Package 'bitriad'

June 30, 2026

Type Package

Title Triadic Analysis of Affiliation Networks

Version 0.4

Maintainer Jason Cory Brunson <cornelioid@gmail.com>

Description Two principal tools are provided for the triadic analysis of affiliation networks: triad census and triadic closure. These include several variations on both classical tools tailored to affiliation network structure; see Opsahl (2013) <doi:10.1016/j.socnet.2011.07.001>, Liebig and Rao (2014) <doi:10.1109/SITIS.2014.15>, and Brunson (2015) <doi:10.1017/nws.2015.38>. Additional functions support manipulation of affiliation networks. Built on 'igraph' with new C++ calculations exposed via 'Rcpp'.

Depends R (>= 3.5.0), igraph (>= 1.1.2)

Imports MASS, Rcpp (>= 0.12.13)

LinkingTo Rcpp

Suggests knitr, testthat

License GPL-2

Encoding UTF-8

VignetteBuilder knitr

Classification/MSC 05C82, 91D30

Config/roxygen2/version 8.0.0

Config/roxygen2/markdown TRUE

URL http://corybrunson.github.io/bitriad/

NeedsCompilation yes

Author Jason Cory Brunson [aut, cre]

Repository CRAN

Date/Publication 2026-06-30 19:10:07 UTC

Contents

`affiliation_network` *Affiliation network structure*

Description

Test `igraph` objects for affiliation network structure or impose such structure if possible.

Usage

```
is_an(graph)

is.an(graph)

as_an(graph, map_type = FALSE)

as.an(graph, map_type = FALSE)
```

Arguments

graph	An igraph object.
map_type	Logical; whether to add a type attribute for bipartite structure if graph admits one.

Details

An affiliation network is a bipartite graph whose nodes are classified as actors and events in such a way that all links are between actors and events. The function is_bipartite() tests an igraph object for a type attribute, which is intended to bipartition of the nodes. It does not test whether the links respect this partition. The function is_an tests this, as well as the condition that actor nodes precede event nodes in their node IDs, which simplifies some other functions. The function as_an coerces an igraph object to an affiliation network by verifying that the object is bipartite and minimally permuting the node IDs. If graph has no type attribute and map_type is FALSE, then as_an throws an error; if map_type is TRUE, then as_an calls bipartite_mapping() to add a logical type attribute that takes the value FALSE at the first node (by node ID) in each connected component and TRUE or FALSE at the remaining nodes. (If this cannot be done, an error is thrown.)

Value

For is_an(), a logical value; for as_an(), the input graph with a "type" attribute.

See Also

Original **igraph** functions: is_igraph()

Other network testing and coercion: dynamic_an

Examples

```
graph <- make_graph(c( 1,2, 1,4, 1,6, 3,6, 5,6, 7,8, 9,8 ))
is_an(graph)
# as_an(graph) # throws an error
an_graph <- as_an(graph, map_type = TRUE)
is_an(an_graph)
```

---

bitriad bitriad: *Triadic analysis of affiliation networks*

Description

Calculate triad censuses and triad closure statistics designed for affiliation networks.

Details

The package contains two principal tools for the triadic analysis of affiliation networks: triad censuses and measures of triad closure. Assorted additional functions, including a measure of dynamic triad closure, are also included.

Triad censuses

Three triad censuses are implemented for affiliation networks:

- The *full triad census* (Brunson, 2015) records the number of triads of each isomorphism class. The classes are indexed by a partition, $\lambda = (\lambda_1 \leq \lambda_2 \leq \lambda_3)$, indicating the number of events attended by both actors in each pair but not the third, and a positive integer, w, indicating the number of events attended by all three actors. The isomorphism classes are organized into a matrix with rows indexed by λ and columns indexed by w, with the partitions λ ordered according to the *revolving door ordering* (Kreher & Stinson, 1999). The main function triad_census_an (called from `triad_census` when the `graph` argument is an `affiliation_network`) defaults to this census.
- For the analysis of sparse affiliation networks, the full triad census may be less useful than information on whether the extent of connectivity through co-attended events differs between each pair of actors. In order to summarize this information, a coarser triad census can be conducted on classes of triads based on the following congruence relation: Using the indices $\lambda = (x \geq y \geq z)$ and w above, note that the numbers of shared events for each pair and for the triad are $x+w \geq y+w \geq z+w \geq w \geq 0$. Consider two triads congruent if the same subset of these weak inequalities are strictly satisfied. The resulting *difference triad census*, previously called the *uniformity triad census*, implemented as triad_census_difference, is organized into a 8×2 matrix with the strictness of the first three inequalities determining the row and that of the last inequality determining the column.
- A still coarser congruence relation can be used to tally how many are connected by at least one event in each distinct way. This relation considers two triads congruent if each corresponding pair of actors both attended or did not attend at least one event not attended by the third, and if the corresponding triads both attended or did not attend at least one event together. The *binary triad census* (Brunson, 2015; therein called the *structural triad census*), implemented as triad_census_binary, records the number of triads in each congruence class.
- The *simple triad census* is the 4-entry triad census on a traditional (non-affiliation) network indicating the number of triads of each isomorphism class, namely whether the triad contains zero, one, two, or three links. The function simple_triad_census computes the classical (undirected) triad census for an undirected traditional network, or for the actor projection of an affiliation network (if provided), using triad_census; if the result doesn't make sense (i.e., the sum of the entries is not the number of triples of nodes), then it instead uses its own, much slower method.

Each of these censuses can be projected from the previous using the function project_census. A fourth census, called the *uniformity triad census* and implemented as unif_triad_census, is deprecated. Three-actor triad affiliation networks can be constructed and plotted using the triad functions.

The default method for the two affiliation network–specific triad censuses is adapted from the algorithm of Batagelj and Mrvar (2001) for calculating the classical triad census for a directed graph.

Measures of triad closure

Each measure of triad closure is defined as the proportion of wedges that are closed, where a *wedge* is the image of a specified two-event triad W under a specified subcategory of graph maps C subject to a specified congruence relation , and where a wedge is *closed* if it is the image of such a map that factors through a canonical inclusion of W to a specified self-dual three-event triad X.

The alcove, wedge, maps, and congruence can be specified by numerical codes as follows (no plans exist to implement more measures than these):

- `alcove`:
  - `0`: $T_{(1,1,1),0}$
  - `1`: $T_{(1,1,0),1}$ (**not yet implemented**)
  - `2`: $T_{(1,0,0),2}$ (**not yet implemented**)
  - `3`: $T_{(0,0,0),3}$ (**not yet implemented**)
- `wedge`:
  - `0`: $T_{(1,1,0),0}$
  - `1`: $T_{(1,0,0),1}$ (**not yet implemented**)
  - `2`: $T_{(0,0,0),2}$ (**not yet implemented**)
- `maps`:
  - `0`: all graph maps (injective on actors)
  - `1`: injective graph maps
  - `2`: induced injective graph maps
- `congruence`:
  - `0`: same actor and event images (equivalence)
  - `1`: same actor images, structurally equivalent event images
  - `2`: same actor images

Some specifications correspond to statistics of especial interest:

- `0,0,0,2`: the classical clustering coefficient (Watts & Strogatz, 1998), evaluated on the unipartite actor projection
- `0,0,1,0`: the two-mode clustering coefficient (Opsahl, 2013)
- `0,0,2,0`: the unconnected clustering coefficient (Liebig & Rao, 2014)
- `3,2,2,0`: the completely connected clustering coefficient (Liebig & Rao, 2014) (**not yet implemented**)
- `0,0,2,1`: the exclusive clustering coefficient (Brunson, 2015)
- `0,0,2,2`: the exclusive clustering coefficient

See Brunson (2015) for a general definition and the aforecited references for discussions of each statistic.

