# Package 'rgph'

February 24, 2026

**Type** Package

**Title** Pair Critical Points and Compute Persistent Homology of Reeb Graphs

**Version** 0.1.0

**Description** Interface to the 'ReebGraphPairing' program to compute critical points of Reeb graphs following Tu, Hajij, & Rosen (2019) <doi:10.1007/978-3-030-33720-9_8> via the 'rJava' package. Also store Reeb graphs in a minimal S3 class, convert between other network data structures, and post-process pairing data to obtain extended persistent homology following Carrière & Oudot (2018) <doi:10.1007/s10208-017-9370-z>.

**Depends** R (>= 2.7.0), rJava (>= 0.5-0), phutil

**Suggests** rlang, tinytest, igraph (>= 0.6-0), network (>= 1.19.0), knitr, rmarkdown, dplyr, tidyr, scales, ggplot2, bench

**SystemRequirements** Java (>= 5.0)

**License** GPL-3

**Repository** CRAN

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Satyajit Mohanty [aut], Shubham Singh [aut], Jason Cory Brunson [aut, cre] (ORCID: <https://orcid.org/0000-0003-3126-9494>), Paul Rosen [cph, ctb] (ReebGraphPairing; GPL-3 license, ORCID: <https://orcid.org/0000-0002-0873-9518>), Junyi Tu [cph, ctb] (ReebGraphPairing; GPL-3 license, ORCID: <https://orcid.org/0000-0001-7026-7454>)

**Maintainer** Jason Cory Brunson <cornelioid@gmail.com>

**Date/Publication** 2026-02-24 19:30:02 UTC

# Contents



---

`as_reeb_graph` *Coerce objects to class* `reeb_graph`

---

## Description

Coerce objects to [reeb_graph]-class objects.

## Usage

```
as_reeb_graph(x, ...)

## S3 method for class 'igraph'
as_reeb_graph(x, values = NULL, names = NULL, ...)

## S3 method for class 'network'
as_reeb_graph(x, values = NULL, names = NULL, ...)

as_igraph(x, ...)

## S3 method for class 'reeb_graph'
as_igraph(x, values = "value", names = "name", ...)

as_network(x, ...)

## S3 method for class 'reeb_graph'
as_network(x, values = "value", names = "vertex.names", ...)
```

## Arguments

| | |
|---|---|
| x | An R object to be coerced. See Details. |
| ... | Additional arguments passed to methods. |
| values | For coercion *to* class reeb_graph, a character value; the node attribute to use as the Reeb graph value function. If NULL (the default), the first numeric node attribute is used. For coercion *from* class reeb_graph, a character value; the name of the node attribute in which to store the Reeb graph value function. |
| names | For coercion *to* class reeb_graph, a character value; the node attribute to use as the Reeb graph node names. If NULL, names are omitted. For coercion *from* class reeb_graph, a character value; the name of the node attribute in which to store the Reeb graph node names. |

## Details

The `as_reeb_graph()` methods require a network (mathematical graph) structure and a real-valued function on the vertex set.

For coercion between external network classes, use the `intergraph` package.

## Value

A reeb_graph object.

## See Also

`reeb_graph()`

## Examples

```
library(igraph)
( g <- make_kautz_graph(2, 1) )
l_g <- layout_with_fr(g)
plot(g, layout = l_g)
( rg <- as_reeb_graph(g, l_g[, 1]) )
vertex_attr(g, "height") <- rg$value
heights <- sort(unique(V(g)$height))
l_rg <- layout_with_sugiyama(g, layers = round(V(g)$height * 100))
plot(g, layout = l_rg)


library(network)
data("emon")
mtsi <- emon$Cheyenne
mtsi_reeb <- as_reeb_graph(
  mtsi,
  values = "Command.Rank.Score",
  names = "vertex.names"
)
print(mtsi_reeb, minlength = 24)
```

---

# reeb_graph *An S3 class and constructors for Reeb graphs*

---

## Description

This is an S3 class with associated constructors for a data structure to represent Reeb graphs in R.

## Usage

```
reeb_graph(values, edgelist)

## S3 method for class 'reeb_graph'
print(x, ..., n = NULL, minlength = 12L)

## S3 method for class 'reeb_graph'
format(x, ..., n = NULL, minlength = 12L)

read_reeb_graph(file)
```

## Arguments

| | |
|---|---|
| values | Numeric vector of function values at vertices; may have names, which may be duplicated and/or missing. |
| edgelist | 2-column integer matrix of linked vertex pairs. |
| x | Object of class reeb_graph. |
| ... | Additional arguments passed to base::format(). |
| n | Integer number of edges to print. |
| minlength | Minimum name abbreviation length; passed to base::abbreviate(). |
| file | A plain text file containing Reeb graph data formatted as at ReebGraphPairing. |

## Details

Vertex indices start at zero, for consistency with examples. The positions of values and the integer values in edgelist will correspond to the same vertices; length(values) must bound max(edgelist).

