural, with the log rates being equal to the potential differences between states. Note that the potential of the equilibrium ERGM is scaled by a factor of 2 from the transition potential (as can be obtained from the `CSTERGM` by letting $q_f = q_d$); intuitively, this arises because states of higher potential are both more stable (lower exit rates) and more attractive (higher entrance rates) than states of lower potential.

Event Rate ($a \to b$): $\exp[q(b) - q(a)]$

Equilibrium: $\exp[2q(a)]/Z$

Further details on each process can be found in Butts (2023). All of the above transition rates are defined up to an arbitrary pacing constant (which is generally specified separately, and taken to be 1 in package tools if not otherwise indicated). Note that the LERGM and Change Inhibition processes have unit-maximum transition rates, and thus the pacing constant sets the maximum rate of change.

Information on functions for simulation or analysis of EGPs is provided in their respective manual pages. Information on ERGMs and their specification can be found within the `ergm` page in the `ergm` library.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, Carter T. (2023). "Continuous Time Graph Processes with Known ERGM Equilibria: Contextual Review, Extensions, and Synthesis." *Journal of Mathematical Sociology*. doi:10.1080/0022250X.2023.2180001

Carnegie, Nicole B.; Krivitsky, Pavel N.; Hunter, David R.; and Goodreau, Steven M. (2015). "An Approximation Method for Improving Dynamic Network Model Fitting." *Journal of Computational and Graphical Statistics*, 24(2):502-519.

Grazioli, Gianmarc; Yu, Yue; Unhelkar, Megha H.; Martin, Rachel W.; and Butts, Carter T. (2019). "Network-based Classification and Modeling of Amyloid Fibrils." *Journal of Physical Chemistry, B*, 123(26):5452-5462.

Koskinen, Johan H. and Snijders, Tom A. (2007). "Bayesian Inference for Dynamic Social Network Data." *Journal of Statistical Planning and Inference*, 137(12):393–3938. 5th St. Petersburg Workshop on Simulation, Part II.

Krivitsky, Pavel N. and Handcock, Mark S. (2014). "A Separable Model for Dynamic Networks." *Journal of the Royal Statistical Society, Series B,* 76(1):29-46.

Robins, Garry L. and Pattison, Philippa E. (2001). "Random Graph Models for Temporal Processes in Social Networks." *Journal of Mathematical Sociology*, 25:5-41.

Snijders, Tom A. B. (2001). "The Statistical Evaluation of Social Network Dynamics." *Sociological Methodology,* 31:361-395.

See Also

simEGP, EGPHazard, EGPRateEst, ergm, durations

`durations` *Obtain Edge Spell Durations from an ERGM Generating Process Trajectory*

Description

Given an input trajectory (in networkDynamic form, or network form with additional attributes), return the set of all edge durations (along with censoring information, if desired).

Usage

```
durations(net, censor = c("obs", "omit"), return.censoring = TRUE)
```

Arguments

`net`	a `network` or `networkDynamic` object containing the trajectory information.
`censor`	how should censoring be handled? (Currently, only returning observed spell lengths and omitting censored spells are supported.)
`return.censoring`	logical; return censoring information?

Details

This function extracts information on edge spells (periods of time in which edges are present) from the input network, and returns the spell durations (optionally, together with censoring information). The durations should not be assumed to be in any particular order; this function is generally invoked to examine duration distributions.

If `net` is a `networkDynamic` object, both spell and censoring information are extracted from its edge activities. If `net` is a `network` object, then its `"LastChangeTime"` network attribute is used to obtain spell information. (Both can be obtained from `simEGP` with appropriate settings.) Currently, `network` objects with `"EventHistory"` attributes are not supported - use the `networkDynamic` output type to examine complete event histories. For the `network` case, the observation period is

assumed to span the interval from 0 to net%n%"Time" (so be sure that temporal offsets were not used if employing that data type).

Spells may be left-censored, right-censored, or both. censor=="obs" results in lengths being reported as-is (subject to truncation to the observation period), and censor=="omit" results in censored spells being omitted. Censoring indicators are also included when return.censoring==TRUE. Note that if "LastChangeTime" information is being used, all spells are censored (we see only the onset times for edges that were present when the simulation was terminated), so the "omit" option will return a zero-length vector.

