

Package: clusterLibreville (via r-universe)

October 21, 2026

Title Parallel and Deterministic Leiden Community Detection

Version 0.99.0

Description A parallel and deterministic implementation of the Leiden community detection algorithm (Traag, Waltman and van Eck 2019, <[doi:10.1038/s41598-019-41695-z](https://doi.org/10.1038/s41598-019-41695-z)>) for large sparse graphs such as single-cell kNN and SNN graphs, implemented in Rust. The same seed gives the same labels at any number of threads. The parallel design follows GVE-Leiden (Sahu et al. 2024, <[doi:10.1145/3673038.3673146](https://doi.org/10.1145/3673038.3673146)>). Works on sparse matrices, Seurat graphs and igraph graphs, and as a bluster clustering function.

License MIT + file LICENSE

URL <https://github.com/btraven00/clust-libreville-rs>

BugReports <https://github.com/btraven00/clust-libreville-rs/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.85.0, xz

biocViews Clustering, SingleCell, GraphAndNetwork, Network, Software

Depends R (>= 4.6.0)

Imports bluster, igraph, Matrix, methods, SingleCellExperiment, stats

Suggests aricode, BiocStyle, ggplot2, knitr, patchwork, rmarkdown, scater, scan, scRNAseq, scuttle, Seurat, sparseMatrixStats, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libglpk-dev libxml2-dev xz-utils zlib1g-dev libclang-dev

Repository <https://btraven00.r-universe.dev>

Date/Publication 2026-10-20 23:16:35 UTC

```
RemoteUrl https://github.com/btraven00/clust-libreville-rs
RemoteRef r-pkg
RemoteSha fbb535383670a743d0d355a6a80b38099f57fa5e
```

Contents

assess	2
champ	3
cluster_libreville	5
find_clusters	5
leiden	6
modularity	8
neosbm	9
with_libreville	10
Index	11

assess	<i>Statistics of a given partition on a graph</i>
--------	---

Description

How well do these labels describe this graph, with the labels taken as fixed (an annotation, a clustering from another modality)? Every quantity is a closed form of the block edge-count matrix, one pass over the edges.

Usage

```
assess(graph, membership, check_symmetry = TRUE)
```

Arguments

graph	An undirected graph: a symmetric sparse matrix (e.g. an SNN graph’s adjacency) or an igraph graph.
membership	Labels, one per vertex (any atomic type).
check_symmetry	Validate symmetry (FALSE for graphs symmetric by construction).

Details

- loglik: the degree-corrected SBM profile log-likelihood with the labels as blocks (Karrer & Newman 2011), the statistic of BESTest (Peel, Larremore & Clauset 2017), whose p-value is a permutation of the labels the caller runs.
- description_length: Peixoto’s (2017) microcanonical DC-SBM description length with uniform priors, in nats, as graph-tool’s BlockState.entropy(degree_dl_kind = "uniform"). It charges for the block matrix and the partition, so it ranks nested annotations: the shorter, the better the graph supports that level.

- significance: Traag, Krings & Van Dooren (2013), nats; leidenalg's `SignificanceVertexPartition.quality()`.
- surprise: per label, $-\ln P(X \geq \text{inner edges})$ with X Poisson at the configuration-model expectation: the label's cohesion beyond chance (Lancichinetti, Radicchi & Ramasco 2010, edge-count form).

Exact on unweighted graphs; weights count as multiplicities.

Value

A list: `loglik`, `description_length`, `significance` (numbers); `labels` (the distinct labels, in order of first appearance) and, per label, `surprise`, `size`, `degree_sum`, `inner_edges`.

References

Peel, Larremore & Clauset (2017) *Sci. Adv.* 3:e1602548; Peixoto (2017) *Phys. Rev. E* 95:012317; Traag, Krings & Van Dooren (2013) *Sci. Rep.* 3:2930; Karrer & Newman (2011) *Phys. Rev. E* 83:016107; Lancichinetti, Radicchi & Ramasco (2010) *Phys. Rev. E* 81:046110.

Examples

```
f <- system.file("extdata", "hao3k.mtx", package = "clusterLibreville")
g <- Matrix::readMM(f)
a <- assess(g, leiden(g))
a$description_length
```

champ

Pick the resolution by CHAMP (experimental)

Description

Experimental: the interface and the scores may change.

