Package 'remstats'

May 15, 2024

Type Package

Title Computes Statistics for Relational Event History Data

Version 3.2.2

Date 2024-05-14

Maintainer Giuseppe Arena <g.arena@tilburguniversity.edu>

Description Computes a variety of statistics for relational event models. Relational event models enable researchers to investigate both exogenous and endogenous factors influencing the evolution of a time-ordered sequence of events. These models are categorized into tie-oriented models (Butts, C., 2008, <doi:10.1111/j.1467-9531.2008.00203.x>), where the probability of a dyad interacting next is modeled in a single step, and actor-oriented models (Stadtfeld, C., & Block, P., 2017, <doi:10.15195/v4.a14>), which first model the probability of a sender initiating an interaction and subsequently the probability of the sender's choice of receiver. The package is designed to compute a variety of statistics that summarize exogenous and endogenous influences on the event stream for both types of models.

License MIT + file LICENSE

URL https://tilburgnetworkgroup.github.io/remstats/

BugReports https://github.com/TilburgNetworkGroup/remstats/issues

LazyData true

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.8.3), stats, graphics, grDevices

LinkingTo Rcpp, RcppArmadillo, RcppProgress

Suggests tinytest, remify (>= 3.2.0), knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Giuseppe Arena [aut, cre] (<https://orcid.org/0000-0001-5204-3326>),
Marlyne Meijerink-Bosman [aut],
Diana Karimova [ctb],

Rumana Lakdawala [ctb],
Mahdi Shafiee Kamalabad [ctb],
Fabio Generoso Vieira [ctb],
Roger Leenders [ctb],
Joris Mulder [ctb]

Repository CRAN

Date/Publication 2024-05-15 09:10:03 UTC

R topics documented:

`actor_effects` *actor_effects*

Description

Overview of statistics in the actor-oriented model, see Details.

Usage

```
actor_effects(step = NULL)
```

Arguments

`step`	outputs all statistics in the sender activity step (if 'step = sender') or receiver choice step (if 'step = receiver').

Details

Overview of statistics in the actor-oriented model.

A list of available effects and their corresponding statistics for the *sender activity rate* step:

- baseline()
- send()
- indegreeSender()
- outdegreeSender()
- totaldegreeSender()
- recencySendSender()
- recencyReceiveSender()
- psABA()
- psABB()
- psABX()

A list of available effects and their corresponding statistics for the *receiver choice* step:

- receive()
- tie()
- same()
- difference()
- average()
- indegreeReceiver()
- outdegreeReceiver()
- totaldegreeReceiver()
- inertia()
- reciprocity()
- otp()
- itp()
- osp()
- isp()
- rrankSend()
- rrankReceive()
- recencySendReceiver()
- recencyReceiveReceiver()
- recencyContinue()
- psABAB()
- psABBA()
- psABXA()
- psABXB()
- psABAY()
- psABBY()
- psABXY()

Value

Returns a list of available effects and their corresponding statistics based on the specified ‘step‘ (sender or receiver).

Examples

```
# List of available effects for both the sender and receiver step
actor_effects()

# List of available effects for the sender step
actor_effects(step = "sender")

# List of available effects for the receiver step
actor_effects(step = "receiver")
```

---

aomstats *aomstats*

Description

Computes statistics for the sender activity rate step and receiver choice step in actor-oriented relational event models (e.g., see Stadtfeld & Block, 2017).

Usage

```
aomstats(
  reh,
  sender_effects = NULL,
  receiver_effects = NULL,
  attr_actors = NULL,
  attr_dyads = NULL,
  method = c("pt", "pe"),
  memory = c("full", "window", "decay", "interval"),
  memory_value = Inf,
  start = 1,
  stop = Inf,
  display_progress = FALSE,
  attr_data,
  attributes,
  edgelist
)
```

Arguments

reh	an object of class "remify" characterizing the relational event history.

`sender_effects` an object of class "formula" (or one that can be coerced to that class): a symbolic description of the effects in the sender activity rate step of the actor-oriented model for which statistics are computed, see 'Details'

`receiver_effects` an object of class "formula" (or one that can be coerced to that class): a symbolic description of the effects in the receiver choice step of model for which statistics are computed, see 'Details'

`attr_actors` optionally, an object of class "data.frame" that contains exogenous attributes for actors (see Details).

`attr_dyads` optionally, an object of class `data.frame` or `matrix` containing attribute information for dyads (see Details).

`method` Specifies the method for managing simultaneous events, i.e., events occurring at the same time. The default 'method' is 'pt' (per timepoint), where statistics are computed once for each unique timepoint in the edgelist. Alternatively, you can choose 'pe' (per event), where statistics are computed once for each unique event observed in the edgelist.

`memory` The memory to be used. See 'Details'.

`memory_value` Numeric value indicating the memory parameter. See 'Details'.

`start` an optional integer value, specifying the index of the first time or event in the relational event history for which statistics must be computed (see 'Details')

`stop` an optional integer value, specifying the index of the last time or event in the relational event history for which statistics must be computed (see 'Details')

`display_progress` should a progress bar for the computation of the endogenous statistics be shown (TRUE) or not (FALSE)?

`attr_data` deprecated, please use "attr_actors" instead

`attributes` deprecated, please use "attr_data" instead

`edgelist` deprecated, please use "reh" instead

Value

An object of class 'aomstats'. List with in the first element the statistics for the sender activity rate step and in the second element the statistics for the receiver choice step. The 'aomstats' object has the following attributes:

`model` Type of model that is estimated.

`formula` Model formula(s), obtained from the formula(s) inputted to 'sender_effects' and/or 'receiver_effects'.

`actors` The set of actors used to construct the statistics, obtained from the remify object inputted to 'reh'.

Effects

The statistics to be computed are defined symbolically and should be supplied to the `sender_effects` and/or `receiver_effects` arguments in the form `~ effects`. The terms are separated by + operators. For example: `receiver_effects = ~ inertia() + otp()`. Interactions between two effects

can be included with * or : operators. For example: `receivereffects = ~ inertia():otp()`. A list of available effects can be obtained with `actor_effects()`.

The majority of the statistics can be scaled in some way, see the documentation of the `scaling` argument in the separate effect functions for more information on this.

attr_actors

For the computation of the *exogenous* statistics an attributes object with the exogenous covariate information has to be supplied to the `attr_actors` argument in either `remstats()` or in the separate effect functions supplied to the `..._effects` arguments (e.g., see `send`). This `attr_actors` object should be constructed as follows: A dataframe with rows referring to the attribute value of actor *i* at timepoint *t*. A 'name' column is required that contains the actor name (corresponding to the actor names in the relational event history). A 'time' column is required that contains the time when attributes change (set to zero if none of the attributes vary over time). Subsequent columns contain the attributes that are called in the specifications of exogenous statistics (column name corresponding to the string supplied to the `variable` argument in the effect function). Note that the procedure for the exogenous effects 'tie' and 'event' deviates from this, here the exogenous covariate information has to be specified in a different way, see `tie` and `event`.

attr_dyads

For the computation of the *dyad exogenous* statistics with `tie()`, an attributes object with the exogenous covariates information per dyad has to be supplied. This is a `data.frame` or `matrix` containing attribute information for dyads. If `attr_dyads` is a `data.frame`, the first two columns should represent "actor1" and "actor2" (for directed events, "actor1" corresponds to the sender, and "actor2" corresponds to the receiver). Additional columns can represent dyads' exogenous attributes. If attributes vary over time, include a column named "time". If `attr_dyads` is a `matrix`, the rows correspond to "actor1", columns to "actor2", and cells contain dyads' exogenous attributes.

Memory

The default 'memory' setting is '"full"', which implies that at each time point t the entire event history before t is included in the computation of the statistics. Alternatively, when 'memory' is set to '"window"', only the past event history within a given time window is considered (see Mulders & Leenders, 2019). This length of this time window is set by the 'memory_value' parameter. For example, when 'memory_value = 100' and 'memory = "window"', at time point t only the past events that happened at most 100 time units ago are included in the computation of the statistics. A third option is to set 'memory' to '"interval"'. In this case, the past event history within a given time interval is considered. For example, when '"memory_value" = c(50, 100)' and 'memory = "window"', at time point t only the past events that happened between 50 and 100 time units ago are included in the computation of the statistics. Finally, the fourth option is to set 'memory' to '"decay"'. In this case, the weight of the past event in the computation of the statistics depend on the elapsed time between t and the past event. This weight is determined based on an exponential decay function with half-life parameter 'memory_value' (see Brandes et al., 2009).

Event weights

Note that if the relational event history contains a column that is named "weight", it is assumed that these affect the endogenous statistics. These affect the computation of all endogenous statistics

with a few exceptions that follow logically from their definition (e.g., the recenyContinue statistic does depend on time since the event and not on event weights).

Subset of the relational event history

Optionally, statistics can be computed for a slice of the relational event sequence - but based on the entire history. This is achieved by setting the start and stop values equal to the index of the first and last event for which statistics are requested. For example, start = 5 and stop = 5 computes the statistics for only the 5th event in the relational event sequence, based on the history that consists of events 1-4.

References

Stadtfeld, C., & Block, P. (2017). Interactions, actors, and time: Dynamic network actor models for relational events. Sociological Science, 4, 318–352. doi:10.15195/v4.a14

Examples

```
library(remstats)

# Load the data
data(history)
data(info)

# Prepare the data
reh <- remify::remify(edgelist = history, model = "actor")

# Define the sender effects
seff <- ~ send("extraversion")

# Define the receiver_effects
reff <- ~ receive("agreeableness") + inertia() + otp()

# Compute the statistics
aomstats(
  reh = reh, sender_effects = seff, receiver_effects = reff,
  attr_actors = info
)
```

---

average *average*

Description

Specifies the statistic for an "average" effect in the tie-oriented model or the receiver choice step of the actor-oriented model. An "average" effect refers to an exogenous actor attribute that affects dyad *(i,j)*'s rate of interacting (tie-oriented model) or actor *j*'s probability of being chosen as a receiver for the event send by the active sender *i* at time *t* (actor-oriented model) based on the average of the values of actors *i* and *j* on this attribute.

Usage

```
average(variable, attr_actors = NULL, scaling = c("none", "std"), attr_data)
```

Arguments

`variable`	string with the name of the column in the `attr_actors` object for which the statistic has to be computed.
`attr_actors`	optionally, an object of class `data.frame` that contains the attribute, see 'Details.'
`scaling`	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
`attr_data`	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* for dyad *(i,j)* is equal to the average of the values of actor *i* and *j* on the attribute at timepoint *t*.

Construct the 'attr_actors' object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the 'difference()' function.