When using durations to estimate equilibrium duration distributions, it is important to bear in mind that EGP trajectories stopped by event count are not terminated at a random time, and hence will provide biased estimates. Consider using EGPRateEst to calibrate a reasonable simulation time, and sampling with a temporal stopping rule.

Value

A vector of spell durations (order not guaranteed), or a matrix containing said durations and censoring indicators (0=uncensored, 1=right-censored, 2=left-censored, and 3=interval censored).

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

simEGP

Examples

```
  #Examples are a bit slow, so not automatically run

#Generate a simple CD-CSTERGM trajectory; equilibrium mean outdegree
#is 2, dissolution rate is 1/3
set.seed(1331)
n <- 25
net <- simulate(network.initialize(n)~edges, coef=log(2/(n-3)))
traj <- simEGP(net~edges, coef=list(formation=log(2/(n-3)*1/3),
    dissolution=log(1/3)), time=5000, process="CDCSTERGM",
    return.changetime=TRUE, verbose=FALSE)
network.edgecount(traj)/(n-1)             #Mean degree apx 2
dur <- durations(traj)                    #Get durations
head(dur)                                 #All are right-censored
mean(dur[,1])                             #Apx 3 (despite censoring)

#Repeat, but now using a networkDynamic object
set.seed(1331)
net <- simulate(network.initialize(n)~edges, coef=log(2/(n-3)))
traj <- simEGP(net~edges, coef=list(formation=log(2/(n-3)*1/3),
    dissolution=log(1/3)), time=500, process="CDCSTERGM",
    return.networkDynamic=TRUE, verbose=FALSE)
slice <- traj %t% 499                     #Take a slice near the end
network.edgecount(slice)/(n-1)            #Mean degree apx 2
```

```
dur <- durations(traj)                    #Get durations
head(dur)                                 #More of a mix
mean(dur[,1])                             #Apx 3
hist(dur[,1], xlab="Time", main="Duration Distribution")  #Visualize
```

EGPHazard *Calculate Transition Hazards for an ERGM Generating Process*

Description

Given an EGP and an initial state, calculate the transition rates to one or more neighboring states.

Usage

```
EGPHazard(form, coef, toggles = NULL, rate.factor = 1, process = c("LERGM",
    "CRSAOM", "CI", "DS", "CDCSTERGM", "CFCSTERGM", "CSTERGM", "CTERGM"))
```

Arguments

`form`	an ERGM formula for the EGP (or a list with `formation` and `dissolution` formulas, for `CSTERGM` processes). The left-hand side is used as the current state when computing transition rates.
`coef`	a vector of EGP coefficients, or a list of vectors with elements `formation` and `dissolution` for `CSTERGM` and variants.
`toggles`	edge variables to evaluate; passing `"all"` or `NULL` leads to all edge variables being evaluated, `"edges"` evaluates only dissolution events, `"nulls"` evaluates only formation events, and passing a two-column matrix of IDs (tail, head order) evaluates the selected dyads.
`rate.factor`	rate or pacing factor (sets the time scale).
`process`	the ERGM generating process to use.

Details

An ERGM generating process (EGP) is a continuous time graph process with an equilibrium distribution having a known ERGM form. See `ergmgp` for an overview of EGPs, including the specifications supported here.

`EGPHazard` calculates the log transition rates (i.e., hazards) from an initial or current state (specified by the left-hand side of the input formula) to one or more target states. These states are specified by the edge variables whose states would change (often called "toggles" in ERGM nomenclature). By default, all possible transitions are evaluated; this can also be obtained by setting `toggles=="all"`. Dissolution rates for all current edges can be obtained by setting `toggles=="edges"`, and formation rates for all current nulls can be obtained by setting `toggles=="nulls"`. Otherwise, the `toggles` argument expects a two-column matrix of tail and head vertex IDs indicating the edge variables to

be evaluated. Note that only instantaneous rates from the origin state are computed; toggles are not cumulative.

