

# Package ‘leidenAlg’

November 19, 2021

**Type** Package

**Title** Implements the Leiden Algorithm via an R Interface

**Version** 1.0.0

## 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, igraph

**Imports** graphics, grDevices, Matrix.utils, parallel, Rcpp (>= 1.0.5), sccore, stats

**Suggests** pbapply, testthat

**LinkingTo** Rcpp, RcppArmadillo, RcppEigen

**SystemRequirements** GNU make

**RoxygenNote** 7.1.2

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

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

**NeedsCompilation** yes

**Author** Peter Kharchenko [aut],  
 Viktor Petukhov [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** 2021-11-19 21:40:02 UTC

## R topics documented:

|   |          |
|---|----------|
| as.dendrogram.fakeCommunities . . . . . | 2        |
| exampleGraph . . . . .                  | 3        |
| find_partition . . . . .                | 3        |
| leiden.community . . . . .              | 4        |
| membership.fakeCommunities . . . . .    | 5        |
| rliden.community . . . . .              | 5        |
| <b>Index</b>                            | <b>7</b> |

---

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 | <i>Conos graph</i> |
|--------------|--------------------|

---

**Description**

Conos graph

**Usage**

```
exampleGraph
```

**Format**

An object of class igraph of length 10.

---

|                |   |
|----------------|---|
| find_partition | <i>Finds the optimal partition using the Leiden algorithm</i> |
|----------------|---|

---

**Description**

Finds the optimal partition using the Leiden algorithm

**Usage**

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

**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. Refer to igraph, weighted graphs.                              |
| resolution   | Integer 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)
```

---

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

---

## Description

Leiden algorithm community detection Detect 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(object, ...)
```

**Arguments**

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

**Value**

membership factor

**Examples**

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

---

```
rleiden.community
```

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

---

**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

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

### 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](#)
- `leiden.community`, [4](#)
- `membership.fakeCommunities`, [5](#)
- `rleiden.community`, [5](#)