

Package ‘edgebundle’

July 24, 2026

Title Algorithms for Bundling Edges in Networks and Visualizing Flow and Metro Maps

Version 1.0.0

Description Implements several algorithms for bundling edges in networks and flow and metro map layouts. This includes force directed edge bundling <doi:10.1111/j.1467-8659.2009.01450.x>, a flow algorithm based on Steiner trees<doi:10.1080/15230406.2018.1437359> and a multicriteria optimization method for metro map layouts <doi:10.1109/TVCG.2010.24>.

URL <https://github.com/schochastics/edgebundle>,
<https://schochastics.github.io/edgebundle/>

BugReports <https://github.com/schochastics/edgebundle/issues>

License MIT + file LICENSE

Suggests testthat (>= 3.0.0), network, tidygraph, interp

Config/testthat/edition 3

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports Rcpp, igraph

Depends R (>= 3.5)

Config/roxygen2/version 8.0.0.9000

NeedsCompilation yes

Author David Schoch [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-2952-4812>>),
Jose Luis Blanco-Claraco [cph] (Author of the bundled nanoflann C++ library)

Maintainer David Schoch <david@schochastics.net>

Repository CRAN

Date/Publication 2026-07-24 15:10:02 UTC

Contents

cali2010	2
convert_edges	3
edge_bundle	3
edge_bundle_force	4
edge_bundle_hammer	6
edge_bundle_mingle	8
edge_bundle_path	9
edge_bundle_stub	10
flow_tree	12
metro_berlin	13
metro_multicriteria	13
tnss_dummies	15
tnss_smooth	16
tnss_tree	17
us_flights	18
us_migration	18
Index	19

cali2010	<i>Migration from California in 2010</i>
----------	--

Description

A dataset containing the number of people who migrated from California to other US states

Usage

cali2010

Format

igraph object

Source

<https://www.census.gov/data/tables/time-series/demo/geographic-mobility/state-to-state-migration.html>

convert_edges	<i>Convert edges</i>
---------------	----------------------

Description

converts edges of an igraph/network/tidygraph object into format useable for edge bundling

Usage

```
convert_edges(object, coords)

## Default S3 method:
convert_edges(object, coords)

## S3 method for class 'igraph'
convert_edges(object, coords)

## S3 method for class 'network'
convert_edges(object, coords)

## S3 method for class 'tbl_graph'
convert_edges(object, coords)
```

Arguments

object	graph object
coords	coordinates of vertices

Value

data frame of edges with coordinates

Author(s)

David Schoch

edge_bundle	<i>Edge bundling</i>
-------------	----------------------

Description

Dispatches to one of the edge bundling algorithms by name. This is a thin wrapper over the individual `edge_bundle_*`() functions and mirrors the way frontends (e.g. `vellumplot`) select a bundling method.

Usage

```
edge_bundle(
  object,
  xy,
  type = c("force", "divided", "stub", "path", "hammer", "mingle"),
  ...
)
```

Arguments

object	a graph object (igraph/network/tbl_graph)
xy	coordinates of vertices
type	bundling algorithm: "force" (force-directed, Holten), "divided" (force-directed for directed graphs, Selassie et al.), "stub" (Nocaj & Brandes), "path" (edge-path, Wallinger et al.), "hammer" (KDE, Hurter et al.), or "mingle" (Gansner et al.)
...	arguments passed on to the selected edge_bundle_*() function

Value

data.frame containing the bundled edges (x, y, index, group)

Author(s)

David Schoch

See Also

[edge_bundle_force](#), [edge_bundle_stub](#), [edge_bundle_path](#), [edge_bundle_hammer](#), [edge_bundle_mingle](#)

Examples

```
library(igraph)
g <- graph_from_edgelist(
  matrix(c(1, 12, 2, 11, 3, 10, 4, 9, 5, 8, 6, 7), ncol = 2, byrow = TRUE), FALSE
)
xy <- cbind(c(rep(0, 6), rep(1, 6)), c(1:6, 1:6))
edge_bundle(g, xy, type = "force")
```

edge_bundle_force	<i>force directed edge bundling</i>
-------------------	-------------------------------------

Description

Implements the classic edge bundling by Holten.