Note that it is possible to omit the 'attr_actors' object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ average("extraversion")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects, attr_actors = info)
```

baseline *baseline*

Description

Specifies an intercept for the tie-oriented model or the sender activity rate step of the actor-oriented model in the same manner as in `lm` (see Details).

Details

A baseline effect is automatically specified for the tie-oriented model and the sender activity rate step of the actor-oriented model when the `ordinal` argument in `remstats`, `tomstats`, `aomstats` is set to FALSE (default) and automatically removed when this argument is set to TRUE. Alternatively, a baseline effect can be explicitly specified by adding '1' to the equation or explicitly removed by adding '-1' to the equation.

The baseline effect refers to the baseline tendency to interact. In the tie-oriented model, the log-inverse of the estimated parameter translates to the average number of observed events per time unit per dyad. In the actor-oriented model, the log-inverse of the estimated parameter translates to the average number of observed events per time unit per actor. The statistic is equal to one for all dyads resp. actors in the riskset at all timepoints.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
remstats(reh = reh_tie, tie_effects = ~1)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = ~1)
```

bind_remstats *Combine two or more remstats objects*

Description

Function to bind any number of remstats objects into one while duplicated statistics in the combined object are removed based on their name.

Usage

```
bind_remstats(...)
```

Arguments

`...`	Any number of `remstats` objects. All the `remstats` objects must have matching dimensions, except for the third dimension. Note that duplicated statistics in the combined object are removed based on their name.

Value

`statistics` array with the combined statistics, where rows refer to time points, columns refer to potential relational event (i.e., potential edges) in the risk set and slices refer to statistics

Examples

```
library(remstats)

# Load the data
data(history)
data(info)

# Prepare the data
reh <- remify::remify(edgelist = history, model = "actor")

# Obtain two different statistics objects
effects1 <- ~ inertia():receive("extraversion") + otp()
stats1 <- remstats(receiver_effects = effects1, reh = reh, attr_actors = info)
effects2 <- ~ reciprocity()
stats2 <- remstats(receiver_effects = effects2, reh = reh, attr_actors = info)

# Bind the two statistics objects
statsC <- bind_remstats(stats1, stats2)
```

---

both_male_long *Exogenous Dyad Attribute in Long Format: both_male_long*

Description

A data frame representing exogenous attributes of dyads in a social network in long format. Each row indicates whether a dyad consists of two male actors (sex=0) in the original matrix 'info_both_male_wide'.

Usage

```
data(both_male_long)
```

Format

A data frame with the following columns:

actor1 Numeric id of the first actor in the dyad.

actor2 Numeric id of the second actor in the dyad.

both_male Binary indicator (1 for male-male dyads, 0 otherwise).

Source

Simulated exogenous information on actors in a social network.

See Also

tie for the function using this data, both_male_wide for the data in wide format, and info for an overview of the actor exogenous attributes.

Examples

```
data(both_male_long)
head(both_male_long)
```

---

both_male_wide *Exogenous Dyad Attribute Matrix: both_male_wide*

Description

A matrix representing exogenous attributes of dyads in a social network. The matrix indicates whether a dyad consists of two male actors (sex=0). Rows and columns correspond to actor IDs, and cells contain binary values (1 for male-male dyads, 0 otherwise).

Usage

```
data(both_male_wide)
```

Format

A square matrix with dimensions equal to the number of unique actors.

Source

Simulated exogenous information on actors in a social network.

See Also

tie for the function using this data, both_male_long for the data in long format, and info for an overview of the actor exogenous attributes.

Examples

```
data(both_male_wide)
print(both_male_wide)
```

`boxplot.aomstats` *Plotting Relational Event Network Statistics*

Description

Generate boxplots for a specified effect in a `aomstats` object.

Usage

```
## S3 method for class 'aomstats'
boxplot(
  x,
  effect,
  model,
  by = "timepoints",
  subset = NULL,
  outliers = TRUE,
  ...
)
```

Arguments

`x`	An object of class `aomstats` containing relational event network statistics.
`effect`	A string specifying the name of the effect in 'x' or an integer indicating the index of the effect to be plotted.
`model`	A string indicating whether the effect is in the 'sender' model or the 'receiver' model.
`by`	A string indicating whether the statistic is plotted across 'timepoints' (default) or 'actors'.
`subset`	An optional vector specifying a subset of timepoints or actors to be used for plotting. Per default, a maximum of 20 unique timepoints or actors are plotted.
`outliers`	A logical value specifying whether to include outliers in the plot.
`...`	additional arguments passed to bxp().

Details

This function produces boxplots to visually represent the distribution of a specified effect in a relational event network, as captured by a `aomstats` object. The 'effect' parameter allows the user to choose a specific effect for visualization, either by providing the effect's name or its index within the 'aomstats' object. The 'model' parameter indicates whether the respective effect is in the 'sender' model or the 'receiver' model. The 'by' parameter determines whether the boxplots are created across different 'timepoints' or 'actors'. At the moment, by 'actors' is only supported for the sender model. Additionally, an optional 'subset' parameter allows the user to focus on specific timepoints or actors. If 'subset' is not specified, a default maximum of 20 unique timepoints or actors are plotted. The 'outliers' argument, when set to TRUE, includes the representation of outliers in the boxplots. If set to FALSE, outliers are omitted from the visualization.

The boxplots are based on the following summary statistics of the data: The box in the middle represents the interquartile range (IQR) between the first (Q1) and third quartile (Q3), and the line inside the box represents the median. The whiskers extend from the box to the minimum and maximum values within 1.5 times the IQR below Q1 or above Q3. Outliers beyond the whiskers are plotted individually.

Examples

```
library(remstats)
# Load data
data(history)
# Prepare data
reh <- remify::remify(edgelist = history[,1:3], model = "actor")
# Compute effects
stats <- remstats(reh, sender_effects = ~ outdegreeSender())
# Plot the 'outdegreeSender' distribution for 20 timepoints
boxplot(stats, effect = "outdegreeSender", model = "sender")
# Plot the 'inertia' distribution for all 10 actors
boxplot(stats, effect = "outdegreeSender", model = "sender", by = "actors")
```

`boxplot.tomstats` *Plotting Relational Event Network Statistics Distributions*

Description

Generate boxplots for a specified effect in a `tomstats` object.

Usage

```
## S3 method for class 'tomstats'
boxplot(x, effect, by = "timepoints", subset = NULL, outliers = TRUE, ...)
```

Arguments

`x`	An object of class `tomstats` containing relational event network statistics.
`effect`	A character string specifying the name of the effect in 'x' or an integer indicating the index of the effect to be plotted.
`by`	A string indicating whether the statistic is plotted across 'timepoints' (default) or 'dyads'.
`subset`	An optional vector specifying a subset of timepoints or dyads to be used for plotting. Per default, a maximum of 20 unique timepoints or dyads are plotted.
`outliers`	A logical value specifying whether to include outliers in the plot.
`...`	Additional arguments passed to bxp().

Details

This function produces boxplots to visually represent the distribution of a specified effect in a relational event network, as captured by a tomstats object. The 'effect' parameter allows the user to choose a specific effect for visualization, either by providing the effect's name or its index within the 'tomstats' object. The 'by' parameter determines whether the boxplots are created across different 'timepoints' or 'dyads'. Additionally, an optional 'subset' parameter allows the user to focus on specific timepoints or dyads. If 'subset' is not specified, a default maximum of 20 unique timepoints or dyads are plotted. The 'outliers' argument, when set to TRUE, includes the representation of outliers in the boxplots. If set to FALSE, outliers are omitted from the visualization.

The boxplots are based on the following summary statistics of the data: The box in the middle represents the interquartile range (IQR) between the first (Q1) and third quartile (Q3), and the line inside the box represents the median. The whiskers extend from the box to the minimum and maximum values within 1.5 times the IQR below Q1 or above Q3. Outliers beyond the whiskers are plotted individually.

Value

no return value

Examples

```
library(remstats)
# Load data
data(history)
# Prepare data
reh <- remify::remify(edgelist = history[,1:3], model = "tie")
# Compute effects
stats <- remstats(reh, tie_effects = ~ inertia())
# Plot the 'inertia' distribution for 20 timepoints
boxplot(stats, effect = "inertia")
# Plot the 'inertia' distribution for 20 dyads
boxplot(stats, effect = "inertia", by = "dyads")
# Plot the 'inertia' distribution for dyads 2:5
boxplot(stats, effect = "inertia", by = "dyads", subset = 2:5)
```

`degreeDiff` *degreeDiff*

Description

Specifies the statistic for a 'degreeDiff' effect in the tie-oriented model.

Usage

```
degreeDiff(scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, standardization of the degree difference per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

A degreeDiff effect refers to the tendency for dyads to increase their interaction rate if the absolute difference in degree for the two actors in the pair increases. The statistic at timepoint *t* for dyad *(i,j)* is equal to the difference between the following two values: the number of events before timepoint *t* that involved actor *i* and actor *j*, respectively. The degreeDiff effect is only defined for undirected events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

degreeMin, degreeMax or totaldegreeDyad for other types of degree effects for undirected events.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ degreeDiff()
remstats(reh = reh_tie, tie_effects = effects)
```

---

degreeMax *degreeMax*

Description

Specifies the statistic for an 'degreeMax' effect in the tie-oriented model with undirected events.

Usage

```
degreeMax(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by two times the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An degreeMax effect refers to the tendency for dyads to increase their interaction rate if the total degree of the most active actor in the pair increases. The statistic at timepoint *t* for dyad *(i,j)* is equal to the maximum of the following two values: the number of events before timepoint *t* that involved actor *i* and actor *j*, respectively. Note that the degreeMax effect is only defined for undirected events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that the most active actor was involved in. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to be involved in an event and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

See Also

`degreeDiff`, `degreeMin` or `totaldegreeDyad` for other types of degree effects for undirected events.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ degreeMax()
remstats(reh = reh_tie, tie_effects = effects)
```

`degreeMin` *degreeMin*

Description

Specifies the statistic for an ‘degreeMin‘ effect in the tie-oriented model with undirected events.

Usage

```
degreeMin(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

`scaling`	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by two times the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
`consider_type`	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An degreeMin effect refers to the tendency for dyads to increase their interaction rate if the total degree of the least active actor in the pair increases. The statistic at timepoint *t* for dyad *(i,j)* is equal to the minimum of the following two values: the number of events before timepoint *t* that involved actor *i* and actor *j*, respectively. Note that the degreeMin effect is only defined for undirected events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that the least active actor was involved in. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to be involved in an event and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

See Also

`degreeDiff`, `degreeMax` or `totaldegreeDyad` for other types of degree effects for undirected events.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ degreeMin()
remstats(reh = reh_tie, tie_effects = effects)
```

`difference` *difference*

Description

Specifies the statistic for a "difference" effect in the tie-oriented model or the receiver choice step of the actor-oriented model. A difference effect refers to an exogenous actor attribute that affects dyad *(i,j)*'s rate of interacting (tie-oriented model) or actor *j*'s probability of being chosen as a receiver for the event send by the active sender *i* at time *t* (actor-oriented model) based on the difference between the values of actors *i* and *j* on this attribute.

