Package ‘RScelestial’

July 21, 2025

Type Package

Title Scelestial: Steiner Tree Based Single-Cell Lineage Tree Inference

Version 1.0.4

Date 2023-11-29

Maintainer Mohammad Hadi Foroughmand Araabi <`foroughmand@gmail.com`>

Description Scelestial infers a lineage tree from single-cell DNA mutation matrix. It generates a tree with approximately maximum parsimony through a Steiner tree approximation algorithm.

License GPL (>= 2)

Imports Rcpp (>= 1.0.1)

LinkingTo Rcpp

RoxygenNote 7.2.3

Suggests igraph, knitr, rmarkdown, stringr, seqinr, spelling

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

Language en-US

NeedsCompilation yes

Author Mohammad Hadi Foroughmand Araabi [aut, cre], Sama Goliaei [aut, ctb], Alice McHardy [ctb]

Repository CRAN

Date/Publication 2023-11-30 21:00:02 UTC

Contents

.scelestial 2
.synthesis 3
as.mutation.matrix 4
as.ten.state.matrix 5

as.ten.state.matrix.from.node.seq 5
distance.matrix.scelestial 6
distance.matrix.tree 7
distance.matrix.true.tree 8
Li 10
my.dfs 10
my.general.dfs 11
read.sequence.table 12
RScelestial 12
scelestial 13
synthesis 15
tree.plot 18

Index **19**

`.scelestial` *Internal function for running scelestial algorithm.*

Description

Internal function for running scelestial algorithm.

Usage

```
.scelestial(data, minK = 3L, maxK = 4L)
```

Arguments

`data`	The data
`minK`, `maxK`	Minimum and maximum number of vertices to be considered for k-restricted Steiner tree.

Value

The tree as well as missing value imputation

`.synthesis` *Internal function for generating synthetic single-cell data through simulation of tumor growth and evolution.*

Description

Internal function for generating synthetic single-cell data through simulation of tumor growth and evolution.

Usage

```
.synthesis(
  sample,
  site,
  evolutionSteps,
  mutationRate = 0.01,
  advantageIncreaseRatio = 1,
  advantageDecreaseRatio = 10,
  advantageKeepRatio = 100,
  advantageIncreaseStep = 0.01,
  advantageDecreaseStep = 0.01,
  mvRate = 0.5,
  fpRate = 0.2,
  fnRate = 0.1,
  seed = -1L
)
```

Arguments

`sample`	Number of samples
`site`	Number of sites
`evolutionSteps`	Number of non-root nodes in the evolutionary tree to be generated.
`mutationRate`	The rate of mutation on each evolutionary step in evolutionary tree synthesis.
`advantageIncreaseRatio`, `advantageDecreaseRatio`, `advantageKeepRatio`	A child node in the evolutionary tree is chosen for increase/decrease/keep its parent advantage with probabilities proportional to `advantage.increase.ratio/advantage.decrease.ra`
`advantageIncreaseStep`, `advantageDecreaseStep`	The amount of increasing or decreasing the advantage of a cell relative to its parent.
`mvRate`	Rate of missing value to be added to the resulting sequences.
`fpRate`, `fnRate`	Rate of false positive (0 -> 1) and false negative (1 -> 0) in the sequences.
`seed`	The seed for randomization.

Value

The function returns a list. The list consists of

- `sequence`: A data frame representing result of sequencing. The data frame has a row for each locus and a column for each sample.
- `true.sequence`: The actual sequence for the sample before adding errors and missing values.
- `true.clone`: A list that stores index of sampled cells for each node in the evolutionary tree.
- `true.tree`: The evolutionary tree that the samples are sampled from. It is a data frame with `src`, `dest`, and `len` columns representing source, destination and weight of edges of the tree, respectively.

`as.mutation.matrix` *Conversion of ten-state sequencing matrix to 0/1-mutation matrix.*

Description

Conversion of ten-state sequencing matrix to 0/1-mutation matrix.

Usage

```
as.mutation.matrix(seq)
```

Arguments

`seq`	A dataframe representing the ten-state sequencing matrix. Elements of the matrix are the from "X/Y" for X and Y being nucleotides or "./." for missing value. Rows represent loci and columns represent samples.

Value

A data frame with exactly the same size as the input `seq` matrix. The most abundant state in each loci (row) translated to 0, and the others are translated to 1. Missing values are translated to 3.