Author(s)

Jason Cory Brunson

References

Kreher, D.L., & Stinson, D.R. (1999). Combinatorial algorithms: generation, enumeration, and search. *SIGACT News*, 30(1), 33–35.

Batagelj, V., & Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks*, 23(3), 237–243.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

Watts, D.J., & Strogatz, S.H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393(6684), 440–442.

Opsahl, T. (2013). Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2), 159–167. Special Issue on Advances in Two-mode Social Networks.

Liebig, J., & Rao, A. (2014). Identifying influential nodes in bipartite networks using the clustering coefficient. Pages 323–330 of: *Proceedings of the tenth international conference on signal-image technology and internet-based systems*.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Useful links:

- http://corybrunson.github.io/bitriad/

`chicago1960s` *Corporate interlocks among directors in Chicago.*

Description

An affiliation network of 20 directors of 24 corporations and social clubs.

Format

An affiliation network; see is_an.

Source

Barnes, R. & Burkett, T. (2010). Structural Redundancy and Multiplicity in Corporate Networks. *Connections*, 30(2), p. 4–20. https://www.insna.org/connections-archives-2000-2015

`dualize` *Take the dual of an affiliation network*

Description

This function obtains the dual of an affiliation network, in which the actors and events have swapped roles. To do this, it negates the logical values of the node `type` attribute and reorders the node ids accordingly.

Usage

```
dualize(graph)

dual_an(graph)

dual.an(graph)
```

Arguments

graph	An affiliation network.

Value

The input graph with the "type" attribute logically negated.

See Also

Other modal queries and manipulations: mode_addition, mode_counts, modes, schedule()

Examples

```
data(women_clique)
( tab <- table(V(women_clique)$type) )
( proj <- actor_projection(dualize(women_clique)) )
vcount(proj) == tab[2]
```

---

dynamic_an *Dynamic affiliation network structure*

Description

An affiliation network is *dynamic*, for present purposes, if its event nodes have time stamps, recorded as a numeric vertex attribute "time" in non-decreasing order.

Usage

```
is_dynamic_an(graph)

is.dyn(graph)

as_dynamic_an(graph, use_attr = NULL)
```

Arguments

graph	An affiliation network.
use_attr	Character; the vertex attribute to coerce to numeric if necessary and use as the "time" attribute. If NULL, the default, time takes the values seq(event_count(graph)), reflecting the order of the event node IDs.

Value

The input graph with a new "time" vertex attribute and its event nodes permuted to respect this attribute.

See Also

Other network testing and coercion: affiliation_network

Examples

```
data(women_group)
is_dynamic_an(women_group)
data(women_clique)
is_dynamic_an(women_clique)
as_dynamic_an(women_clique)
data(nmt_meetings)
nmt_meetings <- set_event_attr(
  nmt_meetings, "meet",
  value = gsub("meet", "", V2(nmt_meetings)$name)
)
as_dynamic_an(nmt_meetings, use_attr = "meet")
```

---

dynamic_triad_closure *Triadic closure for dynamic affiliation networks*

Description

Given an affiliation network with time-stamped events, compute the proportion of centered triples at which an open wedge exists at some time that is closed at a later time.

Usage

```
dynamic_triad_closure(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...,
  measure = NULL
)

dynamic_triad_closure_an(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...,
  measure = NULL
)
```

```
dynamic_transitivity_an(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...,
  measure = NULL
)

dyn.transitivity.an(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...,
  measure = NULL
)

dynamic_triad_closure_projection(graph, memory = Inf, type = "global")
```

Arguments

graph	An affiliation network with time-stamped events.
actors	A vector of actor nodes in graph.
type	The type of statistic, matched to "global", "local", or "raw".
...	Additional parameters passed to specific functions.
measure	Character; the measure of triad closure, used as the suffix * to triad_closure_*. Matched to "classical" (also "watts_strogatz"), "twomode" (also "opsahl"), "unconnected" (also "liebig_rao_0"), "completely_connected" (also "liebig_rao_3"), "exclusive", or "projection".
memory	Numeric; minimum delay of wedge formation since would-have-been closing events.

Value

A numeric vector of the same length as actors.

See Also

Other triad closure functions: project_transitivity(), transitivity_an(), triad_closure(), triad_closure_from_census()

Examples

```
data(women_group)
dynamic_triad_closure(women_group)
cbind(
  transitivity(actor_projection(women_group), type = "local"),
  triad_closure_opsahl(women_group, type = "local"),
```

```
  triad_closure_exclusive(women_group, type = "local"),
  dynamic_triad_closure_projection(women_group, type = "local"),
  dynamic_triad_closure(women_group, type = "local")
)
```

`dynamic_wedges` *Wedge censuses and closure indicators for dynamic affiliation networks*

Description

Given a dynamic affiliation network and an actor node ID, identify all wedges for a specified measure centered at the node and indicate whether each is closed.

Usage

```
dynamic_wedges(
  graph,
  actor,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  memory = Inf,
  wedge.gap = Inf,
  close.after = 0,
  close.before = Inf
)
```

Arguments

`graph`	A dynamic affiliation network.
`actor`	An actor node in `graph`.
`alcove`, `wedge`, `maps`, `congruence`	Choice of alcove, wedge, maps, and congruence (see Details).
`memory`	Numeric; minimum delay of wedge formation since would-have-been closing events.
`wedge.gap`	Numeric; maximum delay between the two events of a wedge.
`close.after`, `close.before`	Numeric; minimum and maximum delays after both events form a wedge for a third event to close it.

Details

The `dynamic_wedges_*` functions implement wedge censuses underlying the several measures of triad closure described below. Each function returns a transversal of wedges from the congruence classes of wedges centered at the index actor and indicators of whether each class is closed. The shell function `dynamic_wedges` determines a unique measure from several coded arguments (see below) and passes the input affiliation network to that measure.

Value

A two-element list consisting of (1) a 3- or 5-row integer matrix of (representatives of) all (congruence classes of) wedges in `graph` centered at `actor`, and (2) a logical vector indicating whether each wedge is closed.

Measures of triad closure

Each measure of triad closure is defined as the proportion of wedges that are closed, where a *wedge* is the image of a specified two-event triad W under a specified subcategory of graph maps C subject to a specified congruence relation , and where a wedge is *closed* if it is the image of such a map that factors through a canonical inclusion of W to a specified self-dual three-event triad X.

