Package: rsetse (via r-universe)

August 28, 2024

Title Strain Elevation Tension Spring Embedding

Version 0.5.0

Description An R implementation for the Strain Elevation and Tension embedding algorithm from Bourne (2020) <doi:10.1007/s41109-020-00329-4>. The package embeds graphs and networks using the Strain Elevation and Tension embedding (SETSe) algorithm. SETSe represents the network as a physical system, where edges are elastic, and nodes exert a force either up or down based on node features. SETSe positions the nodes vertically such that the tension in the edges of a node is equal and opposite to the force it exerts for all nodes in the network. The resultant structure can then be analysed by looking at the node elevation and the edge strain and tension. This algorithm works on weighted and unweighted networks as well as networks with or without explicit node features. Edge elasticity can be created from existing edge weights or kept as a constant.

Depends R (>= 3.4.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports dplyr, Matrix, rlang (>= 0.1.2), igraph, purrr, tibble, minpack.lm, magrittr, methods, stats, Rcpp, RcppArmadillo

RoxygenNote 7.2.0

Suggests knitr, rmarkdown, tidyr, ggplot2, ggraph, roxygen2

VignetteBuilder knitr

URL https://github.com/JonnoB/rSETSe

BugReports https://github.com/JonnoB/rSETSe/issues

LinkingTo Rcpp, RcppArmadillo

Repository https://jonnob.r-universe.dev

RemoteUrl https://github.com/jonnob/rsetse

RemoteRef HEAD

RemoteSha 8ba5e4e43b636d33b7ffa3a98a21fbe351886c18

Contents

`biconnected_network` *A simple network made of three bi-connected components*

Description

The data set can be used to explore simple different embeddings methods on a very simple graph

Usage

```
biconnected_network
```

Format

An igraph network with 7 nodes and 19 edges which forms three biconnected components:

edge_name The name of the edge connecting the two vertices

weight The edge weight connecting the two vertices. This value is 1000 for edges connecting nodes A to D, it is 500 for edges connecting nodes E to G, it is 100 connecting nodes D and E

force The force produced by each node. It was calculated by subtracting the mean node centrality for the network from the node centrality

group The group each node is in. This can be used to generate force if required

Examples

```
## Not run: plot(biconnected_network)
```

---

calc_spring_area *Calculate the cross sectional area of the edge*

Description

This function adds the graph characteristic A which is the cross sectional area of the edge.

Usage

```
calc_spring_area(g, value, minimum_value, range)
```

Arguments

g	an igraph object. The graph representing the network
value	a character string. The name of the edge attribute that is used as value from which Area will be calculated
minimum_value	a numeric value. Indicating the most thinnest edge
range	a numeric value. This gives the range of A values above the minimum.

Details

This function is pretty niche but calculates a cross sectional area of an edge. This is useful when you wish to calculate the spring coefficient k using Young's modulus. The function coerces and edge characteristic to be within a certain range of values preventing negative/zero/infinite values.

Value

a igraph object with the new edge attribute "Area" for each edge

Examples

```
library(igraph)
set.seed(234)
g_prep <- generate_peels_network("A") %>%
 set.edge.attribute(., name = "edge_characteristic", value = rep(1:16, each = 10))

g <- calc_spring_area(g_prep, value = "edge_characteristic", minimum_value = 10, range = 20)

get.edge.attribute(g, "Area")
```

---

calc_spring_constant *Calculate the spring constant*

Description

This function adds the graph characteristic k which is the spring constant for a given Area and Young's modulus.

Usage

```
calc_spring_constant(g, youngs_mod = "E", A = "Area", distance = "distance")
```

Arguments

g	an igraph object. The graph representing the network
youngs_mod	a character string. The Young's modulus of the edge. The default is E
A	a character string. The cross sectional area of the line. The default is A. see details on values of A
distance	A character string. See details on values of distance

Details

When A and distance are both set to 1 k=E and the spring constant is equivalent to Young's modulus. In this case there is no need to call this function as the edge weight representing youngs modulus can be used for k instead.

The values A and distance are edge attributes referring to the cross-sectional area of the edge and the horizontal distance of the edge, in other words the distance between the two nodes at each end of the edge. These values can be set to anything the user wishes, they may be constant or not. However, consider carefully setting the values to anything else other than 1. There needs to be a clear reasoning or the results will be meaningless.

For example setting the distance of an edge that represents an electrical cable to the distance of the electrical cable will return very different results when compared to a constant of one. However, the physical distance between two points does not necessarily have an impact on the loading of the line and so the results would not be interpretable. In contrast setting the distance metric to be some function of the line resistance may have meaning and be appropriate. As a general rule distance and area should be set to 1.

Value

and edge attribute called k with value `EA/distance`

See Also

[calc_spring_area]

Examples

```
library(igraph)
set.seed(234)
g_prep <- generate_peels_network("A") %>%
 set.edge.attribute(., name = "edge_characteristic", value = rep(1:16, each = 10)) %>%
 #set some pretend Young's modulus value
 set.edge.attribute(., name = "E", value = rep(c(1e5, 5e5, 2e5, 3e5), each = 40)) %>%
 #calculate the spring area from another edge characteristic
 calc_spring_area(., value = "edge_characteristic", minimum_value = 10, range = 20) %>%
 prepare_edges() %>%
 prepare_categorical_force(., node_names = "name",
                   force_var = "class")

g <- calc_spring_constant(g_prep, youngs_mod = "E", A = "Area", distance = "distance")
```

---

calc_tension_strain *Calculate line tension and strain from the topology and node embeddings*

Description

This function calculates the line tension and strain characteristics for the edges in a graph. It is called by default by all the embedding functions (SETSe_*) but is included here for completeness.