Examples

```
## A small 10-state matrix
seq = data.frame("C1" = c("C/C", "C/C"), "C2" = c("A/A", NA), "C3" = c("C/C", "A/A"))
## Convert it to mutation matrix
as.mutation.matrix(seq)
#   C1 C2 C3
# 1  0  1  0
# 2  1  3  0
```

`as.ten.state.matrix` *Conversion of 0/1 matrix to 10-state matrix*

Description

It converts 0 to A/A and 1 to C/C. 3 that represents missing values are converted to "./.".

Usage

```
as.ten.state.matrix(mut)
```

Arguments

`mut`	A dataframe representing the mutation matrix.

Value

A data frame with the exact size as `mut`, in which 0, 1 and 3 (or NAs) are replaced with "A/A", "C/C", and "./.", respectively.

Note

Note that following function does not provide inverse of as.mutation.matrix. It could be used to generate input for scelestial.

Examples

```
## A small 0/1/NA mutation matrix
mut = data.frame("C1" = c(0, 0), "C2" = c(0, 3), "C3" = c(1, 0))
## Convert it to 10-state matrix
as.ten.state.matrix(mut)
#    C1  C2  C3
# 1 A/A A/A C/C
# 2 A/A ./. A/A
```

`as.ten.state.matrix.from.node.seq` *Generates 10-state sequence matrix from name/10-char string matrix.*

Description

This function is used for conversion of results of internal scelestial result to 10-state sequence matrices.

Usage

```
as.ten.state.matrix.from.node.seq(n.seq)
```

Arguments

`n.seq` A two column data frame. First column is the name of a node and the second column is a string representation of the sequencing result. Each element of the sequencing result is from a 10-state representation in which each state represented as a character according to the following encoding:

One character representation	10-state representation
"A"	"A/A",
"T"	"T/T",
"C"	"C/C",
"G"	"G/G",
"K"	"A/C",
"L"	"A/G",
"M"	"C/T",
"N"	"C/G",
"O"	"T/G",
"P"	"T/A",
"X"	"./."

Value

A 10-state sequence data frame with samples as columns and loci as rows. Elements of `n.seq` are translated to their 10-state representations.

Examples

```
## A node sequence data frame
n.seq = data.frame("node" = c("C1", "C2"), "seq" = c("AKLTCXAAC", "AKKOCXAPC"))
## Convert it to ten state matrix
as.ten.state.matrix.from.node.seq(n.seq)
#     V1  V2  V3  V4  V5  V6  V7  V8  V9
# C1 A/A A/C A/G T/T C/C ./. A/A A/A C/C
# C2 A/A A/C A/C T/G C/C ./. A/A T/A C/C
```

`distance.matrix.scelestial` *Calculates distance matrix for result of scelestial*

Description

Calculates distance matrix for result of scelestial

Usage

```
distance.matrix.scelestial(SP, normalize = TRUE)
```

Arguments

`SP`	Output of scelestial function
`normalize`	If true, sum of all elements of resulting table is added up to one.

Value

The distance matrix

Examples

```
## Synthesise an evolution
S = synthesis(10, 5, 20, seed=7)
## Run Scelestial
SC = scelestial(as.ten.state.matrix(S$seqeunce))
## Calculate the distance matrix
distance.matrix.scelestial(SC)
#             C1         C10          C2          C3          C4
# C1  0.000000000 0.003512891 0.015222451 0.014051472 0.008196692
# C10 0.003512891 0.000000000 0.011709560 0.010538580 0.004683800
# C2  0.015222451 0.011709560 0.000000000 0.010538627 0.007025759
# C3  0.014051472 0.010538580 0.010538627 0.000000000 0.005854780
# C4  0.008196692 0.004683800 0.007025759 0.005854780 0.000000000
# C5  0.011709560 0.008196668 0.003512891 0.007025736 0.003512868
# C6  0.023419213 0.019906322 0.019906368 0.009367741 0.015222521
# C7  0.018735342 0.015222451 0.015222498 0.004683871 0.010538651
# C8  0.015222474 0.011709583 0.014051542 0.012880562 0.007025783
# C9  0.010538627 0.007025736 0.009367695 0.008196715 0.002341935
# C5          C6          C7          C8          C9
# C1  0.011709560 0.023419213 0.018735342 0.015222474 0.010538627
# C10 0.008196668 0.019906322 0.015222451 0.011709583 0.007025736
# C2  0.003512891 0.019906368 0.015222498 0.014051542 0.009367695
# C3  0.007025736 0.009367741 0.004683871 0.012880562 0.008196715
# C4  0.003512868 0.015222521 0.010538651 0.007025783 0.002341935
# C5  0.000000000 0.016393477 0.011709606 0.010538651 0.005854803
# C6  0.016393477 0.000000000 0.004683871 0.022248304 0.017564457
# C7  0.011709606 0.004683871 0.000000000 0.017564433 0.012880586
# C8  0.010538651 0.022248304 0.017564433 0.000000000 0.004683847
# C9  0.005854803 0.017564457 0.012880586 0.004683847 0.000000000
```

`distance.matrix.tree` *Calculates distance matrix for a nodes on a tree.*

Description

It is used for internal purposes.