Usage

```
edge_bundle_force(
  object,
  xy,
  K = 1,
  C = 6,
  P = 1,
  S = 0.04,
  P_rate = 2,
  I = 50,
  I_rate = 2/3,
  compatibility_threshold = 0.6,
  eps = 1e-08,
  directed = FALSE,
  ...
)
```

Arguments

object	a graph object (igraph/network/tbl_graph)
xy	coordinates of vertices
K	spring constant
C	number of iteration cycles
P	number of initial edge divisions
S	initial step size
P_rate	rate of edge divisions
I	number of initial iterations
I_rate	rate of iteration decrease per cycle
compatibility_threshold	threshold for when edges are considered compatible
eps	accuracy
directed	logical. If TRUE, use divided edge bundling (Selassie et al. 2011) which keeps edges running in opposite directions in separate lanes. Requires an igraph/tbl_graph object with edge directions.
...	arguments passed to the divided bundler when directed = TRUE: lane_width, k_spring, k_charge, decay, friction, step, passes, iterations, use_connectivity.

Details

This is a re-implementation of <https://github.com/upphiminn/d3.ForceBundle>. Force directed edge bundling is slow ($O(E^2)$).

see [online](#) for plotting tips

Value

data.frame containing the bundled edges

Author(s)

David Schoch

References

Holten, Danny, and Jarke J. Van Wijk. "Force-Directed Edge Bundling for Graph Visualization." Computer Graphics Forum (Blackwell Publishing Ltd) 28, no. 3 (2009): 983-990.

Selassie, David, Brandon Heller, and Jeffrey Heer. "Divided edge bundling for directional network data." IEEE Transactions on Visualization and Computer Graphics 17, no. 12 (2011): 2354-2363.

See Also

[edge_bundle_hammer](#), [edge_bundle_stub](#), [edge_bundle_path](#)

Examples

```
library(igraph)
g <- graph_from_edgelist(
  matrix(c(
    1, 12, 2, 11, 3, 10,
    4, 9, 5, 8, 6, 7
  ), ncol = 2, byrow = TRUE), FALSE
)
xy <- cbind(c(rep(0, 6), rep(1, 6)), c(1:6, 1:6))
edge_bundle_force(g, xy)
```

edge_bundle_hammer	<i>hammer edge bundling</i>
--------------------	-----------------------------

Description

Implements hammer edge bundling via kernel density estimation (KDEEB).

Usage

```
edge_bundle_hammer(
  object,
  xy,
  bw = 0.05,
  decay = 0.7,
  npoints = 50,
  iterations = 12,
  grid = 256,
  step = 0.6,
  smooth = 1
)
```

Arguments

object	a graph object (igraph/network/tbl_graph)
xy	coordinates of vertices
bw	initial bandwidth (fraction of the layout extent)
decay	bandwidth decay per iteration (0-1)
npoints	number of points sampled per edge
iterations	number of bundling iterations
grid	resolution of the density grid
step	advection step size (multiple of the bandwidth)
smooth	number of smoothing passes per iteration

Details

Native re-implementation of the KDE-based bundling behind the datashader "hammer" bundler, following Hurter, Ersoy and Telea (2012). Earlier versions of this function wrapped datashader through reticulate; it now runs entirely in C++ with no Python dependency. Edges are sampled into points, a density field is estimated on a grid, points are advected up the density gradient, and the bandwidth shrinks each iteration so bundles emerge along density ridges.

see [online](#) for plotting tips

Value

data.frame containing the bundled edges

Author(s)

David Schoch

References

Hurter, Christophe, Ozan Ersoy, and Alexandru Telea. "Graph Bundling by Kernel Density Estimation." Computer Graphics Forum 31, no. 3 (2012): 865-874.