The alcove, wedge, maps, and congruence can be specified by numerical codes as follows (no plans exist to implement more measures than these):

- `alcove`:
  - `0`: $T_{(1,1,1),0}$
  - `1`: $T_{(1,1,0),1}$ (**not yet implemented**)
  - `2`: $T_{(1,0,0),2}$ (**not yet implemented**)
  - `3`: $T_{(0,0,0),3}$ (**not yet implemented**)
- `wedge`:
  - `0`: $T_{(1,1,0),0}$
  - `1`: $T_{(1,0,0),1}$ (**not yet implemented**)
  - `2`: $T_{(0,0,0),2}$ (**not yet implemented**)
- `maps`:
  - `0`: all graph maps (injective on actors)
  - `1`: injective graph maps
  - `2`: induced injective graph maps
- `congruence`:
  - `0`: same actor and event images (equivalence)
  - `1`: same actor images, structurally equivalent event images
  - `2`: same actor images

Some specifications correspond to statistics of especial interest:

- `0,0,0,2`: the classical clustering coefficient (Watts & Strogatz, 1998), evaluated on the unipartite actor projection
- `0,0,1,0`: the two-mode clustering coefficient (Opsahl, 2013)
- `0,0,2,0`: the unconnected clustering coefficient (Liebig & Rao, 2014)
- `3,2,2,0`: the completely connected clustering coefficient (Liebig & Rao, 2014) (**not yet implemented**)
- `0,0,2,1`: the exclusive clustering coefficient (Brunson, 2015)
- `0,0,2,2`: the exclusive clustering coefficient

See Brunson (2015) for a general definition and the aforecited references for discussions of each statistic.

References

Watts, D.J., & Strogatz, S.H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393(6684), 440–442.

Opsahl, T. (2013). Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2), 159–167. Special Issue on Advances in Two-mode Social Networks.

Liebig, J., & Rao, A. (2014). Identifying influential nodes in bipartite networks using the clustering coefficient. Pages 323–330 of: *Proceedings of the tenth international conference on signal-image technology and internet-based systems*.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Other wedge functions: `indequ_wedges()`

`index_subset` *Combinatorial bijections for affiliation network triad indexing*

Description

These functions biject among partitions of at most 3 parts, 3-subsets of natural numbers, and indices for the lexicographic total orders on them.

Usage

```
index_subset(i)

subset_index(vec)

subset_partition(vec)

partition_subset(lambda)

index_partition(i)

partition_index(lambda)

indexSubset(i)

indexPartition(i)

subsetIndex(vec)

subsetPartition(vec)

partitionIndex(lambda)
```

```
partitionSubset(lambda)
```

Arguments

i	Integer; an index in the total order, starting at 0.
vec	Integer vector; a set of 3 distinct non-negative integers, in decreasing order.
lambda	Integer vector; a partition of at most 3 parts, with parts in non-increasing order.

Value

Numeric vectors.

Examples

```
index_subset(2)
index_partition(2)
subset_index(c(3, 2, 0))
subset_partition(c(3, 2, 0))
partition_index(c(1, 1, 0))
partition_subset(c(1, 1, 0))
```

---

minneapolis1970s *Interlocks among corporate philanthropists in Minneapolis-St. Paul.*

Description

These data record the memberships of 26 directors on the boards of 15 companies. They are a subset of a larger dataset that is not publicly available.

Format

An affiliation network; see is_an.

Source

Wasserman, S. & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press. http://books.google.com/books?id=CAm2DpIqRUIC

modes *Actor and event node iteration and attribute querying and assignment*

Description

These functions return actor and event node lists.

Usage

```
V1(graph)

V2(graph)

actor_attr(graph, name, index = V1(graph))

event_attr(graph, name, index = V2(graph))

set_actor_attr(graph, name, index = V1(graph), value)

set_event_attr(graph, name, index = V2(graph), value)

V1(x) <- value

V2(x) <- value
```

Arguments

graph	An affiliation network.
name	The name of the attribute to set.
index	An optional node sequence to set the attributes of a subset of actor or event nodes.
value	The new value of the attribute for all (or index) actor or event nodes.
x	An affiliation network.

Value

The value of the actor or event attribute, or the input graph with the attribute set.

See Also

Original **igraph** functions: V(), vertex_attr(), set_vertex_attr()

Other modal queries and manipulations: dualize(), mode_addition, mode_counts, schedule()

Examples

```
data(women_clique)
print(V1(women_clique))
print(V2(women_clique))
V1(women_clique)$label <- LETTERS[1:5]
V2(women_clique)$label <- 1:5
plot(prettify_an(women_clique))
```

`mode_addition` *Add actor and event nodes*

Description

These functions add actor and event nodes (as desired) to a graph while maintaining its (temporal) affiliation network structure.

Usage

```
add_modes(graph, mode = 1, nv, ..., attr = list(), affiliations = NULL)

add_actors(graph, nv, ..., attr = list(), events = NULL)

add_events(graph, nv, ..., attr = list(), actors = NULL)
```

Arguments

`graph`	An affiliation network.
`mode`	Numeric or character; whether to project onto actors (`1` or `"actors"`) or onto events (`2` or `"events"`).
`nv, ..., attr`	Arguments passed to `add_vertices()`. Events added to a dynamic affiliation network should be given time attributes.
`affiliations`	A vector, or list of length `nv` of vectors, of nodes in `graph` of mode *not* `mode`, to be linked to the new node(s).
`events`	A vector, or list of length `nv` of vectors, of event nodes in `graph`, to be linked to the new actor(s).
`actors`	A vector, or list of length `nv` of vectors, of actor nodes in `graph`, to be linked to the new event(s).

Value

The input `graph` with additional actor or event nodes added.

See Also

Original **igraph** functions: `add_vertices()`, `add_edges()`

Other modal queries and manipulations: `dualize()`, `mode_counts`, `modes`, `schedule()`

Examples

```
data(women_clique)
plot(prettify_an(add_actors(women_clique, nv = 1, events = c(7, 9))))
data(women_group)
plot(prettify_an(women_group))
actor_names <- c("Frances", "Dorothy")
cbind(
  dynamic_triad_closure(women_group, type = "local"),
  dynamic_triad_closure(
    add_events(women_group, nv = 1, actors = actor_names, time = 0),
    type = "local"
  ),
  dynamic_triad_closure(
    add_events(women_group, nv = 1, actors = actor_names, time = 367),
    type = "local"
  )
)
```

---

mode_counts *Count the actors and events in an affiliation network*

Description

These functions return the number of actors (nodes with `type` attribute `FALSE`) or events (`TRUE`) in an affiliation network.

Usage

```
actor_count(graph)

event_count(graph)

actor.count(graph)

event.count(graph)
```

Arguments

`graph`	An affiliation network.

Value

An integer.

See Also

Original **igraph** functions: `vcount()`, `ecount()`

Other modal queries and manipulations: `dualize()`, `mode_addition`, `modes`, `schedule()`

Examples

```
data(chicago1960s)
actor_count(chicago1960s)
event_count(chicago1960s)
```

`mode_projection` *Project an affiliation network onto its actors*

Description

These functions use `bipartite_projection()` to compute the projections of an affiliation network onto the actor or event nodes.

Usage

```
mode_projection(graph, mode = 1, name = "name")

actor_projection(graph, ...)

event_projection(graph, ...)

actor.projection(graph, ...)

event.projection(graph, ...)
```

Arguments

`graph`	An affiliation network.
`mode`	Numeric or character; whether to project onto actors (`1` or `"actors"`) or onto events (`2` or `"events"`).
`name`	Character; the attribute of the actor or event nodes in `graph` to use as names for the nodes in the projection. If `NA`, node IDs are converted to characters and used. If `NULL`, no names are assigned.
`...`	Arguments passed to `mode_projection`.

Value

An `igraph` object with nodes corresponding to one `"type"` of the input `graph`.