EGP specifications are as per simEGP. Transition rates for all currently implemented EGPs follow the specifications of Butts (2023), with the trivial addition of a pacing constant for all families (which simply sets the timescale).

Value

a matrix containing the toggles, indicators for whether each event would have been a formation event, and the log event hazards (one row per toggle).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, Carter T. (2023). "Continuous Time Graph Processes with Known ERGM Equilibria: Contextual Review, Extensions, and Synthesis." *Journal of Mathematical Sociology*. doi:10.1080/0022250X.2023.2180001

See Also

ergmgp for information on EGPs, ergm for information on ERGM specifications, simEGP

Examples

```
#Simulate a small network with triadic dependence
n <- 25
set.seed(1331)
co <- c(log(2.5/(n-3.5)), -0.75)
net <- simulate(network.initialize(n, directed=FALSE) ~ edges + esp(0),
    coef = co)

#Compute all rates under a LERGM
lr <- EGPHazard(net ~ edges + esp(0), coef = co, process = "LERGM")
head(lr)  #Sender, receiver, formation (1=yes), log rate

#Use a toggle matrix to obtain the same outcome
lrt <- EGPHazard(net ~ edges + esp(0), coef = co, toggles = lr[,1:2],
    process = "LERGM")
all(lrt == lr)   #TRUE

#Examine edge dissolution rates
ldissr <- EGPHazard(net ~ edges + esp(0), coef = co, toggles = "edges",
    process = "LERGM")
a <- function(z){(z-min(z))/diff(range(z))}
plot(net, edge.col = hsv(a(ldissr[,4])*0.6))  #Blue=fast, red=slow
```

EGPRateEst *Estimate Event Rates for an ERGM Generating Process*

Description

Given an EGP, estimate either the expected time required for a specified number of transitions to occur, or the expected number of transitions within a specified time period.

Usage

```
EGPRateEst(formula, coef, process = c("LERGM", "CRSAOM", "CI", "DS",
    "CDCSTERGM", "CFCSTERGM", "CSTERGM", "CTERGM"), time.target = NULL,
    event.target = NULL, reps = 25, cores = 1, rate.factor = 1,
    verbose = FALSE, ...)
```

Arguments

formula	an ERGM formula for the EGP (or a list with formation and dissolution formulas, for CSTERGM processes). The left-hand side is used as the initial state.
coef	a vector of EGP coefficients, or a list of vectors with elements formation and dissolution for CSTERGM and variants.
process	the ERGM generating process to use.
time.target	if specified, the length of the time period for which trajectories should be simulated (in which case the estimand is the number of events within this period).
event.target	if specified, the number of transition events over which trajectories should be simulated (in which case the estimand is the time required for this number of events to be realized).
reps	number of replicate trajectories to use.
cores	number of cores to use for simultaneous simulation of trajectories.
rate.factor	rate or pacing factor (sets the time scale).
verbose	logical; show progress information?
...	additional arguments to simEGP.

Details

This function can be used to estimate the expected amount of time needed for a specific number of transitions to be realized (in which case event.target should be supplied) or the expected number of transition events occurring within a specified time period (in which case time.target should be supplied). Either way, one of time.target and event.target must be given. The function works by simulating reps trajectories (using simEGP) for the specified time/number of events, and returning the mean outcome (along with some other associated statistics).

A typical use case for this function is to calibrate the simulation time needed to obtain a reasonable number of transitions from some starting point (e.g., to ensure burn-in). Simply simulating a fixed number of transition events will result in a biased system state; however, one can avoid this problem

by using this function to determine the average duration needed for the desired number of events to be realized, and then using this duration as a stopping rule for subsequent simulations. Alternately, another use is to estimate the rate at which events accrue, e.g. to estimate compute time or memory requirements for a longer simulation study. Some processes are particularly prone to entering regimes in which they produce very large numbers of events per unit phenomenological time, and it can be useful to identify this issue before committing resources to simulating a long trajectory.

Note that, at present, all trajectories have the same starting point (the network on the left-hand side of the input formula). They are hence coupled by the initial condition (despite being otherwise independent). When equilibrium estimates from short sequences are desired, it may be wise to call this function more than once with different starting networks and integrate the results.

