Package ‘PAFit’

January 11, 2022

Type Package
Title Generative Mechanism Estimation in Temporal Complex Networks
Version 1.2.5
Date 2022-01-11
Author Thong Pham, Paul Sheridan, Hidetoshi Shimodaira
Maintainer Thong Pham <thongphamthe@gmail.com>
URL https://github.com/thongphamthe/PAFit
BugReports https://github.com/thongphamthe/PAFit/issues
License GPL-3
Depends R(>= 2.10.0)
Imports Rcpp (>= 0.11.3) , grDevices, graphics, stats, RColorBrewer, VGAM, MASS, magicaxis, networkDynamic, network, plyr, igraph, mapproj, knitr, methods, ggplot2
LinkingTo Rcpp
LazyData True
Encoding UTF-8
NeedsCompilation yes
Suggests R.rsp
VignetteBuilder R.rsp
Repository CRAN
Date/Publication 2022-01-11 10:02:48 UTC
R topics documented:

PAFit-package .................................................. 3
as.PAFit_net .................................................. 5
Coauthorship network of scientists in the field of complex networks .......... 6
from_igraph ................................................... 7
from_networkDynamic ........................................... 8
generate_BA .................................................. 8
generate_BB .................................................. 10
generate_ER .................................................. 11
generate_fit_only ............................................. 13
generate_net .................................................. 14
generate_simulated_data_from_estimated_model .................................. 17
get_statistics .................................................. 19
graph_from_file ............................................... 22
graph_to_file .................................................. 23
Jeong ............................................................ 24
joint_estimate .................................................. 26
Newman .......................................................... 30
only_A_estimate .................................................. 31
only_F_estimate .................................................. 35
PAFit_oneshot .................................................. 37
plot.Full_PAFit_result ........................................ 38
plot.PAFit_net .................................................. 40
plot.PAFit_result ............................................... 41
plot.PA_result .................................................. 43
plot_contribution .............................................. 45
print.CV_Data .................................................. 47
print.CV_Result .................................................. 48
print.Full_PAFit_result ....................................... 49
print.PAFit_data ............................................... 50
print.PAFit_net .................................................. 51
print.PAFit_result ............................................. 51
print.PA_result .................................................. 52
summary.CV_Data ............................................... 53
summary.CV_Result ............................................. 54
summary.Full_PAFit_result .................................... 55
summary.PAFit_data ........................................... 56
summary.PAFit_net ............................................. 57
summary.PAFit_result .......................................... 58
summary.PA_result ............................................. 59
test_linear_PA .................................................. 59
to_igraph ....................................................... 61
to_networkDynamic ............................................. 62

Index ............................................................. 63
PAFit-package

Generative Mechanism Estimation in Temporal Complex Networks

Description


Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>PAFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>1.2.5</td>
</tr>
<tr>
<td>Authors:</td>
<td>Thong Pham, Paul Sheridan, Hidetoshi Shimodaira</td>
</tr>
<tr>
<td>Maintainer:</td>
<td>Thong Pham <a href="mailto:thongphamthe@gmail.com">thongphamthe@gmail.com</a></td>
</tr>
<tr>
<td>Date:</td>
<td>2022-01-11</td>
</tr>
<tr>
<td>License:</td>
<td>GPL-3</td>
</tr>
</tbody>
</table>

The PAFit package provides a comprehensive framework to deal with growth mechanisms of temporal complex networks. In particular, it implements functions to simulate various temporal network models, gather essential network statistics from raw input data, and use these summarized statistics in the estimation of the attachment function $A_k$ and node fitnesses $\eta_i$. The heavy computational parts of the package are implemented in C++ through the use of the Rcpp package. Furthermore, users with a multi-core machine can enjoy a hassle-free speed up through OpenMP parallelization mechanisms implemented in the code. Apart from the main functions, the package also includes a real-world collaboration network dataset between scientists in the field of complex networks (coauthor.net). The main package functionalities are as follows.

Firstly, most well-known temporal network models based on the preferential attachment (PA) and node fitness mechanisms can be easily simulated using the package. PAFit implements `generate_BA` for the Barabási-Albert (BA) model, `generate_ER` for the growing Erdős–Rényi (ER) model, `generate_BB` for the Bianconi-Barabási (BB) model and `generate_fit_only` for the Caldarelli model. These functions have many customizable options, for example the number of new edges at each time-step are tunable stochastic variables. They are actually wrappers of the more powerful `generate_net` function, which simulates networks with more flexible attachment function and node fitness settings.

Secondly, the function `get_statistics` efficiently collects all temporal network summary statistics. We note that `get_statistics` automatically handles both directed and undirected networks. It returns a list containing many statistics that can be used to characterize the network growth process. Notable fields are $m_{tk}$ containing the number of new edges that connect to a degree-$k$ node at time-step $t$, and `node_degree` containing the degree sequence, i.e., the degree of each node at each time-step.

The most important functionality of the package is estimating the attachment function and node fitnesses.
PAFit-package

fitnesses of a temporal network. This is implemented through various methods. There are three us-
ages: estimation of the attachment function in isolation, estimation of the node fitnesses in isolation,
and the joint estimation of the attachment function and node fitnesses.

• The functions for estimating the attachment function in isolation are: Jeong for Jeong’s
  method (Ref. 1), Newman for Newman’s method (Ref. 2), and only_A_estimate for the
  PAFit method (Ref. 3).

• For estimation of node fitnesses in isolation, only_F_estimate implements a variant of the
  PAFit method (Ref. 4).

• For the joint estimation of the attachment function and node fitnesses, we implement the full
  version of the PAFit method in joint_estimate (Ref. 4).

• For estimating the nonparametric attachment function from a single snapshot, use PAFit_oneshot
  (Ref. 6).

Excluding PAFit_oneshot, the input of the remaining functions is the output object of the function
get_statistics. The output object of these functions contains the estimation results as well as
some additional information pertaining to the estimation process. The estimated attachment func-
tion and/or node fitnesses can be plotted by using the plot command directly on this output object.
This will visualize not only the estimated results but also the remaining uncertainties when possible.

Author(s)
Thong Pham <thongphamthe@gmail.com>, Paul Sheridan, and Hidetoshi Shimodaira.

References
   erential Attachment in Temporal Complex Networks. PLOS ONE 10(9):e0137796. (doi: 10.1371/
   journal.pone.0137796).
   and Node Fitness in Growing Complex Networks. Scientific Reports 6, Article number: 32558.
   (doi: 10.1038/srep32558).
   Estimation of Preferential Attachment and Node Fitness in Temporal Complex Networks. Journal
   of Statistical Software 92 (3). (doi: 10.18637/jss.v092.i03)

See Also
See the accompanying vignette for a tutorial.
See also the GitHub page.
## Not run:
### Jointly estimate the attachment function and node fitnesses
library("PAFit")
set.seed(1)
# a Bianconi-Barabasi network  
# size of initial network = 100  
# number of new nodes at each time-step = 100  
# Ak = k; inverse variance of distribution of fitness: s = 10
net <- generate_BB(N = 1000, m = 10,  
                  num_seed = 100, multiple_node = 100,  
                  s = 10)
net_stats <- get_statistics(net)
# Joint estimation of attachment function Ak and node fitness
result <- joint_estimate(net, net_stats)
summary(result)
# plot the estimated attachment function
plot(result, net_stats)

# true function
true_A <- pmax(result$estimate_result$center_k,1)
lines(result$estimate_result$center_k, true_A, col = "red")  # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")
# plot distribution of estimated node fitnesses
plot(result, net_stats, plot = "f")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

## End(Not run)

---

as.PAFit_net  

*Converting an edgelist matrix to a PAFit_net object*

### Description

This function converts a graph stored in an edgelist matrix format to a PAFit_net object.

### Usage

```r
as.PAFit_net(graph, type = "directed", PA = NULL, fitness = NULL)
```

### Arguments

- **graph**: An edgelist matrix. Each row is assumed to be of the form (from_node_id to_node_id time_stamp). For a directed network, from_node_id is the id of
the source node and to_node_id is the id of the destination node. For an un-directed network, the order is ignored and from_node_id and to_node_id are the ids of two ends. time_stamp is the arrival time of the edge. from_node_id and to_node_id are assumed to be integers that are at least 0. The whole ids need not to be contiguous.

To register a new node i at time t without any edge, add a row with format (i -1 t). This works for both undirected and directed networks. time_stamp can be either numeric or string. The value of a time-stamp can be arbitrary, but we assume that a smaller time_stamp (regarded so by the sort function in R) represents an earlier arrival time. Examples of time-stamps that satisfy this assumption are the integer 0:T, the string format 'yyyy-mm-dd', and the POSIX time.

type String. Indicates whether the network is "directed" or "undirected".

PA Numeric vector. Contains the PA function. Default value is NULL.

fitness Numeric vector. Contains node fitnesses. Default value is NULL.

Value

An object of class PAFit_net

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
as.PAFit_net(net$graph)

Description

The dataset is collaboration network of authors of network science articles with article time-stamps. An edge between two authors represents an article in common. Time stamps denote article publication dates. The network without time-stamps was compiled by Mark Newman in May 2006 from the bibliographies of two review articles on networks. M. E. J. Newman, SIAM Review 45, 167-256 (2003) and S. Boccaletti et al., Physics Reports 424, 175-308 (2006), with a few additional references added by hand. Paul Sheridan independently supplemented the network with time-stamps and some basic metadata in June 2015. The network is undirected with monthly resolution, and contains no duplicated edges. coauthor.net contains the network. coauthor.truetime contains the real times of processed time-stamps. Finally coauthor.author_id contains author names.

The dataset with article time-stamps is available for download at https://www.paulsheridan.net/files/collabnet.zip

Usage

data(ComplexNetCoauthor)

Format

coauteur.net is a matrix with 2849 rows and 3 columns. Each row is an edge with the format (author id 1, author id 2, time_stamp). coauteur.truetime is a two-column matrix whose each row is (time_stamp, real time). coauteur.author_id is a two-column matrix whose each row is (author id, author name).