The S3 class is a list of "values" and "edgelist". The print() method prints one edge per line, with nodes formatted as "index[name] (value)"

## Value

An object of class "reeb_graph", which is a list of two elements:

- values: Numeric vector of function values at vertices, optionally named.
- edgelist: 2-column integer matrix of linked vertex pairs.

## References

https://github.com/USFDataVisualization/ReebGraphPairing/

## See Also

as_reeb_graph()

## Examples

```
x <- reeb_graph(
  values = c(a = 0, b = .4, c = .6, d = 1),
  edgelist = rbind( c(1,2), c(1,3), c(2,4), c(3,4))
)
print(x)

t10 <- system.file("extdata", "10_tree_iterations.txt", package = "rgph")
( y <- read_reeb_graph(t10) )

reeb_graph_pairs(x, method = "multi_pass")
reeb_graph_pairs(y, method = "multi_pass")
```

---

`reeb_graph_examples` *Mesh-Derived Reeb Graphs*

---

## Description

These are some of the Reeb graphs used by Tu &al (2019; Fig. 8) to benchmark the multi-pass and single-pass algorithms. The original meshes were obtained from the `AIM@SHAPE` Shape Repository, while the authors computed Reeb graphs using a custom C++ implementation.

## Usage

```
david
buddha
topology
flower
```

## Format

Objects of class `reeb_graph`.

## Source

http://visionair.ge.imati.cnr.it/ontologies/shapes/ (`AIM@SHAPE`; defunct) https://github.com/USFDataVisualization/ReebGraphPairing

reeb_graph_pairs *Pair Reeb Graph Critical Points via Java*

## Description

This function calls one of two methods, merge-pair and propagate-and-pair, to pair the critical points of a Reeb graph.

## Usage

```
reeb_graph_pairs(
  x,
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  ...
)

## Default S3 method:
reeb_graph_pairs(
  x,
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  ...
)

## S3 method for class 'igraph'
reeb_graph_pairs(
  x,
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  values = NULL,
  ...
)

## S3 method for class 'network'
reeb_graph_pairs(
  x,
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  values = NULL,
  ...
)

## S3 method for class 'reeb_graph'
reeb_graph_pairs(
  x,
  sublevel = TRUE,
```

```
  method = c("single_pass", "multi_pass"),
  ...
)

## S3 method for class 'reeb_graph_pairs'
as.data.frame(x, ...)

## S3 method for class 'reeb_graph_pairs'
print(x, ..., n = NULL, minlength = 12L)

## S3 method for class 'reeb_graph_pairs'
format(x, ..., n = NULL, minlength = 12L)
```

## Arguments

| | |
|---|---|
| x | A reeb_graph object. |
| sublevel | Logical; whether to use the sublevel set filtration (TRUE, the default) or else the superlevel set filtration (via reversing x[["values"]] before paring critical points. |
| method | Character; the pairing method to use. Matched to "single_pass" (the default) or "multi_pass". |
| ... | Additional arguments passed to methods. |
| values | For coercion *to* class reeb_graph, a character value; the node attribute to use as the Reeb graph value function. If NULL (the default), the first numeric node attribute is used. For coercion *from* class reeb_graph, a character value; the name of the node attribute in which to store the Reeb graph value function. |
| n | Integer number of critical pairs to print. |
| minlength | Minimum name abbreviation length; passed to base::abbreviate(). |

## Details

The function uses the rJava package to call either of two Java methods from ReebGraphPairing. Ensure the Java Virtual Machine (JVM) is initialized and the required class is available in the class path.

The Propagate-and-Pair algorithm ("single_pass") performs both join and split merge tree operations along a single sweep through the Reeb graph. It was shown to be more efficient on most test data, and to scale better with graph size, than an algorithm ("multi_pass") that pairs some types along the sublevel filtration and others along the superlevel filtration (Tu &al, 2019).

The output S3 class is a list of 2-column matrices containing the types, values, indices, and orders of persistent pairs, with attributes containing the node names and metadata. The print() method visually expresses each pair, increasing from left to right, with nodes formatted as with reeb_graph.

The names of the coerced data frame use lo_ and hi_ prefixes, in contrast to the Java source code that uses birth_ and death_. This is meant to distinguish the pairs and their metadata from persistent homology, which is here reformulated following Carrière & Oudot (2018).

## Value

A list of subclass reeb_graph_pairs containing 4 2-column matrices characterizing the low- and high-valued critical points of each pair:

`type` Character; the type of critical point, one of `LEAF_MIN`, `LEAF_MAX`, `UPFORK`, and `DOWNFORK`.

`value` Double; the value (stored in `x[["values"]]`) of the critical point.

`index` Integer; the index (used in `x[["edgelist"]]`) of the critical point. Regular points will not appear, while degenerate critical points will appear multiple times.

`order` Integer; the order of the critical point in the pairing. This is based on the conditioned Reeb graph constructed internally so will not be duplicated.

The data frame also has attributes `"names"` for the node names, `"method"` for the method used, and `"elapsed_time"` for the elapsed time.