Usage

```
champ(
  graph,
  resolutions = 10^seq(-1, 1, length.out = 20),
  seeds = 0:4,
  gamma_range = NULL,
  n_iterations = -1L,
  theta = 0.01,
  min_gain = 0,
  threads = NULL,
  check_symmetry = TRUE,
  metric = c("nvi", "ami")
)
```

Arguments

graph	Undirected weighted graph: a symmetric sparse matrix or an igraph graph, as for <code>leiden()</code> .
resolutions	Grid of resolutions (gamma), all > 0.
seeds	Seeds, non-negative whole numbers below 2^{53} .
gamma_range	The range the intervals cover; NULL: the span of resolutions.
n_iterations, theta, min_gain, threads, check_symmetry	Options of <code>leiden()</code> , for every run.
metric	"nvi" (default), normalised variation of information: a distance (0 = identical) and a metric, so steps along the domains add up, as in Markov stability. "ami", adjusted mutual information: a similarity (1 = identical), corrected for the chance agreement between partitions into many small communities.

Details

Runs `leiden()` at every pair of resolutions x seeds, then keeps the partitions that have the highest modularity among that pool somewhere in `gamma_range`, each with its interval of resolutions (CHAMP: Weir et al. 2017, doi:10.3390/a10030093). Modularity is linear in the resolution, so the intervals are exact; they are optimal relative to the pool, not globally. Each partition is also scored by its agreement with its neighbouring domains and with the other seeds, under `metric`. Deterministic, at any number of threads.

Value

A list: `domains`, a data frame with one row per partition on the envelope in increasing resolution (`gamma_lo`, `gamma_hi`, `log_width`, `n_communities`, and the grid point that produced it: `resolution`, `seed`; and the agreement under `metric`: `to_prev` with the previous row's partition, `to_neighbours` the mean with both neighbours' (close: a plateau; far: a boundary), `to_seeds` the mean with the other seeds' partitions at the same grid resolution (algorithmic noise); NA where there is nothing to compare with); `labels`, an integer matrix (vertices x domains, labels 1..k, community 1 the largest); and `best`, the row of domains with the widest interval in log resolution, e.g. `labels[, best]`.

Examples

```
f <- system.file("extdata", "hao3k.mtx", package = "clusterLibreville")
g <- Matrix::readMM(f)
ch <- champ(g, 10^seq(-1, 1, length.out = 10), seeds = 0:2)
ch$domains
table(ch$labels[, ch$best])
```

cluster_libreville	<i>Deterministic Leiden as an igraph community function</i>
--------------------	---

Description

`leiden()` with igraph's `cluster_*`() interface: it returns an igraph communities object, so it plugs into igraph and into bluster's graph clustering, e.g. `bluster::clusterRows(pcs, SNNGraphParam(cluster.fun = cluster_libreville))`.

Usage

```
cluster_libreville(graph, ...)
```

Arguments

<code>graph</code>	An undirected igraph graph (edge attribute weight if present).
<code>...</code>	Options of <code>leiden()</code> : resolution, seed, n_iterations, theta, min_gain, threads.

Value

An igraph communities object (`igraph::membership()`, `igraph::modularity()` and `igraph::sizes()` work on it).

Examples

```
g <- igraph::make_graph("Zachary")
comm <- cluster_libreville(g, seed = 1)
igraph::sizes(comm)
```

find_clusters	<i>leiden() with the interface of Seurat's FindClusters</i>
---------------	---

Description

A drop-in for `Seurat::FindClusters` on a graph, for tools built on it that accept another clustering function (e.g. a patched `CHOIR`: `options(CHOIR.cluster_fun = clusterLibreville::find_clusters)`); for those that do not, see `with_libreville()`. Takes the graph (a Seurat Graph, as `obj@graphs$RNA_snn`, or any symmetric sparse matrix), resolution (a vector gives one column each) and random.seed (the `leiden()` seed), and returns what `FindClusters` returns for a graph: a data frame with one factor column per resolution (`res.<r>`, labels 1..k, 1 the largest community), rows named by cell. As in Seurat, `group.singletons` (default TRUE) merges each one-cell community into the community it is best connected to (highest mean edge weight); Seurat breaks ties at random, here the larger community wins. Seurat's other options (`algorithm`, `method`, `n.start`, `n.iter`, ...) are accepted and ignored: the clustering is always this Leiden.

