

Package ‘leidenAlg’

April 16, 2026

Type Package

Title Implements the Leiden Algorithm via an R Interface

Version 1.1.7

Description

An R interface to the Leiden algorithm, an iterative community detection algorithm on networks. The algorithm is designed to converge to a partition in which all subsets of all communities are locally optimally assigned, yielding communities guaranteed to be connected. The implementation proves to be fast, scales well, and can be run on graphs of millions of nodes (as long as they can fit in memory). The original implementation was constructed as a python interface `leidenalg` found here: <https://github.com/vtraag/leidenalg>. The algorithm was originally described in Traag, V.A., Waltman, L. & van Eck, N.J. "From Louvain to Leiden: guaranteeing well-connected communities". Sci Rep 9, 5233 (2019) [doi:10.1038/s41598-019-41695-z](https://doi.org/10.1038/s41598-019-41695-z).

License GPL-3

Copyright See the file COPYRIGHTS for various leidenAlg copyright details

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), Matrix

Imports graphics, grDevices, igraph, methods, parallel, Rcpp (>= 1.0.5), sccore, stats

Suggests pbapply, testthat (>= 3.1.0)

LinkingTo Rcpp, RcppArmadillo, RcppEigen

SystemRequirements GNU make (optional), libxml2 (optional), glpk (>= 4.57, optional)

RoxygenNote 7.2.3

URL <https://github.com/kharchenkolab/leidenAlg>

BugReports <https://github.com/kharchenkolab/leidenAlg/issues>

NeedsCompilation yes

Author Peter Kharchenko [aut],
 Viktor Petukhov [aut],
 Yichen Wang [aut],
 V.A. Traag [ctb],
 Gábor Csárdi [ctb],
 Tamás Nepusz [ctb],
 Minh Van Nguyen [ctb],
 Evan Biederstedt [cre, aut]

Maintainer Evan Biederstedt <evan.biederstedt@gmail.com>

Repository CRAN

Date/Publication 2026-04-16 05:10:02 UTC

Contents

| | |
|-----------------------------------------|---|
| as.dendrogram.fakeCommunities | 2 |
| exampleGraph | 3 |
| find_partition | 3 |
| find_partition_rcpp | 4 |
| find_partition_with_rep | 5 |
| find_partition_with_rep_rcpp | 6 |
| leiden.community | 7 |
| membership.fakeCommunities | 8 |
| rleiden.community | 9 |

Index **10**

as.dendrogram.fakeCommunities

Returns pre-calculated dendrogram

Description

Returns pre-calculated dendrogram

Usage

```
## S3 method for class 'fakeCommunities'
as.dendrogram(object, ...)
```

Arguments

| | |
|--------|--------------------------------|
| object | fakeCommunities object |
| ... | further parameters for generic |

Value

dendrogram

Examples

```
rLeidenComm = suppressWarnings(rleiden.community(exampleGraph, n.cores=1))
as.dendrogram.fakeCommunities(rLeidenComm)
```

exampleGraph

Conos graph

Description

Conos graph

Usage

```
exampleGraph
```

Format

An object of class igraph of length 100.

find_partition

Finds the optimal partition using the Leiden algorithm

Description

Finds the optimal partition using the Leiden algorithm

Usage

```
find_partition(graph, edge_weights, resolution = 1, niter = 2)
```

Arguments

| | |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| graph | The igraph graph to define the partition on |
| edge_weights | Vector of edge weights. In weighted graphs, a real number is assigned to each (directed or undirected) edge. For an unweighted graph, this is set to 1. Refer to igraph, weighted graphs. |
| resolution | Numeric scalar, resolution parameter controlling communities detected (default=1.0) Higher resolutions lead to more communities, while lower resolutions lead to fewer communities. |
| niter | Number of iterations that the algorithm should be run for (default=2) |

Value

A vector of membership values

Examples

```
library(igraph)
library(leidenAlg)

g <- make_star(10)
E(g)$weight <- seq(ecount(g))
find_partition(g, E(g)$weight)
```

find_partition_rcpp *Refer to the R function find_partition() For notes of the graph object, refer to <https://igraph.org/c/doc/igraph-Basic.html>*

Description

Refer to the R function find_partition() For notes of the graph object, refer to <https://igraph.org/c/doc/igraph-Basic.html>