Usage

```
calc_tension_strain(
  g,
  height_embeddings_df,
  distance = "distance",
  edge_name = "edge_name",
  k = "k"
)
```

Arguments

g	An igraph object of the network.
height_embeddings_df	A data frame. This is the results of Create_stabilised_blocks or Find_network_balance
distance	A character string. The name of the edge attribute that contains the distance between two nodes. The default is "distance"
edge_name	A character string. The name of the edge attribute that contains the edge name. The default is "edge_name".
k	A character string. The name of the edge attribute that contains the spring coefficient

Details

Whilst the node embeddings dataframe contains the elevation of the setse algorithm this function produces a data frame that contains the Tension and Strain. The dataframe that is returned contains a substantial amount of line information so reducing the number of variables may be necessary if the data frame will be merged with previously generated data as there could be multiple columns of the same value. This function is called by default at the end of all setse functions

Value

The function returns a data frame of 7 columns. These columns are the edge name, the change in elevation, The final distance between the two nodes (the hypotenuse of the original distance and the vertical distance), the spring constant k, the edge tension, the edge strain, and the mean elevation.

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")%>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                    force_var = "class")
#embed the network using auto setse
embeddings <- setse_auto(g, force = "class_A")

 edge_embeddings_df <- calc_tension_strain(g, embeddings$node_embeddings)
 all.equal(embeddings$edge_embeddings, edge_embeddings_df)
```

---

calc_tension_strain_hd

Calculate line tension and strain from the topology and node embeddings for high dimensional feature networks

---

Description

This function calculates the line tension and strain characteristics for the edges in a graph. It is called by default by all the embedding functions (SETSe_*) but is included here for completeness.

Usage

```
calc_tension_strain_hd(
  g,
  height_embeddings_df,
  distance = "distance",
  edge_name = "edge_name",
  k = "k"
)
```

Arguments

g	An igraph object of the network.
height_embeddings_df	A data frame. This is the results of Create_stabilised_blocks or Find_network_balance
distance	A character string. The name of the edge attribute that contains the distance between two nodes. The default is "distance"
edge_name	A character string. The name of the edge attribute that contains the edge name. The default is "edge_name".
k	A character string. The name of the edge attribute that contains the spring coefficient

Details

Whilst the node embeddings dataframe contains the elevation of the setse algorithm this function produces a data frame that contains the Tension and Strain. The dataframe that is returned contains a substantial amount of line information so reducing the number of variables may be necessary if the data frame will be merged with previously generated data as there could be multiple columns of the same value. This function is called by default at the end of all setse functions

Value

The function returns a data frame of 7 columns. These columns are the edge name, the change in elevation, The final distance between the two nodes (the hypotenuse of the original distance and the vertical distance), the spring constant k, the edge tension, the edge strain, and the mean elevation.

Examples

```
g <- biconnected_network %>%
 prepare_edges(., k = 1000) %>%
 #prepare the continuous features as normal
 prepare_continuous_force(., node_names = "name", force_var = "force") %>%
 #prepare the categorical features as normal
 prepare_categorical_force(., node_names = "name", force_var = "group")
```

```
#embed them using the high dimensional function
two_dimensional_embeddings <- setse_auto_hd(g, force = c("group_A", "force"), k = "weight")

edge_embeddings_df <- calc_tension_strain_hd(g, two_dimensional_embeddings$node_embeddings)
all.equal(two_dimensional_embeddings$edge_embeddings, edge_embeddings_df)
```

`create_balanced_blocks` *Create balanced blocks*

Description

Separates the network into a series of bi-connected components that can be solved separately. Solving smaller subgraphs using the bi-connected component method reduces the risk of network divergence. This function is seldom called independently of setse_bicomp

Usage

```
create_balanced_blocks(g, force = "force", bigraph = bigraph)
```

Arguments

`g`	An igraph object. The network for which embeddings will be found
`force`	A character vector. The name of the node attribute that is the force exerted by the nodes
`bigraph`	A list. the list of biconnected components produced by the biconnected_components function. This function take a non trivial amount of time on large graphs so this pass through minimises the function being called.

Details

When networks are separated into the bi-connected subgraphs or blocks. The overall network balance needs to be maintained. `create_balanced_blocks` maintains the balance by summing the net force across the all the nodes that are being removed from the subgraph. Therefore a node that is an articulation point has a force value equal to the total of all the nodes on the adjacent bi-connected component.

Value

A list containing all the bi connected component where each component is balanced to have a net force of 0.

Examples

```
library(igraph)
#create a list of balanced network using the biconnected_network dataset
balanced_list <-create_balanced_blocks(biconnected_network,
bigraph = biconnected_components(biconnected_network))