Usage

```
distance.matrix.tree(graph, cell.names, tree.nodes, normalize = TRUE)
```

Arguments

`graph`	The tree
`cell.names`	Name of the cells to be the row and column name of the resulting matrix
`tree.nodes`	For each cell.names a tree node is stored in tree.nodes.
`normalize`	If TRUE the resulting matrix is normalized.

Value

A matrix with equal number of rows and columns, a row/column for each cell. Elements of matrix represent distance between cells on the `graph`.

Examples

```
## Synthesise an evolution
S = synthesis(10, 5, 20, seed=7)
## Run Scelestial
SC = scelestial(as.ten.state.matrix(S$seqeunce))
## Calculate the distance matrix
vertices <- rownames(SC$input);
distance.matrix.tree(SC$tree, vertices, vertices, normalize = TRUE)
#              C1         C10          C2          C3          C4
# C1  0.000000000 0.003512891 0.015222451 0.014051472 0.008196692
# C10 0.003512891 0.000000000 0.011709560 0.010538580 0.004683800
# C2  0.015222451 0.011709560 0.000000000 0.010538627 0.007025759
# C3  0.014051472 0.010538580 0.010538627 0.000000000 0.005854780
# C4  0.008196692 0.004683800 0.007025759 0.005854780 0.000000000
# C5  0.011709560 0.008196668 0.003512891 0.007025736 0.003512868
# C6  0.023419213 0.019906322 0.019906368 0.009367741 0.015222521
# C7  0.018735342 0.015222451 0.015222498 0.004683871 0.010538651
# C8  0.015222474 0.011709583 0.014051542 0.012880562 0.007025783
# C9  0.010538627 0.007025736 0.009367695 0.008196715 0.002341935
# C5          C6          C7          C8          C9
# C1  0.011709560 0.023419213 0.018735342 0.015222474 0.010538627
# C10 0.008196668 0.019906322 0.015222451 0.011709583 0.007025736
# C2  0.003512891 0.019906368 0.015222498 0.014051542 0.009367695
# C3  0.007025736 0.009367741 0.004683871 0.012880562 0.008196715
# C4  0.003512868 0.015222521 0.010538651 0.007025783 0.002341935
# C5  0.000000000 0.016393477 0.011709606 0.010538651 0.005854803
# C6  0.016393477 0.000000000 0.004683871 0.022248304 0.017564457
# C7  0.011709606 0.004683871 0.000000000 0.017564433 0.012880586
# C8  0.010538651 0.022248304 0.017564433 0.000000000 0.004683847
# C9  0.005854803 0.017564457 0.012880586 0.004683847 0.000000000
```

`distance.matrix.true.tree`
Calculates distance matrix for a synthetized data

Description

Calculates distance matrix for a synthetized data

Usage

```
distance.matrix.true.tree(D, normalize = TRUE)
```

Arguments

`D`	Output of synthesis function
`normalize`	If true, sum of all elements of resulting table is added up to one.

Value

The distance matrix of the true tree.

