Package ‘NetOrigin’

July 7, 2016

Title Origin Estimation for Propagation Processes on Complex Networks

Description Performs network-based source estimation. Different approaches are available: effective distance median, recursive backtracking, and centrality-based source estimation. Additionally, we provide public transportation network data as well as methods for data preparation, source estimation performance analysis and visualization.

Version 1.0-2

Depends R (>= 3.2.2)

Imports igraph, Hmisc, colorspace

License GPL-3

LazyData true

RoxygenNote 5.0.1

Collate '0_helper_net.r' 'NetOrigin.r' 'origin_helper.r'
  'origin_methods.r' 'distance.r' 'data.r' 'data_handling.r'
  'robustness.r'

NeedsCompilation no

Author Juliane Manitz [aut, cre], Jonas Harbering [ctb]

Maintainer Juliane Manitz <r@manitz.org>

Repository CRAN

Date/Publication 2016-07-07 07:39:10

R topics documented:

  aggr_data ......................................................... 2
  analyze_ptn ..................................................... 3
  delay-data ....................................................... 4
  eff_dist .......................................................... 5
  NetOrigin ....................................................... 6
  origin ........................................................... 7
  origin-methods .................................................. 10
  origin_multiple ............................................... 11
  performance ..................................................... 12
aggr_data

Description

convert individual event information to aggregated information per network node

Usage

aggr_data(dat, from = NULL, cumsum = TRUE)

Arguments

*dat*  
data.frame with variables 'node', 'time', 'delay', events data with single events with count magnitude

*from*  
character in strftime format, e.g. "2014-06-12 16:15", data is subsetted accordingly before aggregation

*cumsum*  
logical indicating whether data is aggregated by cumulative sum, default is TRUE

Value

data.frame of dimension (T×K), where T is the number of observation times and K the number of network nodes. Thus, each row represents a snapshot of the spreading process at a specific observation time with the event magnitude observed at the network nodes. Rownames are observation times, colnames are node names.

See Also

Other data handling: read_DB_data
**analyze_ptn**

**analyze public transportation network characteristics**

**Description**

analyze public transportation network characteristics

**Usage**

analyze_ptn(g)

**Arguments**

- **g**  
  *igraph* object, network graph representing the public transportation network, vertices represent stations, which are linked by an edge if there is a direct transfer between them

**Value**

'data.frame': 1 obs. of 7 variables:

- **vcount** number of nodes,
- **ecount** number of edges,
- **density** network graph density,
- **av_deg** average degree,
- **av_cent** average unit betweenness,
- **av_spl** average shortest path length,
- **diam** diameter, and
- **trans** transitivity.

**References**

Details to the computation and interpretation can be found in:


**See Also**

Other network helper: plot_ptn
Examples

```r
data(ptnAth)
analyze_pntn(ptnAth)

data(ptnGoe)
analyze_pntn(ptnGoe)
```

delay-data  
*Delay propagation data examples simulated by LinTim software*

Description

Delay propagation data examples simulated by LinTim software

delayAth Delay propagation data generated on the Athens metro network by LinTim software
delayGoe Delay propagation data generated on the Goettingen bus system by LinTim software

Details

delayAth Delay data on the Athens metro network. Propagation simulation under consideration of security distances and fixed-waiting time delay management. 'data.frame' with 510 observations (10 sequential time pictures for delay spreading pattern from 51 stations) of 53 variables (k0 true source, time, delays at 51 stations).
delayGoe Delay data on the directed Goettingen bus system. Propagation simulation under consideration of security distances and fixed-waiting time delay management. 'data.frame’ with 2570 observations (10 sequential time pictures for delay spreading pattern from 257 stations) of 259 variables (k0 true source, time, delays at 257 stations).

Author(s)

Jonas Harbering

Source


References

Manitz, J., J. Harbering, M. Schmidt, T. Kneib, and A. Schoebel (2016). Source Estimation for Propagation Processes on Complex Networks with an Application to Delays in Public Transportation Systems. Accepted at JRSS-C.