Usage

```
find_partition_rcpp(
  edgelist,
  edgelist_length,
  num_vertices,
  direction,
  edge_weights,
  resolution = 1,
  niter = 2L
)
```

Arguments

| | |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| edgelist | The graph edge list |
| edgelist_length | integer The length of the graph edge list |
| num_vertices | integer The number of vertices in the graph |
| direction | boolean Whether the graph is directed or undirected |
| edge_weights | Vector of edge weights. In weighted graphs, a real number is assigned to each (directed or undirected) edge. For an unweighted graph, this is set to 1. Refer to igraph, weighted graphs. |
| resolution | Numeric scalar, resolution parameter controlling communities detected (default=1.0) Higher resolutions lead to more communities, while lower resolutions lead to fewer communities. |
| niter | Number of iterations that the algorithm should be run for (default=2) |

Value

A vector of membership values

Examples

```
library(igraph)
edgelist <- as.vector(t(igraph::as_edgelist(exampleGraph, names=FALSE))) - 1
edgelist_length <- length(edgelist)
num_vertices <- length(igraph::V(exampleGraph)) - 1
direction <- igraph::is_weighted(exampleGraph)
find_partition_rcpp(edgelist, edgelist_length, num_vertices, direction, E(exampleGraph)$weight)
```

find_partition_with_rep

Finds the optimal partition using the Leiden algorithm with replicate starts

Description

This function performs Leiden algorithm `nrep` times and returns the result from the run with the maximum quality.

Since Leiden algorithm has stochastic process, repeating stochastically may improve the result. However, users should be aware of whether there is indeed a community structure with exploration, rather than blindly trusting the returned result that comes with the highest quality value.

The random number generator (RNG) is not re-seeded at each new start of community detection, in order to keep the independence of each replicate. To get reproducible result, users can run `set.seed()` before calling these functions.

`find_partition` only performs the community detection once and the reproducibility can also be ensured with `set.seed()`.

Usage

```
find_partition_with_rep(
  graph,
  edge_weights,
  resolution = 1,
  niter = 2,
  nrep = 10
)
```

Arguments

| | |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>graph</code> | The igraph graph to define the partition on |
| <code>edge_weights</code> | Vector of edge weights. In weighted graphs, a real number is assigned to each (directed or undirected) edge. For an unweighted graph, this is set to 1. Refer to igraph, weighted graphs. |

| | |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| resolution | Numeric scalar, resolution parameter controlling communities detected (default=1.0) Higher resolutions lead to more communities, while lower resolutions lead to fewer communities. |
| niter | Number of iterations that the algorithm should be run for (default=2) |
| nrep | Number of replicate starts with random number being updated. (default=10) The result with the best quality will be returned. |

Value

A vector of membership values

Examples

```
library(igraph)

# To run 10 replicates and get the partitioning with the highest quality
membership <- find_partition_with_rep(exampleGraph, E(exampleGraph)$weight, nrep = 10)

# To get reproducible result for every function call, do `set.seed()` right before calling
set.seed(233)
res1 <- find_partition_with_rep(exampleGraph, E(exampleGraph)$weight, resolution = 2)
# Here, no seed was set...
res2 <- find_partition_with_rep(exampleGraph, E(exampleGraph)$weight, resolution = 2)
set.seed(233)
res3 <- find_partition_with_rep(exampleGraph, E(exampleGraph)$weight, resolution = 2)
identical(res1, res2) # FALSE (usually), as no seed as set
identical(res1, res3) # TRUE (always), as set.seed() was used directly before the function call
```

```
find_partition_with_rep_rcpp
```

Finds the optimal partition using the Leiden algorithm

Description

Finds the optimal partition using the Leiden algorithm

Usage

```
find_partition_with_rep_rcpp(
  edgelist,
  edgelist_length,
  num_vertices,
  direction,
  edge_weights,
  resolution = 1,
  niter = 2L,
  nrep = 1L
)
```