Examples

```
## Synthesise an evolution
S = synthesis(10, 5, 20, seed=7)
## Calculating the distance matrix of the true tree.
distance.matrix.true.tree(S)
#             C3          C6          C4          C2          C7
# C3  0.000000000 0.004587156 0.006880734 0.009174312 0.013761468
# C6  0.004587156 0.000000000 0.002293578 0.009174312 0.013761468
# C4  0.006880734 0.002293578 0.000000000 0.011467890 0.016055046
# C2  0.009174312 0.009174312 0.011467890 0.000000000 0.004587156
# C7  0.013761468 0.013761468 0.016055046 0.004587156 0.000000000
# C10 0.006880734 0.006880734 0.009174312 0.011467890 0.016055046
# C8  0.006880734 0.011467890 0.013761468 0.016055046 0.020642202
# C9  0.006880734 0.011467890 0.013761468 0.016055046 0.020642202
# C1  0.011467890 0.011467890 0.013761468 0.002293578 0.006880734
# C5  0.011467890 0.011467890 0.013761468 0.002293578 0.006880734
# C10         C8          C9          C1          C5
# C3  0.006880734 0.006880734 0.006880734 0.011467890 0.011467890
# C6  0.006880734 0.011467890 0.011467890 0.011467890 0.011467890
# C4  0.009174312 0.013761468 0.013761468 0.013761468 0.013761468
# C2  0.011467890 0.016055046 0.016055046 0.002293578 0.002293578
# C7  0.016055046 0.020642202 0.020642202 0.006880734 0.006880734
# C10 0.000000000 0.013761468 0.013761468 0.013761468 0.013761468
# C8  0.013761468 0.000000000 0.000000000 0.018348624 0.018348624
# C9  0.013761468 0.000000000 0.000000000 0.018348624 0.018348624
# C1  0.013761468 0.018348624 0.018348624 0.000000000 0.000000000
# C5  0.013761468 0.018348624 0.018348624 0.000000000 0.000000000
```

`Li` — *Bladder invasive single cell tumor dataset*

Description

Bladder invasive single cell tumor dataset

Usage

```
data(Li)
```

Format

Each column represent a cell and each row represent a locus. "./." represent the missing value, "A/A" the normal state and "C/C" the mutated state.

Source

QTL Archive

References

Gigascience. 2012 Aug 14;1(1):12. doi: 10.1186/2047-217X-1-12. (PubMed)

Examples

```
data(Li)
```

`my.dfs` — *Runs DFS on tree and calculates parent of each node as well as depth and upper-depth of nodes.*

Description

It is used for internal purposes.

Usage

```
my.dfs(graph, root = NULL)
```

Arguments

`graph`	The tree
`root`	The starting node of DFS.

Value

a list with `father` representing the parent node, and `balance.depth` representing the distance between the node and the farthest node to it, as the elements.

`my.general.dfs` *Running depth first search on a tree and calling functions on entrance/exit events*

Description

It is used for internal purposes.

Usage

```
my.general.dfs(
  nei,
  v,
  f,
  extra,
  in.call,
  mid.call.before,
  mid.call.after,
  out.call
)
```

Arguments

`nei`	Neighbor list for each vertex
`v`	Starting node
`f`	Parent node
`extra`	the shared object for the whole DFS
`in.call`	First function to call
`mid.call.before`	Function to call before calling child DFS
`mid.call.after`	Function to call after calling child DFS
`out.call`	Last function to call

Value

the `extra` parameter modified with `in.call`, `mid.call.before`, `mid.call.after`, and `out.call` functions

`read.sequence.table` *Read mutation table*

Description

A simple read of a sequencing file.

Usage

```
read.sequence.table(file.name)
```

Arguments

`file.name`	Name of the file to be loaded

Value

A table representing the content of the file. First column of the file represents the row names.

Examples

```
# An example input without header could be like following:
# 1 C/C A/A A/A A/A
# 2 ./. A/A C/C C/C
# 3 C/C A/A C/C ./.
# 4 A/A ./. ./. ./.
# 5 ./. A/A A/A A/A
#
# For this file you can run
read.sequence.table(system.file("extdata/sample1.txt", package="RScelestial"))
```

`RScelestial` *RScelestial: An R wrapper for scelestial algorithm for single-cell lineage tree reconstruction through an approximation algorithm based on Steiner tree problem*

Description

This package provides a wrapper for the scelestial which is implemented in C++. The package contains function `scelestial` for running the algorithm and `synthesis` for tumor simulation for providing synthetic data.

`scelestial` *Infer the single-cell phylogenetic tree*

Description

Performs the Scelestial algorithm and calculates the phylogenetic tree reconstruction based on an approximation algorithm for Steiner tree problem.