#count the edges in each of the bi-components
sapply(balanced_list, ecount)
```

---

create_node_edge_df *Create dataframe of node and aggregated edge embeddings*

Description

Aggregates edge strain and tension to node level

Usage

```
create_node_edge_df(embeddings_data, function_names = c("mean", "median"))
```

Arguments

embeddings_data	A list. The output of any of the setse embedding functions
function_names	A string vector. the names of the aggregation methods to be used

Details

Often if can be useful to have edge data at node level, an example of this would be plotting the node and tension or strain. To do this requires that the edge embeddings are aggregated somehow to node level and joined to the appropriate node. This function takes as an argument the output of the setse embedding functions and any number of aggregation functions to produce a dataframe that is convenient to use.

Value

A dataframe with node names, node force, node elevation and strain and tension aggregated using the named functions. The strain and tension columns are returned with names in the form "strain_x" where "x" is the name of the function used to aggregate. The total number of columns is dependent on the number of aggregation functions.

Examples

```
embeddings_data <- biconnected_network %>%
 prepare_edges(.) %>%
 prepare_continuous_force(., node_names = "name", force_var = "force") %>%
 setse_auto(., k = "weight")

out <- create_node_edge_df(embeddings_data, function_names = c("mean", "mode", "sum"))
```

---

create_node_edge_df_hd

Create dataframe of node and aggregated edge embeddings for high dimensional feature networks

---

Description

Aggregates edge strain and tension to node level

Usage

```
create_node_edge_df_hd(embeddings_data, function_names = c("mean", "median"))
```

Arguments

`embeddings_data`	A list. The output of any of the setse embedding functions
`function_names`	A string vector. the names of the aggregation methods to be used

Details

Often if can be useful to have edge data at node level, an example of this would be plotting the node and tension or strain. To do this requires that the edge embeddings are aggregated somehow to node level and joined to the appropriate node. This function takes as an argument the output of the setse embedding functions and any number of aggregation functions to produce a dataframe that is convenient to use.

Value

A dataframe with node names, node force, node elevation and strain and tension aggregated using the named functions. The strain and tension columns are returned with names in the form "strain_x" where "x" is the name of the function used to aggregate. The total number of columns is dependent on the number of aggregation functions.

Examples

```
g <- biconnected_network %>%
 prepare_edges(.) %>%
 #prepare the continuous features as normal
 prepare_continuous_force(., node_names = "name", force_var = "force") %>%
 #prepare the categorical features as normal
 prepare_categorical_force(., node_names = "name", force_var = "group")

 #embed them using the high dimensional function
 two_dimensional_embeddings <- setse_auto_hd(g, force = c("group_A", "force"), k = "weight")

 out <- create_node_edge_df_hd(two_dimensional_embeddings ,
  function_names = c("mean", "mode", "sum"))
```

`generate_peels_network` *Create a random Peel network*

Description

Creates an example of a network from Peel's quintet of the specified type.

Usage

```
generate_peels_network(
  type,
  k_values = c(1000, 500, 100),
  single_component = TRUE
)
```

Arguments

`type`	A character which is any of the capital letters A-E
`k_values`	An integer vector. The spring constant for the edge types within sub class, within class but not sub-class, between classes. The default value is 1000, 500, 100. This means the strongest connection is for nodes in the same sub-class and the weakest connection is for nodes in different classes
`single_component`	Logical. Guarantees a single component network. Set to TRUE as default

Details

This function generates networks matching the 5 types described in Peel et al 2019 (doi:10.1073/pnas.1713019115). All networks have 40 nodes, 60 edges, two node classes and four node sub-classes. The connections between the are equal across all 5 types. As a result all networks generated have identical assortativity. However, as the sub-classes have different connection probability the structures produced by the networks are very different. When projected into SETSe space the network types occupy there own area, see Bourne 2020 (doi:10.1007/s41109020003294) for details.

Value

An igraph object that matches one of the 5 Peel's quintet types. The nodes are labeled with class and sub class. The edges have attribute k which is the spring constant of the edge given relationship between the nodes the edge connects to

Examples

```
set.seed(234)
g <- generate_peels_network(type = "E")
plot(g)
```

---

hello *Hello, World!*

Description

Prints 'Hello, world!'.

Usage

```
hello()
```

Examples

```
hello()
```

---

mass_adjuster *Mass adjuster*

Description

This function adjusts the mass of the nodes so that the force in each direction over the mass for that direction produces an acceleration of 1.

Usage

```
mass_adjuster(g, force = "force", resolution_limit = TRUE)
```

Arguments

g	An igraph object. the network
force	A character string. The name of the network attribute contain the network forces. Default is "force"
resolution_limit	logical. If the forces in the network are smaller than the square root of the machine floating point limit then the mass is set to one. default is true

Details

This function can help stabilise the convergence of networks by preventing major imbalances between the force in the network and the mass of the nodes. in certain cases acceleration can become very large or very small in if force and mass are not well parametrised.

This function means that if the network were reduced to two nodes where each node contained all the mass and all the force of one of the two directions, then each node would have an acceleration of 1ms^-2

The function can become important when using setse_bicomp as the force mass ratio of biconnection components can vary widely from the total force mass ratio of the network.

Value

A numeric value giving the adjusted mass of the nodes in the network.

Examples