See Also

[edge_bundle_force](#), [edge_bundle_stub](#), [edge_bundle_path](#)

Examples

```
library(igraph)
g <- graph_from_edgelist(
  matrix(c(1, 12, 2, 11, 3, 10, 4, 9, 5, 8, 6, 7), ncol = 2, byrow = TRUE), FALSE
)
xy <- cbind(c(rep(0, 6), rep(1, 6)), c(1:6, 1:6))
edge_bundle_hammer(g, xy)
```

edge_bundle_mingle	<i>MINGLE edge bundling</i>
--------------------	-----------------------------

Description

Multilevel agglomerative edge bundling (MINGLE).

Usage

```
edge_bundle_mingle(object, xy, k = 10, segments = 50, bundle_strength = 0.9)
```

Arguments

object	a graph object (igraph/network/tbl_graph)
xy	coordinates of vertices
k	number of nearest neighbours considered as merge candidates per edge
segments	number of points sampled per bundled edge
bundle_strength	strength of bundling between 0 (straight edges) and 1 (route fully through the meeting points)

Details

Groups edges bottom-up: two bundles are merged whenever routing them through shared meeting points reduces the total drawn length ("ink"). Meeting points are the geometric medians of the source-side and target-side endpoints. A kNN proximity graph over the edges is built once with a kd-tree and then coarsened level by level ($O(E \log E)$).

see [online](#) for plotting tips

Value

data.frame containing the bundled edges

Author(s)

David Schoch

References

Gansner, Emden R., Yifan Hu, Stephen North, and Carlos Scheidegger. "Multilevel agglomerative edge bundling for visualizing large graphs." 2011 IEEE Pacific Visualization Symposium (2011): 187-194.

See Also

[edge_bundle_force](#), [edge_bundle_stub](#), [edge_bundle_path](#), [edge_bundle_hammer](#)

Examples

```
library(igraph)
g <- graph_from_edgelist(
  matrix(c(1, 12, 2, 11, 3, 10, 4, 9, 5, 8, 6, 7), ncol = 2, byrow = TRUE), FALSE
)
xy <- cbind(c(rep(0, 6), rep(1, 6)), c(1:6, 1:6))
edge_bundle_mingle(g, xy)
```

edge_bundle_path

*Edge-Path Bundling***Description**

Implements edge-path bundling.

Usage

```
edge_bundle_path(
  g,
  xy,
  max_distortion = 2,
  weight_fac = 2,
  segments = 20,
  bundle_strength = 1,
  mode = "out"
)
```

Arguments

<code>g</code>	an igraph object
<code>xy</code>	coordinates of vertices
<code>max_distortion</code>	maximum distortion
<code>weight_fac</code>	edge weight factor
<code>segments</code>	number of subdivisions of edges
<code>bundle_strength</code>	bundle strength factor
<code>mode</code>	the parameter for <code>shortest_paths</code>

Details

This is a re-implementation of <https://github.com/mwallinger-tu/edge-path-bundling>
 see [online](#) for plotting tips

Value

data.frame containing the bundled edges

Author(s)

David Schoch

References

Wallerger, M., Archambault, D., Auber, D., Nollenburg, M., & Peltonen, J. (2021). Edge-Path Bundling: A Less Ambiguous Edge Bundling Approach. IEEE Transactions on Visualization and Computer Graphics.