Usage

```
find_clusters(
  object,
  resolution = 0.8,
  random.seed = 0,
  group.singletons = TRUE,
  ...,
  threads = NULL
)
```

Arguments

<code>object</code>	A graph, as for <code>leiden()</code> ; not a Seurat object.
<code>resolution</code>	Resolution(s) of the modularity (gamma).
<code>random.seed</code>	Seed, a non-negative whole number below 2^{53} .
<code>group.singletons</code>	Merge one-cell communities into their best-connected community.
<code>...</code>	Ignored (Seurat's options).
<code>threads</code>	Worker threads, as for <code>leiden()</code> .

Value

A data frame, one factor column per resolution.

Examples

```
g <- Matrix::readMM(system.file("extdata", "hao3k.mtx", package = "clusterLibreville"))
cl <- find_clusters(g, resolution = c(0.5, 1), random.seed = 1)
sapply(cl, nlevels)
```

leiden

Deterministic Leiden community detection

Description

Leiden (Traag, Waltman & van Eck 2019) maximising modularity at resolution `resolution`, seeded and deterministic: the same seed gives the same labels at any number of threads.

Usage

```
leiden(x, ...)

## S4 method for signature 'ANY'
leiden(
  x,
  resolution = 1,
```

```

    seed = 0,
    n_iterations = -1L,
    theta = 0.01,
    min_gain = 0,
    threads = NULL,
    check_symmetry = TRUE,
    details = FALSE
)

## S4 method for signature 'SingleCellExperiment'
leiden(x, use.dimred = "PCA", k = 10L, type = "rank", details = FALSE, ...)

```

Arguments

<code>x</code>	A graph or a <code>SingleCellExperiment</code>. - Graph: undirected and weighted, a symmetric sparse matrix (anything methods: <code>as(x, "CsparseMatrix")</code>) accepts: a <code>dgCMatrix</code>, a Seurat Graph such as <code>obj@graphs\$RNA_snn</code>, a symmetric <code>dsCMatrix</code>, a pattern matrix) or an <code>igraph</code> graph (edge attribute <code>weight</code> if present, e.g. from <code>scanr::buildSNNGraph()</code> or <code>bluster::makeSNNGraph()</code>). <code>SingleCellExperiment</code>: the cells are clustered on a shared nearest neighbour graph of <code>reducedDim(x, use.dimred)</code>, built by <code>bluster::makeSNNGraph()</code>.
<code>...</code>	Arguments for the graph method (resolution, seed, ...).
<code>resolution</code>	Resolution of the modularity (gamma).
<code>seed</code>	Seed, a non-negative whole number below 2^{53} .
<code>n_iterations</code>	Leiden iterations; -1: until an iteration changes nothing.
<code>theta</code>	Randomness of the refinement.
<code>min_gain</code>	With <code>n_iterations = -1</code> , also stop once an iteration gains less modularity than this (0: strict).
<code>threads</code>	Worker threads; NULL: all cores. Does not change the result.
<code>check_symmetry</code>	Verify that the matrix is symmetric. FALSE skips only that check, for graphs symmetric by construction (Seurat, scanpy, bluster); bounds and weights are always validated.
<code>details</code>	Also return how the run ended (see Value).
<code>use.dimred</code>	Name of the reduced dimension to build the graph from.
<code>k</code>	Number of nearest neighbours for the SNN graph.
<code>type</code>	Edge weighting of the SNN graph, as in <code>bluster::makeSNNGraph()</code> : "rank", "number" or "jaccard".

Value

For a graph: integer community labels $1..k$, community 1 the largest (ties: by smallest vertex), named by the graph's row names if it has them. For a `SingleCellExperiment`: the same labels as a factor named by cell, e.g. for `colLabels(sce) <- leiden(sce)`. With `details = TRUE`, a list: labels (as above), iterations (Leiden iterations run), converged (whether the last iteration changed nothing: a fixed point), stop (why it stopped: "converged", "min_gain" or "n_iterations") and modularity.

See Also

`cluster_libreville()` for igraph and bluster.

Examples

```
f <- system.file("extdata", "hao3k.mtx", package = "clusterLibreville")
g <- Matrix::readMM(f)
labels <- leiden(g, resolution = 1, seed = 0)
table(labels)

sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(counts = matrix(0, 2, 200)),
  reducedDims = list(PCA = rbind(matrix(rnorm(400), ncol = 4),
                                     matrix(rnorm(400, 5), ncol = 4))))
SingleCellExperiment::colLabels(sce) <- leiden(sce, seed = 1)
table(SingleCellExperiment::colLabels(sce))
```

modularity	<i>Modularity of a partition</i>
------------	----------------------------------

Description

Newman’s modularity at resolution resolution, summed in a fixed order (the quantity `leiden()` maximises).