```
set.seed(234) #set the random see for generating the network

set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E") %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                     force_var = "class")

mass_adjuster(g, force = "class_B", resolution_limit = TRUE)
```

---

prepare_categorical_force

Prepare categorical features for embedding

Description

This function prepares a binary network for SETSe projection.

Usage

```
prepare_categorical_force(g, node_names, force_var, sum_to_one = TRUE)
```

Arguments

g	an igraph object
node_names	a character string. A vertex attribute which contains the node names.
force_var	A vector of force attributes. This describes all the categorical force attributes of the network. All named attributes must be either character or factor attributes.
sum_to_one	Logical. whether the total positive force sums to 1, if FALSE the total is the sum of the positive cases

Details

The network takes in an igraph object and produces an undirected igraph object that can be used with the embedding functions.

The purpose of the function is to easily be able to project categorical features using SETSe. The function creates new variables where each variable represents one level of the categorical variables. For embedding only n-1 of the levels are needed.

The function creates several variables of the format "force_". Vertex attributes representing the force produced by each node for each categorical value, there will be n of these variables representing each level of the categorical values. The variable names will be the the name of the variable and the name of the level seperated by and underscore. For example, with a variable group and levels A and B, the created force variables will be "group_A" and "group_B" The sum of these variables will be 0.

Value

A network with the correct node attributes for the embeddings process.

See Also

setse, setse_auto, setse_bicomp, setse_auto_hd

Other prepare_setse: `prepare_continuous_force()`, `prepare_edges()`

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")
embeddings <- g %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                      force_var = "class") %>%
#embed the network using auto_setse
  setse_auto(., force = "class_A")
```

`prepare_continuous_force` *Prepare continuous features for embedding*

Description

This function prepares a continuous network for SETSe projection. The function works for networks with a single feature or high-dimensional features. The network takes in an igraph object and produces an undirected igraph object that can be used with the embedding functions.

Usage

```
prepare_continuous_force(
  g,
  node_names,
  k = NULL,
  force_var,
  sum_to_one = TRUE,
  distance = 1
)
```

Arguments

g	an igraph object
node_names	a character string. A vertex attribute which contains the node names.
k	The spring constant. This value is either a numeric value giving the spring constant for all edges or NULL. If NULL is used the k value will not be added to the network. This is useful k is made through some other process.
force_var	A character vector. This is the vector of node attributes to be used as the force variables. All the attributes must be a numeric or integer value, and cannot have NA's. On a single variable embedding this is usually "force"
sum_to_one	Logical. whether the total positive force sums to 1, if FALSE the total is the sum of the positive cases
distance	a positive numeric value. The default is 1

Details

The function subtracts the mean from all the values so that the system is balanced. If sum_to_one is true then everything is divided by the absolute sum over two

The function adds the node attribute 'force' and the edge attribute 'k' unless k=NULL. The purpose of the function is to easily be able to project continuous networks using SETSe.

The function creates several variables

- force: a vertex attribute representing the force produced by each node. The sum of this variable will be 0
- k: The spring constant representing the stiffness of the spring.
- edge_name: the name of the edges. it takes the form "from_to" where "from" is the origin node and "to" is the destination node using the as_data_frame function from igraph

Value

A network with the correct edge and node attributes for the embeddings process.

See Also

setse_auto_hd

Other prepare_setse: `prepare_categorical_force()`, `prepare_edges()`

Examples

```
embeddings <- biconnected_network %>%
#prepare the network for a binary embedding
#k is already present in the data so is left null in the preparation function
prepare_edges(k = NULL, distance = 1) %>%
prepare_continuous_force(., node_names = "name", force_var = "force") %>%
#embed the network using auto_setse
#in the biconnected_network dataset the edge weights are used directly as k values
setse_auto(k = "weight")
```

`prepare_edges` *Prepare network edges*

Description

This function helps prepare the network edges for embedding

Usage

```
prepare_edges(g, k = NULL, distance = 1, create_edge_name = TRUE)
```

Arguments

`g`	an igraph object
`k`	The spring constant. This value is either a numeric value giving the spring constant for all edges or NULL. If NULL is used the k value will not be added to the network. This is useful k is made through some other process.
`distance`	The spring constant. This value is either a numeric value giving the spring constant for all edges or NULL. If NULL is used the distance value will not be added to the network. This is useful distance is made through some other process.
`create_edge_name`	Logical. Whether to create and edge name attribute or not.

@details The function prepares the edge characteristics of the network so that they can be embedded using the SETSe_ family of functions.