See Also

* ptn-data
### Examples

```
## Not run:
# compute effective distance
data(ptnAth)
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
# apply source estimation
if (requireNamespace("plyr", quietly = TRUE)) {
  data(delayAth)
  res <- alply(.data=delayAth[-c(1:2)], .margins=1, .fun=origin_edm, distance=eff,
               silent=TRUE, .progress='text')
  perfAth <- ldply(Map(performance, x = res, start = as.list(delayAth$K0),
                    list(graph = ptnAth)))
}

## End(Not run)
```

```
## Not run:
# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply source estimation
if (requireNamespace("plyr", quietly = TRUE)) {
  data(delayGoe)
  res <- alply(.data=delayGoe[-c(1:2)], .margins=1, .fun=origin_edm, distance=eff,
               silent=TRUE, .progress='text')
  perfGoe <- ldply(Map(performance, x = res, start = as.list(delayGoe$K0),
                     list(graph = ptnGoe)))
}

## End(Not run)
```

---

**eff_dist**

**Computation of effective path distance**

---

**Description**

`eff_dist` computes the effective distance between all nodes in the network

`eff_dijkstra` computes the shortest effective paths using the dijkstra algorithm

`spd_dijkstra` computes the shortest paths using the dijkstra algorithm

---

**Usage**

```r
eff_dist(p)

eff_dijkstra(p, start)

spd_dijkstra(p, start)
```
Arguments

\( p \) 
numeric matrix, representing the transition probability matrix for the network graph

\( \text{start} \) 
start of path

Value

A numeric matrix, representing the effective distance between all nodes in the network graph.

References


Examples

```r
# compute effective shortest path distance
data(ptnAth)
require(igraph)
net <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- net/rowSums(net)
eff <- eff_dist(p)

# compute shortest path distance
data(ptnAth)
athnet <- as_adj(ptnAth, sparse=FALSE)
spd <- spd_dijkstra(athnet, start=1)

# compare calculations with the one from igraph
spd_igraph <- igraph::distances(ptnAth, v=1, algorithm='dijkstra')
all(spd[[1]] == spd_igraph)
```

Description

Performs different approaches for network-based source estimation: effective distance median, recursive backtracking, and centrality-based source estimation. Additionally, we provide public transportation network data as well as methods for data preparation, source estimation performance analysis and visualization.
Details

The main function for origin estimation of propagation processes on complex network is `origin`. Different methods are available: effective distance median (`'edm'`), recursive backtracking (`'backtracking'`), and centrality-based source estimation (`'centrality'`). For more details on the methodological background, we refer to the corresponding publications.

Author(s)

Juliane Manitz with contributions by Jonas Harbering

References


Description

This is the main function for origin estimation for propagation processes on complex networks. Different methods are available: effective distance median (`'edm'`), recursive backtracking (`'backtracking'`), and centrality-based source estimation (`'centrality'`). For details on the methodological background, we refer to the corresponding publications.

`origin_edm` for effective distance-median origin estimation (Manitz et al., 2016)

`origin_backtracking` for recursive backtracking origin estimation (Manitz et al., 2016)

`origin_centrality` for centrality-based origin estimation (Comin et al., 2011)

Usage

`origin(events, type = c("edm", "backtracking", "centrality"), ...)`

`origin_edm(events, distance, silent = TRUE)`

`origin_backtracking(events, graph, start_with_event_node = TRUE, silent = TRUE)`

`origin_centrality(events, graph, silent = TRUE)`
Arguments

- **events**: numeric vector of event counts at a specific time point
- **type**: character specifying the method, 'edm', 'backtracking' and 'centrality' are available.
- **distance**: numeric matrix specifying the distance matrix (for type='edm')
- **silent**: logical, should the messages be suppressed?
- **graph**: igraph object specifying the underlying network graph (for type='backtracking' and type='centrality')
- **start_with_event_node**: logical specifying whether backtracking only starts from nodes that experienced events (for type='backtracking')

Value

- **origin_edm** returns an object of class `origin`, list with
  - **est** origin estimate
  - **aux.data.frame** with auxiliary variables
    - `id` as node identifier,
    - `events` for event magnitude,
    - `wmean` for weighted mean,
    - `wvar` for weighted variance, and
    - `mdist` mean distance from a node to all other nodes.
  - **type = 'edm'** effective distance median origin estimation
- **origin_backtracking** returns an object of class `origin`, list with
  - **est** origin estimate
  - **aux.data.frame** with auxiliary variables
    - `id` as node identifier,
    - `events` for event magnitude, and
    - `bcount` for backtracking counts, how often backtracking identifies this source node.
  - **type = 'backtracking'** backtracking origin estimation
- **origin_centrality** returns an object of class `origin`, list with
  - **est** origin estimate
  - **aux.data.frame** with auxiliary variables
    - `id` as node identifier,
    - `events` for event magnitude, and
    - `cent` for node centrality (betweenness divided degree).
  - **type = 'centrality'** centrality-based origin estimation
Author(s)

Juliane Manitz with contributions by Jonas Harbering

References


See Also

Other origin-est: origin_multiple

Examples

data(delayGoe)

# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply effective distance median source estimation
om <- origin(events=delayGoe[10,-c(1:2)], type='edm', distance=eff)
summary(om)
plot(om, 'mdist',start=1)
plot(om, 'wvar',start=1)
performance(om, start=1, graph=ptnGoe)

# backtracking origin estimation (Manitz et al., 2016)
ob <- origin(events=delayGoe[10,-c(1:2)], type='backtracking', graph=ptnGoe)
summary(ob)
plot(ob, start=1)
performance(ob, start=1, graph=ptnGoe)

# centrality-based origin estimation (Comin et al., 2011)
oc <- origin(events=delayGoe[10,-c(1:2)], type='centrality', graph=ptnGoe)
summary.oc)
plot.oc, start=1)
performance.oc, start=1, graph=ptnGoe)
Description

print produces an output for objects of class origin.
summarize produces an object summary for objects of class origin.
plot generates an illustration of an origin object using the variable to be optimized.
performance evaluates an object of class origin and returns a data.frame identifying correct estimation, and computing rank and distance of correct detection.

Usage

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

## S3 method for class 'origin'
summary(object, x = object, ...)

## S3 method for class 'origin'
plot(x, y = "id", start, ...)

## S3 method for class 'origin'
performance(x, start, graph = NULL, ...)
```

Arguments

- `x` object of class `origin`, origin estimation object from function `origin_xxx`
- `...` further arguments to be passed to default plot function
- `object` object of class `origin`, origin estimation object from function `origin_xxx`; passed to `x`
- `y` character specifying the variable being plotted at the y-axis; options are 'id' for node identifier (default), 'mdist' for mean distance (only available for `origin_edm`) or 'wvar' for weighted variance (only available for `origin_edm`)
- `start` numeric, giving the node of the true origin
- `graph` `igraph` object specifying the underlying network graph with attribute 'length' on edges for calculation of distance to the correct origin

Value

`performance.origin` returns a data.frame with variables

- `origin = start` representing the true origin,
- `est` the estimated node of origin,
origin_multiple

- hitt logical indicating whether origin estimation is correct or not,
- rank rank of correct detection,
- spj number of segments from estimated origin to true origin (requires an igraph object),
- dist distance along the shortest path from estimated origin to true origin (igraph edge attribute length)

See Also

origin plot_performance

Examples

data(ptnGoe)
data(delayGoe)

res <- origin(events=delayGoe[,1:2], type='centrality', graph=ptnGoe)
res

summary(res)
plot(res, start=1)
performance(res, start=1, graph=ptnGoe)

origin_multiple Muliple origin estimation using community partitioning

Description

Multiple origin estimation using community partitioning

Usage

origin_multiple(events, type = c("edm", "backtracking", "centrality"), graph, no = 2, distance, fast = TRUE, ...)

Arguments

events numeric vector of event counts at specific time point
type character specifying the method, 'edm', 'backtracking' and 'centrality' are available.
graph igraph object specifying the underlying network graph
no numeric specifying the number of supposed origins
distance numeric matrix specifying the distance matrix
fast logical specifying community partitioning algorithm, default is 'TRUE' that uses fastgreedy.community, 'FALSE' refers to leading.eigenvector.community
... parameters to be passed to origin methods origin_edm, origin_backtracking or origin_centrality
Value

`origin_multiple` returns an list object with objects of class `origin` of length `no`.

References


See Also

Other origin-est: `origin`

Examples

```r
data(ptnAth)
# backtracking
origin_multiple(events=delayAth[10,-c(1:2)], type='backtracking', graph=ptnAth, no=2)
# edm
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
origin_multiple(events=delayAth[10,-c(1:2)], type='edm', graph=ptnAth, no=2, distance=eff)
```

---

**performance**

generic method for performance evaluation

Description

generic method for performance evaluation

Usage

`performance(x, ...)`

Arguments

- `x` object
- `...` further arguments

See Also

`origin-methods plot_performance`
plot  

**Description**

generic method for plots

**Usage**

plot(x, y, ...)

**Arguments**

- **x**: object
- **y**: object
- **...**: further arguments

---

**plot_performance**  

*A plot method combining a time series of performance results.*

**Description**

A plot method combining a time series of performance results.

**Usage**

plot_performance(x, var = "rank", add = FALSE, offset = NULL, 
                 log = FALSE, col = 1, ylim = NULL, text.padding = 0.9, ...)

**Arguments**

- **x**: data.frame obtained by combined results from `performance.origin` with variables `X1` for time point, `start` for true origin, `est` for estimated origin, and performance variables
- **var**: character, variable to be plotted, `performance.origin` returns `rank`, `spj`, and `dist`, default is 'rank'
- **add**: logical, should be added to another performance plot
- **offset**: POSIXct, starting time of spreading
- **log**: logical, should y-axis be logarithmized?
- **col**: numeric or character, color of lines
- **ylim**: numeric vector, range of y axis
- **text.padding**: a numeric value specifying the factor for the text position relative to the y values
- **...**: further graphical parameters passed to default `plot` function
Examples