See Also

[edge_bundle_hammer](#), [edge_bundle_stub](#), [edge_bundle_force](#)

Examples

```
library(igraph)
g <- graph_from_edgelist(matrix(c(
  1, 2, 1, 6,
  1, 4, 2, 3, 3, 4, 4, 5, 5, 6
), ncol = 2, byrow = TRUE), FALSE)
xy <- cbind(c(0, 10, 25, 40, 50, 50), c(0, 15, 25, 15, 0, -10))
edge_bundle_path(g, xy)
```

edge_bundle_stub

stub edge bundling

Description

Implements the stub edge bundling by Nocaj and Brandes

Usage

```
edge_bundle_stub(
  object,
  xy,
  alpha = 11,
  beta = 75,
  gamma = 40,
  t = 0.5,
  tshift = 0.5
)
```

Arguments

object	a graph object (igraph/tbl_graph). Does not support network objects
xy	coordinates of vertices
alpha	maximal angle (in degree) between consecutive edges in a bundle

beta	angle (in degree) at which to connect two stubs
gamma	maximal overall angle (in degree) of an edge bundle
t	numeric between 0 and 1. control point location
tshift	numeric between 0 and 1. The closer to one, the longer the bigger bundle

Details

see [online](#) for plotting tips

Value

data.frame containing the bundled edges

Author(s)

David Schoch

References

Nocaj, Arlind, and Ulrik Brandes. "Stub bundling and confluent spirals for geographic networks." International Symposium on Graph Drawing. Springer, Cham, 2013.

See Also

[edge_bundle_hammer](#), [edge_bundle_force](#), [edge_bundle_path](#)

Examples

```
library(igraph)
g <- make_star(10, "undirected")

xy <- matrix(c(
  0, 0,
  cos(90 * pi / 180), sin(90 * pi / 180),
  cos(80 * pi / 180), sin(80 * pi / 180),
  cos(70 * pi / 180), sin(70 * pi / 180),
  cos(330 * pi / 180), sin(330 * pi / 180),
  cos(320 * pi / 180), sin(320 * pi / 180),
  cos(310 * pi / 180), sin(310 * pi / 180),
  cos(210 * pi / 180), sin(210 * pi / 180),
  cos(200 * pi / 180), sin(200 * pi / 180),
  cos(190 * pi / 180), sin(190 * pi / 180)
), ncol = 2, byrow = TRUE)

edge_bundle_stub(g, xy)
# use ggforce::geom_bezier for plotting
```

flow_tree

Spiral flow tree

Description

Computes a one-to-many flow map as an angle-restricted spiral tree (Verbeek, Buchin and Speckmann 2011). This is the recommended flow map layout; unlike [tnss_tree](#) it needs no dummy nodes or triangulation, keeps the real node positions fixed, produces a planar (crossing-free) tree with smooth, shallow-angle merges, and is tuned by a single parameter alpha.

Usage

```
flow_tree(object, xy, root, alpha = 40, n = 20, optimize = FALSE)
```

Arguments

object	a one-to-many flow network (star graph) as <code>igraph/tbl_graph</code> , with a weight edge attribute giving the flow
xy	coordinates of vertices
root	root node id of the flow
alpha	restricting angle in degrees (0-90). Smaller values bundle more tightly toward the root; typical values are 20-45.
n	number of points sampled per tree edge
optimize	logical. If TRUE, refine the tree with an approximate FLOWTREE optimization (Verbeek et al. 2011, section 5): join points stay fixed while edge interiors are relaxed for smoothness and to keep clear of node obstacles. This can slightly relax the strict alpha bound.

Value

data.frame with columns x, y, flow and edge (one smooth arc per tree edge)

Author(s)

David Schoch

References

Verbeek, Kevin, Kevin Buchin, and Bettina Speckmann. "Flow map layout via spiral trees." IEEE Transactions on Visualization and Computer Graphics 17, no. 12 (2011): 2536-2545.

See Also

[tnss_tree](#)

Examples

```
xy <- cbind(state.center$x, state.center$y)[!state.name %in% c("Alaska", "Hawaii"), ]
flow <- flow_tree(cali2010, xy, root = 4, alpha = 40)
```

metro_berlin	<i>Subway network of Berlin</i>
--------------	---------------------------------

Description

A dataset containing the subway network of Berlin

Usage

```
metro_berlin
```

Format

igraph object

References

Kujala, Rainer, et al. "A collection of public transport network data sets for 25 cities." Scientific data 5 (2018): 180089.

metro_multicriteria	<i>Metro Map Layout (deprecated)</i>
---------------------	--------------------------------------

Description

Metro map layout based on multicriteria optimization. **Deprecated:** use `graphlayouts::layout_as_metromap()` instead, which now provides this layout.