See Also

Original **igraph** functions: `bipartite_projection()`

Examples

```
data(chicago1960s)
( tab <- table(V(chicago1960s)$type) )
( proj <- actor_projection(chicago1960s) )
vcount(proj) == tab[1]
( proj <- event_projection(chicago1960s) )
vcount(proj) == tab[2]
```

`nmt_meetings` *Noordin Top meeting attendance network.*

Description

These data record the attendance of 26 individuals at 20 meetings associated with Noordin Mohammad Top.

Format

An affiliation network; see is_an.

Source

Noordin Top Terrorist Network Data. (2011). The Association of Religious Data Archives. https://web.archive.org/web/20140625114607/http://www.thearda.com/Archive/Files/Descriptions/TERRNET.asp

---

`nmt_organizations` *Noordin Top organization membership network.*

Description

These data record the attendance of 67 members of 32 organizations associated with Noordin Mohammad Top.

Format

An affiliation network; see is_an.

Source

Noordin Top Terrorist Network Data. (2011). The Association of Religious Data Archives. https://web.archive.org/web/20140625114607/http://www.thearda.com/Archive/Files/Descriptions/TERRNET.asp

prettify *Convenient plotting aesthetics for affiliation networks*

Description

Given an affiliation network, assign the node and link aesthetics to values that produce a neater visualization through plot.igraph than the **igraph** defaults.

Usage

```
prettify_an(graph)

prettify.an(graph)
```

Arguments

graph	An affiliation network.

Value

The input graph with aesthetic node and link attributes (re)set.

Examples

```
library(igraph)
data(women_clique)
data(whigs)
for (g in list(women_clique, whigs)) {
    plot(prettify_an(g))
}
```

project_census *Project a higher-resolution triad census to a lower-resolution one*

Description

Given a triad census of any scheme, construct a triad census of a coarser (strictly less informative) scheme.

Usage

```
project_census(census, scheme = NULL, add.names = TRUE)

project.census(census, scheme = NULL, add.names = TRUE)

difference_from_full_census(census)

ftc2utc(census)

binary_from_full_census(census)

ftc2stc(census)

simple_from_full_census(census)

ftc2tc(census)

binary_from_difference_census(census)

utc2stc(census)

simple_from_difference_census(census)

utc2tc(census)

simple_from_binary_census(census)

stc2tc(census)
```

Arguments

census	Numeric matrix or vector; an affiliation network triad census. It is treated as binary or simple if its dimensons are 4-by-2 or 4-by-1, respectively, unless otherwise specified by scheme; otherwise it is treated as full.
scheme	Character; the type of triad census provided, matched to "full", "difference" (also "uniformity"), "binary" (also "structural"), or "simple".
add.names	Logical; whether to label the rows and columns of the output matrix.

Details

This function inputes an affiliation network triad census of any scheme and returns a list of triad censuses projected from it (not icluding itself). The schemes are, in order of resolution, *full* (also called the *affiliation network triad census* without qualification), *difference*, *binary*, and *simple*. A final element of the output list is the total number of triads in the affiliation network. Each summary can be recovered from those before it, specifically by aggregating certain matrix entries to form a smaller matrix. The helper functions *_from_*_census() project a census of each scheme to one of each coarser scheme.

Value

A list of `triad_census()` outputs.

Triad censuses

Three triad censuses are implemented for affiliation networks:

- The *full triad census* (Brunson, 2015) records the number of triads of each isomorphism class. The classes are indexed by a partition, $\lambda = (\lambda_1 \leq \lambda_2 \leq \lambda_3)$, indicating the number of events attended by both actors in each pair but not the third, and a positive integer, w, indicating the number of events attended by all three actors. The isomorphism classes are organized into a matrix with rows indexed by λ and columns indexed by w, with the partitions λ ordered according to the *revolving door ordering* (Kreher & Stinson, 1999). The main function triad_census_an (called from `triad_census` when the `graph` argument is an `affiliation_network`) defaults to this census.
- For the analysis of sparse affiliation networks, the full triad census may be less useful than information on whether the extent of connectivity through co-attended events differs between each pair of actors. In order to summarize this information, a coarser triad census can be conducted on classes of triads based on the following congruence relation: Using the indices $\lambda = (x \geq y \geq z)$ and w above, note that the numbers of shared events for each pair and for the triad are $x+w \geq y+w \geq z+w \geq w \geq 0$. Consider two triads congruent if the same subset of these weak inequalities are strictly satisfied. The resulting *difference triad census*, previously called the *uniformity triad census*, implemented as triad_census_difference, is organized into a 8×2 matrix with the strictness of the first three inequalities determining the row and that of the last inequality determining the column.
- A still coarser congruence relation can be used to tally how many are connected by at least one event in each distinct way. This relation considers two triads congruent if each corresponding pair of actors both attended or did not attend at least one event not attended by the third, and if the corresponding triads both attended or did not attend at least one event together. The *binary triad census* (Brunson, 2015; therein called the *structural triad census*), implemented as triad_census_binary, records the number of triads in each congruence class.
- The *simple triad census* is the 4-entry triad census on a traditional (non-affiliation) network indicating the number of triads of each isomorphism class, namely whether the triad contains zero, one, two, or three links. The function simple_triad_census computes the classical (undirected) triad census for an undirected traditional network, or for the actor projection of an affiliation network (if provided), using triad_census; if the result doesn't make sense (i.e., the sum of the entries is not the number of triples of nodes), then it instead uses its own, much slower method.

Each of these censuses can be projected from the previous using the function project_census. A fourth census, called the *uniformity triad census* and implemented as unif_triad_census, is deprecated. Three-actor triad affiliation networks can be constructed and plotted using the triad functions.

The default method for the two affiliation network–specific triad censuses is adapted from the algorithm of Batagelj and Mrvar (2001) for calculating the classical triad census for a directed graph.

References

Kreher, D.L., & Stinson, D.R. (1999). Combinatorial algorithms: generation, enumeration, and search. *SIGACT News*, 30(1), 33–35.

Batagelj, V., & Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks*, 23(3), 237–243.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Other triad census functions: `triad_census()`, `triad_closure_from_census()`, `triad_tallies`

Examples

```
data(women_group)
group_full_census <- triad_census(women_group, scheme = "full")
project_census(group_full_census, scheme = "full")
group_binary_census <- triad_census(women_group, scheme = "binary")
project_census(group_binary_census, scheme = "binary")
```

`project_transitivity` *Affiliation network clustering coefficients*

Description

Each clustering coefficient can be defined as the proportion of "wedges" that are "closed", for suitable definitions of both terms. The main function, `transitivity_an()`, calls one of the wedge functions and computes the global or local clustering coefficient of the given affiliation network, and if the local, then at the given nodes. (`project_transitivity` cheats by using `transitivity()` but produces output consistent with the other variants of `transitivity_an`.)

Usage

```
project_transitivity(graph, type = "global", vids = which(!V(graph)$type))

project.transitivity(graph, type = "global", vids = which(!V(graph)$type))
```

Arguments

`graph`	An affiliation network.
`type`	The type of clustering coefficient (defaults to "global")
`vids`	A subset of actor node ids at which to evaluate the local clustering coefficient.

Value

If `type` is `"global"`, the global clustering coefficient of the network (a single numeric value); if `"local"`, the local clustering coefficients of the actors (a numeric vector); otherwise, a 2-column matrix, each row of which gives the number of wedges and the number of closed wedges centered at each actor.