@return The function creates several variables

- k: The spring constant representing the stiffness of the spring.
- distance: The minimum distance between nodes. This is the distance between the parallel planes/hyper-planes.
- edge_name: the name of the edges. it takes the form "from_to" where "from" is the origin node and "to" is the destination node using the as_data_frame function from igraph

See Also

setse, setse_auto, setse_bicomp, setse_auto_hd

Other prepare_setse: `prepare_categorical_force()`, `prepare_continuous_force()`

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")
embeddings <- g %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                     force_var = "class") %>%
#embed the network using auto setse
  setse_auto(., force = "class_A")
```

---

rcpp_hello *Hello, Rcpp!*

Description

Returns an R `list` containing the character vector `c("foo", "bar")` and the numeric vector `c(0, 1)`.

Usage

```
rcpp_hello()
```

Examples

```
rcpp_hello()
```

---

remove_small_components *Remove small components*

Description

keep only the largest component of graph

Usage

```
remove_small_components(g)
```

Arguments

g	An igraph object of the graph to embed.

Details

As setse only works on connected components this function removes all but the largest component. This is a helper function to quickly project a network with setse.

Value

An igraph object.

Examples

```
library(igraph)
set.seed(1284)
#generate a random erdos renyi graph with 100 nodes and 150 edges
g <- erdos.renyi.game(n=100, p.or.m = 150, type = "gnm" )
#count the number of components
components(g)$no

#remove all but the largest component
g2 <-remove_small_components(g)

#Now there is only 1 component
igraph::components(g2)$no
```

---

setse *Basic SETSe embedding*

Description

Embeds/smooths a feature network using the basic SETSe algorithm. generally setse_auto or setse_bicomp is preferred.

Usage

```
setse(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
  tstep = 0.02,
  mass = 1,
  max_iter = 20000,
  coef_drag = 1,
  tol = 1e-06,
  sparse = FALSE,
  two_node_solution = TRUE,
  sample = 1,
  static_limit = NULL,
  noisy_termination = TRUE
)
```

Arguments

g	An igraph object
force	A character string. This is the node attribute that contains the force the nodes exert on the network.
distance	A character string. The edge attribute that contains the original/horizontal distance between nodes.
edge_name	A character string. This is the edge attribute that contains the edge_name of the edges.
k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time interval used to iterate through the network dynamics.
mass	A numeric. This is the mass constant of the nodes in normalised networks this is set to 1.
max_iter	An integer. The maximum number of iterations before stopping. Larger networks usually need more iterations.
coef_drag	A numeric.
tol	A numeric. The tolerance factor for early stopping.
sparse	Logical. Whether or not the function should be run using sparse matrices. must match the actual matrix, this could prob be automated
two_node_solution	Logical. The Newton-Raphson algo is used to find the correct angle
sample	Integer. The dynamics will be stored only if the iteration number is a multiple of the sample. This can greatly reduce the size of the results file for large numbers of iterations. Must be a multiple of the max_iter
static_limit	Numeric. The maximum value the static force can reach before the algorithm terminates early. This prevents calculation in a diverging system. The value should be set to some multiple greater than one of the force in the system. If left blank the static limit is twice the system absolute mean force.
noisy_termination	Stop the process if the static force does not monotonically decrease.

Details

This is the basic SETS embeddings algorithm, it outputs all elements of the embeddings as well as convergence dynamics. It is a wrapper around the core SETS algorithm which requires data preparation and only produces node embeddings and network dynamics. There is little reason to use this function as `setse_auto` and `setse_bicomp` are faster and easier to use.

Value

A list containing 4 dataframes.

1. The network dynamics describing several key figures of the network during the convergence process, this includes the static_force.
2. The node embeddings. Includes all data on the nodes the forces exerted on them position and dynamics at simulation termination.

3. time taken. the amount of time taken per component, includes the number of edges and nodes.
4. The edge embeddings. Includes all data on the edges as well as the strain and tension values.

See Also

Other setse: setse_auto_hd(), setse_auto(), setse_bicomp(), setse_expanded()

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")
embeddings <- g %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                      force_var = "class") %>%
#embed the network using auto_setse
  setse(., force = "class_A")
```

---

setse_auto *SETSe embedding with automatic drag and timestep selection*

Description

Embeds/smooths a feature network using the SETSe algorithm automatically finding convergence parameters using a grid search.

Usage

```
setse_auto(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
  tstep = 0.02,
  mass = 1,
  max_iter = 1e+05,
  tol = 0.002,
  sparse = FALSE,
  hyper_iters = 100,
  hyper_tol = 0.01,
  hyper_max = 30000,
  drag_min = 0.01,
  drag_max = 100,
  tstep_change = 0.2,
  sample = 100,
  static_limit = NULL,
```

```
  verbose = FALSE,
  include_edges = TRUE,
  noisy_termination = TRUE
)
```

Arguments

g	An igraph object
force	A character string. This is the node attribute that contains the force the nodes exert on the network.
distance	A character string. The edge attribute that contains the original/horizontal distance between nodes.
edge_name	A character string. This is the edge attribute that contains the edge_name of the edges.
k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time interval used to iterate through the network dynamics.
mass	A numeric. This is the mass constant of the nodes in normalised networks this is set to 1.
max_iter	An integer. The maximum number of iterations before stopping. Larger networks usually need more iterations.
tol	A numeric. The tolerance factor for early stopping.
sparse	Logical. Whether or not the function should be run using sparse matrices. must match the actual matrix, this could prob be automated
hyper_iters	integer. The hyper parameter that determines the number of iterations allowed to find an acceptable convergence value.
hyper_tol	numeric. The convergence tolerance when trying to find the minimum value
hyper_max	integer. The maximum number of iterations that SETSe will go through whilst searching for the minimum.
drag_min	integer. A power of ten. The lowest drag value to be used in the search
drag_max	integer. A power of ten. if the drag exceeds this value the tstep is reduced
tstep_change	numeric. A value between 0 and 1 that determines how much the time step will be reduced by default value is 0.5
sample	Integer. The dynamics will be stored only if the iteration number is a multiple of the sample. This can greatly reduce the size of the results file for large numbers of iterations. Must be a multiple of the max_iter
static_limit	Numeric. The maximum value the static force can reach before the algorithm terminates early. This prevents calculation in a diverging system. The value should be set to some multiple greater than one of the force in the system. If left blank the static limit is the system absolute mean force.
verbose	Logical. This value sets whether messages generated during the process are suppressed or not.
include_edges	logical. An optional variable on whether to calculate the edge tension and strain. Default is TRUE. included for ease of integration into the bicomponent functions.
noisy_termination	Stop the process if the static force does not monotonically decrease.

