Package 'packcircles'

November 22, 2022

Type Package

Version 0.3.5

Title Circle Packing

Description Algorithms to find arrangements of non-overlapping circles.

Date 2022-11-22

URL https://github.com/mbedward/packcircles

BugReports https://github.com/mbedward/packcircles/issues

Depends R (>= 3.2)

Imports Rcpp (>= 1.0.0)

Suggests ggiraph (>= 0.8.4), ggplot2, knitr, rmarkdown, lpSolve

VignetteBuilder knitr

LinkingTo Rcpp (>= 1.0.0)

LazyData true

License MIT + file LICENSE

RoxygenNote 7.2.2

Encoding UTF-8

NeedsCompilation yes

Author Michael Bedward [aut, cre],
David Eppstein [aut] (Author of Python code for graph-based circle packing ported to C++ for this package),
Peter Menzel [aut] (Author of C code for progressive circle packing ported to C++ for this package)

Maintainer Michael Bedward <michael.bedward@gmail.com>

Repository CRAN

Date/Publication 2022-11-22 10:00:02 UTC

R topics documented:

`bacteria` *Abundance of bacteria*

Description

Names and abundances of bacterial taxa as measured in a study of biofilms.

Usage

```
bacteria
```

Format

A data frame with 167 rows and 3 variables:

value measured abundance

colour preferred colour for display

label taxon name

`circleGraphLayout` *Find an arrangement of circles satisfying a graph of adjacencies*

Description

Attempts to derive an arrangement of circles satisfying prior conditions for size and adjacency. Unlike the `circleRepelLayout` function, this is a deterministic algorithm. Circles are classified as either internal or external. Viewing the pattern of adjacencies as a triangulated mesh, external circles are those on the boundary. In the version of the algorithm implemented here, the radii of external circles are provided as inputs, while the radii of internal circles are derived as part of the output arrangement.

Usage

```
circleGraphLayout(internal, external)
```

Arguments

`internal`	A list of vectors of circle ID values where, in each vector, the first element is the ID of an internal circle and the remaining elements are the IDs of that circle's neighbours arranged as a cycle. The cycle may be clockwise or anti-clockwise but the same ordering must be used for all vectors.
`external`	A data.frame or matrix of external circle radii, with circle IDs in the first column and radii in the second column.

Details

The `internal` argument specifies circle adjacencies (ie. tangencies). The format is an concise representation of graph edges, and consists of a list of vectors: one per internal circle. In each vector the first element is the ID value of the internal circle and the remaining values are IDs of neighbouring circles, which may be either internal or external.

The `external` argument is a data.frame which specifies the radii of external circles. Internal circle radii should not be specified as they are derived as part of the fitting algorithm. The function will issue an error if any internal circle IDs are present in the `external` data.

Value

A data.frame with columns for circle ID, centre X and Y ordinate, and radius.

The output arrangement as a data.frame with columns for circle ID, centre X and Y ordinates, and radius. For external circles the radius will equal input values.

Note

Please treat this function as experimental.

References

C.R. Collins & K. Stephenson (2003) An algorithm for circle packing. Computational Geometry Theory and Applications 25:233-256.

Examples

```
## Simple example with two internal circles surrounded by
## four external circles. Internal circle IDs are 1 and 2.
internal <- list( c(1, 3, 4, 5), c(2, 3, 4, 6) )

## Uniform radius for external circles
external <- data.frame(id=3:6, radius=1.0)

## Generate the circle packing
packing <- circleGraphLayout(internal, external)
```

`circleLayout` *Arranges circles by iterative pair-wise repulsion within a bounding rectangle*

Description

This function is deprecated and will be removed in a future release. Please use `circleRepelLayout` instead.