See Also

Other triad closure functions: dynamic_triad_closure(), transitivity_an(), triad_closure(), triad_closure_from_census()

---

schedule *Actors and their shared events*

Description

Given an affiliation network and a vector of actor node IDs, produce the induced subgraph on the actor nodes together with all event nodes incident to at least two of them. This is called the actors' *schedule*.

Usage

```
schedule(graph, actors = V(graph)[V(graph)$type == FALSE])
```

Arguments

graph	An affiliation network.
actors	A vector of actor nodes in graph.

Value

An igraph object induced from the input graph.

See Also

Other modal queries and manipulations: dualize(), mode_addition, mode_counts, modes

Examples

```
data(women_clique)
schedule(women_clique, actors = V1(women_clique)[seq(3)])
```

scotland1920s *Networks of interlocking directorates*

Description

Five affiliation networks constructed from the multiple directors lists of the top >100 Scottish companies over five two-year intervals between 1904 and 1974.

Format

An affiliation network; see is_an.

Source

Scott, J., & Hughes, M. (1980). *The Anatomy of Scottish Capital*. Croom Helm. http://books.google.com/books?id=59mvAwAAQBAJ

transitivity_an *Affiliation network clustering coefficients*

Description

This function computes a given flavor of transitivity (triadic closure) on a given affiliation network. The calculations are performed locally. Each flavor is defined as a proportion of "wedges" that are "closed", for suitable definitions of both terms. The function `transitivity_an` is a shell that proceeds across actors and computes wedges using the provided `wedgeFun`. These functions count the "wedges", and among them the "closed" ones, centered at a given actor node in a given affiliation network. The triads method `transitivity_an_triads` first classifies every triad centered at each node. The appropriate formula then counts the wedges and closed wedges at each. The method is slower for a single flavor but can be used to produce multiple flavors with negligible additional computational cost. The wedges method `transitivity_an_wedges` relies on a separate "wedge function" for each statistic. The algorithm calls the appropriate wedge function to run over the necessary wedge centers and return a wedge count matrix, which is returned back into `transitivity_an` for outputting.

Usage

```
transitivity_an(
  graph,
  type = "global",
  wedgeFun,
  flavor,
  vids = which(!V(graph)$type),
  add.names = FALSE
)
```

```
transitivity_an_triads(graph, vids = which(!V(graph)$type), flavor)

transitivity_an_wedges(graph, vids = which(!V(graph)$type), wedgeFun)

transitivity.an(
  graph,
  type = "global",
  wedgeFun,
  flavor,
  vids = which(!V(graph)$type),
  add.names = FALSE
)

transitivity.an.triads(graph, vids = which(!V(graph)$type), flavor)

transitivity.an.wedges(graph, vids = which(!V(graph)$type), wedgeFun)

indequ_transitivity(graph, type = "global", vids = which(!V(graph)$type))

indequ.transitivity(graph, type = "global", vids = which(!V(graph)$type))

indstr_transitivity(graph, type = "global", vids = which(!V(graph)$type))

indstr.transitivity(graph, type = "global", vids = which(!V(graph)$type))

inject_transitivity(graph, type = "global", vids = which(!V(graph)$type))

inject.transitivity(graph, type = "global", vids = which(!V(graph)$type))

injequ_transitivity(graph, type = "global", vids = which(!V(graph)$type))

injequ.transitivity(graph, type = "global", vids = which(!V(graph)$type))

injstr_transitivity(graph, type = "global", vids = which(!V(graph)$type))

injstr.transitivity(graph, type = "global", vids = which(!V(graph)$type))

opsahl_transitivity(graph, type = "global", vids = which(!V(graph)$type))

opsahl.transitivity(graph, type = "global", vids = which(!V(graph)$type))

excl_transitivity(graph, type = "global", vids = which(!V(graph)$type))

excl.transitivity(graph, type = "global", vids = which(!V(graph)$type))
```

Arguments

graph	An affiliation network; see `is_an`.

type	Character; the type of clustering coefficient (defaults to "global").
wedgeFun	The wedge function; overrides flavor.
flavor	The flavor of transitivity to be used; overridden by wedgeFun.
vids	A subset of actor node ids at which to evaluate the local clustering coefficient.
add.names	Logical; whether to label the matrix rows and columns.
triads	A matrix of centered triads.

Value

If type is "global", the global clustering coefficient of the network (a single numeric value); if "local", the local clustering coefficients of the actors (a numeric vector); otherwise, a 2-column matrix, each row of which gives the number of wedges and the number of closed wedges centered at each actor.

See Also

Other triad closure functions: dynamic_triad_closure(), project_transitivity(), triad_closure(), triad_closure_from_census()

---

triad *Affiliation network triads*

Description

These functions create and operate on triads in affiliation networks. In this context, a *triad* is the schedule of a subset of three distinct actors.

Usage

```
make_triad(
  lambda,
  w,
  actor_names = c("p", "q", "r"),
 event_names = if (sum(c(lambda, w)) == 0) c() else as.character(1:sum(c(lambda, w)))
)

is_triad(graph)

triad_class(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  as.partition = TRUE,
  format = "list"
)

layout_triad(
```

```
  triad = NULL,
  lambda = NULL,
  w = NULL,
  scale = 0.3,
  angdir = -1,
  rot = -pi/2,
  rot_lambda = c(0, 0, 0),
  rot_w = pi/12
)

plot_triad(
  triad = NULL,
  lambda = NULL,
  w = NULL,
  layout = NULL,
  prettify = TRUE,
  cex = 1,
  scale = 0.3,
  angdir = -1,
  rot = -pi/2,
  rot_lambda = c(0, 0, 0),
  rot_w = pi/12,
  actor_names = c("p", "q", "r"),
 event_names = if (sum(c(lambda, w)) == 0) c() else as.character(1:sum(c(lambda, w))),
  xlim = NULL,
  ylim = NULL,
  ...
)

an_triad(...)

is.triad(graph)

triad.class(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  as.partition = TRUE,
  format = "list"
)

an.triad(...)

layout.triad(
  triad = NULL,
  lambda = NULL,
  w = NULL,
  scale = 0.3,
  angdir = -1,
```

```
  rot = -pi/2,
  rot_lambda = c(0, 0, 0),
  rot_w = pi/12
)

plotTriad(
  triad = NULL,
  lambda = NULL,
  w = NULL,
  layout = NULL,
  prettify = TRUE,
  cex = 1,
  scale = 0.3,
  angdir = -1,
  rot = -pi/2,
  rot_lambda = c(0, 0, 0),
  rot_w = pi/12,
  actor_names = c("p", "q", "r"),
 event_names = if (sum(c(lambda, w)) == 0) c() else as.character(1:sum(c(lambda, w))),
  xlim = NULL,
  ylim = NULL,
  ...
)
```

Arguments

`lambda`	A non-negative integer vector of length three indicating the number of events attended by each pair of actors and not by the third (*exclusive* events).
`w`	A non-negative integer indicating the number of events attended by all three actors (*inclusive* events).
`actor_names`, `event_names`	Actor and event names (actor names default to "p", "q", and "r"; event names default to positive integers).
`graph`	An affiliation network, in some cases must be a triad.
`actors`	A vector of three actor nodes in `graph`.
`as.partition`	Whether to sort the exclusive events, versus reporting them in order of the nodes; defaults to `TRUE`.
`format`	Character matched to "list" or "vector"; whether to return the triad class as a list of $\lambda = (x, y, z)$ and w or as a vector of w, $x = \lambda_1$, $y = \lambda_2$, and $z = \lambda_3$.
`triad`	An affiliation network with exactly three distinct actors.
`scale`	A scaling parameter for the entire plot.
`angdir`	A rotation direction parameter (`-1` for clockwise, `1` for counter-clockwise).
`rot`, `rot_lambda`, `rot_w`	Angular orientation parameters for the entire triad, for the exclusive events of two actors, and for the inclusive events of all three actors.
`layout`	A two-column numeric matrix interpretable as a layout.

prettify	Logical; whether to use prettify_an to adjust the aesthetics of a triad before plotting it.
cex	Node size scaling parameter.
xlim, ylim	Custom bounds on the horizontal and vertical axes.
...	Additional arguments passed to plot.igraph.