Details

This is one of the most commonly used SETSe functions. It automatically selects the convergence time-step and drag values to ensure efficient convergence.

The noisy_termination parameter is used as in some cases the convergence process can get stuck in the noisy zone of SETSe space. To prevent this the process is stopped early if the static force does not monotonically decrease. On large networks this greatly speeds up the search for good parameter values. It increases the chance of successful convergence. More detail on auto-SETSe can be found in the paper "The spring bounces back" (Bourne 2020).

Value

A list containing 5 dataframes.

1. The network dynamics describing several key figures of the network during the convergence process, this includes the static_force
2. The node embeddings. Includes all data on the nodes the forces exerted on them position and dynamics at simulation termination
3. time taken. the amount of time taken per component, includes the edge and nodes of each component
4. The edge embeddings. Includes all data on the edges as well as the strain and tension values.
5. memory_df A dataframe recording the iteration history of the convergence of each component.

See Also

Other setse: `setse_auto_hd()`, `setse_bicomp()`, `setse_expanded()`, `setse()`

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")
embeddings <- g %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                      force_var = "class") %>%
#embed the network using auto_setse
  setse_auto(., force = "class_A")
```

`setse_auto_hd` *SETSe embedding with automatic drag and timestep selection for high-dimensional feature vectors*

Usage

```
setse_auto_hd(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
  tstep = 0.02,
  mass = 1,
  max_iter = 1e+05,
  tol = 0.002,
  sparse = FALSE,
  hyper_iters = 100,
  hyper_tol = 0.1,
  hyper_max = 30000,
  drag_min = 0.01,
  drag_max = 100,
  tstep_change = 0.2,
  sample = 100,
  static_limit = NULL,
  verbose = FALSE,
  include_edges = TRUE,
  noisy_termination = TRUE
)
```

Arguments

g	An igraph object
force	A character vector. These are the nodes attributes that contain the force the nodes exert on the network.
distance	A character string. The edge attribute that contains the original/horizontal distance between nodes.
edge_name	A character string. This is the edge attribute that contains the edge_name of the edges.
k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time interval used to iterate through the network dynamics.
mass	A numeric. This is the mass constant of the nodes in normalised networks this is set to 1.
max_iter	An integer. The maximum number of iterations before stopping. Larger networks usually need more iterations.
tol	A numeric. The tolerance factor for early stopping. Setting this value to be 0.1 \itemsparseLogical. Whether or not the function should be run using sparse matrices. must match the actual matrix, this could prob be automated \itemhyper_itersinteger. The hyper parameter that determines the number of iterations allowed to find an acceptable convergence value.

\itemhyper_tolnumeric. The convergence tolerance when trying to find the minimum value. When the ratio between the current static force and the previous static force is smaller than this value the search terminates. values between 0.1-0.3 seem to be often ok, too small and you waste time fine tuning the parameters instead of converging to big and you have poorly parametrised values.

\itemhyper_maxinteger. The maximum number of iterations that SETSe will go through whilst searching for the minimum.

\itemdrag_mininteger. A power of ten. The lowest drag value to be used in the search

\itemdrag_maxinteger. A power of ten. if the drag exceeds this value the tstep is reduced

\itemtstep_changenumeric. A value between 0 and 1 that determines how much the time step will be reduced by default value is 0.5

\itemsampleInteger. The dynamics will be stored only if the iteration number is a multiple of the sample. This can greatly reduce the size of the results file for large numbers of iterations. Must be a multiple of the max_iter

\itemstatic_limitNumeric. The maximum value the static force can reach before the algorithm terminates early. This prevents calculation in a diverging system. The value should be set to some multiple greater than one of the force in the system. If left blank the static limit is the system absolute mean force.

\itemverboseLogical. This value sets whether messages generated during the process are suppressed or not. This is useful for large networks which can take a long time to converge, but for smaller ones can be turned off.

\iteminclude_edgeslogical. An optional variable on whether to calculate the edge tension and strain. Default is TRUE. included for ease of integration into the bicomponent functions.

\itemnoisy_terminationStop the process if the static force does not monotonically decrease.

A list of four elements. A data frame with the height embeddings of the network, a data frame of the edge embeddings, the convergence dynamics dataframe for the network as well as the search history for convergence criteria of the network

Uses a grid search and a binary search to find appropriate convergence conditions.

This is one of the most commonly used SETSe functions. It automatically selects the convergence time-step and drag values to ensure efficient convergence.

The noisy_termination parameter is used as in some cases the convergence process can get stuck in the noisy zone of SETSe space. To prevent this the process is stopped early if the static force does not monotonically decrease. On large networks this greatly speeds up the search for good parameter values. It increases the chance of successful convergence. More detail on auto-SETSe can be found in the paper "The spring bounces back" (Bourne 2020).

g <- biconnected_network %>% prepare_edges(.) %>% #prepare the continuous features as normal prepare_continuous_force(., node_names = "name", force_var = "force") %>% #prepare the categorical features as normal prepare_categorical_force(., node_names = "name", force_var = "group")

#embed them using the high dimensional function two_dimensional_embeddings <- setse_auto_hd(g, force = c("group_A", "force"), k = "weight")

Other setse: `setse_auto()`, `setse_bicomp()`, `setse_expanded()`, `setse()`

setse

`setse_bicomp` *SETSe embedding on each bi-connected component using setse_auto*

Description

Embeds/smooths a feature network using the SETSe algorithm automatically finding convergence parameters using a grid search. In addition it breaks the network into bi-connected component solves each sub-component inidividually and re-assembles them back into a single network. This is the most reliable method to perform SETSe embeddings and can be substantially quicker on certain network topologies.

Usage