Value

A vector containing the mean outcome (time or event count), its standard error, the standard deviation of the outcome, and the number of replicates used.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, Carter T. (2023). "Continuous Time Graph Processes with Known ERGM Equilibria: Contextual Review, Extensions, and Synthesis." *Journal of Mathematical Sociology*. doi:10.1080/0022250X.2023.2180001

See Also

ergmgp for information on EGPs, ergm for information on ERGM specifications, simEGP

Examples

```
#Simulate a small network with triadic dependence
n <- 25
set.seed(1331)
co <- c(log(2.5/(n-3.5)), -0.75)
net <- simulate(network.initialize(n, directed=FALSE) ~ edges + esp(0),
    coef = co)

#Estimate the time needed for 500 events in a LERGM
etime <- EGPRateEst(net ~ edges + esp(0), coef = co, process = "LERGM",
    event.target = 500)
etime

#Estimate the mean number of events in the above time
eevents <- EGPRateEst(net ~ edges + esp(0), coef = co, process = "LERGM",
    time.target = etime[1])
eevents   #Expectation should be close to 500
```

simEGP *Simulate Trajectories from an ERGM Generating Process*

Description

Given an ergm formula, simulate trajectories from a continuous time graph process having the specified ERGM as a limiting distribution. A number of different processes are supported, and termination may be specified either by phenomenological time or event counts.

Usage

```
simEGP(form, coef, events = 1, time = NULL, rate.factor = 1,
    time.offset = 0, event.offset = 0, process = c("LERGM", "CRSAOM",
    "CI", "DS", "CDCSTERGM", "CFCSTERGM", "CSTERGM", "CTERGM"),
    use.logtime = FALSE, return.changetime = FALSE,
    changetime.offset = NULL, return.history = FALSE,
    return.networkDynamic = FALSE, verbose = TRUE,  trace.interval = 100,
    ...)

simEGPTraj(form, coef, events = 1, time = NULL, checkpoints = 1,
    rate.factor = 1, trajectories = 1, mc.cores = 1,
    log.sampling = FALSE, process = c("LERGM", "CRSAOM", "CI", "DS",
    "CDCSTERGM", "CFCSTERGM", "CSTERGM", "CTERGM"), use.logtime = FALSE,
    return.changetime = FALSE, return.history = FALSE, verbose = TRUE,
    trace.interval = 100, statsonly = FALSE, monitor = NULL)
```

Arguments

form	an ergm formula defining terms for the EGP; the left-hand side must be a network object, whose properties are used to determine the state space. For the CSTERGM process, a list containing two such formulas must be used, with named elements formation (for the formation model) and dissolution (for the dissolution model).
coef	vector of coefficients for the EGP; for the CSTERGMs, this should be a containing named elements formation and dissolution, each of which should be the coefficient vector for its respective model.
events	optionally, the number of simulated events to draw (if time==NULL); if time is specified, this is ignored.
time	optionally, the temporal length of the simulation; if not supplied, events is used instead to determine when to stop.
rate.factor	a multiplicative factor scaling the time evolution of the system; higher values correspond to faster dynamics.
time.offset	optionally, an initial "clock" offset for the start of a trajectory; this allows time 0 (the start of the simulation interval) to be set to an arbitrary time point. This is only used for book-keeping (e.g., when a trajectory is run as multiple segments),

and does not affect e.g. the meaning of the `time` argument (which is always interpreted as units after the start time).

`event.offset` optionally, an initial offset to the step or event count for the start of a trajectory (e.g., for trajectories being run in segments). As with `time.offset`, this only affects book-keeping, and has no other effect.

`process` the ERGM generating process to use (described below).

`use.logtime` logical; internally, use logarithmic timescale? This can potentially protect against overflow or underflow when rates are extreme, but can reduce precision and adds some overhead.

`return.changetime`
logical; should we return a matrix with the last update times for each edge variable as a network attribute?

`changetime.offset`
optionally, an `n` x `n` matrix of last change times (for trajectories being resumed in process).