Usage

```
difference(
  variable,
  attr_actors = NULL,
  scaling = c("none", "std"),
  absolute = TRUE,
  attr_data
)
```

Arguments

variable	string with the name of the column in the attr_actors object for which the statistic has to be computed.
attr_actors	optionally, an object of class data.frame that contains the attribute, see 'Details.'
scaling	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
absolute	Logical value indicating whether the difference values should be converted to the absolute difference (default is TRUE).
attr_data	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* is equal to the (absolute) difference between the values of actor *i* and *j* on the attribute at timepoint *t*.

Construct the 'attr_actors' object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the variable argument in the 'difference()' function.

Note that it is possible to omit the 'attr_actors' object in the call of difference() and, instead, supply it in the call of remstats() for multiple exogenous effects.

Note that it is possible to omit the 'attr_actors' object in the call of difference() and, instead, supply it in the call of remstats() for multiple exogenous effects.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
# Example for tie-oriented model
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ difference("extraversion", absolute = TRUE)
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)

# Example for actor-oriented model
reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects, attr_actors = info)
```

event *event*

Description

Specifies the statistic for an "event" effect in the tie-oriented model. An "event" effect refers to an exogenous event attribute that affects the waiting time between events.

Usage

```
event(x, variableName = NULL)
```

Arguments

x	vector with the event attribute
variableName	optionally, a string indicating the variable name, used for the dimnames of the output statistics object

Details

The statistic at timepoint *t* is for all dyads in the risk set equal to the attribute of the event at timepoint *t*.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

See Also

FEtype

Examples

```
reh_tie <- remify::remify(history, model = "tie")
data(history, package = "remstats")
history$work <- ifelse(history$setting == "work", 1, 0)
effects <- ~ event(x = history$work, variableName = "setting_is_work")
remstats(reh = reh_tie, tie_effects = effects)
```

FEtype *FEtype*

Description

Specifies the statistic for fixed effects for event types in the tie-oriented model.

Usage

```
FEtype()
```

Details

Fixed effects for event types capture the variation in event rate across different event types (e.g., see Butts, 2008). The specification of FEtype results in the creation of C-1 statistics, were C is the number of different event types in the riskset. Let one of the event types, e.g. $c = 1$, represent the reference category. Than, for every event type $c = 2, ..., C$, a statistic is created that at timepoint t for dyad (i,j,c) is equal to 1 if c is equal to the respective event type and equal to 0 otherwise (i.e., dummy variables are created). Note that specifying fixed effects for event types is only available when event types are modeled in the dependent variable.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

event

Examples

```
history$type <- history$setting
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ FEtype()
remstats(reh = reh_tie, tie_effects = effects)
```

history *Simulated relational event history*

Description

A dataset containing a small example of a relational event history. Data is simulated.

Usage

```
data(history)
```

Format

A dataframe with 115 rows and 5 variables:

time time of the event since onset of observation (e.g., in minutes)

actor1 the first actor involved in the event

actor2 the second actor involved in the event

setting the setting for the event

weight the intensity of the event (e.g., based on the duration)

Source

Simulated relational event history for actors in a social network.

See Also

`info` for exogenous information on the actors in the social network.

Examples

```
data(history)
```

`indegreeReceiver` *indegreeReceiver*

Description

Specifies the statistic for an 'indegreeReceiver' effect in the tie-oriented model or the receiver choice step of the actor-oriented model.

Usage

```
indegreeReceiver(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

`scaling`	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
`consider_type`	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An indegree of the receiver effect refers to the tendency for actors to receive events if they have received more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or receiver *j* (actor-oriented model) is equal to the number of events received by actor *j* before timepoint *t*. Note that the 'indegreeReceiver' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that were received by actor j. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to receive a message and the statistic is set equal to 1/ n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeSender`, `outdegreeSender`, `outdegreeReceiver`, `totaldegreeSender`, or `totaldegreeReceiver` for other types of degree effects.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ indegreeReceiver()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

`indegreeSender` *indegreeSender*

Description

Specifies the statistic for an 'indegreeSender' effect in the tie-oriented model or the sender activity rate step of the actor-oriented model.

Usage

```
indegreeSender(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An indegree of the sender effect refers to the tendency for actors to send events if they have received more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or sender *i* (actor-oriented model) is equal to the number of events received by actor *i* before timepoint *t*. Note that the 'indegreeSender' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that were received by actor i. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to send a message and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeReceiver`, `outdegreeSender`, `outdegreeReceiver`, `totaldegreeSender`, or `totaldegreeReceiver` for other types of degree effects.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ indegreeSender()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = effects)
```

`inertia` *inertia*

Description

Specifies the statistic for an inertia effect in the tie-oriented model or the receiver choice step of the actor-oriented model.

Usage

```
inertia(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the inertia statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, the statistics can be scaled by specifying 'prop', in which raw counts are divided by the outdegree of the sender at time t (see 'details') or standardization of the raw counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the number of past events separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

An inertia effect refers to the tendency for dyads to repeatedly interact with each other (tie-oriented model) or for actors to repeatedly choose the same actor as receiver of their events (actor-oriented model). The statistic at timepoint *t* for dyad *(i,j)* resp. receiver *j* is equal to the number of *(i,j)* events before timepoint *t*.

Optionally, a scaling method can be set with `scaling`. By scaling the inertia count by the outdegree of the sender ("prop"), the statistic refers to the fraction of messages send by actor i that were send to actor j. If actor i hasn't send any messages yet it can be assumed that every actor is equally likely to receive a message from i and the statistic is set equal to 1/(n-1), where n refers to the number of actors. The resulting statistic is similar to the "FrPSndSnd" statistic in the R package 'relevent', or the persistence statistic in Section 2.2.2 of Butts (2008). Note that this scaling method is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ inertia()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

info *Simulated exogenous information on actors in a social network.*

Description

A dataset containing exogenous information on the actors in the social network of a relational event history. Data is simulated.

Usage

```
data(info)
```

Format

A dataframe with 10 rows and 5 variables:

id numeric id of the actor

time numeric value, describes when the value of the covariate changes, if it changes

age dichotomized age of the actor (e.g., 0 = below 25, 1 = 25 or older)

sex dichotomized sex of the actor (e.g., 0 = male, 1 = female)

extraversion standardized extraversion score of the actor

agreeableness standardized agreeableness score of the actor

Source

Simulated exogenous information on actors in a social network.

See Also

history for the relational event history.

Examples

```
data(info)
```

---

isp *isp*

Description

Specifies the statistic for an incoming shared partners effect.

Usage

```
isp(unique = FALSE, scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

unique	A logical value indicating whether to sum the minimum of events with third actors (FALSE, default) or the number of third actors that create a new, unique shared partner (TRUE). See details for more information.
scaling	the method for scaling the triad statistic. Default is to not scale the statistic but keep the raw 'counts'. Alternatively, standardization of the raw counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the shared partners separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

The incoming shared partners effect describes the propensity of dyads to interact based on the number of past incoming shared partners between them. By default, the statistic at timepoint t for the dyad (i,j) is computed as the sum of the minimum occurrences of past (h,i) and (h,j) events across all actors h.

When the unique parameter is set to TRUE, a different approach is taken. In this case, the statistic counts the number of actors h that contribute to the creation of a new, distinct shared partner between actors i and j.

Additionally, it is possible to specify a scaling method using the scaling parameter.

Please note that the incoming shared partners effect, 'isp', is exclusively defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

References

Butts, C. (2008). A relational event framework for social action. Sociological Methodology.

See Also

`otp`, `itp`, or `osp` for other types of triadic effects for directed relational events and `sp` for triadic effects for undirected relational events.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ isp()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

itp *itp*

Description

Specifies the statistic for an incoming two-path effect.

Usage

```
itp(unique = FALSE, scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

unique	A logical value indicating whether to sum the minimum of events with third actors (FALSE, default) or the number of third actors that create a new, unique two-path (TRUE). See details for more information.
scaling	The method for scaling the triad statistic. The default value is "none", which means the statistic is not scaled. Alternatively, you can set it to "std" to request standardization of the raw counts per time point.
consider_type	A logical value indicating whether to count the two-paths separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

The incoming two-path effect describes the propensity of dyads to interact based on the number of past incoming two-paths between them. By default, the statistic at timepoint t for the dyad (i,j) is computed as the sum of the minimum occurrences of past (j,h) and (h,i) events across all actors h.

When the unique parameter is set to TRUE, a different approach is taken. In this case, the statistic counts the number of actors h that contribute to the creation of a new, distinct two-path between actors i and j.

Additionally, it is possible to specify a scaling method using the scaling parameter.

Please note that the incoming two-path effect, 'itp', is exclusively defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

References

Butts, C. (2008). A relational event framework for social action. Sociological Methodology.

See Also

otp, osp, or isp for other types of triadic effects for directed relational events and sp for triadic effects for undirected relational events.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ itp()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

maximum *maximum*

Description

Specifies the statistic for a "maximum" effect in the tie-oriented model. A "maximum" effect refers to an exogenous actor attribute that affects dyad *(i,j)*'s rate of interacting based on the maximum of the values of actors *i* and *j* on this attribute.

Usage

```
maximum(variable, attr_actors = NULL, scaling = c("none", "std"), attr_data)
```

Arguments

`variable`	string with the name of the column in the `attr_actors` object for which the statistic has to be computed.
`attr_actors`	optionally, an object of class `data.frame` that contains the attribute, see 'Details.'
`scaling`	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
`attr_data`	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* for dyad *(i,j)* is equal to the maximum of the values of actor *i* and *j* on the attribute at timepoint *t*.

Construct the 'attr_actors' object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the 'difference()' function.

Note that it is possible to omit the 'attr_actors' object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ maximum("extraversion")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)
```

`minimum` *minimum*

Description

Specifies the statistic for a "minimum" effect in the tie-oriented model. A "minimum" effect refers to an exogenous actor attribute that affects dyad *(i,j)*'s rate of interacting based on the minimum of the values of actors *i* and *j* on this attribute.

Usage

```
minimum(variable, attr_actors = NULL, scaling = c("none", "std"), attr_data)
```

Arguments

`variable`	string with the name of the column in the `attr_actors` object for which the statistic has to be computed.
`attr_actors`	optionally, an object of class `data.frame` that contains the attribute, see 'Details.'
`scaling`	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
`attr_data`	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* for dyad *(i,j)* is equal to the minimum of the values of actor *i* and *j* on the attribute at timepoint *t*.

Construct the 'attr_actors' object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the 'difference()' function.