## References

https://github.com/USFDataVisualization/ReebGraphPairing/

Tu J, Hajij M, Rosen P. Propagate and Pair: A Single-Pass Approach to Critical Point Pairing in Reeb Graphs. In: Bebis G, Boyle R, Parvin B, &al, eds. *Advances in Visual Computing. Lecture Notes in Computer Science*. Springer International Publishing; 2019:99–113. doi:10.1007/9783-030337209_8

Carrière M & Oudot S (2018) "Structure and Stability of the One-Dimensional Mapper". *Foundations of Computational Mathematics* 18(6): 1333–1396. doi:10.1007/s102080179370z

## See Also

reeb_graph_persistence()

## Examples

```
ex_sf <- system.file("extdata", "running_example.txt", package = "rgph")
( ex_rg <- read_reeb_graph(ex_sf) )
( ex_cp <- reeb_graph_pairs(ex_rg) )
attr(ex_cp, "method")
attr(ex_cp, "elapsed_time")

reeb_graph_pairs(ex_rg, sublevel = FALSE)

x <- reeb_graph(
  values = c(0, .4, .6, 1),
  edgelist = c( 1,2, 1,3, 2,4, 3,4 )
)
( mp <- reeb_graph_pairs(x) )
class(mp)
as.data.frame(mp)

names(x$values) <- letters[seq_along(x$values)]
( mp <- reeb_graph_pairs(x) )
as.data.frame(mp)
```

```
library(network)
data("emon")
mtsi <- emon$Cheyenne
mtsi_reeb <- as_reeb_graph(
  mtsi,
  values = "Command.Rank.Score",
  names = "vertex.names"
)
mtsi_cp <- reeb_graph_pairs(mtsi_reeb, sublevel = FALSE)
print(mtsi_cp, minlength = 20)
```

---

`reeb_graph_persistence` *Compute Extended Persistent Homology of a Reeb Graph*

---

## Description

This function obtains extended persistent homology of a Reeb graph by way of pairing critical points.

## Usage

```
reeb_graph_persistence(x, scale = c("value", "index", "order"), ...)

## Default S3 method:
reeb_graph_persistence(x, scale = c("value", "index", "order"), ...)

## S3 method for class 'igraph'
reeb_graph_persistence(
  x,
  scale = c("value", "index", "order"),
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  values = NULL,
  ...
)

## S3 method for class 'network'
reeb_graph_persistence(
  x,
  scale = c("value", "index", "order"),
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  values = NULL,
  ...
```

```
)

## S3 method for class 'reeb_graph'
reeb_graph_persistence(
  x,
  scale = c("value", "index", "order"),
  sublevel = TRUE,
  method = c("single_pass", "multi_pass"),
  ...
)

## S3 method for class 'reeb_graph_pairs'
reeb_graph_persistence(x, scale = c("value", "index", "order"), ...)
```

## Arguments

| | |
|---|---|
| x | A reeb_graph or reeb_graph_pairs object, or an object that can be coerced to class "reeb_graph". |
| scale | Character; the scale parameter used by the persistent pairs. Matched to "value" (the default), "index", or "order". |
| ... | Additional arguments passed to methods. |
| sublevel | Logical; whether to use the sublevel set filtration (TRUE, the default) or else the superlevel set filtration (via reversing x[["values"]] before paring critical points. |
| method | Character; the pairing method to use. Matched to "single_pass" (the default) or "multi_pass". |
| values | For coercion *to* class reeb_graph, a character value; the node attribute to use as the Reeb graph value function. If NULL (the default), the first numeric node attribute is used. For coercion *from* class reeb_graph, a character value; the name of the node attribute in which to store the Reeb graph value function. |

## Details

The types, values, and indices of critical pairs are obtained by reeb_graph_pairs(). reeb_graph_persistence() calls this function internally with the prescribed method, then restructures the values or indices as phutil::persistence data.

This function may be deprecated once a reeb_graph_pairs method is written for phutil::as_persistence().

## Value

A phutil::persistence object.

## See Also

reeb_graph_pairs()

## Examples

```
ex_sf <- system.file("extdata", "running_example.txt", package = "rgph")
( ex_rg <- read_reeb_graph(ex_sf) )
( ex_ph <- reeb_graph_persistence(ex_rg) )
phutil::get_pairs(ex_ph, dimension = 0)
phutil::get_pairs(ex_ph, dimension = 1)

t10_f <- system.file("extdata", "10_tree_iterations.txt", package = "rgph")
( t10 <- read_reeb_graph(t10_f) )
( t10_ph <- reeb_graph_persistence(t10) )
phutil::get_pairs(t10_ph, dimension = 0)
( t10_ph <- reeb_graph_persistence(t10, scale = "index") )
phutil::get_pairs(t10_ph, dimension = 0)
( t10_ph <- reeb_graph_persistence(t10, scale = "order") )
phutil::get_pairs(t10_ph, dimension = 0)
```

# Index