Source

https://www.paulsheridan.net/files/collabnet.zip

from_igraph

Convert an igraph object to a PAFit_net object

Description

This function converts an igraph object (of package igraph) to a PAFit_net object.

Usage

from_igraph(net)

Arguments

net An object of class igraph.

Value

The function returns a PAFit_net object.

Author(s)

Thong Pham <thongphanthe@gmail.com>

Examples

library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50, m = 10, s = 10)
igraph_graph <- to_igraph(net)
back <- from_igraph(igraph_graph)
from_networkDynamic  
*Convert a networkDynamic object to a PAFit_net object*

**Description**

This function converts a networkDynamic object (of package networkDynamic) to a PAFit_net object.

**Usage**

```r
from_networkDynamic(net)
```

**Arguments**

- `net`  
  An object of class networkDynamic.

**Value**

The function returns a PAFit_net object.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```r
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
nD_graph <- to_networkDynamic(net)
back <- from_networkDynamic(nD_graph)
```

---

**generate_BA**  
*Simulating networks from the generalized Barabasi-Albert model*

**Description**

This function generates networks from the generalized Barabási-Albert model. In this model, the preferential attachment function is power-law, i.e. \( A_k = k^\alpha \), and node fitnesses are all equal to 1. It is a wrapper of the more powerful function generate.net.

**Usage**

```r
generate_BA(N = 1000, num_seed = 2 , multiple_node = 1 , m = 1 , alpha = 1)
```
Arguments

N  Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
num_seed  Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
multiple_node  Positive integer. The number of new nodes at each time-step. Default value is 1.
m  Positive integer. The number of edges of each new node. Default value is 1.
alpha  Numeric. This is the attachment exponent in the attachment function $A_k = k^\alpha$.

Value

The output is a PAFit_net object, which is a List contains the following four fields:

graph  a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
type  a string indicates whether the network is “directed” or “undirected”.
PA  a numeric vector contains the true PA function.
fitness  fitness values of nodes in the network. The fitnesses are all equal to 1.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

For subsequent estimation procedures, see get_statistics.
For other functions to generate networks, see generate_net, generate_ER, generate_BB and generate_fit_only.

Examples

```r
library("PAFit")
# generate a network from the BA model with alpha = 1, N = 100, m = 1
net <- generate_BA(N = 100)
str(net)
plot(net)
```
**generate_BB**

*Simulating networks from the Bianconi-Barabási model*

**Description**

This function generates networks from the Bianconi-Barabási model. It is a ‘preferential attachment with fitness’ model. In this model, the preferential attachment function is linear, i.e. $A_k = k$, and node fitnesses are sampled from some probability distribution.

**Usage**

```r
generate_BB(N = 1000 ,
             num_seed = 2 ,
             multiple_node = 1 ,
             m = 1 ,
             mode_f = "gamma",
             s = 10 )
```

**Arguments**

The parameters can be divided into two groups.

The first group specifies basic properties of the network:

- **N** Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
- **num_seed** Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
- **multiple_node** Positive integer. The number of new nodes at each time-step. Default value is 1.
- **m** Positive integer. The number of edges of each new node. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

- **mode_f** String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution. "power_law" = power-law (pareto) distribution. Default value is "gamma".
- **s** Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is $1/s$ (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to $s$; setting mean and standard deviation in log-scale of the log-normal distribution to $-1/2 \times \log(1/s + 1)$ and $(\log(1/s + 1))^{0.5}$; and setting shape and scale parameters of the pareto distribution to $(s + 1)^{0.5} + 1$ and $(s + 1)^{0.5}/((s + 1)^{0.5} + 1)$. If $s$ is 0, all node fitnesses $\eta$ are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.
The output is a PAFit_net object, which is a List contains the following four fields:

- **graph**: a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.
- **type**: a string indicates whether the network is "directed" or "undirected".
- **PA**: a numeric vector contains the true PA function.
- **fitness**: fitness values of nodes in the network. The name of each value is the ID of the node.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**


**See Also**

For subsequent estimation procedures, see `get_statistics`.

For other functions to generate networks, see `generate_net`, `generate_BA`, `generate_ER` and `generate_fit_only`.

**Examples**

```r
library("PAFit")
# generate a network from the BB model with alpha = 1, N = 100, m = 1
# The inverse variance of the Gamma distribution of node fitnesses is s = 10
net <- generate_BB(N = 100,m = 1,mode = 1, s = 10)
str(net)
plot(net)
```

---

**Description**

This function generates networks from the Erdős–Rényi model. In this model, the preferential attachment function is a constant function, i.e. $A_k = 1$, and node fitnesses are all equal to 1. It is a wrapper of the more powerful function `generate_net`.
Usage

```r
generate_ER(N = 1000, num_seed = 2, multiple_node = 1, m = 1)
```

Arguments

- **N**: Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is `1000`.
- **num_seed**: Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is `2`.
- **multiple_node**: Positive integer. The number of new nodes at each time-step. Default value is `1`.
- **m**: Positive integer. The number of edges of each new node. Default value is `1`.

Value

The output is a `PAFit_net` object, which is a List contains the following four fields:

- **graph**: A three-column matrix, where each row contains information of one edge, in the form of `(from_id, to_id, time_stamp)`. `from_id` is the id of the source, `to_id` is the id of the destination.
- **type**: A string indicates whether the network is "directed" or "undirected".
- **PA**: A numeric vector contains the true PA function.
- **fitness**: Fitness values of nodes in the network. The fitnesses are all equal to `1`.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

For subsequent estimation procedures, see `get_statistics`.

For other functions to generate networks, see `generate_net`, `generate_BA`, `generate_BB` and `generate_fit_only`.

Examples

```r
library("PAFit")
# generate a network from the ER model with N = 1000 nodes
net <- generate_ER(N = 1000)
str(net)
plot(net)
```
**generate_fit_only**  
*Simulating networks from the Caldarelli model*

**Description**

This function generates networks from the Caldarelli model. In this model, the preferential attachment function is constant, i.e. $A_k = 1$, and node fitnesses are sampled from some probability distribution.

**Usage**

```r
generate_fit_only(N = 1000 ,
                  num_seed = 2 ,
                  multiple_node = 1 ,
                  m = 1 ,
                  mode_f = "gamma",
                  s = 10 )
```

**Arguments**

The parameters can be divided into two groups.

The first group specifies basic properties of the network:

- **N**: Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
- **num_seed**: Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
- **multiple_node**: Positive integer. The number of new nodes at each time-step. Default value is 1.
- **m**: Positive integer. The number of edges of each new node. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

- **mode_f**: String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution. "power_law" = power-law (pareto) distribution. Default value is "gamma".
- **s**: Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is $1/s$ (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to $s$; setting mean and standard deviation in log-scale of the log-normal distribution to $-1/2 \times \log(1/s + 1)$ and $(\log(1/s + 1))^{0.5}$; and setting shape and scale parameters of the pareto distribution to $(s + 1)^{0.5} + 1$ and $(s + 1)^{0.5}/((s + 1)^{0.5} + 1)$. If s is 0, all node fitnesses $\eta$ are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.
generate_net

Value

The output is a PAFit_net object, which is a List contains the following four fields:

- **graph**: a three-column matrix, where each row contains information of one edge, in the form of \((\text{from}_id, \text{to}_id, \text{time} \_ \text{stamp})\). \text{from}_id is the id of the source, \text{to}_id is the id of the destination.
- **type**: a string indicates whether the network is "directed" or "undirected".
- **PA**: a numeric vector contains the true PA function.
- **fitness**: fitness values of nodes in the network. The name of each value is the ID of the node.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

For subsequent estimation procedures, see `get_statistics`.

For other functions to generate networks, see `generate_net`, `generate_BA`, `generate_ER` and `generate_BB`.

Examples

```r
library("PAFit")
# generate a network from the Caldarelli model with alpha = 1, N = 100, m = 1
# the inverse variance of distribution of node fitnesses is s = 10
net <- generate_fit_only(N = 100,m = 1,mode = 1, s = 10)
str(net)
plot(net)
```

Description

This function generates networks from the General Temporal model, a generative temporal network model that includes many well-known models such as the Erdős-Rényi model, the Barabási-Albert model or the Bianconi-Barabási model as special cases. This function also includes some flexible mechanisms to vary the number of new nodes and new edges at each time-step in order to generate realistic networks.
Usage