Note that it is possible to omit the 'attr_actors' object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ minimum("extraversion")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)
```

`osp` *osp*

Description

Specifies the statistic for an outgoing shared partners effect.

Usage

```
osp(unique = FALSE, scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

`unique`	A logical value indicating whether to sum the minimum of events with third actors (FALSE, default) or the number of third actors that create a new, unique shared partner (TRUE). See details for more information.
`scaling`	the method for scaling the triad statistic. Default is to not scale the statistic but keep the raw 'counts'. Alternatively, standardization of the raw counts per time point can be requested with 'std'.
`consider_type`	logical, indicates whether to count the shared partners separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

The outgoing shared partners effect describes the propensity of dyads to interact based on the number of past outgoing shared partners between them. By default, the statistic at timepoint t for the dyad (i,j) is computed as the sum of the minimum occurrences of past (i,h) and (j,h) events across all actors h.

When the unique parameter is set to TRUE, a different approach is taken. In this case, the statistic counts the number of actors h that contribute to the creation of a new, distinct shared partner between actors i and j.

Additionally, it is possible to specify a scaling method using the scaling parameter.

Please note that the outgoing shared partners effect, 'osp', is exclusively defined for directed events.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

References

Butts, C. (2008). A relational event framework for social action. Sociological Methodology.

See Also

otp, itp, or isp for other types of triadic effects for directed relational events and sp for triadic effects for undirected relational events.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ osp()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

otp *otp*

Description

Specifies the statistic for an outgoing two-path effect.

Usage

```
otp(unique = FALSE, scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

unique	A logical value indicating whether to sum the minimum of events with third actors (FALSE, default) or the number of third actors that create a new, unique two-path (TRUE). See details for more information.
scaling	The method for scaling the triad statistic. The default value is "none", which means the statistic is not scaled. Alternatively, you can set it to "std" to request standardization of the raw counts per time point.
consider_type	A logical value indicating whether to count the two-paths separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

The outgoing two-path effect describes the propensity of dyads to interact based on the number of past outgoing two-paths between them. By default, the statistic at timepoint t for the dyad (i,j) is computed as the sum of the minimum occurrences of past (i,h) and (h,j) events across all actors h.

When the unique parameter is set to TRUE, a different approach is taken. In this case, the statistic counts the number of actors h that contribute to the creation of a new, distinct two-path between actors i and j.

Additionally, it is possible to specify a scaling method using the scaling parameter.

Please note that the outgoing two-path effect, 'otp', is exclusively defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

References

Butts, C. (2008). A relational event framework for social action. Sociological Methodology.

See Also

itp, osp, or isp for other types of triadic effects for directed relational events and sp for triadic effects for undirected relational events.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ otp()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

outdegreeReceiver *outdegreeReceiver*

Description

Specifies the statistic for an 'outdegreeReceiver' effect in the tie-oriented model or the receiver choice step of the actor-oriented model.

Usage

```
outdegreeReceiver(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An outdegree of the receiver effect refers to the tendency for actors to receive events if they have send more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or receiver *j* (actor-oriented model) is equal to the number of events send by actor *j* before timepoint *t*. Note that the 'outdegreeReceiver' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that were send by actor j. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to receive a message and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeSender`, `indegreeReceiver`, `outdegreeSender`, `totaldegreeSender`, or `totaldegreeReceiver` for other types of degree effects.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ outdegreeReceiver()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

outdegreeSender *outdegreeSender*

Description

Specifies the statistic for an 'outdegreeSender' effect in the tie-oriented model or the sender activity rate step of the actor-oriented model.

Usage

```
outdegreeSender(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

An outdegree of the sender effect refers to the tendency for actors to send events if they have send more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or sender *i* (actor-oriented model) is equal to the number of events send by actor *i* before timepoint *t*. Note that the 'outdegreeSender' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events, the statistic refers to the fraction of past events that were send by actor i. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to send a message and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeSender`, `indegreeReceiver`, `outdegreeReceiver`, `totaldegreeSender`, or `totaldegreeReceiver` for other types of degree effects.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ outdegreeSender()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = effects)
```

---

plot.aomstats *Plotting Relational Event Network Statistics Trajectories*

Description

Generate line plots to visualize the trajectories of a specified effect in the sender model of a `aomstats` object.

Usage

```
## S3 method for class 'aomstats'
plot(x, effect, subset = NULL, ...)
```

Arguments

x	An object of class aomstats containing relational event network statistics.
effect	A character string specifying the name of the effect in 'x' or an integer indicating the index of the effect to be plotted.
subset	An optional vector specifying a subset of actors to be used for plotting. By default, a maximum of 5 unique actors are used for plotting.
...	Additional arguments passed to plot().

Details

This function creates line plots to illustrate the temporal trajectories of a specified effect in a relational event network, as captured in the sender model by a aomstats object. The 'effect' parameter allows users to choose a specific effect for visualization, either by providing the effect's name or its index within the 'aomstats' object. An optional 'subset' parameter enables users to focus on specific actors. If 'subset' is not specified, a default maximum of 5 unique actors is plotted. These actors are randomly selected to represent trajectories across the range of different endpoints for the effect (excluding zero).

Examples

```
library(remstats)
# Load data
data(history)
# Prepare data
reh <- remify::remify(edgelist = history[,1:3], model = "actor")
# Compute effects
stats <- remstats(reh, sender_effects = ~ outdegreeSender())
# Plot the 'outdegreeSender' trajectories 5 actors
plot(stats, effect = "outdegreeSender")
# Plot the 'outdegreeSender' trajectory for a specific actor
plot(stats, effect = "outdegreeSender", subset = 10)
```

---

plot.tomstats *Plotting Relational Event Network Statistics Trajectories*

Description

Generate line plots to visualize the trajectories of a specified effect in a tomstats object.

Usage

```
## S3 method for class 'tomstats'
plot(x, effect, subset = NULL, ...)
```

Arguments

x	An object of class tomstats containing relational event network statistics.
effect	A character string specifying the name of the effect in 'x' or an integer indicating the index of the effect to be plotted.
subset	An optional vector specifying a subset of dyads to be used for plotting. By default, a maximum of 5 unique dyads are used for plotting.
...	Additional arguments passed to plot().

Details

This function creates line plots to illustrate the temporal trajectories of a specified effect in a relational event network, as captured by a tomstats object. The 'effect' parameter allows users to choose a specific effect for visualization, either by providing the effect's name or its index within the 'tomstats' object. An optional 'subset' parameter enables users to focus on specific dyads. If 'subset' is not specified, a default maximum of 5 unique dyads is plotted. These dyads are randomly selected to represent trajectories across the range of different endpoints for the effect (excluding zero).

Examples

```
library(remstats)
# Load data
data(history)
# Prepare data
reh <- remify::remify(edgelist = history[,1:3], model = "tie")
# Compute effects
stats <- remstats(reh, tie_effects = ~ inertia())
# Plot the 'inertia' trajectories for 5 dyads
plot(stats, effect = "inertia")
# Plot the 'inertia' trajectory for a specific dyad
plot(stats, effect = "inertia", subset = 60)
```

---

print.remstats *Printing Relational Event Network Statistics*

Description

Print a remstats object in a user-friendly format.

Usage

```
## S3 method for class 'remstats'
print(x, ...)
```

Arguments

x	object of class remstats.
...	further arguments passed to or from other methods.

Value

The function prints formatted information about the remstats object to the console, presenting details about the relational event network statistics in a user-friendly format.

Examples

```
rehObject <- remify::remify(edgelist = history, model = "tie")
remstatsObject <- remstats::remstats(reh = rehObject, tie_effects = ~ remstats::inertia())
print(remstatsObject)

rehObject <- remify::remify(edgelist = history, model = "actor")
remstatsObject <- remstats::remstats(reh = rehObject, receiver_effects = ~ inertia())
print(remstatsObject)
```

---

psABA *psABA*

Description

Specifies the statistic for a participation shift AB-A in the sender step of the actor-oriented model.

Usage

```
psABA()
```

Details

Refers to the tendency for the same actor to keep initiating events: The next sender is equal to the previous sender. For each timepoint t, the psABA statistic is equal to one for the actor that will create the participation shift if they would occur in the edgelist as the sender at time t and equal to zero for the actors that will not create this participation shift. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

See Also

psABB or psABX for exploring alternative participation shifts in the sender step of the actor-oriented model.

Examples

```
reh_actor <- remify::remify(history, model = "actor")
remstats(sender_effects = ~ psABA(), reh = reh_actor)
```

---

psABAB *psABAB*

Description

Specifies the statistic for a pshift AB-AB effect.

Usage

```
psABAB(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

Refers to the tendency for the same dyads to keep interacting. For directed events, the next sender and receiver are equal to the previous sender and receiver. For undirected events, the next actor pair is equal to the current actor pair. For each timepoint t, the psABAB statistic is equal to one for the dyads that will create the participation shift if they would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the two subsequent AB events have to be equal. If it is set to FALSE, the participation shift is set to one for every AB event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

See Also

psABBA, psABBY, psABXA, psABXB, psABXY or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ psABAB()
remstats(tie_effects = effects, reh = reh_tie)
```

---

psABAY *psABAY*

Description

Specifies the statistic for a participation shift AB-AY.

Usage

```
psABAY(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

One of Gibson's (2003) dyadic participation shifts. The AB-AY participation shift refers to a tendency for *turn continuing*. For directed events, the sender (A) in the current event is the same as the sender in the previous event (A), and the receiver (Y) is different from the previous receiver (B). In undirected events, one of the current actors (A) matches one of the actors in the previous events (A or B), while the other actor (Y) is different.

To identify these shifts, a statistic 'psABAY' is calculated for each pair of actors at a given timepoint (t). If the pair's interaction follows the AB-AY pattern, the statistic is set equal to one; otherwise, it's set to zero.

Additionally, the types of the AB and AY events can be taken into account. If 'consider_type' is 'TRUE', the type of the AB event and the type of the AY event must match for the shift to occur. If 'consider_type' is 'FALSE', the shift happens for every AY event, regardless of the event type.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBA, psABBY, psABXA, psABXB, psABXY or psABAB for other dyadic participation shifts.

Examples

```
reh <- remify::remify(history, model = "tie")
effects <- ~ psABAY()
remstats(reh = reh, tie_effects = effects)
```

---

psABB *psABB*

Description

Specifies the statistic for a participation shift AB-B in the sender step of the actor-oriented model.

Usage

```
psABB()
```

Details

The AB-B participation shift refers to the tendency for immediate reciprocation (the next sender is the previous receiver). For each timepoint t, the psABBA statistic is equal to one for the actor (i.e, the previous event receiver) that will create the participation shift if it would occur as sender in the edgelist at time t and equal to zero for the actors that will not create this participation shift. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABA or psABX for exploring alternative participation shifts in the sender step of the actor-oriented model.

Examples

```
reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = ~ psABB())
```

psABBA *psABBA*

Description

Specifies the statistic for a participation shift AB-BA.

Usage