```r
# Not run:
### delays on Goettingen bus network
# compute effective distance
data(ptnGoe)
goenet <- igraph::as_adjacency_matrix(ptnGoe, sparse=FALSE)
p <- goenet/rowSums(goenet)
eff <- eff_dist(p)
# apply source estimation
data(delayGoe)
if (requireNamespace("aplyr", quietly = TRUE)) {
  res <- alply(.data=delayGoe[11:20,-c(1:2)], .margins=1, .fun=origin_edm,
              distance=eff, silent=TRUE, .progress='text')
  perfGoe <- ldply(Map(performance, x = res, start = 2, list(graph = ptnGoe)))
  # performance plots
  plot_performance(perfGoe, var='rank', ylab='rank of correct detection', text.padding=0.5)
  plot_performance(perfGoe, var='dist', ylab='distance to correct detection')
}

### delays on Athens metro network
# compute effective distance
data(ptnAth)
athnet <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- athnet/rowSums(athnet)
eff <- eff_dist(p)
# apply source estimation
data(delayAth)
if (requireNamespace("aplyr", quietly = TRUE)) {
  res <- alply(.data=delayAth[11:20,-c(1:2)], .margins=1, .fun=origin_edm,
              distance=eff, silent=TRUE, .progress='text')
  perfAth <- ldply(Map(performance, x = res, start = as.list(delayAth$k0),
                   list(graph = ptnAth)))
  # performance plots
  plot_performance(perfAth, var='rank', ylab='rank of correct detection', text.padding=0.5)
  plot_performance(perfAth, var='dist', ylab='distance to correct detection')
}

# End(Not run)
```

---

**plot_ptn**

A plot method for public transportation networks (PTNs).

**Description**

A plot method for public transportation networks (PTNs).

**Usage**

```r
plot_ptn(g, color.coding = NULL, color.scheme = rev(sequential_hcl(5)),
         legend = FALSE, ...)```

Arguments

- **g**: igraph object, network graph representing the public transportation network, vertices represent stations, which are linked by an edge if there is a direct transfer between them.
- **color.coding**: numeric vector with length equal to the number of network nodes.
- **color.scheme**: character vector of length 5 indicating the vertex.color, default is rev(sequential_hcl(5)).
- **legend**: logical indicating whether legend for color-coding should be added or not.
- ... further arguments to be passed to plot.igraph.

See Also

Other network helper: analyze_ptn

Examples

```r
data(ptnAth)
plot_ptn(ptnAth)

data(ptnGoe)
plot_ptn(ptnGoe)
```

ptn-data

*Public transportation network datasets from LinTim software (Integrated Optimization in Public Transportation)*

Description

Public transportation network datasets from LinTim software (Integrated Optimization in Public Transportation).

- **ptnAth**: The data of the Athens Metro, consisting of 51 nodes and 52 edges.
  - Vertex attributes: station name, additional station info.
  - Edge attributes: track length (in meter), minimal and maximal time required to pass the track (in minutes).

- **ptnGoe**: The data of the Goettingen bus network, consisting of 257 nodes and 548 edges.
  - Vertex attributes: station name.
  - Edge attributes: track length (in meter), minimal and maximal time required to pass the track (in minutes).

Author(s)

Juliane Manitz and Jonas Harbering
Source

Public transportation network datasets are extracted from LinTim software (Integrated Optimization in Public Transportation; http://lintim.math.uni-goettingen.de/index.php?go=data&lang=en). Special thanks to Anita Schoebel for making the data available.

The Athens Metro data was collected by Konstantinos Gkoumas.

The Goettingen bus network data was collected by Barbara Michalski.

See Also

delay-data

Examples

# Athens metro system
data(ptnAth)
analyze_ptn(ptnAth)
plot_ptn(ptnAth)

# Goettingen bus system
data(ptnGoe)
analyze_ptn(ptnGoe)
plot_ptn(ptnGoe)

read_DB_data

Description

Reads a data file as provided by 'Deutsche Bahn' (for internal use).