```r
gen = generate_net (N = 1000 ,
  num_seed = 2 ,
  multiple_node = 1 ,
  specific_start = NULL ,
  m = 1 ,
  prob_m = FALSE ,
  increase = FALSE ,
  log = FALSE ,
  no_new_node_step = 0 ,
  m_no_new_node_step = m ,
  custom_PA = NULL ,
  mode = 1 ,
  alpha = 1 ,
  beta = 2 ,
  sat_at = 100 ,
  offset = 1 ,
  mode_f = "gamma" ,
  s = 10 )
```

Arguments

The parameters can be divided into four groups.

The first group specifies basic properties of the network:

- **N** Integer. Total number of nodes in the network (including the nodes in the seed graph). Default value is 1000.
- **num_seed** Integer. The number of nodes of the seed graph (the initial state of the network). The seed graph is a cycle. Default value is 2.
- **multiple_node** Positive integer. The number of new nodes at each time-step. Default value is 1.
- **specific_start** Positive integer. If `specific_start` is specified, then all the time-steps from time-step 1 to `specific_start` are grouped to become the initial time-step in the final output. This option is useful when we want to create a network with a large initial network that follows a scale-free degree distribution. Default value is NULL.

The second group specifies the number of new edges at each time-step:

- **m** Positive integer. The number of edges of each new node. Default value is 1.
- **prob_m** Logical. Indicates whether we fix the number of edges of each new node as a constant, or let it follow a Poisson distribution. If `prob_m == TRUE`, the number of edges of each new node follows a Poisson distribution. The mean of this distribution depends on the value of `increase` and `log`. Default value is FALSE.
- **increase** Logical. Indicates whether we increase the mean of the Poisson distribution over time. If `increase == FALSE`, the mean is fixed at m. If `increase == TRUE`, the way the mean increases depends on the value of `log`. Default value is FALSE.
- **log** Logical. Indicates how to increase the mean of the Poisson distribution. If `log == TRUE`, the mean increases logarithmically with the number of current nodes.
If log == FALSE, the mean increases linearly with the number of current nodes. Default value is FALSE.

no_new_node_step
Non-negative integer. The number of time-steps in which no new node is added, while new edges are added between existing nodes. Default value is 0, i.e., new nodes are always added at each time-step.

m_no_new_node_step
Positive integer. The number of new edges in the no-new-node steps. Default value is equal to m. Note that the number of new edges in the no-new-node steps is not affected by the parameters increase or prob_m; this number is always the constant specified by m_no_new_node_step.

The third group of parameters specifies the preferential attachment function:

custom_PA
Numeric vector. This is the user-input PA function: A_0, A_1, ..., A_K. If custom_PA is specified, then mode is ignored, and we grow the network using the PA function custom_PA. Degrees greater than K will have attachment value A_K. Default value is NULL.

mode
Integer. Indicates the parametric attachment function to be used in generating the network. If mode == 1, the attachment function is A_k = k^α. If mode == 2, the attachment function is A_k = \min(k, sat.at)^α. If mode == 3, the attachment function is A_k = a log(k)^β. Default value is 1.

alpha
Numeric. If mode == 1, this is the attachment exponent in the attachment function A_k = k^α. If mode == 2, this is the attachment exponent in the attachment function A_k = \min(k, sat.at)^α. If mode == 3, this is the α in the attachment function A_k = a log(k)^β + 1.

beta
Numeric. This is the beta in the attachment function A_k = a log(k)^β + 1.

sat_at
Integer. This is the saturation position sat.at in the attachment function A_k = \min(k, sat.at)^α.

offset
Numeric. The attachment value of degree 0. Default value is 1.

The final group of parameters specifies the distribution from which node fitnesses are generated:

mode_f
String. Possible values: "gamma", "log_normal" or "power_law". This parameter indicates the true distribution for node fitness. "gamma" = gamma distribution, "log_normal" = log-normal distribution, "power_law" = power-law (pareto) distribution. Default value is "gamma".

s
Non-negative numeric. The inverse variance parameter. The mean of the distribution is kept at 1 and the variance is 1/s (since node fitnesses are only meaningful up to scale). This is achieved by setting shape and rate parameters of the Gamma distribution to s; setting mean and standard deviation in log-scale of the log-normal distribution to \(-1/2 \times log(1/s + 1)\) and \((log(1/s + 1))^{0.5}\); and setting shape and scale parameters of the pareto distribution to \((s + 1)^{0.5} + 1\) and \((s + 1)^{0.5}/((s + 1)^{0.5} + 1)\). If s is 0, all node fitnesses \eta are fixed at 1 (i.e., Barabási-Albert model). The default value is 10.

Value
The output is a PAFit_net object, which is a List contains the following four fields:
graph a three-column matrix, where each row contains information of one edge, in the form of (from_id, to_id, time_stamp). from_id is the id of the source, to_id is the id of the destination.

type a string indicates whether the network is "directed" or "undirected".

PA a numeric vector contains the true PA function.

fitness fitness values of nodes in the network. The name of each value is the ID of the node.

Author(s)
Thong Pham <thongphamthe@gmail.com>

See Also
For subsequent estimation procedures, see get_statistics.

For simpler functions to generate networks from well-known models, see generate_BA, generate_ER, generate_BB and generate_fit_only.

Examples

library("PAFit")
#Generate a network from the original BA model with alpha = 1, N = 100, m = 1
net <- generate_net(N = 100, m = 1, mode = 1, alpha = 1, s = 0)
str(net)
plot(net)
generate_simulated_data_from_estimated_model

Arguments

net_object  an object of class PAFit_net that contains the original network.

net_stat  An object of class PAFit_data which contains summarized statistics of the original network. This object is created by the function get_statistics.

result  An object of class Full_PAFit_result which contains the fitted model obtained by applying the function joint_estimate.

M  integer. The number of simulated networks. Default value is 5.

Value

Outputs a Simulated_Data_From_Fitted_Model object, which is a list containing the following fields:

- graph_list: a list containing \( M \) simulated graphs.
- stats_list: a list containing \( M \) objects of class PAFit_data, which are the results of applying get_statistics on the simulated graphs.
- result_list: a list containing \( M \) objects of class Full_PAFit_result, which are the results of applying joint_estimate on the simulated graphs.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

generate_simulated_data_from_estimated_model, get_statistics, joint_estimate, plot_contribution
get_statistics

Examples

## Not run:

```r
library("PAFit")
net_object <- generate_net(N = 500, m = 10, s = 10, alpha = 0.5)
net_stat <- get_statistics(net_object)
result <- joint_estimate(net_object, net_stat)
simulated_data <- generate_simulated_data_from_estimated_model(net_object, net_stat, result)
plot_contribution(simulated_data, result, which_plot = "PA")
plot_contribution(simulated_data, result, which_plot = "fit")
```

## End(Not run)

get_statistics Getting summarized statistics from input data

Description

The function summarizes input data into sufficient statistics for estimating the attachment function and node fitness, together with additional information about the data, such as total number of nodes, number of time-steps, maximum degree, and the final degree of the network, etc. It also provides mechanisms to automatically deal with very large datasets by binning the degree, setting a degree threshold, or grouping time-steps.

Usage

```r
get_statistics(net_object, only_PA = FALSE ,
only_true_deg_matrix = FALSE ,
binning = TRUE , g = 50 ,
deg_threshold = 0 ,
compress_mode = 0 , compress_ratio = 0.5 ,
custom_time = NULL)
```

Arguments

The parameters can be divided into four groups. The first group specifies input data and how the data will be summarized:

An object of class PAFit_net. You can use the function `as.PAFit_net` to convert from an edgelist matrix, function `from_igraph` to convert from an igraph object, function `from_networkDynamic` to convert from a networkDynamic object, and function `graph_from_file` to read from a file.

### onlyObject

Logical. Indicates whether only the statistics for estimating \( A_k \) are summarized. If TRUE, the statistics for estimating \( \eta_i \) are NOT collected. This will save memory at the cost of unable to estimate node fitness). Default value is FALSE.

### only_true_deg_matrix

Logical. Return only the true degree matrix (without binning), and no other statistics is returned. The result cannot be used in PAFit function to estimate PA
get_statistics

or fitness. The motivation for this option is that sometimes we only want to get a degree matrix that summarizes the growth process of a very big network for plotting etc. Default value is FALSE.

Second group of parameters specifies how to bin the degrees:

**binning** Logical. Indicates whether the degree should be binned together. Default value is TRUE.

**g** Positive integer. Number of bins. Should be at least 3. Default value is 50.

Third group contains a single parameter specifying how to reduce the number of node fitnesses:

**deg_threshold** Integer. We only estimate the fitnesses of nodes whose number of new edges acquired is at least deg_threshold. The fitnesses of all other nodes are fixed at 1. Default value is 0.

Last group of parameters specifies how to group the time-stamps:

**compress_mode** Integer. Indicates whether the timeline should be compressed. The value of CompressMode:

0: No compression
1: Compressed by using a subset of time-steps. The time stamps in this subset are equally spaced. The size of this subset is CompressRatio times the size of the set of all time stamps.
2: Compressed by only starting from the first time-step when CompressRatio * 100 percentages of the total number of edges (in the final state of the network) had already been added to the network.
3: This mode offers the most flexibility, but requires user to supply the time stamps in CustomTime. Only time stamps in this CustomTime will be used. This mode can be used, for example, when investigating the change of the attachment function or node fitness in different time intervals. Default value is 0, i.e. no compression.

**compress_ratio** Numeric. Indicates how much we should compress if CompressMode is 1 or 2. Default value is 0.5.

**custom_time** Vector. Custom time stamps. This vector is a subset of the vector that contains all time-stamps. Only effective if CompressMode == 3. In that case, only these time stamps are used.

**Value**

An object of class PAFit_data, which is a list. Some important fields are:

**offset_tk** A matrix where the \((t,k+1)\) element is the number of nodes with degree \(k\) at time \(t\), counting among all the nodes whose number of new edges acquired is less than deg_threshold

**n_tk** A matrix where the \((t,k+1)\) element is the number of nodes with degree \(k\) at time \(t\)

**m_tk** A matrix where the \((t,k+1)\) element is the number of new edges connect to a degree-\(k\) node at time \(t\)

**sum_m_k** A vector where the \((k+1)\)-th element is the total number of edges that linked to a degree \(k\) node, counting over all time steps
get_statistics

node_degree A matrix recording the degree of all nodes (that satisfy degree_threshold condition) at each time step
m_t A vector where the t-th element is the number of new edges at time t
z_j A vector where the j-th element is the total number of edges that linked to node j
N Numeric. The number of nodes in the network
T Numeric. The number of time steps
deg_max Numeric. The maximum degree in the final network
node_id A vector contains the id of all nodes
final_deg A vector contains the final degree of all nodes (including those that do not satisfy the degree_threshold condition)
deg_thresh Integer. The specified degree threshold.
f_position Numeric vector. The index in the node_id vector of the nodes we want to estimate (i.e. nodes whose number of new edges acquired is at least deg_thresh)
start_deg Integer. The specified degree at which we start binning.
begin_deg Numeric vector contains the beginning degree of each bin
end_deg Numeric vector contains the ending degree of each bin
interval_length Numeric vector contains the length of each bin.
binning Logical. Indicates whether binning was applied or not.
g Integer. Number of bins
time_compress_mode Integer. The mode of time compression.
t_compressed Integer. The number of time stamps actually used
compressed_unique_time The time stamps that are actually used
compress_ratio Numeric.
custom_time Vector. The time stamps specified by user.

Author(s)
Thong Pham <thongphamthe@gmail.com>

See Also
For creating the needed input for this function (a PAFit_net object), see as.PAFit_net, from_igraph, from_networkDynamic, and graph_from_file.

For the next step, see Newman, Jeong or only_A_estimate for estimating the attachment function in isolation, only_F_estimate for estimating node fitnesses in isolation, and joint_estimate for joint estimation of the attachment function and node fitnesses.
Examples