Usage

```
metro_multicriteria(object, xy, l = 2, gr = 0.0025, w = rep(1, 5), bsize = 5)
```

Arguments

object	original graph
xy	initial layout of the original graph
l	desired multiple of grid point spacing. ($l \cdot gr$ determines desired edge length)
gr	grid spacing. ($l \cdot gr$ determines desired edge length)
w	weight vector for criteria (see details)
bsize	number of grid points a station can move away from its original position

Details

The function optimizes the following five criteria using a hill climbing algorithm:

- *Angular Resolution Criterion*: The angles of incident edges at each station should be maximized, because if there is only a small angle between any two adjacent edges, then it can become difficult to distinguish between them
- *Edge Length Criterion*: The edge lengths across the whole map should be approximately equal to ensure regular spacing between stations. It is based on the preferred multiple, l , of the grid spacing, g . The purpose of the criterion is to penalize edges that are longer than or shorter than lg .
- *Balanced Edge Length Criterion*: The length of edges incident to a particular station should be similar
- *Line Straightness Criterion*: (not yet implemented) Edges that form part of a line should, where possible, be co-linear either side of each station that the line passes through
- *Octilinearity Criterion*: Each edge should be drawn horizontally, vertically, or diagonally at 45 degree, so we penalize edges that are not at a desired angle see [online](#) for more plotting tips

Value

new coordinates for stations

Author(s)

David Schoch

References

Stott, Jonathan, et al. "Automatic metro map layout using multicriteria optimization." IEEE Transactions on Visualization and Computer Graphics 17.1 (2010): 101-114.

Examples

```
# the algorithm has problems with parallel edges
library(igraph)
g <- simplify(metro_berlin)
xy <- cbind(V(g)$lon, V(g)$lat) * 100

# the algorithm is not very stable. try playing with the parameters
xy_new <- metro_multicriteria(g, xy, l = 2, gr = 0.5, w = c(100, 100, 1, 1, 100), bsize = 35)
```

tnss_dummies

Sample points for triangulated networks

Description

uses various sampling strategies to create dummy nodes for the [tnss_tree](#)

Usage

```
tnss_dummies(
  xy,
  root,
  circ = TRUE,
  line = TRUE,
  diag = TRUE,
  grid = FALSE,
  rand = FALSE,
  ncirc = 9,
  rcirc = 2,
  nline = 10,
  ndiag = 50,
  ngrid = 50,
  nrnd = 50
)
```

Arguments

xy	coordinates of "real" nodes
root	root node id
circ	logical. create circular dummy nodes around leafs.
line	logical. create dummy nodes on a straight line between root and leafs.
diag	logical. create dummy nodes diagonally through space.
grid	logical. create dummy nodes on a grid.
rand	logical. create random dummy nodes.
ncirc	numeric. number of circular dummy nodes per leaf.
rcirc	numeric. radius of circles around leaf nodes.
nline	numeric. number of straight line nodes per leaf.
ndiag	numeric. number of dummy nodes on diagonals.
ngrid	numeric. number of dummy nodes per dim on grid.
nrnd	numeric. number of random nodes to create.

