Package ‘cronyNets’

September 12, 2020

Title Crony Capitalism Network Simulations

Version 1.0

Description This packages includes functions to run simulations for Crony Capitalism Paper.

Depends R (>= 3.1.0), network, sna, parallel, doParallel, foreach, iterators

ByteCompile true

BuildManual yes

LazyData true

RoxygenNote 7.1.1

License MIT (Open Source) License

Suggests testthat

Imports network, sna, parallel, doParallel, foreach, iterators

R topics documented:

`a0.Stats` *Summary statistics corresponding to an attacked firm*

Description

This function calculates various summary statistics for the first A attacked by D. This output can be useful to assess whether there are idiosyncratic factors that affect the incidence and extent of predation. For example, a firm with multiple G's will enable the spread of predation to multiple firms.

Usage

```
a0.Stats(victim, GA.net, AA, A.dat, GG, G.dat)
```

Arguments

`victim`	a firm that was initially targeted for predation
`GA.net`	affiliation network matrix of G's and A's
`AA`	a network of A's with common protectors
`A.dat`	Firm-specific data
`GG`	a network of G's with common protection duties
`G.dat`	Third-party enforcer data

Value

a vector with this information:

(1) identify FIRST victim a0 (2) a0's Rents (3) degree centrality (4) betweenness centrality (5) how many other notes are reachable? (6) No.of G protectors (7) how many other firms do related G's protect? (8) avg RHO of related G's (9) cumulative RHO of related G's (10) avg degree centrality of related G's (11) avg betweenness centrality of related G's (12) average reach of related G's

See Also

Other Simulation summary statistics: `attackSummary()`, `netStats()`, `sessionVars()`

`activateG` *Identifies a new set of (activated) G's that can impose additional penalties*

Description

This function traverses the GG network to identify G's that can assist the firms in a given vector A.victims.

Usage

```
activateG(A.victims, GA.net, protectors = c())
```

Arguments

`A.victims` is a vector of firms to be attacked

`GA.net` the affiliation network of private protection

`protectors` a vector (possibly empty) of previously activated G's

Value

a vector with new G id numbers

`attackList` *Identify victims of predation attacks (as many as the number of firms)*

Description

This function delivers either a single firm ID is attack.mode equals "single" or all firms ID's if attack.mode equals "all".

Usage

```
attackList(numA, attack.mode = "rnd", attack.N = "single", biasAttr)
```

Arguments

`numA` No. of firms

`attack.mode` "rnd"=choose victims at random;"biased"=choose victims proportional to their rents; "all": all firms will be attacked

`attack.N` 1 for as single random attack, society$numA to attack all firms

`biasAttr` A firm attribute \(e.g., rents\) for biased selection

Value

a vector of firm ID's of length equal to 1 if single="TRUE" or no. of firms \(in effect, numA separate attacks\)

Note

This function replaces pickFirm, but keep pickFirm to have the ability to select just one firm.

`attackNet` *Simulates an isolated attack on a firm A, which spreads to its neighbors*

Description

This is the main function to simulate an attack on one firm.

This function can be called multiple times within a single society attack if `societyAttack` is using the attack.N="all" option.

Usage

```
attackNet(AA.net, GA.net, A.dat, G.dat, D.dat, victim.0)
```

Arguments

`AA.net`	Network of A's that share G's
`GA.net`	Affiliation Network of G's and A's
`A.dat`	Network of A's
`G.dat`	Network of A's
`D.dat`	Network of A's
`victim.0`	number that identifies targeted A

Value

A list with these items:

a0: initial victim's ID victims: a vector of all affected firms protectors: a vector of activated G's s: the last predation step prey.gain: cumulative predation gains penalty: cumulative penalty

`attackSummary` *Summarize output from a given predation attack*

Description

This function combines parameter information from a society object and information from a particular attack.

Some of the variable names are repeated so the attackNet function can be tested separately without calling the whole society object

Usage

```
attackSummary(society, attack.dat)
```

Arguments

`society`	output of SocietySetup
`attack.dat`	output of attackNet

Value

Summarize output from a given attack

See Also

Other Simulation summary statistics: `a0.Stats()`, `netStats()`, `sessionVars()`

`A_Data` *Collect data to describe all A's*

Description

This function creates a data frame with a variety of firm-level attributes such as rent and the number of associated G's.