Usage

```
circleLayout(xyr, xlim, ylim, maxiter = 1000, wrap = TRUE, weights = 1)
```

Arguments

`xyr`	A 3-column matrix or data frame (centre X, centre Y, radius).
`xlim`	The bounds in the X direction; either a vector for [xmin, xmax) or a single value interpreted as [0, xmax). Alternatively, omitting this argument or passing any of `NULL`, a vector of `NA` or an empty vector will result in unbounded movement in the X direction.
`ylim`	The bounds in the Y direction; either a vector for [ymin, ymax) or a single value interpreted as [0, ymax). Alternatively, omitting this argument or passing any of `NULL`, a vector of `NA` or an empty vector will result in unbounded movement in the Y direction.
`maxiter`	The maximum number of iterations.
`wrap`	Whether to treat the bounding rectangle as a toroid (default `TRUE`). When this is in effect, a circle leaving the bounds on one side re-enters on the opposite side.
`weights`	An optional vector of numeric weights (0 to 1 inclusive) to apply to the distance each circle moves during pair-repulsion. A weight of 0 prevents any movement. A weight of 1 gives the default movement distance. A single value can be supplied for uniform weights. A vector with length less than the number of circles will be silently extended by repeating the final value. Any values outside the range [0, 1] will be clamped to 0 or 1.

Value

A list with components:

layout A 3-column matrix or data.frame (centre x, centre y, radius).

niter Number of iterations performed.

Note

This function assumes that circle sizes are expressed as radii whereas the default for `circleRepelLayout` is area.

See Also

circleRepelLayout

`circleLayoutVertices` *Generate a set of circle vertices suitable for plotting*

Description

Given a matrix or data frame for a circle layout, with columns for centre x-y coordinates and sizes, this function generates a data set of vertices which can then be used with ggplot or base graphics functions.

Usage

```
circleLayoutVertices(
  layout,
  npoints = 25,
  xysizecols = 1:3,
  sizetype = c("radius", "area"),
  idcol = NULL
)
```

Arguments

`layout`	A matrix or data.frame of circle data (x, y, size). May also contain other columns including an optional identifier column.
`npoints`	The number of vertices to generate for each circle.
`xysizecols`	The integer indices or names of columns for the centre X, centre Y and size values. Default is 'c(1,2,3)'.
`sizetype`	The type of size values: either `"radius"` (default) or `"area"`. May be abbreviated.
`idcol`	Optional index or name of column for circle identifiers. These may be numeric or character but must be unique. If not provided, the output circle IDs will be the row numbers of the input circle data.

Value

A data.frame with columns: id, x, y; where id is the unique integer identifier for each circle.

Note

Input sizes are assumed to be radii. This is slightly confusing because the layout functions `circleRepelLayout` and `circleProgressiveLayout` treat their input sizes as areas by default. To be safe, you can always set the `sizetype` argument explicitly for both this function and layout functions.

See Also

circleVertices

Examples

```
xmax <- 100
ymax <- 100
rmin <- 10
rmax <- 20
N <- 20

## Random centre coordinates and radii
layout <- data.frame(id = 1:N,
                     x = runif(N, 0, xmax),
                     y = runif(N, 0, ymax),
                     radius = runif(N, rmin, rmax))

## Get data for circle vertices
verts <- circleLayoutVertices(layout, idcol=1, xysizecols=2:4,
                              sizetype = "radius")

## Not run:
library(ggplot2)

## Draw circles annotated with their IDs
ggplot() +
  geom_polygon(data = verts, aes(x, y, group = id),
               fill = "grey90",
               colour = "black") +

  geom_text(data = layout, aes(x, y, label = id)) +

  coord_equal() +
  theme_bw()

## End(Not run)
```

---

circlePlotData *Generate a set of circle vertices suitable for plotting*

Description

This function is deprecated and will be removed in a future release. Please use `circleLayoutVertices` instead.

Usage

```
circlePlotData(layout, npoints = 25, xyr.cols = 1:3, id.col = NULL)
```

Arguments

layout	A matrix or data.frame of circle data (x, y, radius). May contain other columns, including an optional ID column.
npoints	The number of vertices to generate for each circle.
xyr.cols	Indices or names of columns for x, y, radius (in that order). Default is columns 1-3.
id.col	Optional index or name of column for circle IDs in output. If not provided, the output circle IDs will be the row numbers of the input circle data.

Value

A data.frame with columns: id, x, y; where id is the unique integer identifier for each circle.