Usage

```
modularity(graph, membership, resolution = 1)
```

Arguments

- graph Undirected weighted graph: a symmetric sparse matrix or an igraph graph, as for `leiden()`.
- membership Labels, one per vertex (any atomic type).
- resolution Resolution of the modularity (gamma).

Value

A number.

Examples

```
f <- system.file("extdata", "hao3k.mtx", package = "clusterLibreville")
g <- Matrix::readMM(f)
modularity(g, leiden(g))
```

neosbm	<i>neoSBM: how many vertices leave an annotation when leaving gets cheaper</i>
--------	--

Description

Peel, Larremore & Clauset's (2017) neoSBM: every vertex keeps its label's block or is free to sit in any block, at a prior cost of $\log((1 - \theta) / \theta)$ nats of degree-corrected SBM likelihood. Sweeping θ towards 1/2 traces the fraction of vertices that defect from the annotation; at $\theta = 1/2$ the prior is neutral and that fraction is "vertices whose label is not locally optimal for this graph". Greedy single-vertex moves with exact likelihood deltas, warm-started along θ s; a maximiser of the published objective, not Peel's reference code.

Usage

```
neosbm(
  graph,
  membership,
  thetas = c(1e-06, 1e-05, 1e-04, 0.001, 0.003, 0.01, 0.03, 0.1, 0.2, 0.3, 0.4, 0.5),
  max_sweeps = 20L,
  seed = 0L,
  check_symmetry = TRUE
)
```

Arguments

graph	An undirected graph: a symmetric sparse matrix (e.g. an SNN graph's adjacency) or an igraph graph.
membership	Labels, one per vertex (any atomic type).
thetas	The grid, in (0, 1), ascending.
max_sweeps	Sweeps per theta before giving up on convergence.
seed	Fixes the vertex order of the sweeps; NULL sweeps in vertex order (the deterministic choice for cross-checks).
check_symmetry	Validate symmetry (FALSE for graphs symmetric by construction).

Value

A list: thetas; per theta defect (fraction off their label), loglik_gain (nats recovered) and sweeps; blocks (the block of every vertex at the last theta, as indices into labels); labels.

Examples

```
f <- system.file("extdata", "hao3k.mtx", package = "clusterLibreville")
g <- Matrix::readMM(f)
r <- neosbm(g, leiden(g))
r$defect
```

with_libreville	<i>Run code with Seurat's FindClusters replaced by find_clusters()</i>
-----------------	--

Description

Evaluates `expr` with `Seurat::FindClusters` swapped for `find_clusters()` and restores it afterwards (also on error). For tools that cluster through `Seurat::FindClusters` and have no option to choose the clustering function, such as `CHOIR`: `with_libreville(CHOIR::CHOIR(obj))` makes every clustering in its tree deterministic Leiden, seeded with `CHOIR`'s `random_seed`.

Usage

```
with_libreville(expr, threads = NULL)
```

Arguments

<code>expr</code>	Code to run.
<code>threads</code>	Worker threads for every clustering, as for <code>leiden()</code> .

Details

The function is replaced in `Seurat`'s namespace, for the whole R session, while `expr` runs.

Value

The value of `expr`.

Examples

```
g <- Matrix::readMM(system.file("extdata", "hao3k.mtx", package = "clusterLibreville"))
cl <- with_libreville(Seurat::FindClusters(g, resolution = 1, random.seed = 1))
table(cl$res.1)
```

Index

`assess`, [2](#)

`bluster::makeSNNGraph()`, [7](#)

`champ`, [3](#)

`cluster_libreville`, [5](#)

`cluster_libreville()`, [8](#)

`find_clusters`, [5](#)

`find_clusters()`, [10](#)

`leiden`, [6](#)

`leiden()`, [4–6](#), [8](#), [10](#)

`leiden`, ANY-method (`leiden`), [6](#)

`leiden`, SingleCellExperiment-method
(`leiden`), [6](#)

`modularity`, [8](#)

`neosbm`, [9](#)

`with_libreville`, [10](#)

`with_libreville()`, [5](#)