Usage

```
scelestial(
  seq,
  mink = 3,
  maxk = 3,
  root.assign.method = c("none", "balance", "fix"),
  root = NULL,
  return.graph = FALSE
)
```

Arguments

`seq`	The sequence matrix. Rows represent loci and columns represent samples. Elements of the matrix represent 10-state genome sequencing results, or missing values. I.e each element is in the format "X/Y" where X and Y are from the set {A, T, C, G}. There is a special case "./." that represents the missing value.
`mink`	The minimum k used in the calculation of k-restricted Steiner trees. It is supposed to be 3.
`maxk`	The maximum k used in the calculation of k-restricted Steiner trees. When maxk=3, the approximation algorithm produces an 11/6-approximation result. Increasing k increases the running time as well as the approximation ratio of the algorithm. maxk should be not less than mink.
`root.assign.method`, `root`	root.assign.method is the method for choosing the root. • "none" for undirected tree, • "fix" for a tree with `root` as its root. • "balance" to let the root to be chosen to produce the most balanced tree.
`return.graph`	If TRUE, the actual graph through igraph library is generated and produced.

Value

Returns a list containing following elements:

- `tree`: A data frame representing edges of the tree. `tree$src` is the source of the edge, `tree$dest` represents the destination of the edge, and `tree$len` represents its weight (evolutionary distance).
- `input`: input sequences.

- `sequence`: inferred or imputed sequences for the tree nodes. If the node is already in the input, sequence represents its missing value imputation, in the case of presence of missing values, and if the node is not an input node, the sequence represents inferred sequence for the tree node.
- `graph`: graph. If the return.graph is TRUE, there is an element G that represents the graph from the igraph library.

Examples

```
## simulates tumor evolution
S = synthesis(10, 10, 2, seed=7)
## convert to 10-state matrix
seq = as.ten.state.matrix(S$seqeunce)
## runs the scelestial to generate 4-restricted Steiner trees. It represents the tree and graph
SP = scelestial(seq, mink=3, maxk=4, return.graph = TRUE)
SP
## Expected output:
# $input
#    node   sequence
# 1     0 AAXACAAXXA
# 2     1 AXXXAXAAXA
# 3     2 AXAXCAXXAX
# 4     3 AXCCCAXAAX
# 5     4 AXCXAXXCAX
# 6     5 XXCAXXXXXX
# 7     6 XACXACAAAC
# 8     7 AXAXXAXAXA
# 9     8 AXAAXXAXXX
# 10    9 AAXXXXCXCX
#
# $sequence
#    node   sequence
# 1     0 AAAACAAACA
# 2     1 AACAAAAAAA
# 3     2 AAAACAAAAA
# 4     3 AACCCAAAAA
# 5     4 AACAACACAC
# 6     5 AACAACAAAC
# 7     6 AACAACAAAC
# 8     7 AAAACAAACA
# 9     8 AAAACAAACA
# 10    9 AAAACACACA
# 11   10 AAAACAAACA
# 12   16 AACAAAAAAA
# 13   18 AACACAAAAA
#
# $tree
#    src dest     len
# 1    9   10 4.00006
# 2    8   10 3.00006
# 3    7   10 2.50005
# 4    0   10 1.50003
# 5    6   16 3.00002
# 6    1   16 2.50005
# 7    3   18 2.50003
# 8    0   18 1.50003
# 9   16   18 1.00000
# 10   0    2 3.50008
# 11   4    6 4.00007
# 12   5    6 4.50010
#
# $graph
# IGRAPH 6ba60f3 DNW- 13 12 --
# + attr: name (v/c), weight (e/n)
# + edges from 6ba60f3 (vertex names):
#  [1] 9 ->10 8 ->10 7 ->10 0 ->10 6 ->16 1 ->16 3 ->18 0 ->18 16->18 0 ->2
# [11] 4 ->6  5 ->6
#
```

`synthesis` *Synthesize single-cell data through tumor simulation*

Description

This function simulates a evolution in a tumor through two phases: 1) simulation of evolution, 2) sampling.

Usage

```
synthesis(
  sample,
  site,
  evolution.step,
  mutation.rate = 1,
  advantage.increase.ratio = 1,
  advantage.decrease.ratio = 10,
  advantage.keep.ratio = 100,
  advantage.increase.step = 0.01,
  advantage.decrease.step = 0.01,
  mv.rate = 0.5,
  fp.rate = 0.2,
  fn.rate = 0.1,
  seed = -1
)
```

Arguments

`sample`	Number of samples.
`site`	number of sites (loci)

`evolution.step`	Number of evolutionary steps in the process of production of the evolutionary tree.
`mutation.rate`	The rate of mutation on each evolutionary step in evolutionary tree synthesis.
`advantage.increase.ratio`, `advantage.decrease.ratio`, `advantage.keep.ratio`	A child node in the evolutionary tree is chosen for increase/decrease/keep its parent advantage with probabilities proportional to `advantage.increase.ratio`/`advantage.decrease.ra`
`advantage.increase.step`, `advantage.decrease.step`	The amount of increasing or decreasing the advantage of a cell relative to its parent.
`mv.rate`	Rate of missing value to be added to the resulting sequences.
`fp.rate`, `fn.rate`	Rate of false positive (0 -> 1) and false negative (1 -> 0) in the sequences.
`seed`	The seed for randomization.