See Also

circleLayoutVertices circleVertices

---

circleProgressiveLayout

Progressive layout algorithm

Description

Arranges a set of circles, which are denoted by their sizes, by consecutively placing each circle externally tangent to two previously placed circles while avoiding overlaps.

Usage

```
circleProgressiveLayout(x, sizecol = 1, sizetype = c("area", "radius"))
```

Arguments

x	Either a vector of circle sizes, or a matrix or data frame with one column for circle sizes.
sizecol	The index or name of the column in x for circle sizes. Ignored if x is a vector.
sizetype	The type of size values: either "area" (default) or "radius". May be abbreviated.

Details

Based on an algorithm described in the paper: *Visualization of large hierarchical data by circle packing* by Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. Published in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006, pp. 517-520 doi:10.1145/1124772.1124851

The implementation here was adapted from a version written in C by Peter Menzel: https://github.com/pmenzel/packCircles.

Value

A data frame with columns: x, y, radius. If any of the input size values were non-positive or missing, the corresponding rows of the output data frame will be filled with NAs.

Examples

```
areas <- sample(c(4, 16, 64), 100, rep = TRUE, prob = c(60, 30, 10))
packing <- circleProgressiveLayout(areas)

## Not run:

# Graph the result with ggplot
library(ggplot2)

dat.gg <- circleLayoutVertices(packing)

ggplot(data = dat.gg, aes(x, y, group = id)) +
  geom_polygon(colour = "black", fill = "grey90") +
  coord_equal() +
  theme_void()


## End(Not run)
```

---

circleRemoveOverlaps *Filters a set of circles to remove all overlaps*

Description

Given an initial set of circles, this function identifies a subset of non-overlapping circles using a simple heuristic algorithm. Circle positions remain fixed.

Usage

```
circleRemoveOverlaps(
  x,
  xysizecols = 1:3,
  sizetype = c("area", "radius"),
  tolerance = 1,
 method = c("maxov", "minov", "largest", "smallest", "random", "lparea", "lpnum")
)
```

Arguments

x	A matrix or data frame containing circle x-y centre coordinates and sizes (area or radius).

`xysizecols`	The integer indices or names of the columns in `x` for the centre x-y coordinates and sizes of circles. Default is `c(1,2,3)`.
`sizetype`	The type of size values: either `"area"` (default) or `"radius"`. May be abbreviated.
`tolerance`	Controls the amount of overlap allowed. Set to 1 for simple exclusion of overlaps. Values lower than 1 allow more overlap. Values > 1 have the effect of expanding the influence of circles so that more space is required between them. The input value must be > 0.
`method`	Specifies whether to use linear programming (default) or one of the variants of the heuristic algorithm. Alternatives are: `"maxov"`, `"minov"`, `"largest"`, `"smallest"`, `"random"`, `"lparea"`, `"lpnum"`. See Details for further explanation.

Details

The `method` argument specifies whether to use the heuristic algorithm or linear programming. The following options select the heuristic algorithm and specify how to choose an overlapping circle for rejection at each iteration:

maxov Choose one of the circles with the greatest number of overlaps.

minov Choose one of the circle with the least number of overlaps.

largest Choose one of the largest circles.

smallest Choose one of the smallest circles.

random Choose a circle at random.

At each iteration the number of overlaps is checked for each candidate circle and any non-overlapping circles added to the selected subset. Then a single overlapping circle is chosen, based on the method being used, from among the remainder and marked as rejected. Iterations continue until all circles have been either selected or rejected. The 'maxov' option (default) generally seems to perform best at maximizing the number of circles retained. The other options are provided for comparison and experiment. Beware that some can perform surprisingly poorly, especially 'minov'.

Two further options select linear programming:

lparea Maximise the total area of circles in the subset.

lpnum Maximise the total number of circles in the subset.

The 'lpSolve' package must be installed to use the linear programming options. These options will find an optimal subset, but for anything other than a small number of initial circles the running time can be prohibitive.

Value

A data frame with centre coordinates and radii of selected circles.

Note

This function is experimental and will almost certainly change before the next package release. In particular, it will probably return something other than a data frame.