```
setse_bicomp(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
  tstep = 0.02,
  tol = 0.01,
  max_iter = 20000,
  mass = NULL,
  sparse = FALSE,
  sample = 100,
  static_limit = NULL,
  hyper_iters = 100,
  hyper_tol = 0.1,
  hyper_max = 30000,
  drag_min = 0.01,
  drag_max = 100,
  tstep_change = 0.2,
  verbose = FALSE,
  noisy_termination = TRUE
)
```

Arguments

`g`	An igraph object
`force`	A character string. This is the node attribute that contains the force the nodes exert on the network.
`distance`	A character string. The edge attribute that contains the original/horizontal distance between nodes.
`edge_name`	A character string. This is the edge attribute that contains the edge_name of the edges.

k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time interval used to iterate through the network dynamics.
tol	A numeric. The tolerance factor for early stopping.
max_iter	An integer. The maximum number of iterations before stopping. Larger networks usually need more iterations.
mass	A numeric. This is the mass constant of the nodes in normalised networks. Default is set to NULL and call mass_adjuster to set the mass for each biconnected component
sparse	Logical. Whether sparse matrices will be used. This becomes valuable for larger networks
sample	Integer. The dynamics will be stored only if the iteration number is a multiple of the sample. This can greatly reduce the size of the results file for large numbers of iterations. Must be a multiple of the max_iter
static_limit	Numeric. The maximum value the static force can reach before the algorithm terminates early. This prevents calculation in a diverging system. The value should be set to some multiple greater than one of the force in the system. If left blank the static limit is the system absolute mean force.
hyper_iters	integer. The hyper parameter that determines the number of iterations allowed to find an acceptable convergence value.
hyper_tol	numeric. The convergence tolerance when trying to find the minimum value
hyper_max	integer. The maximum number of iterations that SETSe will go through whilst searching for the minimum.
drag_min	integer. A power of ten. The lowest drag value to be used in the search
drag_max	integer. A power of ten. if the drag exceeds this value the tstep is reduced
tstep_change	numeric. A value between 0 and 1 that determines how much the time step will be reduced by default value is 0.5
verbose	Logical. This value sets whether messages generated during the process are suppressed or not.
noisy_termination	Stop the process if the static force does not monotonically decrease.

Details

Embedding the network by solving each bi-connected component then re-assembling can be faster for larger graphs, graphs with many nodes of degree 2, or networks with a low clustering coefficient. This is because although SETSe is very efficient the topology of larger graphs make them more difficult to converge. Large graph tend to be made of 1 very large biconnected component and many very small biconnected components. As the mass of the system is concentrated in the major biconnected component smaller ones can be knocked around by minor movements of the largest component. This can lead to long convergence times. By solving all biconnected components separately and then reassembling the block tree at the end, the system can be converged considerably faster.

Setting mass to the absolute system force divided by the total nodes, often leads to faster convergence. As such When mass is left to the default of NULL, the mean absolute force value is used.

Value

A list containing 5 dataframes.

1. The node embeddings. Includes all data on the nodes the forces exerted on them position and dynamics at simulation termination
2. The network dynamics describing several key figures of the network during the convergence process, this includes the static_force
3. memory_df A dataframe recording the iteration history of the convergence of each component.
4. Time taken. A data frame giving the time taken for the simulation as well as the number of nodes and edges. Node and edge data is given as this may differ from the total number of nodes and edges in the network depending on the method used for convergence. For example if setse_bicomp is used then some simulations may contain as little as two nodes and 1 edge
5. The edge embeddings. Includes all data on the edges as well as the strain and tension values.

See Also

Other setse: `setse_auto_hd()`, `setse_auto()`, `setse_expanded()`, `setse()`

Examples

```
set.seed(234) #set the random see for generating the network
g <- generate_peels_network(type = "E")
embeddings <- g %>%
prepare_edges(k = 500, distance = 1) %>%
#prepare the network for a binary embedding
prepare_categorical_force(., node_names = "name",
                      force_var = "class") %>%
#embed the network
  setse_bicomp(., force = "class_A")
```

---

setse_expanded *SETSe embedding showing full convergence history*

Description

This is a special case function of SETSe which keeps the history of all node movements during convergence0. It is useful for demonstrations, or parametrising difficult networks.

Usage