Usage

read_DB_data(file)

Arguments

file character with path and file name containing the variables for 'stationID', 'date', 'hour', 'minutes', and 'delay'

Value

data.frame with variables 'node', 'time', 'delay'

See Also

Other data Handling: aggr_data
robustness

Description

run robustness analysis for a source estimate by subsampling individual events.

Usage

robustness(x, type = c("edm", "backtracking", "centrality"), prop, n = 100, ...)

Arguments

x data.frame, dataset with individual events and their magnitude, to be passed to aggr_data

type character, specifying the method, 'edm', 'backtracking' and 'centrality' are available.

prop numeric, value between zero and one, proportion of events to be sampled

n numeric, number of resamplings

... parameters to be passed to origin methods origin_edm, origin_backtracking or origin_centrality

Details

We create subsamples of individual events and their magnitude using a sampling proportion p in [0, 1]. After aggregating the data, we apply the source estimation approach. Using this result, we deduce the relative frequency of how often the source estimate obtained with the complete data set can be recovered by source estimation based on the subsample. Thus, the estimate robustness is assessed by the proportion of estimate recovery.

Value

data.frame with columns

- est origin estimated when all data is evaluated

- rob estimate uncertainty, computed as the proportion of resamplings when origin estimate was recovered

See Also

robustness-methods
**Examples**

```r
# generate random delay data
data <- data.frame(node = sample(size = 500, make.names(V(ptnAth)$name), replace = TRUE),
                  time = sample(size = 500, 1:10, replace = TRUE),
                  delay = rexp(500, rate=10))

# compute effective distance
net <- igraph::as_adjacency_matrix(ptnAth, sparse=FALSE)
p <- net/rowSums(net)
co <- eff_dist(p)
colnames(co) <- paste('x.', colnames(co), sep='')

# run robustness analysis
r5 <- robustness(x=dat, type='edm', prop=0.5, n=10, distance=co)
summary(r5)
plot(r5)

# compare results
r9 <- robustness(x=dat, type='edm', prop=0.9, n=10, distance=co)
plot(r9, add=TRUE, col='gray')
```

---

**robustness-methods**

*methods for robustness estimation objects of class robustness*

**Description**

- `print` produces an output for objects of class `robustness`
- `summary` produces an object summary for objects of class `robustness`
- `plot` produces a time series plot of the robustness estimate object

**Usage**

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

## S3 method for class 'robustness'
summary(object, x = object, ...)

## S3 method for class 'robustness'
plot(x, y = NULL, add = FALSE, ...)
```

**Arguments**

- `x` data.frame obtained by `robustness`, robustness estimation object for source estimation from function `robustness`
Computes the variance of a weighted mean following the definition by Cochran (1977; see Gatz and Smith, 1995)

Description

This is a helper method for weighted variance computation in `origin_edm`, which is the closest to the bootstrap.

Usage

```r
var_wtd_mean_cochran(x, w)
```

Arguments

- `x` numeric vector of values
- `w` numeric vector of weights

Value

numeric value of weighted variance

References

Index

*Topic data
   - delay-data, 4
   - ptn-data, 15

aggr_data, 2, 16, 17
analyze_ptn, 3, 15

delay-data, 4
delayAth (delay-data), 4
delayGoe (delay-data), 4

eff_dijkstra (eff_dist), 5
eff_dist, 5

fastgreedy.community, 11

igraph, 3, 10, 11, 15

leading.eigenvector.community, 11

NetOrigin, 6
NetOrigin-package (NetOrigin), 6

origin, 7, 7, 10–12, 19
origin-methods, 10
origin_backtracking, 8, 11, 17
origin_backtracking (origin), 7
origin_centrality, 8, 11, 17
origin_centrality (origin), 7
origin_edm, 8, 10, 11, 17, 19
origin_edm (origin), 7
origin_multiple, 9, 11

performance, 12
performance.origin, 13
performance.origin (origin-methods), 10
plot, 13
plot.igraph, 15
plot.origin (origin-methods), 10
plot.robustness (robustness-methods), 18
plot_performance, 11, 12, 13

plot_ptn, 7, 14
print.origin (origin-methods), 10
print.robustness (robustness-methods), 18

ptn-data, 15
ptnAth (ptn-data), 15
ptnGoe (ptn-data), 15

read_DB_data, 2, 16
robustness, 17, 18, 19
robustness-methods, 18

spd_dijkstra (eff_dist), 5
strftime, 2
summary.origin (origin-methods), 10
summary.robustness (robustness-methods), 18

var_wtd_mean_cochran, 19