```r
library("PAFit")
net <- generate_BA(N = 100, m = 1)
net_stats <- get_statistics(net)
summary(net_stats)
```

---

### Description

This function reads an input file to a PAFit_net object. Accepted formats are the edgelist format or the gml format.

### Usage

```r
graph_from_file(file_name, format = "edgelist", type = "directed")
```

### Arguments

- **file_name**: A string indicates the file name.
- **format**: String. Possible values are "edgelist" and "gml". If format is "edgelist", we assume the following edgelist matrix format. Each row is assumed to be of the form \((from\_node\_id \rightarrow to\_node\_id \text{ time\_stamp})\). \(from\_node\_id\) is the id of the source node. \(to\_node\_id\) is the id of the destination node. \text{time\_stamp} is the arrival time of the edge. \(from\_node\_id\) and \(to\_node\_id\) are assumed to be integers that are at least 0. They need not to be contiguous.

To register a new node \(i\) at time \(t\) without any edge, add a row with format \((i-1 \rightarrow -1 \text{ time\_stamp})\). This works for both undirected and directed networks.

\text{time\_stamp} can be either numeric or string. The value of a time-stamp can be arbitrary, but we assume that a smaller \text{time\_stamp} (regarded so by the sort function in R) represents an earlier arrival time. Examples of time-stamps that satisfy this assumption are the integer 0:T, the string format ‘yyyy-mm-dd’, and the POSIX time.

If format is "gml", there must be a binary field directed indicating the type of the network (0: undirected, 1: directed). The required fields for an edge are: source, target, and time. source and target are the ID of the source node and the target node, respectively. time is the time-stamp of the edge. The required fields for a node are: id, isolated (binary) and time. The binary field isolated indicates whether this node is an isolated node when it enters the system or not. If isolated is 1, then time must contain the node’s appearance time. If isolated is 0, then we can automatically infer the node’s appearance time from its edges, so the field time in this case can be NULL. The assumptions on node IDs and the format of time-stamps are the same as in the case when format = "edgelist". See `graph_to_file` to see detail on the format of the gml file this package outputs.
**graph_to_file**

String. Indicates whether the network is "directed" or "undirected". This option is ignored if format is "gml", since the information is assumed to be contained in the gml file.

**Value**

An object of class PAFit_net containing the network.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```r
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)

#graph_to_file(net, file_name = "test.gml", format = "gml")
#reread <- graph_from_file(file_name = "test.gml", format = "gml")
```

**Description**

This function writes a graph in a PAFit_net object to an output file. Accepted file formats are the edgelist format or the gml format.

**Usage**

```r
graph_to_file(net_object, file_name, format = "edgelist")
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net_object</td>
<td>An object of class PAFit_net.</td>
</tr>
<tr>
<td>file_name</td>
<td>A string indicates the file name.</td>
</tr>
<tr>
<td>format</td>
<td>String. Possible values are &quot;edgelist&quot; and &quot;gml&quot;.</td>
</tr>
</tbody>
</table>

If format = "edgelist", we just output the edgelist matrix contained in the PAFit_net object as it is.

If format = "gml", here is the specification of the gml file. There is a binary field directed indicating the type of the network (0: undirected, 1: directed). There are three attributes for an edge: source, target, and time. There are three attributes for a node: id, isolated (binary) and time. The attribute time is NULL if the attribute isolated is 0 (since this is not an isolated node, we do not need to record its first appearance time). On the other hand, time is the node’s appearance time if attribute isolated is 1.
Value

The function writes directly to the output file.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
# graph_to_file(net, file_name = "test.gml", format = "gml")

Description

This function estimates the preferential attachment function by Jeong’s method.

Usage

```r
Jeong(net_object ,
      net_stat = get_statistics(net_object) ,
      T_0_start = 0 ,
      T_0_end = round(net_stat$T * 0.75) ,
      T_1_start = T_0_end + 1 ,
      T_1_end = net_stat$T ,
      interpolate = FALSE)
```

Arguments

- `net_object` an object of class `PAFit_net` that contains the network.
- `net_stat` An object of class `PAFit_data` which contains summerized statistics needed in estimation. This object is created by the function `get_statistics`. Default value is `get_statistics(net_object)`. 
- `T_0_start` Positive integer. The starting time-step of the `T_0_interval`. Default value is 0.
- `T_0_end` Positive integer. The ending time-step of `T_0_interval`. Default value is `round(net_stat$T * 0.75)`.
- `T_1_start` Positive integer. The starting time-step of the `T_1_interval`. Default value is `T_0_end + 1`.
- `T_1_end` Positive integer. The ending time-step of `T_1_interval`. Default value is `net_stat$T`.
- `interpolate` Logical. If `TRUE` then all the gaps in the estimated PA function are interpolated by linear interpolating in logarithm scale. Default value is `FALSE`. 
Jeong

Value

Outputs an PA_result object which contains the estimated attachment function. In particular, it contains the following field:

• k and $A$: a degree vector and the estimated PA function.
• center_k and theta: when we perform binning, these are the centers of the bins and the estimated PA values for those bins.
• g: the number of bins used.
• alpha and ci: alpha is the estimated attachment exponent $\alpha$ (when assume $A_k = k^\alpha$), while ci is the confidence interval.
• loglinear_fit: this is the fitting result when we estimate $\alpha$.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

See get_statistics for how to create summerized statistics needed in this function.

See Newman and only_A_estimate for other methods to estimate the attachment function in isolation.

Examples

library("PAFit")
net <- generate_net(N = 1000, m = 1, mode = 1, alpha = 1, s = 0)
net_stats <- get_statistics(net)
result <- Jeong(net, net_stats)
# true function
true_A <- result$center_k
# plot the estimated attachment function
plot(result, net_stats)
lines(result$center_k, true_A, col = "red") # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")
Joint inference of attachment function and node fitnesses

Description

This function jointly estimates the attachment function $A_k$ and node fitnesses $\eta_i$. It first performs a cross-validation to select the optimal parameters $r$ and $s$, then estimates $A_k$ and $\eta_i$ using that optimal pair with the full data (Ref. 2).

Usage

\[
\text{joint_estimate}(\text{net_object}, \text{net_stat} = \text{get_statistics(\text{net_object})}, p = 0.75, \text{stop_cond} = 10^{-8}, \text{mode_reg_A} = 0, \ldots)
\]

Arguments

- **net_object** an object of class `PAFit_net` that contains the network.
- **net_stat** An object of class `PAFit_data` which contains summarized statistics needed in estimation. This object is created by the function `get_statistics`. The default value is `get_statistics(net_object)`. 
- **p** Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on $p$. The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is $0.75$.
- **stop_cond** Numeric. The iterative algorithm stops when \( \frac{\text{abs}(h(ii)} - h(ii+1))}{\text{abs}(h(ii))+1} < \text{stop.cond} \) where $h(ii)$ is the value of the objective function at iteration $ii$. We recommend to choose $\text{stop.cond}$ at most equal to $10^{(\text{-number of digits of } h - 2)}$, in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is $10^{-8}$. This threshold is good enough for most applications.
- **mode_reg_A** Binary. Indicates which regularization term is used for $A_k$:
  - 0: This is the regularization term used in Ref. 1 and 2. Please refer to Eq. (4) in the tutorial for the definition of the term. It approximately enforces the power-law form $A_k = k^\alpha$. This is the default value.
  - 1: Unlike the default, this regularization term exactly enforces the functional form $A_k = k^\alpha$. Please refer to Eq. (6) in the tutorial for the definition of the term. Its main drawback is it is significantly slower to converge, while its gain over the default one is marginal in most cases.

...
Value

Outputs a Full_PAFit_result object, which is a list containing the following fields:

- `cv_data`: a CV_Data object which contains the cross-validation data. This is the testing data.
- `cv_result`: a CV_Result object which contains the cross-validation result. Normally the user does not need to pay attention to this data.
- `estimate_result`: this is a PAFit_result object which contains the estimated attachment function $A_k$, the estimated fitnesses $\eta_i$ and their confidence intervals. In particular, the important fields are:
  - `ratio`: this is the selected value for the hyper-parameter $r$.
  - `shape`: this is the selected value for the hyper-parameter $s$.
  - `k` and $A$; a degree vector and the estimated PA function.
  - `var_A`: the estimated variance of $A$.
  - `var_logA`: the estimated variance of $\log A$.
  - `upper_A`: the upper value of the interval of two standard deviations around $A$.
  - `lower_A`: the lower value of the interval of two standard deviations around $A$.
  - `center_k` and `theta`: when we perform binning, these are the centers of the bins and the estimated PA values for those bins. `theta` is similar to $A$ but with duplicated values removed.
  - `var_bin`: the variance of `theta`. Same as `var_A` but with duplicated values removed.
  - `upper_bin`: the upper value of the interval of two standard deviations around `theta`. Same as `upper_A` but with duplicated values removed.
  - `lower_bin`: the lower value of the interval of two standard deviations around `theta`. Same as `lower_A` but with duplicated values removed.
  - `g`: the number of bins used.
  - `alpha` and `ci`: `alpha` is the estimated attachment exponent $\alpha$ (when assume $A_k = k^\alpha$), while `ci` is the confidence interval.
  - `loglinear_fit`: this is the fitting result when we estimate $\alpha$.
  - `f`: the estimated node fitnesses.
  - `var_f`: the estimated variance of $\eta_i$.
  - `upper_f`: the estimated upper value of the interval of two standard deviations around $\eta_i$.
  - `lower_f`: the estimated lower value of the interval of two standard deviations around $\eta_i$.
  - `objective_value`: values of the objective function over iterations in the final run with the full data.
  - `diverge_zero`: logical value indicates whether the algorithm diverged in the final run with the full data.
- `contribution`: a list containing an estimate of the contributions of preferential attachment and fitness mechanisms in the growth process of the network. The calculation adapts a quantification method proposed in Section 3 of Ref. 4, which is for preferential attachment and transitivity, to preferential attachment and fitness.
  - `PA_contribution`: an array containing the contributions of preferential attachment at each time-step
  - `fit_contribution`: an array containing the contributions of the fitness mechanism at each time-step
- mean_PA_contrib: the average contribution of preferential attachment through the whole growth process
- mean_fit_contrib: the average contribution of the fitness mechanism through the whole growth process

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**References**


**See Also**

See `get_statistics` for how to create summarized statistics needed in this function.

See `Jeong`, `Newman` and `only_A_estimate` for functions to estimate the attachment function in isolation.

See `only_F_estimate` for a function to estimate node fitnesses in isolation.

**Examples**