`return.history` logical; return the entire event history as a network attribute?

`return.networkDynamic`
logical; retain the entire event history and return as a `networkDynamic` object?

`verbose` logical; provide trace messages regarding simulation progress?

`trace.interval` for verbose output, the interval at which messages should be printed (in events).

`checkpoints` number of checkpoints at which the trajectory should be sampled (in addition to the initial state).

`trajectories` number of independent trajectories to simulate (all start from the seed network, but evolve independently).

`mc.cores` number of cores to use when simulating trajectories.

`log.sampling` logical; should time points to sample be logarithmically spaced?

`statsonly` logical; should only network statistics be retained (and not the graphs themselves)?

`monitor` optionally, an `ergm` formula with additional statistics to track.

`...` additional arguments (currently unused).

Details

An ERGM generating process (EGP) is a continuous time graph process with an equilibrium distribution having a known ERGM form. See `ergmgp` for an overview of EGPs, including the specifications supported here.

`simEGP` generates a single trajectory from an EGP, with the EGP being specified via its graph potential (as a `ergm` formula or pair thereof and associated coefficients) and its initial state being given by the left-hand side of the input formula. The trajectory length can be specified either in terms of the number of transitions to be simulated (`events`) or the length of the trajectory in phenomenological time (`time`); only the latter leads to the specified ERGM equilibrium (since event times are not "random" times, stopping after a fixed number of events biases the final state). If desired for bookkeeping purposes, an offset can be added to the simulation clock (which otherwise starts at 0), event count (likewise), and most recent change times (also likewise). By default, the return value

is a network object containing the final graph state, with network attributes giving the final time ("Time"), event count ("Events"), ERGM potential ("Potential"). A square matrix containing the time of the most recent transition experienced by each edge variable can be returned as a network attribute ("LastChangeTime") if return.changetime is selected. By default, the entire event history is not stored (as it can become extremely large). However, if return.history==TRUE, a matrix containing the event history is saved and returned as a network attribute ("EventHistory"). Alternately, setting return.networkDynamic=TRUE will lead to the event history being stored and the entire trajectory being returned as a networkDynamic object, with edge activities set based on the observed transitions. This format may be easier to use for visualization, or to query the state of the network at an arbitrary point in the trajectory. The durations function can be used to extract edge durations from such objects, as well.

For models with extreme transition rates, the option use.logtime may be useful for avoiding overflow or underflow; this only affects internal calculation, and not reported event times. Note that logscale calculations add some overhead, and may be less precise in some cases than the default, so this option is not suggested unless specifically needed.

simEGPTraj is a wrapper for simEGP, which adds additional capabilities for simulation of multiple trajectories and/or sampling of longer trajectories. Each returned trajectory contains the initial state, as well as simEGP output from checkpoints points along the trajectory (including the end). The default behavior (checkpoints==1) returns the initial and final states. Checkpoints are evenly spaced (with termination criteria indicated as per simEGP) by default, or logarithmically spaced if log.sampling==TRUE. Multiple independent trajectories can be simulated by setting trajectories>1; these can be run in parallel by setting mc.cores>1. If desired, the model statistics can be returned without the graph state by choosing statsonly=TRUE, and a one-sided monitor formula can likewise be used to calculate additional statistics if desired (with similar functionality to the ergm simulate method). Otherwise, network.list objects are returned containing the states in the respective trajectories.

Simulation itself follows the discrete event approach described in Butts (2023). Transition hazards are computed for all edge variables (making the scaling no better then $O(N^2)$ for each update, and are used to draw both the next event and the event time. Because the cost of computing each transition is unrelated to waiting time, this algorithm can be quite efficient at simulating long time periods when events are sparse (unlike, e.g. a discrete-time algorithm that updates in every period). By turns, however, trajectories can become quite expensive (per unit phenomenological time) when event rates are high. This issue is especially pronounced for the CRSAOM and DS processes, which can both generate very high transition rates in some cases. Unless otherwise specified, event histories are not stored, so storage costs are by default unrelated to trajectory length. Care should be taken when storing event histories, as they can become quite large when transition rates are high.