Value

An igraph object, a logical value, a matrix of plotting coordinates, or a list of summary parameters lambda and w.

Examples

```
tr <- make_triad(lambda = c(3,1,1), w = 2)
is_triad(tr)
triad_class(tr)
layout_triad(tr)
plot_triad(tr)
```

---

triad_census *Triad census for affiliation networks*

Description

Given an affiliation network, tally all actor triads by isomorphism or other congruence class.

Usage

```
triad_census(graph, ..., add.names = TRUE)

triad_census_an(
  graph,
  scheme = "full",
  method = "batagelj_mrvar",
  ...,
  add.names = TRUE
)

triad.census.an(...)

triad_census_full(graph, method = "batagelj_mrvar", ..., add.names = TRUE)

triad_census_full_batagelj_mrvar(graph, use.integer = FALSE)

triad_census_full_projection(graph, verbose = FALSE)
```

```
triad_census_difference(
  graph,
  method = "batagelj_mrvar",
  ...,
  add.names = TRUE
)

triad_census_difference_batagelj_mrvar(graph, use.integer = FALSE)

triad_census_difference_projection(graph)

unif_triad_census(graph)

unif.triad.census(graph)

triad_census_binary(graph, method = "batagelj_mrvar", ..., add.names = TRUE)

triad_census_binary_batagelj_mrvar(graph, use.integer = FALSE)

triad_census_binary_projection(graph, verbose = FALSE)

str_triad_census(graph)

structural.triad.census(graph)

simple_triad_census(graph, add.names = TRUE)

simple.triad.census(graph, add.names = TRUE)
```

Arguments

graph	An **igraph** object, usually an affiliation network.
...	Additional arguments (currently use.integer and verbose) passed to the method function.
add.names	Logical; whether to label the rows and columns of the output matrix.
scheme	Character; the type of triad census to calculate, matched to "full", "difference" (also "uniformity"), "binary" (also "structural"), or "simple".
method	Character; the triad census method to use. Currently only "batagelj_mrvar" is implemented. "projection" calls an inefficient but reliable implementation in R from the first package version that invokes the simple_triad_census() of the actor_projection() of graph.
use.integer	Logical; whether to use the IntegerMatrix class in **Rcpp** rather than the default NumericMatrix.
verbose	Logical; whether to display progress bars.

Details

The `triad_census_*()` functions implement the several triad censuses described below. Each census is based on a congruence relation among the triads in an affiliation network, and each function returns a matrix (or, in the "simple" case, a vector) recording the number of triads in each congruence class.

The function `triad_census()` masks `triad_census()` but calls it in case `graph` is not an affiliation network.

Value

An integer matrix of counts of triad congruence classes, with row indices reflecting pairwise exclusive events and column indices reflecting triadwise events.

Triad censuses

Three triad censuses are implemented for affiliation networks:

- The *full triad census* (Brunson, 2015) records the number of triads of each isomorphism class. The classes are indexed by a partition, $\lambda = (\lambda_1 \leq \lambda_2 \leq \lambda_3)$, indicating the number of events attended by both actors in each pair but not the third, and a positive integer, w, indicating the number of events attended by all three actors. The isomorphism classes are organized into a matrix with rows indexed by λ and columns indexed by w, with the partitions λ ordered according to the *revolving door ordering* (Kreher & Stinson, 1999). The main function triad_census_an (called from `triad_census` when the `graph` argument is an `affiliation_network`) defaults to this census.
- For the analysis of sparse affiliation networks, the full triad census may be less useful than information on whether the extent of connectivity through co-attended events differs between each pair of actors. In order to summarize this information, a coarser triad census can be conducted on classes of triads based on the following congruence relation: Using the indices $\lambda = (x \geq y \geq z)$ and w above, note that the numbers of shared events for each pair and for the triad are $x+w \geq y+w \geq z+w \geq w \geq 0$. Consider two triads congruent if the same subset of these weak inequalities are strictly satisfied. The resulting *difference triad census*, previously called the *uniformity triad census*, implemented as triad_census_difference, is organized into a 8×2 matrix with the strictness of the first three inequalities determining the row and that of the last inequality determining the column.
- A still coarser congruence relation can be used to tally how many are connected by at least one event in each distinct way. This relation considers two triads congruent if each corresponding pair of actors both attended or did not attend at least one event not attended by the third, and if the corresponding triads both attended or did not attend at least one event together. The *binary triad census* (Brunson, 2015; therein called the *structural triad census*), implemented as triad_census_binary, records the number of triads in each congruence class.
- The *simple triad census* is the 4-entry triad census on a traditional (non-affiliation) network indicating the number of triads of each isomorphism class, namely whether the triad contains zero, one, two, or three links. The function simple_triad_census computes the classical (undirected) triad census for an undirected traditional network, or for the actor projection of an affiliation network (if provided), using triad_census; if the result doesn't make sense (i.e., the sum of the entries is not the number of triples of nodes), then it instead uses its own, much slower method.

Each of these censuses can be projected from the previous using the function project_census. A fourth census, called the *uniformity triad census* and implemented as unif_triad_census, is deprecated. Three-actor triad affiliation networks can be constructed and plotted using the triad functions.

The default method for the two affiliation network–specific triad censuses is adapted from the algorithm of Batagelj and Mrvar (2001) for calculating the classical triad census for a directed graph.

References

Kreher, D.L., & Stinson, D.R. (1999). Combinatorial algorithms: generation, enumeration, and search. *SIGACT News*, 30(1), 33–35.

Batagelj, V., & Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks*, 23(3), 237–243.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Original **igraph** functions: `triad_census()`

Other triad census functions: `project_census()`, `triad_closure_from_census()`, `triad_tallies`

Examples

```
data(women_clique)
(tc <- triad_census(women_clique, add.names = TRUE))
sum(tc) == choose(vcount(actor_projection(women_clique)), 3)
```

---

triad_closure *Triad closure for affiliation networks*

Description

Given an affiliation network and a vector of actor node IDs, calculate a specified measure of triad closure centered at the nodes.