```r
## Not run:
library("PAFit")
# Example 1: a linear preferential attachment kernel, i.e., A_k = k
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
# Ak = k; inverse variance of the distribution of node fitness = 5
net <- generate_BB(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 5)
net_stats <- get_statistics(net)

# Joint estimation of attachment function Ak and node fitness
result <- joint_estimate(net, net_stats)
```
summary(result)

# plot the estimated attachment function
true_A <- 2 * log(pmax(result$estimate_result$center_k,1))^2 + 1 # true function
plot(result, net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

#############################################################################
#### Example 2: a non-log-linear preferential attachment kernel ###############
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
# A_k = alpha* log (max(k,1))^beta + 1, with alpha = 2, and beta = 2
# inverse variance of the distribution of node fitness = 10
net <- generate_net(N = 1000, m = 50, num_seed = 100, multiple_node = 100, s = 10, mode = 3, alpha = 2, beta = 2)
net_stats <- get_statistics(net)
# Joint estimation of attachment function A_k and node fitness
result <- joint_estimate(net, net_stats)
summary(result)

# plot the estimated attachment function
true_A <- 2 * log(pmax(result$estimate_result$center_k,1))^2 + 1 # true function
plot(result, net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

#############################################################################
#### Example 3: another non-log-linear preferential attachment kernel ###############
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
# A_k = min(max(k,1),sat_at)^alpha, with alpha = 1, and sat_at = 100
# inverse variance of the distribution of node fitness = 10
net <- generate_net(N = 1000, m = 50, num_seed = 100, multiple_node = 100, s = 10, mode = 2, alpha = 1, sat_at = 100)
net_stats <- get_statistics(net)
# Joint estimation of attachment function A_k and node fitness
result <- joint_estimate(net, net_stats)
summary(result)
# plot the estimated attachment function
true_A <- pmin(pmax(result$estimate_result$center_k,1),100)^1 # true function
plot(result , net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")

## End(Not run)

---

Newman | Corrected Newman's method for estimating the preferential attachment function

### Description

This function implements a correction proposed in [1] of the original Newman’s method in [2] to estimate the preferential attachment function.

### Usage

```
Newman(net_object ,
       net_stat = get_statistics(net_object),
       start = 1 ,
       interpolate = FALSE)
```

### Arguments

- **net_object** an object of class PAFit_net that contains the network.
- **net_stat** An object of class PAFit_data which contains summerized statistics needed in estimation. This object is created by the function `get_statistics`. Default value is `get_statistics(net_object)`.
- **start** Positive integer. The starting time from which the method is applied. Default value is 1.
- **interpolate** Logical. If TRUE then all the gaps in the estimated PA function are interpolated by linear interpolating in logarithm scale. Default value is FALSE.

### Value

Outputs an PA_result object which contains the estimated attachment function. In particular, it contains the following field:

- k and A: a degree vector and the estimated PA function.
- center_k and theta: when we perform binning, these are the centers of the bins and the estimated PA values for those bins.
- g: the number of bins used.
only_A_estimate

- alpha and ci: alpha is the estimated attachment exponent $\alpha$ (when assume $A_k = k^\alpha$), while ci is the mean plus/minus two-standard-deviation interval.
- loglinear_fit: this is the fitting result when we estimate $\alpha$.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

See get_statistics for how to create summarized statistics needed in this function.

See Jeong, only_A_estimate for other methods to estimate the attachment function in isolation.

Examples

```r
library("PAFit")
net <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result <- Newman(net, net_stats)
summary(result)
# true function
true_A <- result$center_k
#plot the estimated attachment function
plot(result , net_stats)
lines(result$center_k, true_A, col = "red") # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")
```

---

only_A_estimate

Estimating the attachment function in isolation by PAFit method

Description

This function estimates the attachment function $A_k$ by PAFit method. The method has a hyper-parameter $r$. It first performs a cross-validation step to select the optimal parameter $r$ for the regularization of $A_k$, then uses that $r$ to estimate the attachment function with the full data.
Usage

only_A_estimate(net_object = get_statistics(net_object),
    p = 0.75,
    stop_cond = 10^-8,
    mode_reg_A = 0,
    MLE = FALSE,
    ...)

Arguments

net_object an object of class PAFit_net that contains the network.

net_stat An object of class PAFit_data which contains summerized statistics needed in estimation. This object is created by the function get_statistics. The default value is get_statistics(net_object).

p Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on p. The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is 0.75.

stop_cond Numeric. The iterative algorithm stops when \( \frac{|h(i) - h(i+1)|}{|h(i)| + 1} < \text{stop.cond} \) where \( h(i) \) is the value of the objective function at iteration \( i \). We recommend to choose \( \text{stop.cond} \) at most equal to \( 10^{(-\text{number of digits of } h - 2)} \), in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is \( 10^{-8} \). This threshold is good enough for most applications.

mode_reg_A Binary. Indicates which regularization term is used for \( A_k \):

- 0: This is the regularization term used in Ref. 1 and 2. Please refer to Eq. (4) in the tutorial for the definition of the term. It approximately enforces the power-law form \( A_k = k^\alpha \). This is the default value.
- 1: Unlike the default, this regularization term exactly enforces the functional form \( A_k = k^\alpha \). Please refer to Eq. (6) in the tutorial for the definition of the term. Its main drawback is it is significantly slower to converge, while its gain over the default one is marginal in most cases.

MLE Logical. If \( \text{TRUE} \), then not perform cross-validation and estimate the PA function with \( r = 0 \), i.e., maximum likelihood estimation. Default is \( \text{FALSE} \). One might want to set this option to \( \text{TRUE} \) when one believes that there are sufficient data to get a reasonable MLE result, or when one wants to compare the default, regularized result with the MLE result.

Value

Outputs a Full_PAFit_result object, which is a list containing the following fields:

- \( \text{cv.data} \): a CV_Data object which contains the cross-validation data. This is the final Normally the user does not need to pay attention to this data. \( \text{NULL} \) if \( \text{MLE = TRUE} \).
only_A_estimate

• cv_result: a CV_Result object which contains the cross-validation result. Normally the user does not need to pay attention to this data. NULL if MLE = TRUE.
• estimate_result: this is a PAFit_result object which contains the estimated PA function and its confidence interval. It also includes the estimated attachment exponent $\alpha$ (assuming the model $A_k = k^\alpha$) in the field alpha, and the confidence interval of $\alpha$ (in the field ci) when possible. In particular, the important fields are:
  – ratio: this is the selected value for the hyper-parameter $r$.
  – k and A: a degree vector and the estimated PA function.
  – var_A: the estimated variance of $A$.
  – var_logA: the estimated variance of $\log A$.
  – upper_A: the upper value of the interval of two standard deviations around $A$.
  – lower_A: the lower value of the interval of two standard deviations around $A$.
  – center_k and theta: when we perform binning, these are the centers of the bins and the estimated PA values for those bins. theta is similar to A but with duplicated values removed.
  – var_bin: the variance of theta. Same as var_A but with duplicated values removed.
  – upper_bin: the upper value of the interval of two standard deviations around theta. Same as upper_A but with duplicated values removed.
  – lower_lower: the lower value of the interval of two standard deviations around theta. Same as lower_A but with duplicated values removed.
  – g: the number of bins used.
  – alpha and ci: alpha is the estimated attachment exponent $\alpha$ (when assume $A_k = k^\alpha$), while ci is the confidence interval.
  – loglinear_fit: this is the fitting result when we estimate $\alpha$.
  – objective_value: values of the objective function over iterations in the final run with the full data.
  – diverge_zero: logical value indicates whether the algorithm diverged in the final run with the full data.

Author(s)
Thong Pham <thongphamthe@gmail.com>

References


See Also

See get_statistics for how to create summarized statistics needed in this function.
See Newman and Jeong for other methods to estimate the attachment function $A_k$ in isolation.
Examples

```r
## Not run:
library("PAFit")
set.seed(1)
#### Example 1: Linear preferential attachment ⬤inkel
# a network from BA model
net <- generate_net(N = 1000 , m = 50 , mode = 1, alpha = 1, s = 0)
net_stats <- get_statistics(net, only_PA = TRUE)
result <- only_A_estimate(net, net_stats)
# plot the estimated attachment function
plot(result, net_stats)
# true function
true_A <- result$estimate_result$center_k
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

#### Example 2: a non-log-linear preferential attachment ⬤inkel
# A_k = alpha* log (max(k,1))^beta + 1, with alpha = 2, and beta = 2
set.seed(1)
net <- generate_net(N = 1000 , m = 50 , mode = 3, alpha = 2, beta = 2, s = 0)
net_stats <- get_statistics(net, only_PA = TRUE)
result <- only_A_estimate(net, net_stats)
# plot the estimated attachment function
plot(result, net_stats)
# true function
true_A <- 2 * log(pmax(result$estimate_result$center_k,1))^2 + 1 # true function
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")