```
setse_expanded(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
```

```
  tstep = 0.02,
  mass = 1,
  max_iter = 20000,
  coef_drag = 1,
  tol = 1e-06,
  sparse = FALSE,
  verbose = TRUE,
  two_node_solution = TRUE
)
```

Arguments

g	An igraph object. The network
force	A character string
distance	A character string. The name of the graph attribute that contains the graph distance
edge_name	A character string. This is the edge attribute that contains the edge_name of the edges.
k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time in seconds that elapses between each iteration
mass	A numeric. The mass in kg of the nodes, this is arbitrary and commonly 1 is used.
max_iter	An integer. The maximum number of iterations before terminating the simulation
coef_drag	A numeric. A multiplier used to tune the damping. Generally no need to twiddle
tol	A numeric. Early termination. If the dynamics of the nodes fall below this value the algorithm will be classed as "converged" and the simulation terminates.
sparse	Logical. Whether or not the function should be run using sparse matrices. must match the actual matrix, this could prob be automated
verbose	Logical value. Whether the function should output messages or run quietly.
two_node_solution	Logical. The Newton-Raphson algo is used to find the correct angle

Value

A dataframe equivalent to the node_embeddings dataframe for the other SETSe methods. However, the dataframe includes a row for each node in each iteration of the simulation, as well as an additional column identifying the iteration number. This dataframe can be very large as it contains nxm rows where n is the number of nodes and m is the number of iterations in the simulation.

See Also

Other setse: setse_auto_hd(), setse_auto(), setse_bicomp(), setse()

Examples

```
g_prep <- biconnected_network%>%
prepare_edges(.) %>%
prepare_continuous_force(., node_names = "name", force_var = "force", k = NULL)

#the base configuration does not work
divergent_result <- setse_expanded(g_prep, k = "weight", tstep = 0.1)

#with a smaller timestep the algorithm converges
convergent_result <- setse_expanded(g_prep, k = "weight", tstep = 0.01)

## Not run:
library(ggplot2)
#plot the results for a given node
convergent_result %>%
 ggplot(aes(x = t, y = net_force, colour = node)) + geom_line()
#re-plot with divergent_result to see what it looks like

## End(Not run)
```

`setse_shift` *setse algorithm with automatic timestep adjustment*

Description

The basic setse function with added timestep adjustment. The time shift functionality automatically adjusts the timestep if the convergence process is noisy

Usage

```
setse_shift(
  g,
  force = "force",
  distance = "distance",
  edge_name = "edge_name",
  k = "k",
  tstep = 0.02,
  mass = 1,
  max_iter = 20000,
  coef_drag = 1,
  tol = 1e-06,
  sparse = FALSE,
  two_node_solution = TRUE,
  sample = 1,
  static_limit = NULL,
  tstep_change = 0.5
)
```

Arguments

g	An igraph object
force	A character string. This is the node attribute that contains the force the nodes exert on the network.
distance	A character string. The edge attribute that contains the original/horizontal distance between nodes.
edge_name	A character string. This is the edge attribute that contains the edge_name of the edges.
k	A character string. This is k for the moment don't change it.
tstep	A numeric. The time interval used to iterate through the network dynamics.
mass	A numeric. This is the mass constant of the nodes in normalised networks this is set to 1.
max_iter	An integer. The maximum number of iterations before stopping. Larger networks usually need more iterations.
coef_drag	A numeric.
tol	A numeric. The tolerance factor for early stopping.
sparse	Logical. Whether or not the function should be run using sparse matrices. must match the actual matrix, this could prob be automated
two_node_solution	Logical. The Newton-Raphson algo is used to find the correct angle
sample	Integer. The dynamics will be stored only if the iteration number is a multiple of the sample. This can greatly reduce the size of the results file for large numbers of iterations. Must be a multiple of the max_iter
static_limit	Numeric. The maximum value the static force can reach before the algorithm terminates early. This prevents calculation in a diverging system. The value should be set to some multiple greater than one of the force in the system. If left blank the static limit is twice the system absolute mean force.
tstep_change	a numeric scaler. A value between 0 and one, the fraction the new timestep will be relative to the previous one this can stop the momentum of the nodes forcing a divergence, but also can slow down the process. default is TRUE.

Details

This is the basic SETS embeddings algorithm, it outputs all elements of the embeddings as well as convergence dynamics. It is a wrapper around the core SETS algorithm which requires data preparation and only produces node embeddings and network dynamics. There is little reason to use this function as `setse_auto` and `setse_bicomp` are faster and easier to use.

Value

A list of three elements. A data frame with the height embeddings of the network, a data frame of the edge embeddings as well as the convergence dynamics dataframe for the network.

See Also

setse_auto setse_bicomp

Examples

```
## Not run:
biconnected_network %>%
prepare_continuous_force(., node_names = "name", force_var = "force") %>%
#embed the network using setse
setse_shift(., k = "weight", tstep = 0.000029)

## End(Not run)
```

Index