To obtain equilibrium graph distributions from an EGP, it is generally (much) more efficient to use the simulate functions in the ergm package: they employ MCMC algorithms that are unconstrained by the need to follow realistic trajectories, and that are optimized for rapid mixing. (In particular, note that many systems can become *kinetically trapped*, spending very long periods in metastable states that are far from equilibrium. This can be a real-world phenomenon, but is not always desirable from a computational point of view. Functions such as simEGP are intended to faithfully reproduce such dynamics, while MCMC algorithms are intended to avoid them.) Comparison of late-phase draws from a simERGMPot trajectory with equilibrium ERGM draws can be used to evaluate convergence to equilibrium behavior (where desired); alternately, simEGP can be seeded with ERGM draws to follow trajectories from equilibrated states. Consult the ergm package documentation for details.

Value

For `simEGP`, a network object containing the final graph state, with network attributes `Time`, `Events`, and `Potential` listing the time, event count, and ERGM potential at the end of the simulation interval. See above for additional attributes that may be added if history retention is activated. If `return.networkDynamic==TRUE`, then the return value is instead a `networkDynamic` object containing the event history as edge activity data; be aware that an edge will exist in this object if any corresponding edge is ever active, so the raw graph state should not be used to access the final system state. Instead, use the `network.extract` method to query the network state at the desired time point.

For `simERGMPotTraj`, a list containing the simulated trajectories. These are either *network.list* objects containing the networks at each checkpoint (with time, step, and potential attribute as described above), or else matrices of trace statistics (if `statsonly==TRUE`). Note that the statistics are in any event included as an attribute to each network list, so the effect of `statsonly==TRUE` is simply not to retain the graph states.

Note

Using `steps` to control trajectory termination will lead to biased samples (sometimes severely so); this is because transitions are not random times. If your goal is to obtain equilibrium draws (or draws en route thereto), use `time` to set the stopping point. See `EGPRateEst` for a simple tool for calibrating simulation times.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, Carter T. (2023). "Continuous Time Graph Processes with Known ERGM Equilibria: Contextual Review, Extensions, and Synthesis." *Journal of Mathematical Sociology*. doi:10.1080/0022250X.2023.2180001

See Also

`ergmgp` for information on EGPs, `ergm` for information on ERGM specifications, `EGPHazard`, `EGPRateEst`, `networkDynamic`

Examples

```
#Small example of 2-ribbon generation
n<-100
set.seed(1331)
net<-network.initialize(n,directed=FALSE)
sim<-simEGP(net~edges+kstar(2)+nsp(1:2),
    coef=c(109-log(n),-25,-1.25,3.25), time=100, process="LERGM",
    verbose = TRUE)
plot(sim) #Return value is a single network

#Generate a trajectory showing the process at multiple stages
```

```
set.seed(1331)
sim<-simEGPTraj(net~edges+kstar(2)+nsp(1:2),
    coef=c(109-log(n),-25,-1.25,3.25), time=100, checkpoints = 5,
    trajectories = 2, mc.cores = 1, log.sampling = TRUE,
    process = "LERGM", verbose = TRUE)
length(sim)==2        #One entry per simulated trajectory
op<-par(mfrow=c(2,3))
for(i in 1:6)         #Show the first trajectory
  plot(sim[[1]][[i]],main=paste("Time",round(sim[[1]][[i]]%n%"Time",2)))
summary(sim[[2]]~edges+kstar(2))  #Show selected stats from the second
attributes(sim[[1]])  #Show precomputed statistics
par(op)

#A simple example with statsonly
set.seed(1331)
sim<-simEGPTraj(net~edges+esp(0), coef = c(log(2)-log(n), -1), time = 200,
    checkpoints = 25, process = "LERGM", statsonly = TRUE,
    monitor = ~triangle)
sim                   #Note the monitor stat
op<-par(mfrow=c(1,1))
plot(sim[,"Time"], sim[,"edges"], type = "l")  #Time by edge count
lines(sim[,"Time"], sim[,"esp0"], col = 2)     #Add ESP(0)s
par(op)
```

Index