Arguments

| | |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| edgelist | The graph edge list |
| edgelist_length | integer The length of the graph edge list |
| num_vertices | integer The number of vertices in the graph |
| direction | boolean Whether the graph is directed or undirected |
| edge_weights | Vector of edge weights. In weighted graphs, a real number is assigned to each (directed or undirected) edge. For an unweighted graph, this is set to 1. Refer to igraph, weighted graphs. |
| resolution | Numeric scalar, resolution parameter controlling communities detected (default=1.0) Higher resolutions lead to more communities, while lower resolutions lead to fewer communities. |
| niter | Number of iterations that the algorithm should be run for (default=2) |
| nrep | Number of replicate starts with random number being updated. (default=10) The result with the best quality will be returned. |

Details

For notes of the graph object, refer to <https://igraph.org/c/doc/igraph-Basic.html>

Examples

```
library(igraph)

edgelist <- as.vector(t(igraph::as_edgelist(exampleGraph, names=FALSE))) - 1
edgelist_len <- length(edgelist) ## The length of the graph edge list
n_vertices <- length(igraph::V(exampleGraph)) - 1 ## The number of vertices in the graph
direct <- igraph::is_weighted(exampleGraph) ## Whether the graph is directed or undirected
edge_weights <- E(exampleGraph)$weight
find_partition_with_rep_rcpp(edgelist, edgelist_len, n_vertices, direct, edge_weights, nrep = 10)
```

| | |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| leiden.community | <i>Leiden algorithm community detection Detects communities using Leiden algorithm (implementation copied from https://github.com/vtraag/leidenalg)</i> |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Description

Leiden algorithm community detection Detects communities using Leiden algorithm (implementation copied from <https://github.com/vtraag/leidenalg>)

Usage

```
leiden.community(graph, resolution = 1, n.iterations = 2)
```

Arguments

graph graph on which communities should be detected
resolution resolution parameter (default=1.0) - higher numbers lead to more communities
n.iterations number of iterations that the algorithm should be run for (default=2)

Value

a fakeCommunities object that returns membership and dendrogram

Examples

```
leiden.community(exampleGraph)
```

```
membership.fakeCommunities
```

Returns pre-calculated membership factor

Description

Returns pre-calculated membership factor

Usage

```
## S3 method for class 'fakeCommunities'  
membership(communities, ...)
```

Arguments

communities fakeCommunities object
... further parameters for generic

Value

membership factor

Examples

```
leidenComm = leiden.community(exampleGraph)  
membership.fakeCommunities(leidenComm)
```

| | |
|-------------------|-------------------------------------------------------------------------------------------------------|
| rleiden.community | <i>Recursive leiden communities Constructs an n-step recursive clustering, using leiden.community</i> |
|-------------------|-------------------------------------------------------------------------------------------------------|

Description

Recursive leiden communities Constructs an n-step recursive clustering, using leiden.community

Usage

```
rleiden.community(
  graph,
  max.depth = 2,
  n.cores = parallel::detectCores(logical = FALSE),
  min.community.size = 10,
  verbose = FALSE,
  resolution = 1,
  cur.depth = 1,
  hierarchical = TRUE,
  ...
)
```

Arguments

| | |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| graph | graph |
| max.depth | Recursive depth (default=2) |
| n.cores | integer Number of cores to use (default = parallel::detectCores(logical=FALSE)). If logical=FALSE, uses the number of physical CPUs/cores. If logical=TRUE, uses the logical number of CPUS/cores. See parallel::detectCores() |
| min.community.size | integer Minimal community size parameter for the walktrap communities—Communities smaller than that will be merged (default=10) |
| verbose | boolean Whether to output progress messages (default=FALSE) |
| resolution | resolution parameter passed to leiden.community (either a single value, or a value equivalent to max.depth) (default=1) |
| cur.depth | integer Current depth of clustering (default=1) |
| hierarchical | boolean If TRUE, calculate hierarchy on the multilevel clusters (default=TRUE) |
| ... | passed to leiden.community |

Value

a fakeCommunities object that returns membership and dendrogram

Examples

```
rleiden.community(exampleGraph, n.cores=1)
```

Index

* datasets

- exampleGraph, 3
- as.dendrogram.fakeCommunities, 2
- exampleGraph, 3
- find_partition, 3, 5
- find_partition_rcpp, 4
- find_partition_with_rep, 5
- find_partition_with_rep_rcpp, 6
- leiden.community, 7
- membership.fakeCommunities, 8
- rleiden.community, 9