```
psABBA(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

The AB-BA pshift effect refers to one of Gibson's (2003) dyadic participation shifts. The AB-BA pshift refers to the tendency for immediate reciprocation (the next sender is the previous receiver and the next receiver is the previous sender). For each timepoint t, the psABBA statistic is equal to one for the dyad that will create the participation shift if it would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the AB event and the type of the BA event have to be equal. If it is set to FALSE, the participation shift is set to one for every BA event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed. Note that the AB-BA pshift is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBY, psABXA, psABXB, psABXY or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ psABBA()
remstats(reh = reh_tie, tie_effects = effects)
```

psABBY *psABBY*

Description

Specifies the statistic for a participation shift AB-BY.

Usage

```
psABBY(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

The AB-BY participation shift refers to one of Gibson's (2003) dyadic participation shifts. The AB-BY pshift refers to a tendency for turn receiving (here, the next sender is the previous receiver and the next receiver is not in the current previous). For each timepoint t, the psABBY statistic is equal to one for the dyads that will create the participation shift if they would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the AB event and the type of the BY events have to be equal. If it is set to FALSE, the participation shift is set to one for every BY event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed. Note that the AB-BY pshift is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBA, psABXA, psABXB, psABXY or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ psABBY()
remstats(reh = reh_tie, tie_effects = effects)
```

psABX *psABX*

Description

Specifies the statistic for a participation shift AB-X in the sender step of the actor-oriented model.

Usage

```
psABX()
```

Details

The AB-X participation shift refers to a tendency for turn usurping (here, the next sender is not in the previous event). For each timepoint t, the psABX statistic is equal to one for the actors that will create the participation shift if they would occur as the sender in the edgelist at time t and equal to zero for the actors that will not create this participation shift. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABA or psABB for exploring alternative participation shifts in the sender step of the actor-oriented model.

Examples

```
reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = ~ psABX())
```

psABXA *psABXA*

Description

Specifies the statistic for a participation shift AB-XA.

Usage

```
psABXA(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

The AB-XA participation shift refers to one of Gibson's (2003) dyadic participation shifts. The AB-XA pshift refers to a tendency for turn usurping (here, the next sender is not in the previous event and the next receiver is the previous sender). For each timepoint t, the psABXA statistic is equal to one for the dyads that will create the participation shift if they would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the AB event and the type of the XA events have to be equal. If it is set to FALSE, the participation shift is set to one for every XA event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the pshift is observed. Note that the AB-XA pshift is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBA, psABBY, psABXB, psABXY or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ psABXA()
remstats(reh = reh_tie, tie_effects = effects)
```

---

psABXB *psABXB*

Description

Specifies the statistic for a participation shift AB-XB.

Usage

```
psABXB(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

The AB-XB participation shift refers to one of Gibson's (2003) dyadic participation shifts. The AB-XB pshift refers to a tendency for turn usurping (here, the next sender is not in the previous event and the next receiver is the previous receiver). For each timepoint t, the psABXB statistic is equal to one for the dyads that will create the participation shift if they would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the AB event and the type of the XB events have to be equal. If it is set to FALSE, the participation shift is set to one for every XB event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed. Note that the AB-XB pshift is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBA, psABBY, psABXA, psABXY or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ psABXB()
remstats(reh = reh_tie, tie_effects = effects)
```

---

psABXY *psABXY*

Description

Specifies the statistic for a participation shift AB-XY.

Usage

```
psABXY(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to consider the event type in determining which dyads create a pshift (TRUE, default) or not (FALSE).

Details

The AB-XY participation shift refers to one of Gibson's (2003) dyadic participation shifts. The AB-XY pshift refers to a tendency for turn usurping (here, the next sender and the next receiver are not in the previous event). For each timepoint t, the psABXY statistic is equal to one for the dyads that will create the participation shift if they would occur in the edgelist at time t and equal to zero for the dyads that will not create this participation shift. If consider_type is set to TRUE, the type of the AB event and the type of the XY events have to be equal. If it is set to FALSE, the participation shift is set to one for every XY event, regardless of the event type. If multiple events in the edgelist occur at the same time point, the order of these events determines whether the p-shift is observed. Note that the AB-XY pshift is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

psABBA, psABBY, psABXA, psABXB or psABAY for other dyadic participation shifts.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ psABXY()
remstats(reh = reh_tie, tie_effects = effects)
```

---

receive *receive*

Description

Specifies the statistic for a "receive" effect in the tie-oriented model or the receiver choice step of the actor-oriented model. A "receive" effect refers to an exogenous actor attribute that affects actor *i*'s rate of receiving events.

Usage

```
receive(variable, attr_actors = NULL, scaling = c("none", "std"), attr_data)
```

Arguments

variable	string with the name of the column in the attr_actors object for which the statistic has to be computed.
attr_actors	optionally, an object of class data.frame that contains the attribute, see 'Details.'
scaling	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
attr_data	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* is equal to the value of the exogenous attribute for actor *i* at time *t* for all dyads in the riskset that have actor *i* as receiver. Note that a "receive" effect is only defined for directed relational events.

Construct the ‘attr_actors‘ object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors’ name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the ‘difference()‘ function.

Note that it is possible to omit the ‘attr_actors‘ object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

Examples

```
data(history)
data(info)

# Tie-oriented model
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ receive("extraversion")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)

# Actor-oriented model
reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects, attr_actors = info)
```

`recencyContinue` *recencyContinue*

Description

Specifies the statistic for a recency continue effect in the `effects` argument of `tomstats` or the `receiver_effects` argument of `aomstats`.

Usage

```
recencyContinue(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to compute the recency separately for each event type (TRUE, default) or regardless of event types (FALSE).

Details

The recencyContinue effect refers to a recency statistic similar to what is described in Vu et al. (2017) and Mulder and Leenders (2019). For each timepoint t, for directed dyad (i,j) the statistic is equal to 1/(the time that has past since the dyad was last active + 1).

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

rrankSend, rrankReceive, recencySendSender, recencyReceiveSender, recencyReceiveSender and recencyReceiveReceiver for other type of recency effects

Examples

```
effects <- ~ recencyContinue()
reh_tie <- remify::remify(history, model = "tie")
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(receiver_effects = effects, reh = reh_actor)
```

---

recencyReceiveReceiver

recencyReceiveReceiver

Description

Specifies the statistic for a recency receive of receiver effect in the effects argument of tomstats or the receiver_effects argument of aomstats.

Usage

```
recencyReceiveReceiver(consider_type = TRUE)
```

Arguments

consider_type	logical, indicates whether to compute the recency separately for each event type (TRUE, default) or regardless of event types (FALSE).

Details

The recencyReceiveReceiver effect refers to a recency statistic similar to what is described in Vu et al. (2017) and Mulder and Leenders (2019). For each timepoint t, for directed dyad (i,j) the statistic is equal to 1/(the time that has past since receiver j was last active as receiver + 1). Note that the 'recencyReceiveReceiver' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

rrankSend, rrankReceive, recencySendSender, recencyReceiveSender, recencyReceiveSender and recencyContinue for other type of recency effects

Examples

```
effects <- ~ recencyReceiveReceiver()
reh_tie <- remify::remify(history, model = "tie")
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(receiver_effects = effects, reh = reh_actor)
```

---

recencyReceiveSender *recencyReceiveSender*

Description

Specifies the statistic for a recency receive of sender effect in the `effects` argument of `tomstats` or the `sender_effects` argument of `aomstats`.

Usage

```
recencyReceiveSender(consider_type = TRUE)
```

Arguments

`consider_type`	logical, indicates whether to compute the recency separately for each event type (TRUE, default) or regardless of event types (FALSE).

Details

The recencyReceiveSender effect refers to a recency statistic similar to what is described in Vu et al. (2017) and Mulder and Leenders (2019). For each timepoint t, for directed dyad (i,j) the statistic is equal to 1/(the time that has past since sender i was last active as receiver + 1). Note that the 'recencyReceiveSender' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

rrankSend, rrankReceive, recencySendSender, recencySendReceiver, recencyReceiveReceiver and recencyContinue for other type of recency effects

Examples

```
effects <- ~ recencyReceiveSender()
reh_tie <- remify::remify(history, model = "tie")
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(sender_effects = effects, reh = reh_actor)
```

---

recencySendReceiver *recencySendReceiver*

Description

Specifies the statistic for a recency send of receiver effect in the `effects` argument of `tomstats` or the `receiver_effects` argument of `aomstats`.

Usage

```
recencySendReceiver(consider_type = TRUE)
```

Arguments

`consider_type`	logical, indicates whether to compute the recency separately for each event type (TRUE, default) or regardless of event types (FALSE).

Details

The recencySendReceiver effect refers to a recency statistic similar to what is described in Vu et al. (2017) and Mulder and Leenders (2019). For each timepoint t, for directed dyad (i,j) the statistic is equal to 1/(the time that has past since receiver j was last active as sender + 1). Note that the 'recencySendReceiver' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

rrankSend, rrankReceive, recencySendSender, recencyReceiveSender, recencyReceiveReceiver and recencyContinue for other type of recency effects

Examples

```
effects <- ~ recencySendReceiver()
reh_tie <- remify::remify(history, model = "tie")
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(receiver_effects = effects, reh = reh_actor)
```

---

recencySendSender *recencySendSender*

Description

Specifies the statistic for a recency send of sender effect in the `effects` argument of `tomstats` or the `sender_effects` argument of `aomstats`.

Usage

```
recencySendSender(consider_type = TRUE)
```

Arguments

`consider_type`	logical, indicates whether to compute the recency separately for each event type (TRUE, default) or regardless of event types (FALSE).

Details

The recencySendSender effect refers to a recency statistic similar to what is described in Vu et al. (2017) and Mulder and Leenders (2019). For each timepoint t, for directed dyad (i,j) the statistic is equal to 1/(the time that has past since sender i was last active as sender + 1). Note that the 'recencySendSender' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

rrankSend, rrankReceive, recencySendReceiver, recencyReceiveSender, recencyReceiveReceiver and recencyContinue for other type of recency effects

Examples

```
effects <- ~ recencySendSender()
reh_tie <- remify::remify(history, model = "tie")
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(sender_effects = effects, reh = reh_actor)
```

`reciprocity` *reciprocity*

Description

Specifies the statistic for a reciprocity effect in the tie-oriented model or the receiver choice step of the actor-oriented model.