#### Example 3: another non-log-linear preferential attachment kernel ⬤inkel
set.seed(1)
# A_k = min(max(k,1),sat_at)^alpha, with alpha = 1, and sat_at = 200
# inverse variance of the distribution of node fitness = 10
net <- generate_net(N = 1000 , m = 50 , mode = 2, alpha = 1, sat_at = 200, s = 0)
net_stats <- get_statistics(net, only_PA = TRUE)
result <- only_A_estimate(net, net_stats)
# plot the estimated attachment function
true_A <- pmin(pmax(result$estimate_result$center_k,1),200)^1 # true function
plot(result, net_stats, max_A = max(true_A,result$estimate_result$theta))
lines(result$estimate_result$center_k, true_A, col = "red") # true line
legend("topleft" , legend = "True function" , col = "red" , lty = 1 , bty = "n")
```
only_F_estimate

Estimating node fitnesses in isolation

Description

This function estimates node fitnesses $\eta_i$ assuming either $A_k = k$ (i.e. linear preferential attachment) or $A_k = 1$ (i.e. no preferential attachment). The method has a hyper-parameter $s$. It first performs a cross-validation to select the optimal parameter $s$ for the prior of $\eta_i$, then estimates $\eta_i$ with the full data (Ref. 1).

Usage

```r
only_F_estimate(net_object ,
net_stat = get_statistics(net_object),
p = 0.75 ,
stop_cond = 10^-8 ,
model_A = "Linear" ,
...)
```

Arguments

- `net_object` an object of class PAFit_net that contains the network.
- `net_stat` An object of class PAFit_data which contains summerized statistics needed in estimation. This object is created by the function `get_statistics`. The default value is `get_statistics(net_object)`.
- `p` Numeric. This is the ratio of the number of new edges in the learning data to that of the full data. The data is then divided into two parts: learning data and testing data based on $p$. The learning data is used to learn the node fitnesses and the testing data is then used in cross-validation. Default value is 0.75.
- `stop_cond` Numeric. The iterative algorithm stops when $\frac{\text{abs}(h(ii)−h(ii+1))/(\text{abs}(h(ii)) + 1)}{\text{abs}(h(ii))}$ < `stop_cond` where $h(ii)$ is the value of the objective function at iteration $ii$. We recommend to choose `stop.cond` at most equal to $10^{-(\text{number of digits of } h−2)}$, in order to ensure that when the algorithm stops, the increase in posterior probability is less than 1% of the current posterior probability. Default is $10^-8$. This threshold is good enough for most applications.
- `model_A` String. Indicates which attachment function $A_k$ we assume:
  - "Linear": We assume $A_k = k$, i.e. the Bianconi-Barabási model (Ref. 2).
  - "Constant": We assume $A_k = 1$, i.e. the Caldarelli model (Ref. 3).

...
Value

Outputs a `Full_PAFit_result` object, which is a list containing the following fields:

- `cv_data`: a `CV_Data` object which contains the cross-validation data. Normally the user does not need to pay attention to this data.
- `cv_result`: a `CV_Result` object which contains the cross-validation result. Normally the user does not need to pay attention to this data.
- `estimate_result`: this is a `PAFit_result` object which contains the estimated node fitnesses and their confidence intervals. In particular, the important fields are:
  - `shape`: this is the selected value for the hyper-parameter $s$.
  - `g`: the number of bins used.
  - `f`: the estimated node fitnesses.
  - `var_f`: the estimated variance of $\eta_i$.
  - `upper_f`: the estimated upper value of the interval of two standard deviations around $\eta_i$.
  - `lower_f`: the estimated lower value of the interval of two standard deviations around $\eta_i$.
  - `objective_value`: values of the objective function over iterations in the final run with the full data.
  - `diverge_zero`: logical value indicates whether the algorithm diverged in the final run with the full data.

Author(s)

Thong Pham <thongphamthe@gmail.com>

References


See Also

See `get_statistics` for how to create summarized statistics needed in this function.

See `joint_estimate` for the method to jointly estimate the attachment function $A_k$ and node fitnesses $\eta_i$.

Examples

```r
## Not run:
library("PAFit")
set.seed(1)
# size of initial network = 100
# number of new nodes at each time-step = 100
```
```r
# Ak = k; inverse variance of the distribution of node fitnesses = 10
net <- generate_BB(N = 1000, m = 50, num_seed = 100, multiple_node = 100, s = 10)
net_stats <- get_statistics(net)

# estimate node fitnesses in isolation, assuming Ak = k
result <- only_F_estimate(net, net_stats)

# plot the estimated node fitnesses and true node fitnesses
plot(result, net_stats, true = net$fitness, plot = "true_f")
```

## End(Not run)

---

### Description

This function estimates the attachment function $A_k$ from one snapshot.

### Usage

```r
PAFit_oneshot(net_object,
               M = 10,
               S = 5,
               loop = 5,
               G = 1000)
```

### Arguments

- `net_object` an object of class `PAFit_net` that contains the network. Any time-step information, if available, will be ignored.
- `M` Integer. Number of simulated networks in each iteration. Default is 10.
- `S` Integer. Number of iterations inside each loop. Default is 5.
- `loop` Integer. Number of loops of the whole process. Default is 5.
- `G` Integer. Number of bins for the PA function. Default is 1000.

### Value

Outputs a `PAFit_result` object.

### Author(s)

Thong Pham <thongphamthe@gmail.com>
References


Examples

```r
## Not run:
library("PAFit")
net_1  &lt;- generate_BA(N = 10000, alpha = 1) # true attachment exponent = 1.0
result_1 &lt;- PAFit_oneshot(net_1)
print(result_1)

net_2 &lt;- generate_BA(N = 10000, alpha = 0.5) # true attachment exponent = 0.5
result_2 &lt;- PAFit_oneshot(net_2)
print(result_2)
## End(Not run)
```

---

plot.Full_PAFit_result

Plotting the estimated attachment function and node fitness

Description

This function plots the estimated attachment function $A_k$ and node fitness $\eta_i$, together with additional information such as their confidence intervals or the estimated attachment exponent ($\alpha$ when assuming $A_k = k^\alpha$).

Usage

```r
## S3 method for class 'Full_PAFit_result'
plot(x, net_stat , true_f = NULL, plot = "A", plot_bin = TRUE, line = FALSE, confidence = TRUE, high_deg_A = 1, high_deg_f = 5, shade_point = 0.5, col_point = "grey25", pch = 16, shade_interval = 0.5, col_interval = "lightsteelblue", label_x = NULL, label_y = NULL, max_A = NULL, min_A = NULL, f_min = NULL, f_max = NULL, plot_true_degree = FALSE, ...)
```
Arguments

- **x**
  An object of class `Full_PAFit_result`, containing the estimated results from `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

- **net_stat**
  An object of class `PAFit_data`, containing the summerized statistics.

- **true_f**
  Vector. Optional parameter for the true value of node fitnesses (only available in simulated datasets). If this parameter is specified and `plot == "true_f"`, a plot of estimated $\eta$ versus true $\eta$ is produced (after a suitable rescaling of the estimated $f$).

- **plot**
  String. Indicates which plot is produced.
  - If "A" then PA function is plotted.
  - If "f" then the histogram of estimated fitness is plotted.
  - If "true_f" then estimated fitness and true fitness are plotted together (require supplement of true fitness).
  Default value is "A".

- **plot_bin**
  Logical. If TRUE then only the center of each bin is plotted. Default is TRUE.

- **line**
  Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is TRUE.

- **confidence**
  Logical. Indicates whether to plot the confidence intervals of $A_k$ and $\eta_i$ or not. If confidence == TRUE, a 2-sigma confidence interval will be plotted at each $A_k$ and $\eta_i$.

- **high_deg_A**
  Integer. The estimated PA function is plotted starting from `high_deg_A`. Default value is 1.

- **high_deg_f**
  Integer. If `plot == "true_f"`, only nodes whose number of edges acquired is not less than `high_deg_f` are plotted. Default value is 5.

- **col_point**
  String. The name of the color of the points. Default value is "black".

- **shade_point**
  Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.

- **pch**
  Numeric. The plot symbol. Default value is 16.

- **shade_interval**
  Numeric. Value between 0 and 1. This is the transparency level of the confidence intervals. Default value is 0.5.

- **max_A**
  Numeric. Specify the maximum of the axis of PA.

- **min_A**
  Numeric. Specify the minimum of the axis of PA.

- **f_min**
  Numeric. Specify the minimum of the axis of fitness.

- **f_max**
  Numeric. Specify the maximum of the axis of fitness.

- **plot_true_degree**
  Logical. The degree of each node is plotted or not.

- **label_x**
  String. The label of x-axis.

- **label_y**
  String. The label of y-axis.

- **col_interval**
  String. The name of the color of the confidence intervals. Default value is "lightsteelblue".

...
plot.PAFit_net

Plot a PAFit_net object

Description

This function plots a PAFit_net object. There are four options of plot to specify the type of plot.

The first two concern plotting the graph in $graph of the PAFit_net object. Option plot = "graph" plots the graph, while plot = "degree" plots the degree distribution. Option slice allows selection of the time-step at which the temporal graph is plotted.

The last two options concern plotting the PA function and node fitnesses (if they are not NULL).

Usage

## S3 method for class 'PAFit_net'
plot(x,
     plot = "graph",
     slice = length(unique(x$graph[,3])) - 1,
     ...)
Arguments

- **x**: An object of class PAFit_net.
- **plot**: String. Possible values are "graph", "degree", "PA", and "fit". Default value is "graph".
- **slice**: Integer. Ignored when plot is not "graph" or "degree". Specifies the time-step at which the graph is plotted. Default value is the final time-step.

Value

Outputs the desired plot.

Author(s)

Thong Pham <thongphanthe@gmail.com>. When plot = "graph", the function uses `plot.network.default` in the network package.

Examples

```r
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50, m = 10, s = 10)
plot(net, plot = "graph")
plot(net, plot = "degree")
plot(net, plot = "PA")
plot(net, plot = "fit")
```

Description

This function plots the estimated attachment function $A_k$ and node fitness $\eta_i$, together with additional information such as their confidence intervals or the estimated attachment exponent ($\alpha$ when assuming $A_k = k^\alpha$) of a PAFit_result object. This object is stored in the field `$estimate_result` of a Full_PAFit_result object, which in turn is the returning value of `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

Usage