Value

coordinates of dummy nodes

Author(s)

David Schoch

Examples

```
# dummy nodes for tree rooted in California
xy <- cbind(state.center$x, state.center$y)
xy_dummy <- tnss_dummies(xy, 4)
```

tnss_smooth

*Smooth a Steiner tree***Description**

Converts the Steiner tree to smooth paths

Usage

```
tnss_smooth(g, bw = 3, n = 10)
```

Arguments

g	Steiner tree computed with tnss_tree
bw	bandwidth of Gaussian Kernel
n	number of extra nodes to include per edge

Detailssee see [online](#) for tips on plotting the result**Value**

data.frame containing the smoothed paths

Author(s)

David Schoch

Examples

```
xy <- cbind(state.center$x, state.center$y)[!state.name %in% c("Alaska", "Hawaii"), ]
xy_dummy <- tnss_dummies(xy, root = 4)
gtree <- tnss_tree(cali2010, xy, xy_dummy, root = 4, gamma = 0.9)
tree_smooth <- tnss_smooth(gtree, bw = 10, n = 10)
```

tnss_tree	Create Steiner tree from real and dummy points
-----------	--

Description

Creates an approximated Steiner tree for a flow map visualization. For most purposes [flow_tree](#) is now the recommended flow map layout (planar, angle-restricted, no dummy nodes); `tnss_tree` is kept as an alternative and requires the `interp` package for triangulation.

Usage

```
tnss_tree(  
  g,  
  xy,  
  xydummy,  
  root,  
  gamma = 0.9,  
  epsilon = 0.3,  
  elen = Inf,  
  order = "random"  
)
```

Arguments

<code>g</code>	original flow network (must be a one-to-many flow network, i.e star graph). Must have a weight attribute indicating the flow
<code>xy</code>	coordinates of "real" nodes
<code>xydummy</code>	coordinates of "dummy" nodes
<code>root</code>	root node id of the flow
<code>gamma</code>	edge length decay parameter
<code>epsilon</code>	percentage of points kept on a line after straightening with Visvalingam Algorithm
<code>elen</code>	maximal length of edges in triangulation
<code>order</code>	in which order shortest paths are calculated ("random", "weight", "near", "far")

Details

Use [tnss_smooth](#) to smooth the edges of the tree

Value

approximated Steiner tree from dummy and real nodes as `igraph` object

Author(s)

David Schoch

References

Sun, Shipeng. "An automated spatial flow layout algorithm using triangulation, approximate Steiner tree, and path smoothing." AutoCarto, 2016.

Examples

```
xy <- cbind(state.center$x, state.center$y)[!state.name %in% c("Alaska", "Hawaii"), ]
xy_dummy <- tnss_dummies(xy, root = 4)
gtree <- tnss_tree(cali2010, xy, xy_dummy, root = 4, gamma = 0.9)
```

us_flights	<i>Flights within the US</i>
------------	------------------------------

Description

A dataset containing flights between US airports as igraph object

Usage

```
us_flights
```

Format

igraph object

Source

<https://gist.githubusercontent.com/mbostock/7608400/raw>

us_migration	<i>Migration within the US 2010-2019</i>
--------------	--

Description

A dataset containing the number of people migrating between US states from 2010-2019

Usage

```
us_migration
```

Format

data.frame

Source

<https://www.census.gov/data/tables/time-series/demo/geographic-mobility/state-to-state-migration.html>

Index

* datasets

- cali2010, [2](#)
- metro_berlin, [13](#)
- us_flights, [18](#)
- us_migration, [18](#)

cali2010, [2](#)

convert_edges, [3](#)

edge_bundle, [3](#)

edge_bundle_force, [4](#), [4](#), [7](#), [8](#), [10](#), [11](#)

edge_bundle_hammer, [4](#), [6](#), [6](#), [8](#), [10](#), [11](#)

edge_bundle_mingle, [4](#), [8](#)

edge_bundle_path, [4](#), [6–8](#), [9](#), [11](#)

edge_bundle_stub, [4](#), [6–8](#), [10](#), [10](#)

flow_tree, [12](#), [17](#)

metro_berlin, [13](#)

metro_multicriteria, [13](#)

tnss_dummies, [15](#)

tnss_smooth, [16](#), [17](#)

tnss_tree, [12](#), [15](#), [16](#), [17](#)

us_flights, [18](#)

us_migration, [18](#)