Usage

```
triad_closure(graph, ...)

triad_closure_an(graph, method = "wedges", ...)

triad_closure_via_triads(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...
)

triad_closure_from_centered_triads(
```

```
  triad_list,
  type = "global",
  ...,
  measure = NULL,
  triads.fun = NULL
)

triad_closure_via_wedges(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global",
  ...,
  measure = NULL,
  wedges.fun = NULL
)

triad_closure_watts_strogatz(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_classical(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_opsahl(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_twomode(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_liebig_rao_0(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_unconnected(
  graph,
```

```
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_liebig_rao_3(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_completely_connected(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_exclusive(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)

triad_closure_projection(
  graph,
  actors = V(graph)[V(graph)$type == FALSE],
  type = "global"
)
```

Arguments

graph	An affiliation network.
...	Measure specifications passed to wedges.
method	Character; for a given measure, whether to use the measure-specific wedge census ("wedges") or the measure-specific calculation on the centered triad census ("triads").
actors	A vector of actor nodes in graph.
type	The type of statistic, matched to "global", "local", or "raw".
triad_list	A list of triad isomorphism classes in matrix format, as produced by centered_triads.
measure	Character; the measure of triad closure, used as the suffix * to triad_closure_* Matched to "classical" (also "watts_strogatz"), "twomode" (also "opsahl"), "unconnected" (also "liebig_rao_0"), "completely_connected" (also "liebig_rao_3"), or "exclusive".
triads.fun	A custom triad closure calculation. It must accept a vector of *centered* triad isomorphism classes, encoded as vectors w, x, y, and z, and return a 2-row integer matrix recording the number of wedges of the desired measure centered at the second actor, and involving the other two actors, of each triad.

`wedges.fun` A custom wedge census function. It must accept an affiliation network `graph` and a single actor node ID `actor` and may have any additional parameters. It must return a named list with values `wedges` a numeric matrix of node IDs whose columns record the wedges centered at `actor` and `closed` a logical vector recording whether each wedge is closed. Overrides `measure`.

Details

The `triad_closure_*` functions implement the several measures of triad closure described below. Each function returns a single global statistic, a vector of local statistics, or a matrix of local denominators and numerators from which the global and local statistics can be recovered.

The function `triad_closure_projection` recapitulates `triad_closure_watts_strogatz()` by invoking the `bipartite_projection()` and `transitivity()` functions in **igraph**.

Value

If `type` is `"global"`, the global statistic for `graph` (a single numeric value); if `"local"`, the local statistics for `actors` (a numeric vector); if `"raw"`, a 2-column matrix, each row of which gives the number of wedges and of closed wedges centered at `actors`.

Measures of triad closure

Each measure of triad closure is defined as the proportion of wedges that are closed, where a *wedge* is the image of a specified two-event triad W under a specified subcategory of graph maps C subject to a specified congruence relation , and where a wedge is *closed* if it is the image of such a map that factors through a canonical inclusion of W to a specified self-dual three-event triad X.

The alcove, wedge, maps, and congruence can be specified by numerical codes as follows (no plans exist to implement more measures than these):

- `alcove`:
  - `0`: $T_{(1,1,1),0}$
  - `1`: $T_{(1,1,0),1}$ (**not yet implemented**)
  - `2`: $T_{(1,0,0),2}$ (**not yet implemented**)
  - `3`: $T_{(0,0,0),3}$ (**not yet implemented**)
- `wedge`:
  - `0`: $T_{(1,1,0),0}$
  - `1`: $T_{(1,0,0),1}$ (**not yet implemented**)
  - `2`: $T_{(0,0,0),2}$ (**not yet implemented**)
- `maps`:
  - `0`: all graph maps (injective on actors)
  - `1`: injective graph maps
  - `2`: induced injective graph maps
- `congruence`:
  - `0`: same actor and event images (equivalence)
  - `1`: same actor images, structurally equivalent event images

  - 2: same actor images

Some specifications correspond to statistics of especial interest:

- `0,0,0,2`: the classical clustering coefficient (Watts & Strogatz, 1998), evaluated on the unipartite actor projection
- `0,0,1,0`: the two-mode clustering coefficient (Opsahl, 2013)
- `0,0,2,0`: the unconnected clustering coefficient (Liebig & Rao, 2014)
- `3,2,2,0`: the completely connected clustering coefficient (Liebig & Rao, 2014) (**not yet implemented**)
- `0,0,2,1`: the exclusive clustering coefficient (Brunson, 2015)
- `0,0,2,2`: the exclusive clustering coefficient

See Brunson (2015) for a general definition and the aforecited references for discussions of each statistic.

References

Watts, D.J., & Strogatz, S.H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393(6684), 440–442.

Opsahl, T. (2013). Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2), 159–167. Special Issue on Advances in Two-mode Social Networks.

Liebig, J., & Rao, A. (2014). Identifying influential nodes in bipartite networks using the clustering coefficient. Pages 323–330 of: *Proceedings of the tenth international conference on signal-image technology and internet-based systems*.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Original **igraph** functions: `transitivity()`

Other triad closure functions: `dynamic_triad_closure()`, `project_transitivity()`, `transitivity_an()`, `triad_closure_from_census()`

Examples

```
data(women_clique)
mapply(
  triad_closure,
  measure = c("classical", "twomode", "unconnected", "exclusive"),
  MoreArgs = list(graph = women_clique, type = "local")
)
data(women_group)
cbind(
  triad_closure_watts_strogatz(women_group, type = "local"),
  triad_closure_opsahl(women_group, type = "local"),
  triad_closure_liebig_rao_0(women_group, type = "local"),
  triad_closure_exclusive(women_group, type = "local")
)
```

`triad_closure_from_census` *Global triad closure from a triad census*

Description

Given a triad census of a suitable scheme, calculate a global measure of triad closure for the associated affiliation network.

Usage

```
triad_closure_from_census(
  census,
  scheme = NULL,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  measure = NULL,
  open.fun = NULL,
  closed.fun = NULL,
  counts = FALSE
)

triad_closure_from_simple_census(
  census,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  open.fun = NULL,
  closed.fun = NULL,
  counts = FALSE
)

triad_closure_from_binary_census(
  census,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  open.fun = NULL,
  closed.fun = NULL,
  counts = FALSE
)

triad_closure_from_difference_census(
```

```
  census,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  open.fun = NULL,
  closed.fun = NULL,
  counts = FALSE
)

triad_closure_from_full_census(
  census,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  open.fun = NULL,
  closed.fun = NULL,
  counts = FALSE
)

wedges_from_full_census(census, open.fun, closed.fun)

wedges_from_census(...)

wedgecount_census(...)

wedgecount.census(...)

triad_closure_from_census_original(
  census,
  scheme = NULL,
  alcove = 0,
  wedge = 0,
  maps = 0,
  congruence = 0,
  measure,
  open.fun,
  closed.fun,
  counts = FALSE
)

transitivity_from_census(...)

transitivity.census(...)
```

Arguments

census	Numeric matrix or vector; an affiliation network triad census. It is treated as binary or simple if its dimensons are 4-by-2 or 4-by-1, respectively, unless otherwise specified by scheme; otherwise it is treated as full.
scheme	Character; the type of triad census provided, matched to "full", "difference" (also "uniformity"), "binary" (also "structural"), or "simple".
alcove, wedge, maps, congruence	Choice of alcove, wedge, maps, and congruence (see Details).
measure	Character; the measure of triad closure (matched to "classical", "watts_strogatz", "twomode", "opsahl", "unconnected", "liebig_rao_0", "completely_connected", "liebig_rao_3", "exclusive", "allact", "indequ", "indstr", "inject", "injequ", or "injstr"). Overrides alcove, wedge, maps, and congruence.
open.fun, closed.fun	Functions to calculate the open and closed wedge count for a triad (when scheme is "full") or a triad census (otherwise), in order to calculate a custom measure of triad closure. Override measure.
counts	Logical; whether to return open and closed wedge counts instead of the quotient.
...	Arguments passed from deprecated functions to their replacements.