Details

The simulation of evolution starts with a single cell. Then for `evolution.step` steps, on each step a cell is selected for duplication. A new cell as its child is added to the evolutionary tree. To each node in the evolutionary tree an advantage is assigned representing its relative advantage in replication and in being sampled. Advantage of a node is calculated by increasing (decreasing) its parents advantage by `advantage.increase.step` (`advantage.decrease.step`) with probability proportional to `advantage.increase.ratio` (`advantage.decrease.ratio`). With a probability proportional to `advantage.keep.ratio` the advantage of a node is equal to its parent's advantage.

Sequences for each node is build based on its parent's sequence by adding some mutations. Mutations are added for each locus independently with rate `mutation.rate`.

In the sampling phase, `sample` cells are selected from the evolutionary tree nodes. Result of the sequencing process for a cell is determined by the sequence of the node in the evolutionary tree with addition of some random errors. Errors are result of applying some false positives with rate `fp.rate`, applying some false negatives with rate `fn.rate`, and adding some missing values with rate `mv.rate`.

Value

The function returns a list. The list consists of

- `sequence`: A data frame representing result of sequencing. The data frame has a row for each locus and a column for each sample.
- `true.sequence`: The actual sequence for the sample before adding errors and missing values.
- `true.clone`: A list that stores index of sampled cells for each node in the evolutionary tree.
- `true.tree`: The evolutionary tree that the samples are sampled from. It is a data frame with `src`, `dest`, and `len` columns representing source, destination and weight of edges of the tree, respectively.

Examples

```
## generating a data set with 10 samples and 5 loci through simulation of
## 20-step evolution.
synthesis(10, 5, 20, seed=7)
## The result is
# $seqeunce
#     C1 C2 C3 C4 C5
# L1   1  1  1  1  1
# L2   3  1  3  3  0
# L3   3  1  3  3  1
# L4   3  0  1  0  0
# L5   1  3  0  3  3
# L6   3  1  3  1  0
# L7   3  3  1  0  3
# L8   3  1  1  3  3
# L9   3  3  1  3  1
# L10  0  3  0  3  0
#
# $true.sequence
#     C1 C2 C3 C4 C5
# L1   0  1  1  1  1
# L2   0  1  0  0  1
# L3   0  1  0  0  1
# L4   0  1  1  1  1
# L5   1  1  0  1  0
# L6   0  1  0  1  0
# L7   0  1  0  0  1
# L8   0  1  1  1  1
# L9   0  1  1  1  1
# L10  0  0  0  0  0
#
# $true.clone
# $true.clone[[1]]
# [1] 4
#
# $true.clone[[2]]
# [1] 1
#
# $true.clone[[3]]
# [1] 6
#
# $true.clone[[4]]
# [1] 10
#
# $true.clone[[5]]
# [1] 2
#
# $true.clone[[6]]
# [1] 3
#
# $true.clone[[7]]
# [1] 8 9
```

```
#
# $true.clone[[8]]
# [1] 7
#
# $true.clone[[9]]
# [1] 5
#
#
# $true.tree
#   src dest len
# 1   1    5   3
# 2   5    7   1
# 3   5   10   2
# 4   1   11   3
# 5   1   12   2
# 6   1   13   3
# 7   7   14   2
# 8  12   19   1
# 9  10   20   1
#
```

`tree.plot` *Plotting the tree*

Description

Plotting the igraph tree created by scelestial.

Usage

```
tree.plot(graph, ...)
```

Arguments

`graph`	Output of scelestial or the G element of the scelestial output.
`...`	Parameters passing to the plot function

Index

∗ **datasets**
    Li, 10
.scelestial, 2
.synthesis, 3

as.mutation.matrix, 4
as.ten.state.matrix, 5
as.ten.state.matrix.from.node.seq, 5

distance.matrix.scelestial, 6
distance.matrix.tree, 7
distance.matrix.true.tree, 8

Li, 10

my.dfs, 10
my.general.dfs, 11

read.sequence.table, 12
RScelestial, 12

scelestial, 13
synthesis, 15

tree.plot, 18