Usage

```
A_Data(Cd, exog, society.constraints, GA.mat)
```

Arguments

`Cd`	D's incumbency cost
`exog`	exogenous rent generation flag
`society.constraints`	societal parameters to induce participation
`GA.mat`	affiliation matrix (Gs: rows, As: columns)

Value

A data frame with NINE columns (1) id: a firm number (2) R: rents (3) R.base: minimum individual rent to induce G, D to participate (4) v: reservation value (as (5) t: tax rate (paid to Cd) (6) b: total private protection fees (paid to linked G's) (7) I: equals 1 if firm wants to invest (8) Gs: Number of (hired) private enforcers (9) prey: equals 1 if D prays on firm "id"

See Also

Other participant data creation functions: `D_Data()`, `G_Data()`, `societalConstraints()`

cmp_centralization *Find the Centralization of a Given Network, for Some Measure of Centrality*

Description

Find the Centralization of a Given Network, for Some Measure of Centrality

Usage

```
cmp_centralization(
  dat,
  FUN,
  g = NULL,
  mode = "digraph",
  diag = FALSE,
  normalize = TRUE,
  ...
)
```

Arguments

`dat` one or more input graphs.

`FUN` Function to return nodal centrality scores.

`g` Integer indicating the index of the graph for which centralization should be computed. By default, all graphs are employed.

`mode` String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `mode` is set to "digraph" by default.

`diag` Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.

`normalize` Boolean indicating whether or not the centralization score should be normalized to the theoretical maximum. (Note that this function relies on `FUN` to return this value when called with `tmaxdev==TRUE`.) By default, `tmaxdev==TRUE`.

`...` Additional arguments to `FUN`.

See Also

`centralization`

`cmp_component.dist` *Calculate the Component Size Distribution of a Graph*

Description

Calculate the Component Size Distribution of a Graph

Usage

```
cmp_component.dist(
  dat,
  connected = c("strong", "weak", "unilateral", "recursive")
)
```

Arguments

`dat`	one or more input graphs.
`connected`	a string selecting strong, weak, unilateral or recursively connected components; by default, `"strong"` components are used.

See Also

`component.dist`

`cmp_is.connected` *Is a Given Graph Connected?*

Description

Is a Given Graph Connected?

Usage

```
cmp_is.connected(g, connected = "strong", comp.dist.precomp = NULL)
```

Arguments

`g`	one or more input graphs.
`connected`	definition of connectedness to use; must be one of `"strong"`, `"weak"`, `"unilateral"`, or `"recursive"`.
`comp.dist.precomp`	a `component.dist` object precomputed for the graph to be analyzed (optional).

See Also

`sna::is.connected`

cmp_reachability *Find the Reachability Matrix of a Graph*

Description

Find the Reachability Matrix of a Graph

Usage

```
cmp_reachability(
  dat,
  geodist.precomp = NULL,
  return.as.edgelist = FALSE,
  na.omit = TRUE
)
```

Arguments

`dat`	one or more graphs (directed or otherwise).
`geodist.precomp`	optionally, a precomputed `geodist` object.
`return.as.edgelist`	logical; return the result as an sna edgelist?
`na.omit`	logical; omit missing edges when computing reach?

See Also

`reachability`

compileFunction *Creates byte-compiled version of a function*

Description

This function is used to create compiled versions of R functions, for possible substitution if they can run faster than original functions.

Usage

```
compileFunction(fun)
```

Arguments

`fun:`	a function

Value

A byte compiled version of ’fun’ function

Note

Used to bytecompile some sna functionst that take a lot of time

createNetworks — *Create affiliation (two-mode) and one-mode networks to link A's and G's*

Description

This is a wrapper function that creates all the embedded networks in a society object GA, AA, and GG) by passing required inputs to the functions `link{networkAA}`, `link{networkAA}`, and `link{networkAA}`, respectively.