`circleRepelLayout` *Arranges circles by iterative pair-wise repulsion within a bounding rectangle*

Description

This function takes a set of circles, defined by a data frame of initial centre positions and radii, and uses iterative pair-wise repulsion to try to find a non-overlapping arrangement where all circle centres lie inside a bounding rectangle. If no such arrangement can be found within the specified maximum number of iterations, the last attempt is returned.

Usage

```
circleRepelLayout(
  x,
  xlim,
  ylim,
  xysizecols = c(1, 2, 3),
  sizetype = c("area", "radius"),
  maxiter = 1000,
  wrap = TRUE,
  weights = 1
)
```

Arguments

`x`	Either a vector of circle sizes (areas or radii) or a matrix or data frame with a column of sizes and, optionally, columns for initial x-y coordinates of circle centres.
`xlim`	The bounds in the X direction; either a vector for [xmin, xmax) or a single value interpreted as [0, xmax). Alternatively, omitting this argument or passing any of `NULL`, a vector of `NA` or an empty vector will result in unbounded movement in the X direction.
`ylim`	The bounds in the Y direction; either a vector for [ymin, ymax) or a single value interpreted as [0, ymax). Alternatively, omitting this argument or passing any of `NULL`, a vector of `NA` or an empty vector will result in unbounded movement in the Y direction.
`xysizecols`	The integer indices or names of the columns in `x` for the centre x-y coordinates and sizes of circles. This argument is ignored if `x` is a vector. If `x` is a matrix or data frame but does not contain initial x-y coordinates, this can be indicated as `xysizecols = c(NA, NA, 1)` for example.
`sizetype`	The type of size values: either `"area"` or `"radius"`. May be abbreviated.
`maxiter`	The maximum number of iterations.
`wrap`	Whether to treat the bounding rectangle as a toroid (default `TRUE`). When this is in effect, a circle leaving the bounds on one side re-enters on the opposite side.

`weights` An optional vector of numeric weights (0 to 1 inclusive) to apply to the distance each circle moves during pair-repulsion. A weight of 0 prevents any movement. A weight of 1 gives the default movement distance. A single value can be supplied for uniform weights. A vector with length less than the number of circles will be silently extended by repeating the final value. Any values outside the range [0, 1] will be clamped to 0 or 1.

Details

The algorithm is adapted from a demo written in the Processing language by Sean McCullough (this no longer seems to be available online). Each circle in the input data is compared to those following it. If two circles overlap, they are moved apart such that the distance moved by each is proportional to the radius of the other, loosely simulating inertia. So when a small circle is overlapped by a larger circle, the small circle moves furthest. This process is repeated until no more movement takes place (acceptable layout) or the maximum number of iterations is reached (layout failure).

To avoid edge effects, the bounding rectangle can be treated as a toroid by setting the `wrap` argument to `TRUE`. With this option, a circle moving outside the bounds re-enters at the opposite side.

Value

A list with components:

layout A 3-column matrix or data frame (centre x, centre y, radius).

niter Number of iterations performed.

`circleVertices` *Generate vertex coordinates for a circle*

Description

Generates vertex coordinates for a circle given its centre coordinates and radius.

Usage

```
circleVertices(xc, yc, radius, npoints = 25)
```

Arguments

`xc`	Circle centre X ordinate.
`yc`	Circle centre Y ordinate.
`radius`	Circle radius.
`npoints`	Number of distinct vertices required.

Value

A 2-column matrix of X and Y values. The final row is a copy of the first row to create a closed polygon, so the matrix has `npoints + 1` rows.

See Also

circleLayoutVertices

---

packcircles *packcircles: Simple algorithms for circle packing*

Description

This package provides several algorithms to find non-overlapping arrangements of circles:

circleRepelLayout Arranges circles within a bounding rectangle by pairwise repulsion.

circleProgressiveLayout Arranges circles in an unbounded area by progressive placement. This is a very efficient algorithm that can handle large numbers of circles.

circleGraphLayout Finds an arrangement of circles conforming to a graph specification.

Index