Usage

```
reciprocity(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

`scaling`	the method for scaling the reciprocity statistic. Default is to not scale the statistic but keep the raw 'counts'. Alternatively, the statistics can be scaled by 'prop', in which raw counts are divided by the indegree of the sender at time t (see 'details') or standardization of the raw counts per time point can be requested with 'std'.
`consider_type`	logical, indicates whether to count the number of past reciprocal events separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

A reciprocity effect refers to the tendency for actors to reciprocate past interactions. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or receiver *j* (actor-oriented model) is equal to the number of *(j,i)* events before timepoint *t*. Note that a reciprocity effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the reciprocity count by the indegree of the sender, the statistic refers to the fraction of messages received by actor i that were received from actor j. If actor i hasn't received any messages yet it can be assumed that actor i is equally likely to receive a message from every actor and the statistic is set equal to 1/(n-1), where n refers to the number of actors. The resulting statistic is similar to the "FrRecSnd" statistic in the R package 'relevent'.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ reciprocity()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

`remstats` *remstats*

Description

Computes statistics for modeling relational events with a tie-oriented or actor-oriented approach.

Usage

```
remstats(
  reh,
  tie_effects = NULL,
  sender_effects = NULL,
  receiver_effects = NULL,
  attr_actors = NULL,
  attr_dyads = NULL,
  method = c("pt", "pe"),
  memory = c("full", "window", "decay", "interval"),
  memory_value = NA,
  start = 1,
  stop = Inf,
  display_progress = FALSE,
  adjmat = NULL,
  get_adjmat = FALSE,
  attr_data,
  attributes,
  edgelist
)
```

Arguments

`reh`	an object of class "`remify`" characterizing the relational event history.
`tie_effects`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the effects in the tie-oriented model for which statistics are computed, see 'Details' for the available effects and their corresponding statistics
`sender_effects`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the effects in the sender activity rate step of the actor-oriented model for which statistics are computed, see 'Details'

`receiver_effects`
: an object of class "formula" (or one that can be coerced to that class): a symbolic description of the effects in the receiver choice step of model for which statistics are computed, see 'Details'

`attr_actors`
: optionally, an object of class "data.frame" that contains exogenous attributes for actors (see Details).

`attr_dyads`
: optionally, an object of class `data.frame` or `matrix` containing attribute information for dyads (see Details).

`method`
: Specifies the method for managing simultaneous events, i.e., events occurring at the same time. The default 'method' is 'pt' (per timepoint), where statistics are computed once for each unique timepoint in the edgelist. Alternatively, you can choose 'pe' (per event), where statistics are computed once for each unique event observed in the edgelist.

`memory`
: The memory to be used. See 'Details'.

`memory_value`
: Numeric value indicating the memory parameter. See 'Details'.

`start`
: an optional integer value, specifying the index of the first time or event in the relational event history for which statistics must be computed (see 'Details')

`stop`
: an optional integer value, specifying the index of the last time or event in the relational event history for which statistics must be computed (see 'Details')

`display_progress`
: should a progress bar for the computation of the endogenous statistics be shown (TRUE) or not (FALSE)?

`adjmat`
: optionally, for a tie-oriented model a previously computed adjacency matrix with on the rows the time points and on the columns the risk set entries

`get_adjmat`
: for a tie-oriented model, whether the adjmat computed by remstats should be outputted as an attribute of the statistics.

`attr_data`
: deprecated, please use "attr_actors" instead

`attributes`
: deprecated, please use "attr_data" instead

`edgelist`
: deprecated, please use "reh" instead

Value

An object of class 'remstats'. In case of the tie-oriented model, an array with the computed statistics, where rows refer to time points, columns refer to potential relational event (i.e., potential edges) in the risk set and slices refer to statistics. In case of the actor-oriented model, list with in the first element the statistics for the sender activity rate step and in the second element the statistics for the receiver choice step, where rows refer to time points, columns refer to potential senders or receivers, respectively. The 'remstats' object has the following attributes:

`model` Type of model that is estimated, obtained from the remify object inputted to 'reh'.

`formula` Model formula, obtained from the formula inputted to 'tie_effects', 'sender_effects' and/or 'receiver_effects', depending on the model.

`riskset` For the tie-oriented model, the risk set used to construct the statistics.

`actors` For the actor-oriented model, the set of actors used to construct the statistics, obtained from the remify object inputted to 'reh'.

`adjmat` [Optional], for the tie-oriented model, if "get_adjmat = TRUE", the matrix with the accumulated event weights for each time point (on the rows) and each dyad (in the columns).

Effects

The statistics to be computed are defined symbolically and should be supplied to the `tie_effects` (for the tie-oriented model), or `sender_effects` and/or `receiver_effects` (for the actor-oriented model) argument in the form `~ effects`. The terms are separated by + operators. For example: `effects = ~ inertia() + otp()`. Interactions between two effects can be included with * or : operators. For example: `effects = ~ inertia():otp()`. A list of available effects can be obtained with `tie_effects()` and `actor_effects()`.

The majority of the statistics can be scaled in some way, see the documentation of the `scaling` argument in the separate effect functions for more information on this.

The majority of the statistics can account for the event type included as a dependent variable, see the documentation of the `consider_type` argument in the separate effect functions for more information on this. Note that this option is only available for the tie-oriented model.

Note that events in the relational event history can be directed or undirected. Some statistics are only defined for either directed or undirected events (see the documentation of the statistics). Note that undirected events are only available for the tie-oriented model.

attr_actors

For the computation of the *exogenous* statistics an attributes object with the exogenous covariate information has to be supplied to the `attr_actors` argument in either `remstats()` or in the separate effect functions supplied to the `..._effects` arguments (e.g., see `send`). This `attr_actors` object should be constructed as follows: A dataframe with rows referring to the attribute value of actor *i* at timepoint *t*. A 'name' column is required that contains the actor name (corresponding to the actor names in the relational event history). A 'time' column is required that contains the time when attributes change (set to zero if none of the attributes vary over time). Subsequent columns contain the attributes that are called in the specifications of exogenous statistics (column name corresponding to the string supplied to the `variable` argument in the effect function). Note that the procedure for the exogenous effects 'tie' and 'event' deviates from this, here the exogenous covariate information has to be specified in a different way, see `tie` and `event`.

attr_dyads

For the computation of the *dyad exogenous* statistics with `tie()`, an attributes object with the exogenous covariates information per dyad has to be supplied. This is a `data.frame` or `matrix` containing attribute information for dyads. If `attr_dyads` is a `data.frame`, the first two columns should represent "actor1" and "actor2" (for directed events, "actor1" corresponds to the sender, and "actor2" corresponds to the receiver). Additional columns can represent dyads' exogenous attributes. If attributes vary over time, include a column named "time". If `attr_dyads` is a `matrix`, the rows correspond to "actor1", columns to "actor2", and cells contain dyads' exogenous attributes.

Memory

The default 'memory' setting is '"full"', which implies that at each time point t the entire event history before t is included in the computation of the statistics. Alternatively, when 'memory' is set to '"window"', only the past event history within a given time window is considered (see Mulders & Leenders, 2019). This length of this time window is set by the 'memory_value' parameter. For example, when 'memory_value = 100' and 'memory = "window"', at time point t only the past events that happened at most 100 time units ago are included in the computation of the statistics.

A third option is to set ‘memory‘ to ‘"interval"‘. In this case, the past event history within a given time interval is considered. For example, when ‘"memory_value" = c(50, 100)‘ and ‘memory = "window"‘, at time point t only the past events that happened between 50 and 100 time units ago are included in the computation of the statistics. Finally, the fourth option is to set ‘memory‘ to ‘"decay"‘. In this case, the weight of the past event in the computation of the statistics depend on the elapsed time between t and the past event. This weight is determined based on an exponential decay function with half-life parameter ‘memory_value‘ (see Brandes et al., 2009).

Event weights

Note that if the relational event history contains a column that is named “weight”, it is assumed that these affect the endogenous statistics. These affect the computation of all endogenous statistics with a few exceptions that follow logically from their definition (e.g., the recenyContinue statistic does depend on time since the event and not on event weights).

Subset the event history using ’start’ and ’stop’

It is possible to compute statistics for a segment of the relational event sequence, based on the entire event history. This is done by specifying the ’start’ and ’stop’ values as the indices for the first and last event times for which statistics are needed. For instance, setting ’start = 5’ and ’stop = 5’ calculates statistics for the 5th event in the relational event sequence, considering events 1-4 in the history. Note that in cases of simultaneous events with the ’method’ set to ’pt’ (per timepoint), ’start’ and ’stop’ should correspond to the indices of the first and last *unique* event timepoints for which statistics are needed. For example, if ’start = 5’ and ’stop = 5’, statistics are computed for the 5th unique timepoint in the relational event sequence, considering all events occurring at unique timepoints 1-4.

Adjacency matrix

Optionally, a previously computed adjacency matrix can be supplied. Note that the endogenous statistics will be computed based on this adjacency matrix. Hence, supplying a previously computed adjacency matrix can reduce computation time but the user should be absolutely sure the adjacency matrix is accurate.

References

Butts, C. T. (2008). A relational event framework for social action. Sociological Methodology, 38(1), 155–200. doi:10.1111/j.14679531.2008.00203.x, Stadtfeld, C., & Block, P. (2017). Interactions, actors, and time: Dynamic network actor models for relational events. Sociological Science, 4, 318–352. doi:10.15195/v4.a14

Examples

```
library(remstats)

# Tie-oriented model
eff <- ~ inertia():send("extraversion") + otp()
reh_tie <- remify::remify(edgelist = history, model = "tie")
remstats(reh = reh_tie, tie_effects = eff, attr_actors = info)
```

```
# Actor-oriented model
seff <- ~ send("extraversion")
reff <- ~ receive("agreeableness") + inertia() + otp()
reh_actor <- remify::remify(edgelist = history, model = "actor")
remstats(
    reh = reh_actor, sender_effects = seff, receiver_effects = reff,
    attr_actors = info
)
```

---

rrankReceive *rrankReceive*

Description

Specifies the statistic for a recency rank receive effect in the `effects` argument of `tomstats` or the `receiver_effects` argument of `aomstats`.

Usage

```
rrankReceive(consider_type = TRUE)
```

Arguments

`consider_type`	logical, indicates whether to discriminate between event types in determining the event rank (TRUE, default) or not (FALSE).

Details

The rrankSend effect refers to a rank-based recency effect, as described in section 2.2.5 of Butts (2008). For each timepoint t, for directed dyad (i,j) the statistic is equal to the inverse of the rank of receiver j among the actors from which sender i has most recently received past events. Note that the 'rrankReceive' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`rrankSend`, `recencySendSender`, `recencySendReceiver`, `recencyReceiveSender`, `recencyReceiveReceiver` and `recencyContinue` for other type of recency effects

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ rrankReceive()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(receiver_effects = effects, reh = reh_actor)
```

---

rrankSend *rrankSend*

Description

Specifies the statistic for a recency rank send effect in the `effects` argument of `tomstats` or the `receiver_effects` argument of `aomstats`.

Usage

```
rrankSend(consider_type = TRUE)
```

Arguments

`consider_type`	logical, indicates whether to discriminate between event types in determining the event rank (TRUE, default) or not (FALSE).