Usage

```
createNetworks(numA, numG, ptie.max, ptie.mode)
```

Arguments

`numA`	No. of firms
`numG`	No. of third-party enforcers
`ptie.max`	(maximum) probability of a tie between G and A
`ptie.mode`	equal tie probabilities ('rnd') or 'biased' net formation?

Value

A list with three network structures and a dummy variable (i) An affiliation matrix linking G's to A's (ii) A one-mode network of A's (iii) A one-mode network of G's (iv) empty.net=1 if there are no links between G's and A's

cronyNets — *Simulate predation impact of GA network structure(s)*

Description

This is the main cronyNets function to simulate a predation attack and return simulation results.

The end user does not need to use any other functions, except to generate a parameter GRID using `societyParamsGrid`.

Usage

```
cronyNets(params.GRID, REPS = 1)
```

Arguments

`params.GRID`	Data frame with parametric configurations for societySetup
`REPS`	number of repetitions per societal configuration

Value

A data frame with simulation results for REPS repetitions for each params.GRID configuration

Note

A wrapper function to run various configurations

Examples

```
## Not run:
sim1 <- cronyNets(params.GRID, reps = 50)

## End(Not run)
```

---

cronysimple *A simpler version of cronyNets that takes three basic inputs*

Description

This is a simpler version of cronyNets, which does not require a parameter vector. The user only needs to provide numbers for A's and G's and a probability of G<->A ties.

Missing societal parameters are supplied automatically using default options in societyParamsGrid

Usage

```
cronysimple(numA, numG, ptie.max)
```

Arguments

numA	No. of firms
numG	No. of third-party enforcers
ptie.max	Probability of a tie between G's and A's

Value

a data matrix with simulation results

Examples

```
## Not run:
sim1 <- cronysimple(100,50,0.05)  # 100 A's, 50 G's, Pro(G<->A) = 0.05

## End(Not run)
```

denseGA *Calculate density for given affiliation matrix or two-mode network object*

Description

This is a generic function that takes either a matrix or network object and calculates the corresponding network density.

Especially when a matrix GA is available, this function is mean to avoid expensive calls to network.density(as.network(GA)).

Usage

```
denseGA(GA)
```

Arguments

`GA.mat`	A two-mode affiliation matrix or affiliation network object

Value

A number between 0 and 1, denoting density of two-mode connections

D_Data *Collect data to describe D*

Description

This function creates a list with government-specific attributes.

At this point, these variables are immutable, but in future development, these variables could be updated (e.g., prey.gain)

Usage

```
D_Data(Cd, A.dat)
```

Arguments

`Cd:`	D's incumbency cost
`A.dat:`	a data frame with firm-level information, esp. paid tax rates and individual rents

Value

a data frame with SIX columns (1) Cd: D's imcumbency cost (2) t: average "tax rate" charged to A's (3) tR: total revenue from all firms (4) promise: equals 1 if D promises protection (Cd is covered) (5) prey.gain: initialized at zero (before game begins) (6) prey.loss: initialized at zero (before game begins)

See Also

Other participant data creation functions: `A_Data()`, `G_Data()`, `societalConstraints()`

`G_Data` *Collect data to describe all G’s*

Description

This function creates a data frame with various attributes such as a protector’s own capacity to punish the dictator, the number of protected A’s, etc.

Usage

```
G_Data(rho, exog, society.constraints, GA.mat, A.dat)
```

Arguments

`society.constraints`
societal parameters to induce participation

`rho:` individual punishment capacity

`exog:` exogenous rent generation

`GA.mat:` affiliation matrix (Gs: rows, As: columns)

`A.dat:` firm-level data

Value

a data frame with FIVE columns (1) id: G’s "name" (2) rho: private punishment capacity (3) As: number of protected firms (4) bR: private protection payoffs (5) enforce: equals 1 if G is willing to enforce

See Also

Other participant data creation functions: `A_Data()`, `D_Data()`, `societalConstraints()`

`matDensity` *Calculate density for a sociomatrix*

Description

This function takes a one-mode (symmetric) matrix representing a network and calculates the density function (ratio of actual ties to max number of possible ties).