Details

Each global measure of triad closure can be recovered from the full triad census, and some can be recovered from smaller censuses. This function verifies that a given census is sufficient to recover a given measure of triad closure and, if it is, returns its value.

Value

Output equivalent to that of triad_closure().

Triad censuses

Three triad censuses are implemented for affiliation networks:

- The *full triad census* (Brunson, 2015) records the number of triads of each isomorphism class. The classes are indexed by a partition, $\lambda = (\lambda_1 \leq \lambda_2 \leq \lambda_3)$, indicating the number of events attended by both actors in each pair but not the third, and a positive integer, w, indicating the number of events attended by all three actors. The isomorphism classes are organized into a matrix with rows indexed by λ and columns indexed by w, with the partitions λ ordered according to the *revolving door ordering* (Kreher & Stinson, 1999). The main function triad_census_an (called from triad_census when the graph argument is an affiliation_network) defaults to this census.
- For the analysis of sparse affiliation networks, the full triad census may be less useful than information on whether the extent of connectivity through co-attended events differs between each pair of actors. In order to summarize this information, a coarser triad census can be conducted on classes of triads based on the following congruence relation: Using the indices $\lambda = (x \geq y \geq z)$ and w above, note that the numbers of shared events for each pair and for the triad are $x+w \geq y+w \geq z+w \geq w \geq 0$. Consider two triads congruent if the same subset of

these weak inequalities are strictly satisfied. The resulting *difference triad census*, previously called the *uniformity triad census*, implemented as triad_census_difference, is organized into a 8×2 matrix with the strictness of the first three inequalities determining the row and that of the last inequality determining the column.

- A still coarser congruence relation can be used to tally how many are connected by at least one event in each distinct way. This relation considers two triads congruent if each corresponding pair of actors both attended or did not attend at least one event not attended by the third, and if the corresponding triads both attended or did not attend at least one event together. The *binary triad census* (Brunson, 2015; therein called the *structural triad census*), implemented as triad_census_binary, records the number of triads in each congruence class.
- The *simple triad census* is the 4-entry triad census on a traditional (non-affiliation) network indicating the number of triads of each isomorphism class, namely whether the triad contains zero, one, two, or three links. The function simple_triad_census computes the classical (undirected) triad census for an undirected traditional network, or for the actor projection of an affiliation network (if provided), using triad_census; if the result doesn't make sense (i.e., the sum of the entries is not the number of triples of nodes), then it instead uses its own, much slower method.

Each of these censuses can be projected from the previous using the function project_census. A fourth census, called the *uniformity triad census* and implemented as unif_triad_census, is deprecated. Three-actor triad affiliation networks can be constructed and plotted using the triad functions.

The default method for the two affiliation network–specific triad censuses is adapted from the algorithm of Batagelj and Mrvar (2001) for calculating the classical triad census for a directed graph.

Measures of triad closure

Each measure of triad closure is defined as the proportion of wedges that are closed, where a *wedge* is the image of a specified two-event triad W under a specified subcategory of graph maps C subject to a specified congruence relation , and where a wedge is *closed* if it is the image of such a map that factors through a canonical inclusion of W to a specified self-dual three-event triad X.

The alcove, wedge, maps, and congruence can be specified by numerical codes as follows (no plans exist to implement more measures than these):

- `alcove`:
  - `0`: $T_{(1,1,1),0}$
  - `1`: $T_{(1,1,0),1}$ **(not yet implemented)**
  - `2`: $T_{(1,0,0),2}$ **(not yet implemented)**
  - `3`: $T_{(0,0,0),3}$ **(not yet implemented)**
- `wedge`:
  - `0`: $T_{(1,1,0),0}$
  - `1`: $T_{(1,0,0),1}$ **(not yet implemented)**
  - `2`: $T_{(0,0,0),2}$ **(not yet implemented)**
- `maps`:
  - `0`: all graph maps (injective on actors)
  - `1`: injective graph maps
  - `2`: induced injective graph maps

- `congruence`:
  - `0`: same actor and event images (equivalence)
  - `1`: same actor images, structurally equivalent event images
  - `2`: same actor images

Some specifications correspond to statistics of especial interest:

- `0,0,0,2`: the classical clustering coefficient (Watts & Strogatz, 1998), evaluated on the unipartite actor projection
- `0,0,1,0`: the two-mode clustering coefficient (Opsahl, 2013)
- `0,0,2,0`: the unconnected clustering coefficient (Liebig & Rao, 2014)
- `3,2,2,0`: the completely connected clustering coefficient (Liebig & Rao, 2014) (**not yet implemented**)
- `0,0,2,1`: the exclusive clustering coefficient (Brunson, 2015)
- `0,0,2,2`: the exclusive clustering coefficient

See Brunson (2015) for a general definition and the aforecited references for discussions of each statistic.

References

Kreher, D.L., & Stinson, D.R. (1999). Combinatorial algorithms: generation, enumeration, and search. *SIGACT News*, 30(1), 33–35.

Batagelj, V., & Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Social Networks*, 23(3), 237–243.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

Watts, D.J., & Strogatz, S.H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393(6684), 440–442.

Opsahl, T. (2013). Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2), 159–167. Special Issue on Advances in Two-mode Social Networks.

Liebig, J., & Rao, A. (2014). Identifying influential nodes in bipartite networks using the clustering coefficient. Pages 323–330 of: *Proceedings of the tenth international conference on signal-image technology and internet-based systems*.

Brunson, J.C. (2015). Triadic analysis of affiliation networks. *Network Science*, 3(4), 480–508.

See Also

Other triad census functions: `project_census()`, `triad_census()`, `triad_tallies`

Other triad closure functions: `dynamic_triad_closure()`, `project_transitivity()`, `transitivity_an()`, `triad_closure()`

whigs *Membership network of American Whigs.*

Description

These data record the membership of 136 colonial Americans in 5 Whig organizations.

Format

An affiliation network; see is_an.

Source

Fischer, D.H. (1994). *Paul Revere's Ride*. Oxford University Press. https://books.google.com/books?id=knC-kTFI9_gC https://github.com/kjhealy/revere

women_clique *Clique of women connected by event coattendance in Old City.*

Description

These data record the participation of 5 African-American women in 5 social activities.

Format

An affiliation network; see is_an.

Source

Davis, A., Gardner, B.B., & Gardner, M.R. (1941). *Deep South: A Social Anthropological Study of Caste and Class*. Columbia, SC: University of South Carolina Press. http://books.google.com/books?id=HGIdAAAAIAAJ

`women_group` *Group of women connected by event coattendance in Old City.*

Description

These data record the attendance of 18 white women at 14 social events over a 9-month period.

Format

An affiliation network; see is_an. Events carry a `time` attribute equal to the number of days since the previous 31 December.

Source

Davis, A., Gardner, B.B., & Gardner, M.R. (1941). *Deep South: A Social Anthropological Study of Caste and Class*. Columbia, SC: University of South Carolina Press. http://books.google.com/books?id=HGIdAAAAIAAJ

Index