Details

The rrankSend effect refers to a rank-based recency effect, as described in section 2.2.5 of Butts (2008). For each timepoint t, for directed dyad (i,j) the statistic is equal to the inverse of the rank of receiver j among the actors to which sender i has most recently send past events. Note that the 'rrankSend' effect is only defined for directed events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`rrankReceive`, `recencySendSender`, `recencySendReceiver`, `recencyReceiveSender`, `recencyReceiveReceiver` and `recencyContinue` for other type of recency effects

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ rrankSend()
remstats(tie_effects = effects, reh = reh_tie)

reh_actor <- remify::remify(history, model = "actor")
remstats(receiver_effects = effects, reh = reh_actor)
```

`same` *same*

Description

Specifies the statistic for a "same" effect in the tie-oriented model or the receiver choice step of the actor-oriented model. A "same" effect refers to an exogenous actor attribute that affects dyad *(i,j)*'s rate of interacting (tie-oriented model) or actor *j*'s probability of being chosen as a receiver for the event send by the active sender *i* at time *t* (actor-oriented model) based on whether actors *i* and *j* have the same value (or not) on this attribute.

Usage

```
same(variable, attr_actors = NULL, attr_data)
```

Arguments

`variable`	string with the name of the column in the `attr_actors` object for which the statistic has to be computed.
`attr_actors`	optionally, an object of class `data.frame` that contains the attribute, see 'Details.'
`attr_data`	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* is equal to one for dyads *(i,j)* that have the same value on the attribute at timepoint *t* (tie-oriented model) or one for receivers *j* that have the same value on the attribute as the active sender *i* at timepoint *t* (actor-oriented model) and equal to 0 for dyads and receivers that do not have the same value.

Construct the 'attr_actors' object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.
- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the 'difference()' function.

Note that it is possible to omit the 'attr_actors' object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ same("age")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects, attr_actors = info)
```

---

send *send*

Description

Specifies the statistic for a "send" effect in the tie-oriented model or the actor activity rate step of the actor-oriented model. A "send" effect refers to an exogenous actor attribute that affects actor *i*'s rate of sending events.

Usage

```
send(variable, attr_actors = NULL, scaling = c("none", "std"), attr_data)
```

Arguments

`variable`	string with the name of the column in the `attr_actors` object for which the statistic has to be computed.
`attr_actors`	optionally, an object of class `data.frame` that contains the attribute, see 'Details.'
`scaling`	the method for scaling the statistic. Default is to not scale the statistic. Alternatively, standardization of the statistic per time point can be requested with "std".
`attr_data`	Deprecated argument. Please use 'attr_actors' instead.

Details

The statistic at timepoint *t* is equal to the value of the exogenous attribute for actor *i* at time *t* for all dyads in the risk set that have actor *i* as sender. Note that a "send" effect is only defined for directed relational events.

Construct the ‘attr_actors‘ object as a data frame where each row represents the attribute value of actor *i* at timepoint *t*:

- name: The actors' name.
- time: The time when the attribute values change.

- variable: The third column contains the attribute used in the specification of the "difference" effect. The column name should correspond to the string supplied to the `variable` argument in the ‘difference()‘ function.

Note that it is possible to omit the ‘attr_actors‘ object in the call of `difference()` and, instead, supply it in the call of `remstats()` for multiple exogenous effects.

Value

List with all information required by ‘remstats::remstats()‘ to compute the statistic.

Examples

```
data(history)
data(info)

# Tie-oriented model
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ send("extraversion")
remstats(reh = reh_tie, tie_effects = effects, attr_actors = info)

# Actor-oriented model
reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = effects, attr_actors = info)
```

`sp` *sp*

Description

Specifies the statistic for a shared partners effect for undirected events.

Usage

```
sp(unique = FALSE, scaling = c("none", "std"), consider_type = TRUE)
```

Arguments

`unique`	A logical value indicating whether to sum the minimum of events with third actors (FALSE, default) or the number of third actors that create a new, unique shared partner (TRUE). See details for more information.
`scaling`	the method for scaling the triad statistic. Default is to not scale the statistic but keep the raw 'counts'. Alternatively, standardization of the raw counts per time point can be requested with 'std'.
`consider_type`	logical, indicates whether to count the shared partners separately for each event type (TRUE, default) or sum across different event types (FALSE).

Details

The shared partners effect describes the propensity of dyads to interact based on the number of past shared partners between them. By default, the statistic at timepoint t for the undirected dyad (i,j) is computed as the sum of the minimum occurrences of past undirected (i,h) and undirected (j,h) events across all actors h.

When the unique parameter is set to TRUE, a different approach is taken. In this case, the statistic counts the number of actors h that contribute to the creation of a new, distinct shared partner between actors i and j.

Additionally, it is possible to specify a scaling method using the scaling parameter.

Please note that the shared partners effect, 'sp', is exclusively defined for undirected events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`otp`, `itp`, `osp`, or `isp` for triadic effects for directed relational events.

Examples

```
reh_tie <- remify::remify(history, model = "tie", directed = FALSE)
effects <- ~ sp()
remstats(tie_effects = effects, reh = reh_tie)
```

`spUnique` *spUnique*

Description

Deprecated. Use `sp`.

Usage

```
spUnique()
```

Value

Warning.

summary.remstats *Relational Event Network Statistics Summaries*

Description

Produce summaries of each statistic from a remstats object.

Usage

```
## S3 method for class 'remstats'
summary(object, ...)
```

Arguments

object	object of class remstats.
...	additional arguments affecting the summary produced.

Value

The summaries provide information for each statistic included in the remstats object, offering insights into the distribution and characteristics of the data.

Examples

```
rehObject <- remify::remify(edgelist = history, model = "tie")
remstatsObject <- remstats::remstats(reh = rehObject, tie_effects = ~ remstats::inertia())
summary(remstatsObject)

rehObject <- remify::remify(edgelist = history, model = "actor")
remstatsObject <- remstats::remstats(reh = rehObject, receiver_effects = ~ inertia())
summary(remstatsObject)
```

tie *tie*

Description

Specifies the statistic for a "tie" (or, "dyad") effect.

Usage

```
tie(variable, attr_dyads = NULL, scaling = c("none", "std"), x, variableName)
```

Arguments

variable	A string specifying the attribute to compute the statistic. If attr_dyads is a data.frame, this refers to the column name in attr_actors. If attr_dyads is a matrix, this corresponds to the name of the exogenous attribute, used to label the statistic in the resulting remstats object.
attr_dyads	A data.frame or matrix containing attribute information for dyads. If attr_dyads is a data.frame, the first two columns should represent "actor1" and "actor2" (for directed events, "actor1" corresponds to the sender, and "actor2" corresponds to the receiver). Additional columns can represent dyads' exogenous attributes. If attributes vary over time, include a column named "time". If attr_dyads is a matrix, the rows correspond to "actor1", columns to "actor2", and cells contain dyads' exogenous attributes.
scaling	The method for scaling the statistic. The default is no scaling. Alternatively, standardization of the statistic per time point can be requested with "std".
x	Deprecated argument. Please use 'attr_dyads' instead.
variableName	Deprecated argument. Please use 'variable' instead.

Details

The "tie" effect or "dyad" effect refers to an exogenous dyad attribute that influences dyad *(i,j)*'s interaction rate (in tie-oriented models) or the probability of actor *j* being chosen as a receiver for the event sent by the active sender *i* (in actor-oriented models). The statistic represents the value of the exogenous attribute for dyad *(i,j)* in the attr_dyads data.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
data(history)
data(both_male_long)
effect <- ~ tie(variable = "both_male", attr_dyads = both_male_long)
reh <- remify::remify(history, model = "tie")
remstats(reh = reh, tie_effects = effect)

data(both_male_wide)
effect <- ~ tie(variable = "both_male", attr_dyads = both_male_wide)
reh <- remify::remify(history, model = "tie")
remstats(reh = reh, tie_effects = effect)
```

tie_effects *tie_effects*

Description

Overview of statistics in the tie-oriented model, see Details.

Usage

```
tie_effects(directed = NULL, endogenous = NULL)
```

Arguments

directed	logical value. The function outputs all statistics in the tie-oriented model for directed events if true, or all statistics in the tie-oriented model for undirected events if false.
endogenous	logical value. The function outputs all endogenous statistics in the tie-oriented model if true, or all exogenous statistics if false

Details

Overview of statistics in the tie-oriented model.

Baseline:

- baseline

Exogenous statistics:

- send()
- receive()
- tie()
- same()
- difference()
- average()
- minimum()
- maximum()
- event()
- userStat()

Endogenous statistics:

- indegreeSender()
- indegreeReceiver()
- outdegreeSender()
- outdegreeReceiver()

- totaldegreeSender()
- totaldegreeReceiver()
- totaldegreeDyad()
- degreeMin()
- degreeMax()
- degreeDiff()
- inertia()
- reciprocity()
- otp()
- itp()
- osp()
- isp()
- sp()
- psABBA()
- psABBY()
- psABXA()
- psABXB()
- psABXY()
- psABAY()
- psABAB()
- rrankSend()
- rrankReceive()
- recencySendSender()
- recencySendReceiver()
- recencyReceiveSender()
- recencyReceiveReceiver()
- recencyContinue()
- FEtype()

Value

Returns a list of available effects and their corresponding statistics.

Examples

```
# List of available effects
tie_effects()

# List of available effects for undirected networks
tie_effects(directed = FALSE)

# List of available endogenous effects for undirected networks
tie_effects(directed = FALSE, endogenous = TRUE)
```

tomstats *tomstats*

Description

Computes statistics for modeling relational event history data with the tie-oriented relational event model.

Usage