This function is meant to quickly calculate density without having to call `network.density`.

Usage

```
matDensity(mat.net)
```

Arguments

`mat.net` A square matrix representing an undirected one-mode network

Value

A number between 0 and 1, denoting density of one-mode network

Note

Added this function to speed up netClustering function, which had made expensive calls to as.network of extracted ego networks

---

`matrixGA` *Derive random GA matrix with density equal to ptie.max*

Description

This function creates a rectangular affiliation matrix linking G's (rows) and A's (columns). Each cell in this matrix is equal to 1 if the corresponding (row) G protects a given (column) A. These cell values are determined according to ptie.mode. If ptie.mode="indep", then each GA[i,j] is generated separately with probability ptie.max. If ptie.mode="density", then this function samples from the conditional distribution of GA random matrices with density equal to ptie.max in one of two ways, depending on the argument ptie.mode. For example, if numG=10 and numA=10, then ptie.max=0.1 and ptie.mode="density" will sample from GA matrices that have 10 ties, the cells are therefore dependent in order to ensure that the overall density equals ptie.max.

Usage

```
matrixGA(numA, numG, ptie.max, ptie.mode = "density")
```

Arguments

`numA`	No. of firms
`numG`	No. of third-party enforcers
`ptie.max`	(maximum) probability of a tie between G and A
`ptie.mode`	equal tie probabilities ('indep') or ('density') net formation?

Value

A rectangular GA affiliation matrix where GA[i, j]=1 if G_i protects A_j

---

`netClustering` *Calculate clustering coefficient for one-mode network*

Description

This is an own function to calculate the clustering coefficient as the "...the tendency towards dense local neighborhoods..." as explained by Hanneman in `http://www.faculty.ucr.edu/~hanneman/nettext/C8_Embedding.html#transitivity`. See notes.

Usage

```
netClustering(net, type = "overall")
```

Arguments

`net`	a network object
`type`	calculation option in c("overall", "weighted")

Value

A clustering coefficient between 0 and 1

Note

Based on average density of ego neighborhoods Source: `http://www.faculty.ucr.edu/~hanneman/nettext/C8_Embedding.html#transitivity` "...the tendency towards dense local neighborhoods, or what is now thought of as "clustering.

One common way of measuring the extent to which a graph displays clustering is to examine the local neighborhood of an actor (that is, all the actors who are directly connected to ego), and to calculate the density in this neighborhood (leaving out ego). After doing this for all actors in the whole network, we can characterize the degree of clustering as an average of all the neighborhoods."

`netStats` *Calculates various network-centric statistics*

Description

This function calculates 10 network-analytic variables for given network.

Among these variables, the function delivers density, centralization, number of components, etc.

Usage

```
netStats(net)
```

Arguments

`net:`	a network object

Value

A vector with selected network-centric statistics

(1) n: network size (2) d: density (3) c.deg: sna::centralization using degree (4) c.bet: sna::centralization using betweenness (5) connected: Dummy variable for connected networks (6) comp: Number of components (7) comp.large: Size of largest component (8) clustering: clustering coefficient (9) trans: Transitivity score (10) avgReach: average of node-level reachability

See Also

Other Simulation summary statistics: `a0.Stats()`, `attackSummary()`, `sessionVars()`

`networkAA` *Derive A to A projection network from affiliation matrix*

Description

This function takes a rectangular GA matrix, and creates a network object out of GA' * GA

Usage

```
networkAA(GA.mat)
```

Arguments

`GA.mat`	A two-mode affiliation matrix

Value

A 'network' object with connections among A's

`networkGA` *Convert a GA rectangular matrix into a network object*

Description

This function takes a rectangular GA matrix and creates a bipartite network object with the first mode equal to the number of rows in GA.

Usage

```
networkGA(GA.mat)
```

Arguments

`GA.mat`	A two-mode affiliation matrix

Value

A 'network' object

networkGG *Derive G to G projection network from affiliation matrix*

Description

This function takes a rectangular GA matrix, and creates a network object out of GA * GA'

Usage

```
networkGG(GA.mat)
```

Arguments

`GA.mat`	A two-mode affiliation matrix

Value

A 'network' object with connections among G's

newPenalty *Calculate cumulative penalty from activated G's*

Description

This function calcualtes the collective penalty that can be imposed by a set of activated G's.