```r
## S3 method for class 'PAFit_result'
plot(x,
     net_stat = NULL,
     true_f = NULL, plot = "A", plot_bin = TRUE,
     line = FALSE, confidence = TRUE, high_deg_A = 1,
     high_deg_f = 5)
```
Arguments

x An object of class PAFit_result.
net_stat An object of class PAFit_data, containing the summerized statistics.
true_f Vector. Optional parameter for the true value of node fitnesses (only available in simulated datasets). If this parameter is specified and plot == "true_f", a plot of estimated η versus true η is produced (after a suitable rescaling of the estimated f).
plot String. Indicates which plot is produced.
  • If "A" then PA function is plotted.
  • If "f" then the histogram of estimated fitness is plotted.
  • If "true_f" then estimated fitness and true fitness are plotted together (require supplement of true fitness).
  Default value is "A".
plot_bin Logical. If TRUE then only the center of each bin is plotted. Default is TRUE.
line Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is TRUE.
confidence Logical. Indicates whether to plot the confidence intervals of A_k and eta_i or not. If confidence == TRUE, a 2-sigma confidence interval will be plotted at each A_k and eta_i.
high_deg_A Integer. The estimated PA function is plotted starting from high_deg_A. Default value is 1.
high_deg_f Integer. If plot == "true_f", only nodes whose number of edges acquired is not less than high_deg_f are plotted. Default value is 5.
col_point String. The name of the color of the points. Default value is "black".
shade_point Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.
pch Numeric. The plot symbol. Default value is 16.
shade_interval Numeric. Value between 0 and 1. This is the transparency level of the confidence intervals. Default value is 0.5.
max_A Numeric. Specify the maximum of the axis of PA.
min_A Numeric. Specify the minimum of the axis of PA.
f_min Numeric. Specify the minimum of the axis of fitness.
f_max Numeric. Specify the maximum of the axis of fitness.
plot_true_degree Logical. The degree of each node is plotted or not.
plot.PA_result

label_x  String. The label of x-axis.
label_y  String. The label of y-axis.
col_interval  String. The name of the color of the confidence intervals. Default value is "lightsteelblue".
...

Value

Outputs the desired plot.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50,
                   num_seed = 100, multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
#plot A
plot(result$estimate_result, net_stats, plot = "A")
true_A <- c(1, result$estimate_result$center_k[-1])
lines(result$estimate_result$center_k + 1, true_A, col = "red") # true line
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")
#plot true_f
plot(result, net_stats, net$fitness, plot = "true_f")
## End(Not run)
```

---

plot.PA_result

Plotting the estimated attachment function

Description

This function plots the estimated attachment function from the corrected Newman’s method or the Jeong’s method. Its also plots additional information such as the estimated attachment exponenten ($\alpha$ when assuming $A_k = k^\alpha$).
Usage

```r
## S3 method for class 'PA_result'
plot(x,
     net_stat = NULL,
     plot_bin = TRUE,
     high_deg = 1,
     line = FALSE,
     col_point = "black",
     shade_point = 0.5,
     pch = 16,
     max_A = NULL,
     min_A = NULL,
     label_x = NULL,
     label_y = NULL,
     ...)
```

Arguments

- `x`: An object of class `PA_result`, containing the estimated attachment function and the estimated attachment exponent from either Newman or Jeong functions.

- `net_stat`: An object of class `PA_data`, containing the summerized statistics. This object is created from the function `get_statistics`.

- `plot_bin`: Logical. If `TRUE` then only the center of each bin is plotted. Default is `TRUE`.

- `high_deg`: Integer. Specifies the starting degree from which $A_k$ is plotted. If this parameter is specified, the estimated attachment function is plotted from $k = \text{high_deg}$.

- `line`: Logical. Indicates whether to plot the line fitted from the log-linear model or not. Default value is `FALSE`.

- `col_point`: String. The name of the color of the points. Default value is "black".

- `shade_point`: Numeric. Value between 0 and 1. This is the transparency level of the points. Default value is 0.5.

- `pch`: Numeric. The plot symbol. Default value is 16.

- `max_A`: Numeric. Specify the maximum of the horizontal axis.

- `min_A`: Numeric. Specify the minimum of the horizontal axis.

- `label_x`: String. The label of x-axis. If `NULL`, then "Degree k" is used.

- `label_y`: String. The label of y-axis. If `NULL`, then "Attachment function" is used.

Value

Outputs the desired plot.

Author(s)

Thong Pham <thongphanthe@gmail.com>
Examples

```r
library("PAFit")
net <- generate_net(N = 1000, m = 1, mode = 1, alpha = 1, s = 0)
net_stats <- get_statistics(net)
result <- Newman(net, net_stats)
# true function
true_A <- result$center_k
# plot the estimated attachment function
plot(result, net_stats)
lines(result$center_k, true_A, col = "red") # true attachment function
legend("topleft", legend = "True function", col = "red", lty = 1, bty = "n")
```

Description

This function extracts from a `Simulated_Data_From_Fitted_Model` object contributions of rich-get-richer and fit-get-richer effects calculated using simulated networks and plots these contributions versus the contributions calculated from the original observed network. See `joint_estimate` for a description of how the contributions are calculated.

Usage

```r
plot_contribution(simulated_object, original_result, which_plot = "PA", y_label = ifelse("PA" == which_plot,
"Contribution of the rich-get-richer effect",
"Contribution of the fit-get-richer effect"),
legend_pos_x = 0.75, legend_pos_y = 0.9)
```

Arguments

- `simulated_object`: an object of class `Simulated_Data_From_Fitted_Model` that contains simulated data.
- `original_result`: an object of class `Full_PAFit_result` that contains the estimation results from the original observed data.
- `which_plot`: String. "PA": plots contributions of rich-get-richer effect, "fit": plots contribution of fit-get-richer effect. Default is "PA".
- `y_label`: String. The label for y-axis. Default is "Contribution of rich-get-richer effect".
- `legend_pos_x`: Numeric. The horizontal position, between (0,1), of the legend. Default value is 0.75.
legend_pos_y Numeric. The vertical position, between (0,1), of the legend. Default value is 0.9.

Value
Output a plot.

Author(s)
Thong Pham <thongphamthe@gmail.com>

References


See Also
joint_estimate, plot_contribution

Examples
```r
## Not run:
library("PAFit")
net_object <- generate_net(N = 500, m = 10, s = 10, alpha = 0.5)
net_stat <- get_statistics(net_object)
result <- joint_estimate(net_object, net_stat)
simulated_data <- generate_simulated_data_from_estimated_model(net_object, net_stat, result)
plot_contribution(simulated_data, result, which_plot = "PA")
plot_contribution(simulated_data, result, which_plot = "fit")

## End(Not run)
```
**Description**

This function prints simple information of the cross-validation data stored in a `CV_Data` object. This object is the field `$cv_data` of a `Full_PAFit_result` object, which in turn is the returning value of `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

**Usage**

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

**Arguments**

- `x` An object of class `CV_Data`.
- `...`

**Value**

Prints simple information of the cross-validation data.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>

**Examples**

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50, 
                   num_seed = 100, multiple_node = 100, 
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
print(result$cv_data)
## End(Not run)
```
Description

This function prints simple information of the cross-validation result stored in a CV_Result object. This object is the field $cv_result of a Full_PAFit_result object, which in turn is the returning value of only_A_estimate, only_F_estimate or joint_estimate.

Usage

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

Arguments

- `x` An object of class CV_Result.
- `...`

Value

Prints simple information of the cross-validation result.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50,
                   num_seed = 100, multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
print(result$cv_result)
```

## End(Not run)
print.Full_PAFit_result

printing information on the estimation result

Description

This function outputs simple information of the estimation result.

Usage

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

Arguments

- `x`
  - An object of class `Full_PAFit_result`, containing the estimated results from `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

- `...`

Value

Outputs summary information on the estimation result.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50,
                   num_seed = 100, multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
print(result)
## End(Not run)
```
print.PAFit_data  

Printing simple information on the statistics of the network stored in a PAFit_data object

Description

This function prints simple information of the statistics stored in a PAFit_data object. This object is the returning value of get_statistics.

Usage

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

Arguments

- `x` An object of class PAFit_data.
- `...`

Value

Prints simple information of the network statistics.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
print(net_stats)

## End(Not run)
```
print.PAFit_net  

Printing simple information of a PAFit_net object

Description

This function outputs simple information of a PAFit_net object.

Usage

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

Arguments

- `x` An object of class PAFit_net.
- `...`

Value

Outputs simple information of the network.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50, m = 10, s = 10)
print(net)
```

print.PAFit_result  

printing information on the estimation result stored in a PAFit_result object

Description

This function outputs simple information of the estimation result stored in a PAFit_result object. This object is stored in the field $estimate_result of a Full_PAFit_result object, which in turn is the returning value of only_A_estimate, only_F_estimate or joint_estimate.

Usage

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

x An object of class PAFit_result.

Value

Outputs summary information on the estimation result.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50, 
num_seed = 100, multiple_node = 100,
s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
print(result$estimate_result)

## End(Not run)

print.PA_result

Printing information of the estimated attachment function

Description

This function outputs simple information of the estimated attachment function from the corrected Newman’s method or the Jeong’s method.

Usage

## S3 method for class ’PA_result’
print(x, 

...)

Arguments

x An object of class PA_result, containing the estimated attachment function and the estimated attachment exponent from either Newman or Jeong functions.

... Additional parameters to pass onto the plot function.
Value

Simple information of the estimated attachment function.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
library("PAFit")
net <- generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
net_stats <- get_statistics(net)
result <- Newman(net, net_stats)
print(result)
```

Description

This function outputs summary information of the cross-validation data stored in a `CV_Data` object. This object is the field `$cv_data` of a `Full_PAFit_result` object, which in turn is the returning value of `only_A_estimate`, `only_F_estimate` or `joint_estimate`.

Usage

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

Arguments

- `object` An object of class `CV_Data`.
- `...`

Value

Outputs summary information of the cross-validation data.

Author(s)

Thong Pham <thongphamthe@gmail.com>
Examples

## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50,
                   num_seed = 100, multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
summary(result$cv_data)
## End(Not run)

summary.CV_Result

Output summary information of the cross-validation result

Description

This function outputs summary information of the cross-validation result stored in a CV_Result object. This object is the field $cv_result of a Full_PAFit_result object, which in turn is the returning value of only_A_estimate, only_F_estimate or joint_estimate.