```
tomstats(
  effects,
  reh,
  attr_actors = NULL,
  attr_dyads = NULL,
  method = c("pt", "pe"),
  memory = c("full", "window", "decay", "interval"),
  memory_value = NA,
  start = 1,
  stop = Inf,
  display_progress = FALSE,
  adjmat = NULL,
  get_adjmat = FALSE,
  attr_data,
  attributes,
  edgelist
)
```

Arguments

effects	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the effects in the model for which statistics are computed, see 'Details' for the available effects and their corresponding statistics
reh	an object of class "remify" characterizing the relational event history.
attr_actors	optionally, an object of class "data.frame" that contains exogenous attributes for actors (see Details).
attr_dyads	optionally, an object of class data.frame or matrix containing attribute information for dyads (see Details).
method	Specifies the method for managing simultaneous events, i.e., events occurring at the same time. The default 'method' is 'pt' (per timepoint), where statistics are computed once for each unique timepoint in the edgelist. Alternatively, you can choose 'pe' (per event), where statistics are computed once for each unique event observed in the edgelist.
memory	The memory to be used. See 'Details'.
memory_value	Numeric value indicating the memory parameter. See 'Details'.

`start` an optional integer value, specifying the index of the first time or event in the relational event history for which statistics must be computed (see 'Details')

`stop` an optional integer value, specifying the index of the last time or event in the relational event history for which statistics must be computed (see 'Details')

`display_progress` should a progress bar for the computation of the endogenous statistics be shown (TRUE) or not (FALSE)?

`adjmat` optionally, a previously computed adjacency matrix with on the rows the time points and on the columns the risk set entries

`get_adjmat` whether the adjmat computed by tomstats should be outputted as an attribute of the statistics.

`attr_data` deprecated, please use "attr_actors" instead

`attributes` deprecated, please use "attr_data" instead

`edgelist` deprecated, please use "reh" instead

Value

An object of class 'tomstats'. Array with the computed statistics, where rows refer to time points, columns refer to potential relational event (i.e., potential edges) in the risk set and slices refer to statistics. The 'tomstats' object has the following attributes:

`model` Type of model that is estimated.

`formula` Model formula, obtained from the formula inputted to 'tie_effects'.

`riskset` The risk set used to construct the statistics.

`adjmat` [Optional], if "get_adjmat = TRUE", the matrix with the accumulated event weights for each time point (on the rows) and each dyad (in the columns).

Effects

The statistics to be computed are defined symbolically and should be supplied to the `effects` argument in the form `~ effects`. The terms are separated by + operators. For example: `effects = ~ inertia() + otp()`. Interactions between two effects can be included with * operators. For example: `effects = ~ inertia()*otp()`. A list of available effects can be obtained with `tie_effects()`.

The majority of the statistics can be scaled in some way, see the documentation of the `scaling` argument in the separate effect functions for more information on this.

The majority of the statistics can account for the event type included as a dependent variable, see the documentation of the `consider_type` argument in the separate effect functions for more information on this.

Note that events in the relational event history can be directed or undirected. Some statistics are only defined for either directed or undirected events (see the documentation of the statistics). Note that undirected events are only available for the tie-oriented model.

attr_actors

For the computation of the *exogenous* statistics an attributes object with the exogenous covariate information has to be supplied to the `attr_actors` argument in either `remstats()` or in the separate effect functions supplied to the `..._effects` arguments (e.g., see `send`). This `attr_actors` object should be constructed as follows: A dataframe with rows referring to the attribute value of actor *i* at timepoint *t*. A 'name' column is required that contains the actor name (corresponding to the actor names in the relational event history). A 'time' column is required that contains the time when attributes change (set to zero if none of the attributes vary over time). Subsequent columns contain the attributes that are called in the specifications of exogenous statistics (column name corresponding to the string supplied to the `variable` argument in the effect function). Note that the procedure for the exogenous effects 'tie' and 'event' deviates from this, here the exogenous covariate information has to be specified in a different way, see `tie` and `event`.

attr_dyads

For the computation of the *dyad exogenous* statistics with `tie()`, an attributes object with the exogenous covariates information per dyad has to be supplied. This is a `data.frame` or `matrix` containing attribute information for dyads. If `attr_dyads` is a `data.frame`, the first two columns should represent "actor1" and "actor2" (for directed events, "actor1" corresponds to the sender, and "actor2" corresponds to the receiver). Additional columns can represent dyads' exogenous attributes. If attributes vary over time, include a column named "time". If `attr_dyads` is a `matrix`, the rows correspond to "actor1", columns to "actor2", and cells contain dyads' exogenous attributes.

Memory

The default 'memory' setting is '"full"', which implies that at each time point t the entire event history before t is included in the computation of the statistics. Alternatively, when 'memory' is set to '"window"', only the past event history within a given time window is considered (see Mulders & Leenders, 2019). This length of this time window is set by the 'memory_value' parameter. For example, when 'memory_value = 100' and 'memory = "window"', at time point t only the past events that happened at most 100 time units ago are included in the computation of the statistics. A third option is to set 'memory' to '"interval"'. In this case, the past event history within a given time interval is considered. For example, when '"memory_value" = c(50, 100)' and 'memory = "window"', at time point t only the past events that happened between 50 and 100 time units ago are included in the computation of the statistics. Finally, the fourth option is to set 'memory' to '"decay"'. In this case, the weight of the past event in the computation of the statistics depend on the elapsed time between t and the past event. This weight is determined based on an exponential decay function with half-life parameter 'memory_value' (see Brandes et al., 2009).

Event weights

Note that if the relational event history contains a column that is named "weight", it is assumed that these affect the endogenous statistics. These affect the computation of all endogenous statistics with a few exceptions that follow logically from their definition (e.g., the recenyContinue statistic does depend on time since the event and not on event weights).

Subset the event history using 'start' and 'stop'

It is possible to compute statistics for a segment of the relational event sequence, based on the entire event history. This is done by specifying the 'start' and 'stop' values as the indices for the first and last event times for which statistics are needed. For instance, setting 'start = 5' and 'stop = 5' calculates statistics for the 5th event in the relational event sequence, considering events 1-4 in the history. Note that in cases of simultaneous events with the 'method' set to 'pt' (per timepoint), 'start' and 'stop' should correspond to the indices of the first and last *unique* event timepoints for which statistics are needed. For example, if 'start = 5' and 'stop = 5', statistics are computed for the 5th unique timepoint in the relational event sequence, considering all events occurring at unique timepoints 1-4.

Adjacency matrix

Optionally, a previously computed adjacency matrix can be supplied. Note that the endogenous statistics will be computed based on this adjacency matrix. Hence, supplying a previously computed adjacency matrix can reduce computation time but the user should be absolutely sure the adjacency matrix is accurate.

References

Butts, C. T. (2008). A relational event framework for social action. Sociological Methodology, 38(1), 155–200. doi:10.1111/j.14679531.2008.00203.x

Examples

```
library(remstats)

# Load data
data(history)
data(info)

# Prepare data
reh <- remify::remify(edgelist = history, model = "tie")

# Compute effects
effects <- ~ inertia():send("extraversion") + otp()
tomstats(effects, reh = reh, attr_actors = info)
```

---

totaldegreeDyad *totaldegreeDyad*

Description

Specifies the statistic for a 'totaldegreeDyad' effect.

Usage

```
totaldegreeDyad(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by two times the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

The 'totaldegreeDyad' effect refers to the tendency of pairs of actors (dyads) to increase their interaction rate as the total degree (number of interactions) of both actors in the pair goes up. To calculate this effect for a specific pair (i,j) at a given timepoint (t), we sum the degrees of the two actors in the dyad (i,j).

Additionally, there is an optional scaling method, which can be chosen using the 'scaling' method. When the 'prop' scaling method is applied, the degree count is divided by two times the total number of past events. This scaling converts the statistic into a fraction, representing the proportion of past events in which at least one actor in the dyad was involved. For the first timepoint, where no events have previously occurred, it is assumed that each actor is equally likely to be involved in an event. In this case, the statistic is set to 1 divided by the total number of actors (N).

The totaldegreeDyad effect is defined for the tie-oriented model and is applicable to both directed and undirected events.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ totaldegreeDyad()
remstats(reh = reh_tie, tie_effects = effects)
```

---

totaldegreeReceiver *totaldegreeReceiver*

Description

Specifies the statistic for an 'totaldegreeReceiver' effect in the tie-oriented model or the receiver choice step of the actor-oriented model.

Usage

```
totaldegreeReceiver(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by two times the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

A total degree of the receiver effect refers to the tendency for actors to receive events if they have send and received more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or receiver *j* (actor-oriented model) is equal to the number of events send and received by actor *j* before timepoint *t*. Note that the 'totaldegreeReceiver' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events times two, the statistic refers to the fraction of past events times two that involved actor j. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to receive a message and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeSender`, `indegreeReceiver`, `outdegreeSender`, `outdegreeReceiver`, or `totaldegreeSender` for other types of degree effects.

Examples

```
reh_tie <- remify::remify(history, model = "tie")
effects <- ~ totaldegreeReceiver()
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, receiver_effects = effects)
```

---

totaldegreeSender *totaldegreeSender*

Description

Specifies the statistic for an 'totaldegreeSender' effect in the tie-oriented model or the sender activity rate step of the actor-oriented model.

Usage

```
totaldegreeSender(scaling = c("none", "prop", "std"), consider_type = TRUE)
```

Arguments

scaling	the method for scaling the degree statistic. Default is to not scale the statistic (scaling = "none"). Alternatively, scaling of the raw degree counts by two times the number of past events at time t can be requested with 'prop' or standardization of the raw degree counts per time point can be requested with 'std'.
consider_type	logical, indicates whether to count the degrees separately for each event type (TRUE, default) or sum degrees across different event types (FALSE).

Details

A total degree of the sender effect refers to the tendency for actors to send events if they have send and received more past events. The statistic at timepoint *t* for dyad *(i,j)* (tie-oriented model) or sender *i* (actor-oriented model) is equal to the number of events send and received by actor *i* before timepoint *t*. Note that the 'totaldegreeSender' effect is only defined for directed events.

Optionally, a scaling method can be set with `scaling`. By scaling the degree count by the total number of past events times two, the statistic refers to the fraction of past events times two that involved actor i. At the first time point, when no events did previously occur, it is assumed that every actor is equally likely to send a message and the statistic is set equal to 1/n, where n refers to the number of actors.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

See Also

`indegreeSender`, `indegreeReceiver`, `outdegreeSender`, `outdegreeReceiver`, or `totaldegreeReceiver` for other types of degree effects.

Examples

```
effects <- ~ totaldegreeSender()
reh_tie <- remify::remify(history, model = "tie")
remstats(reh = reh_tie, tie_effects = effects)

reh_actor <- remify::remify(history, model = "actor")
remstats(reh = reh_actor, sender_effects = effects)
```

userStat *userStat*

Description

Allows the user to add its own pre-computed statistic to the statistics object and, optionally, interact this statistic with other statistics in the formula.

Usage

```
userStat(x, variableName = NULL)
```

Arguments

x	Matrix with number of rows equal to the number of events and number of columns equal to the number of dyads in the network (tie-oriented model) or the number of actors in the network (actor-oriented model)
variableName	Optionally, a string with the name of the statistic.

Value

List with all information required by 'remstats::remstats()' to compute the statistic.

Examples

```
reh <- remify::remify(history, model = "tie")
actor101Events <- which(history$actor1 == "101" | history$actor2 == "101")
actor101_stat <- t(sapply(seq_len(nrow(history)), function(i) {
  rep(i %in% actor101Events, reh$D)
}))

# Main effects only
effects <- ~ userStat(x = actor101_stat, variableName = "actor101event")
remstats(reh = reh, tie_effects = effects)

# Model with interaction effects
interaction_effects <- ~ inertia() *
  userStat(x = actor101_stat, variableName = "actor101event")
remstats(reh = reh, tie_effects = interaction_effects)
```

Index