Usage

```
newPenalty(activatedG, G.dat)
```

Arguments

`activatedG`	a vector of G id numbers
`G.dat`	a data frame with information on G's

Value

a non-negative cumulative penalty

newRents *Calculate rents from the next set of (networked) victims*

Description

This function takes a set of firm ID's and calculates their collective rents.

Usage

```
newRents(newVictims, A.dat)
```

Arguments

A.dat	a matrix with firm-level data, including rents
victims	a (possibly empty) set of preyed upon firms

Value

The sum of victims' available rents

newSociety *Create a society object to simulate predation attacks*

Description

This function creates an S3 'society' class, a list with various societal-level components such as the associated GA network, the derived AA network that ties A's with common protectors, the derived overlapping protection GG network that links G's. This class also contains node-specific attributes such as a firm's rents and a protector's rho (penalty) capacity.

Usage

```
newSociety(
  numA,
  numG,
  ptie.max,
  Cd = 100,
  rho = 1,
  attack.mode = "rnd",
  attack.N = "single",
  ptie.mode = "density",
  exog = 1
)
```

Arguments

`numA`	No. of firms
`numG`	No. of third-party enforcers
`ptie.max`	(maximum) probability of a tie between G and A
`Cd`	Dictator's incumbency cost
`rho`	baseline strength for a given G (penalty per firm)
`attack.mode`	in c("rnd", "biased") specifies how to select potential predation victims
`attack.N`	in c("single", "all") specifies how many firms to use for attack summary
`ptie.mode`	equal tie probabilities ('rnd') or ('biased') net formation?
`exog`	dummy variable to account for exogenous or endogenous rents

Value

A society list, which includes: (i) three network structures for GA affiliation network, related A's (AA), related G's (GG) (ii) three data tables for actors A, G, and D (iii) four original parameters about tie probabilities and attack type

Examples

```
# set up a society with 20 asset holders and 10 private enforcers with prob of GA ties to 0.01
#' mySociety <- newSociety(numA=20, numG=10, ptie.max=0.01)
# set up a society with 20 asset holders and 10 private enforcers with prob of GA ties to 0.01
# plus other specified arguments
## Not run:
mySociety <- newSociety(numA=20, numG=10, ptie.max=0.01, Cd=100,rho=1,
                        attack.mode="rnd",attack.n="single", ptie.mode="indep",exog=0)

## End(Not run)
```

`nextVictims` *Identifies set of new firms to attack*

Description

This function takes a given firm as a starting point to traverse the network and identify all connected firms within a certain neighborhood of size *step*.

Usage

```
nextVictims(victim.0, step, AA.net)
```

Arguments

`victim.0`	is the numerical ID of first attacked firm
`step`	An integer > 1, the neighborhood size for subsequent predation
`AA.net`	is the one-mode network of related firms

pickFirm *Pick a firm to be attacked by D*

Description

Pick a firm to be attacked by D

Usage

```
pickFirm(numA, biasAttr, attack.mode = "rnd")
```

Arguments

`numA`	No. of firms
`biasAttr`	Firm-level attribute to determine proportational selection probabilities
`attack.mode`	"rnd": pick at random // "biased": pick proportional to biasAttr

protectedFirms *Identify firms protected by a set of G protectors*

Description

This function traverses the network to find all firms that are protected by a set of G protectors.

Usage

```
protectedFirms(protectors, GA.net)
```

Arguments

`protectors`	A vector of IDs for active G's
`GA.net`	An affiliated network bewteen G's and A's

Value

a vector with related firm IDs

reachedNodes *Count nodes reached from nodeset by traversing net*

Description

This function is used by `a0.Stats` to take a given set of nodes and calculate their /strongcollective reach to other nodes in a given network /emphnet.