Usage

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

Arguments

object An object of class CV_Result.
...

Value

Outputs summary information of the cross-validation result.

Author(s)

Thong Pham <thongphanthe@gmail.com>
Examples

## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50,
                  num_seed = 100, multiple_node = 100,
                  s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
summary(result$cv_result)
## End(Not run)

---

**summary.Full_PAFit_result**

*Summary information on the estimation result*

**Description**

This function outputs a summary on the estimation result.

**Usage**

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

**Arguments**

- `object` An object of class Full_PAFit_result, containing the estimated results from only_A_estimate, only_F_estimate or joint_estimate.

**Value**

Outputs summary information on the estimation result.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>
Examples

## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000, m = 50, 
                  num_seed = 100, multiple_node = 100, 
                  s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
summary(result)

## End(Not run)

---

**summary.PAFit_data**

Output summary information on the statistics of the network stored in a PAFit_data object

**Description**

This function outputs summary information of the statistics stored in a PAFit_data object. This object is the returning value of `get_statistics`.

**Usage**

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

**Arguments**

- `object` An object of class PAFit_data.
- `...`

**Value**

Outputs summary information of the network statistics.

**Author(s)**

Thong Pham <thongphamthe@gmail.com>
## Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
summary(net_stats)

## End(Not run)
```

---

### summary.PAFit_net

#### Summary information of a PAFit_net object

**Description**

This function outputs summary information of a `PAFit_net` object.

**Usage**

```r
## S3 method for class 'PAFit_net'
summary(object, ...
```

**Arguments**

- `object` An object of class `PAFit_net`.

**Value**

Outputs summary information of the network.

**Author(s)**

Thong Pham `<thongphamthe@gmail.com>`

**Examples**

```r
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
summary(net)
```
summary.PAFit_result  

Output summary information on the estimation result stored in a PAFit_result object

Description

This function outputs summary information of the estimation result stored in a PAFit_result object. This object is stored in the field $estimate_result of a Full_PAFit_result object, which in turn is the returning value of only_A_estimate, only_F_estimate or joint_estimate.

Usage

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

Arguments

- `object`: An object of class PAFit_result.
- `...`

Value

Outputs summary information on the estimation result.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
## Since the runtime is long, we do not let this example run on CRAN
## Not run:
library("PAFit")
set.seed(1)
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 1000 , m = 50 ,
                   num_seed = 100 , multiple_node = 100,
                   s = 10)
net_stats <- get_statistics(net)
result <- joint_estimate(net, net_stats)
summary(result$estimate_result)
```

## End(Not run)
Description

This function outputs summary information of the estimated attachment function from the corrected Newman’s method or the Jeong’s method.

Usage

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

Arguments

- `object` An object of class `PA_result`, containing the estimated attachment function and the estimated attachment exponent from either Newman or Jeong functions.
- `...` Additional parameters to pass onto the `plot` function.

Value

Summary information of the estimated attachment function.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```r
test_with = generate_net(N = 1000 , m = 1 , mode = 1 , alpha = 1 , s = 0)
test_stats <- get_statistics(test_with)
result <- Newman(test_with, test_stats)
summary(result)
```
Description

This function implements the method in Handcock and Jones (2004) to fit various distributions to a degree vector. The implemented distributions are Yule, Waring, Poisson, geometric and negative binomial. The Yule and Waring distributions correspond to a preferential attachment situation. In particular, the two distributions correspond to the case of $A_k = k$ for $k \geq 1$ and $\eta_i = 1$ for all $i$ (note that, the number of new edges and new nodes at each time-step are implicitly assumed to be 1).

Thus, if the best fitted distribution, which is chosen by either the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC), is NOT Yule or Waring, then the case of $A_k = k$ for $k \geq 1$ and $\eta_i = 1$ for all $i$ is NOT consistent with the observed degree vector.

The method allows the low-tail probabilities to NOT follow the parametric distribution, i.e., $P(K = k) = \pi_k$ for all $k \leq k_{\text{min}}$ and $P(K = k) = f(k; \theta)$ for all $k > k_{\text{min}}$. Here $k_{\text{min}}$ is the degree threshold above which the parametric distribution holds, $\pi_k$ are probabilities of the low-tail, $f(\cdot, \theta)$ is the parametric distribution with parameter vector $\theta$.

For fixed $k_{\text{min}}$ and $f$, $\pi_k$ and $\theta$ can be estimated by Maximum Likelihood Estimation. We can choose the best $k_{\text{min}}$ for each $f$ by comparing the AIC (or BIC). More details can be founded in Handcock and Jones (2004).

Usage

test_linear_PA(degree_vector)

Arguments

degree_vector a degree vector

Value

Outputs a Linear_PA_test_result object which contains the fitting of five distributions to the degree vector: Yule (yule), Waring (waring), Poisson (pois), geometric (geom) and negative binomial (nb). In particular, for each distribution, the AIC and BIC are calculated for each $k_{\text{min}}$.

Author(s)

Thong Pham <thongphanthe@gmail.com>

References


Examples

```r
## Not run:
library("PAFit")
set.seed(1)
net <- generate_BA(n = 1000)
```
stats <- get_statistics(net, only_PA = TRUE)
u <- test_linear_PA(stats$final_deg)
print(u)

## End(Not run)

to_igraph

Convert a PAFit_net object to an igraph object

Description

This function converts a PAFit_net object to an igraph object (of package igraph).

Usage

```
to_igraph(net_object)
```

Arguments

net_object An object of class PAFit_net.

Value

The function returns an igraph object.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

```
library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50 , m = 10 , s = 10)
igraph_graph <- to_igraph(net)
```
to_networkDynamic

Convert a PAFit_net object to a networkDynamic object

Description

This function converts a PAFit_net object to a networkDynamic object (of package networkDynamic).

Usage

to_networkDynamic(net_object)

Arguments

net_object  An object of class PAFit_net.

Value

The function returns a networkDynamic object.

Author(s)

Thong Pham <thongphamthe@gmail.com>

Examples

library("PAFit")
# a network from Bianconi-Barabasi model
net <- generate_BB(N = 50, m = 10, s = 10)
nD_graph <- to_networkDynamic(net)
Index

* Barabasi-Albert model
  generate_BA, 8
  generate_net, 14
  PAFit-package, 3
* Bianconi-Barabasi model
  generate_BB, 10
  generate_net, 14
  PAFit-package, 3
* Corrected Newman’s method
  PAFit-package, 3
* ER model
  generate_ER, 11
* Jeong’s method
  PAFit-package, 3
* attachment function
  generate_simulated_data_from_estimated_model,
  17
  Jeong, 24
  joint_estimate, 26
  Newman, 30
  only_A_estimate, 31
  PAFit-package, 3
  PAFit_oneshot, 37
  plot_contribution, 45
* fitness model
  generate_fit_only, 13
  generate_net, 14
  only_F_estimate, 35
  PAFit-package, 3
* fitness
  generate_simulated_data_from_estimated_model,
  17
  joint_estimate, 26
  plot_contribution, 45
* fitting degree distributions
  test_linear_PA, 59
* linear preferential attachment
  test_linear_PA, 59
* preferential attachment
  generate_simulated_data_from_estimated_model,
  17
  Jeong, 24
  joint_estimate, 26
  Newman, 30
  only_A_estimate, 31
  PAFit-package, 3
  PAFit_oneshot, 37
  plot_contribution, 45
* temporal complex networks
  get_statistics, 19
  PAFit-package, 3
as.PAFit_net, 5, 19, 21
coauthor.author_id(Coauthorship
  network of scientists in the field of complex networks), 6
coauthor.net, 3
coauthor.net(Coauthorship network of
  scientists in the field of complex networks), 6
coauthor.truetime(Coauthorship
  network of scientists in the field of complex networks), 6
Coauthorship network of scientists in
  the field of complex networks, 6
ComplexNetCoauthor(Coauthorship
  network of scientists in the field of complex networks), 6
from_igraph, 7, 19, 21
from_networkDynamic, 8, 19, 21
generate_BA, 3, 8, 11, 12, 14, 17
generate_BB, 3, 9, 10, 12, 14, 17
generate_ER, 3, 9, 11, 11, 14, 17
generate_fit_only, 3, 9, 11, 12, 13, 17
generate_net, 3, 8, 9, 11, 12, 14, 14
generate_simulated_data_from_estimated_model, 17
get_statistics, 3, 4, 9, 11, 12, 14, 17, 18, 19, 24–26, 28, 30–33, 35, 36, 44, 50, 56
graph_from_file, 19, 21, 22
graph_to_file, 22, 23
Jeong, 4, 21, 24, 28, 31, 33, 44, 52, 59
joint_estimate, 4, 18, 21, 26, 36, 39, 41, 45–49, 51, 53–55, 58
Newman, 4, 21, 25, 28, 30, 33, 44, 52, 59
only_A_estimate, 4, 21, 25, 28, 31, 39, 41, 47–49, 51, 53–55, 58
only_F_estimate, 4, 21, 28, 35, 39, 41, 47–49, 51, 53–55, 58
PAFit (PAFit-package), 3
PAFit-package, 3
PAFit_data (get_statistics), 19
PAFit_oneshot, 4, 37
plot.Full_PAFit_result, 38
plot.network.default, 41
plot.PA_result, 43
plot.PAFit_net, 40
plot.PAFit_result, 41
plot_contribution, 18, 45, 46
print.CV_Data, 47
print.CV_Result, 48
print.Full_PAFit_result, 49
print.PA_result, 52
print.PAFit_data, 50
print.PAFit_net, 51
print.PAFit_result, 51
summary.CV_Data, 53
summary.CV_Result, 54
summary.Full_PAFit_result, 55
summary.PA_result, 59
summary.PAFit_data, 56
summary.PAFit_net, 57
summary.PAFit_result, 58
test_linear_PA, 59
to_igraph, 61
to_networkDynamic, 62