Usage

```
reachedNodes(nodeset, net)
```

Arguments

`nodeset`	a vector of nodes
`net`	a square network matrix

Value

An integer that counts reached nodes

sessionVars *Wrapper function to define variable names for a simulation run*

Description

This function provides a one-stop collection of all variables that are collected from a single attack through `attackNet`

Usage

```
sessionVars()
```

Value

a vector of variable names to match output from `attackSummary`.

See Also

Other Simulation summary statistics: `a0.Stats()`, `attackSummary()`, `netStats()`

`societalConstraints` *Establish societal parameters (taxes, etc.) to induce players to participate*

Description

This function calculates values that ensure that this society has the right parameters for all actors to participate, and thus run a simulation.

Usage

```
societalConstraints(Cd, numG)
```

Arguments

`Cd`	D's incumbency cost
`numG`	No. of third-party enforcers

Value

four societal parameters that ensure participation constraints

See Also

Other participant data creation functions: `A_Data()`, `D_Data()`, `G_Data()`

`societyAttack` *Simulates the impact of a predation attack*

Description

Simulates the impact of a predation attack on selected firms, which are chosen according to the attack.N attribute of society, which can be either "single" (for a random attack on a single A) or "all" (to attack all firms), which averages the results of multiple attacks.

This function is the main output for each simulation run created by `cronyNets` or `cronysimple`.

Usage

```
societyAttack(society)
```

Arguments

`society`	A society (list) object, which includes an attack list of size numA

Value

An attackSummary for a given list of firms to attack (averages if attack.N="all")

societyAttackMC *Simulates the impact of a predation attack using DoMC parallel backend*

Description

Simulates the impact of a predation attack on selected firms, which are chosen according to the attack.N attribute of society, which can be either "single" (for a random attack on a single A) or "all" (to attack all firms), which averages the results of multiple attacks.

Usage

```
societyAttackMC(society)
```

Arguments

`society`	A society (list) object, which includes an attack list of size numA

Value

An "average" of attackSummary for a given list of firms to attack

Note

Adds a

societyParamsGrid *Create multiple societal configurations for predation simulations*

Description

This function takes possible values for 9 configuration variable values to construct a data frame with all value combinations.

Usage

```
societyParamsGrid(
  numA,
  numG,
  ptie.max,
  Cd = 100,
  rho.base = 1,
  attack.mode = "rnd",
  attack.N = "single",
  ptie.mode = "rnd",
  exog = 1,
  saveFile = "simParams"
)
```

Arguments

`numA` num of A's (asset holders), either a single number or a vector of numbers

`numG` num of G's (private enforcers), either a single number or a vector of numbers

`ptie.max` probability of a tie between one A & one G, either a single number or a vector of numbers

`Cd` D's cost of staying in office, either a single number or a vector of numbers

`rho.base` baseline (per firm) private penalty imposed by each G, either a single number or a vector of numbers

`attack.mode` a value in c("rnd", "biased"), used to specify how predation victims will be selected

`attack.N` in c("single", "all") to denote how many firms to attack for simulated society

`ptie.mode` any or all values c("density", "indep"), used to determine sampling scheme for GA network formation process

`exog` any or all values in c(0,1): exog=1 if rent (and rho distributions) are exogenous, used as a flag variable to trigger different society setups

`saveFile` a string "fileName" to save results as "fileName.RData"

Value

A data frame with all combinations of given configuration variables

Examples

```
## Not run:
grid <- societyParamsGrid(numA=c(25,50), # will try societies with 25 and 50 asset holders
              numG=10,                        # but a fixed number of G's: 50
          ptie.max=seq(0.01,0.25,0.01),  # variable probabilities of ties between A's & G's
              Cd=100,                         # a fixed cost of running the government
              rho.base=0.1,                   # a fixed rho
          attack.mode=c("rnd", "biased"),# D will attack both in random and biased fashion
              attack.N=c("single", "all"),    # D will attack single and all firms
              ptie.mode=c("indep"),           # independent GA[i,j] probabilities
           exog=c(0,1),                # will try both exogenous and endogenous rent setup
           saveFile="simGrid")         # save grid as "simGrid.RData" for future retrieval

## End(Not run)
```

Index