Rgraphviz

October 25, 2011

`AgEdge-class` *Class "AgEdge": A class to describe an edge for a Ragraph object*

Description

This class is used to represent edges for the `Ragraph` class. One can retrieve various pieces of information as well as draw them.

Objects from the Class

Objects can be created by calls of the form `new("AgEdge", ...)`.

Slots

`arrowhead`: Object of class `"character"` The style of arrowhead for this edge.

`arrowsize`: Object of class `"character"` A scale factor for the length of the arrow heads & tails

`arrowtail`: Object of class `"character"` The style of arrowtail for this edge.

`color`: Object of class `"character"` The edge color.

`dir`: Object of class `"character"` The edge direction.

`ep`: Object of class `"xyPoint"` The end point of the edge.

`head`: Object of class `"character"` The head node for this edge.

`lty`: Object of class `"character"` The edge line type.

`lwd`: Object of class `"numeric"` The edge line width.

`sp`: Object of class `"xyPoint"` The starting point of the edge.

`splines`: Object of class `"list"` A list of `BezierCurve` objects

`tail`: Object of class `"character"` The tail node for this edge.

`txtLabel`: Object of class `"character"` The edge text label.

Methods

splines Returns the `splines` slot

sp Returns the `sp` slot

ep Returns the `ep` slot

numSplines Returns the number of splines

getSpline Convenience method to retrieve a specific spline

show Displays a concise description of the object

lines Draws the edge

head Gets the head slot

tail Gets the tail slot

txtLabel Returns any label for this edge

arrowhead Retrieves the `arrowhead` slot.

arrowtail Retrieves the `arrowtail` slot.

arrowsize Retrieves the `arrowsize` slot.

Author(s)

Jeff Gentry

See Also

`Ragraph`, `BezierCurve`, `xyPoint`

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,name="foo")
x <- AgEdge(z)  ## list of AgEdge objects

vv <- x[[1]]
vv
## Demonstrate the methods of this class
splines(vv)
sp(vv)
ep(vv)
numSplines(vv)
getSpline(vv, 1)
head(vv)
tail(vv)
txtLabel(vv)
arrowhead(vv)
arrowtail(vv)
arrowsize(vv)
```

AgNode-class *Class "AgNode": A class to describe a node for a Ragraph object*

Description

This class is used to represent nodes for the `Ragraph` class. One can retrieve various pieces of information as well as draw them.

Objects from the Class

Objects can be created by calls of the form `new("AgNode", ...)`.

Slots

`center`: Object of class `"xyPoint"`: The center point of the node

`name`: Object of class `"character"`: The name of the node, used to reference it

`txtLabel`: Object of class `"AgTextLabel"`: Label for this edge

`height`: Object of class `"integer"`: Height of the node in points

`rWidth`: Object of class `"integer"`: The right half of the node in points.

`lWidth`: Object of class `"integer"`: The left half of the node in points.

`color`: Object of class `"character"`: The drawing color of the node.

`fillcolor`: Object of class `"character"`: The color to fill in the node with.

`shape`: Object of class `"character"`: The shape of the node.

`style`: Object of class `"character"`: The style of the node.

Methods

color `signature(object = "AgNode")`: Retrieves the drawing color for the node.

fillcolor `signature(object = "AgNode")`: Retrieves the color to fill in the node image with.

getNodeCenter `signature(object = "AgNode")`: Returns the center point of the node.

getNodeXY `signature(object = "AgNode")`: Returns the center as a two element list, with the first element containing the 'x' value and the second element containing the 'y' value.

getNodeHeight `signature(object = "AgNode")`: Returns the height of the node.

getNodeLW `signature(object = "AgNode")`: Returns the left width of the node.

getNodeRW `signature(object = "AgNode")`: Returns the right width of the node.

name `signature(object = "AgNode")`: Retrieves the name of the node.

shape `signature(object = "AgNode")`: Returns the shape of the node.

style `signature(object = "AgNode")`: Returns the style of the node.

txtLabel `signature(object = "AgNode")`: Retrieves the node label.

Author(s)

Jeff Gentry

See Also

Ragraph

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,name="foo")
x <- AgNode(z) ## list of AgNode objects
vv <- x[[1]]

## The methods in use
color(vv)
fillcolor(vv)
getNodeCenter(vv)
getNodeXY(vv)
getNodeHeight(vv)
getNodeLW(vv)
getNodeRW(vv)
name(vv)
shape(vv)
style(vv)
txtLabel(vv)
```

AgTextLabel-class *Class "AgTextLabel": Represents a graphviz text label*

Description

This class is used to represent the textlabel object in C from the Graphviz library

Objects from the Class

Objects can be created by calls of the form `new("AgTextLabel", ...)`.

Slots

`labelText`: Object of class `"character"` The actual label text

`labelLoc`: Object of class `"xyPoint"` The location of the label

`labelJust`: Object of class `"character"` The justification of the label

`labelWidth`: Object of class `"integer"` The width of the label

`labelColor`: Object of class `"character"` The color to print the label

`labelFontsize`: Object of class `"numeric"` The font size for the label

Methods

labelText Retrieves the labelText slot

labelLoc Retrieves the labelLoc slot

labelJust Retrieves the labelJust slot

labelWidth Retrieves the labelWidth slot

labelColor Retrieves the labelColor slot

labelFontsize Retrieves the labelFontsize slot

Author(s)

Jeff Gentry

See Also

AgEdge-class, AgNode-class

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)

## Make the labels be the edge weights.  This code is from the vignette
eAttrs <- list()
ew <- edgeWeights(g1)
lw <- unlist(unlist(ew))
toRemove <- removedEdges(g1)
lw <- lw[-toRemove]
names(lw) <- edgeNames(g1)
eAttrs$label <- lw

z <- agopen(g1,"foo", edgeAttrs=eAttrs)
x <- AgEdge(z)  ## list of AgEdge objects
x[[1]]  ## AgEdge
a <- txtLabel(x[[1]])
a ## txtLabel object

labelText(a)
labelLoc(a)
labelJust(a)
labelWidth(a)
labelColor(a)
labelFontsize(a)
```

BezierCurve-class *Class "BezierCurve": A class to describe a Bezier curve*

Description

This class is used to represent a Bezier curve in R, which can then be used for other applications, plotted, etc

Objects from the Class

Objects can be created by calls of the form `new("BezierCurve", ...)`.

Slots

`cPoints`: Object of class `"list"`: A list of `xyPoint` objects, representing control points for the curve

Methods

`cPoints`: Returns the `cPoints` slot

`pointList`: Returns a list of all points having been processed with teh `getPoints` method of `xyPoint`

`bezierPoints`: Returns a matrix giving x & y points (by column) for the complete Bezier curve

`lines`: Draws the Bezier curve

`show`: Provides a concise display of information

Author(s)

Jeff Gentry

See Also

`xyPoint`

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,"foo")
x <- AgEdge(z)  ## list of AgEdge objects
x[[1]]  ## AgEdge
a <- splines(x[[1]])[[1]]
a  ## BezierCurve

cPoints(a)
pointList(a)
bezierPoints(a)
```

`GraphvizAttributes` *Graph Attributes for Rgraphviz*

Description

The following describes the allowable attributes to be used with Rgraphviz. Most of these are sent directly to Graphviz and will influence the plot layout, some of these are only cosmetic and stay in R. Users are referred to the Graphviz web documentation which contains some more detailed information on these attributes (and is our source).

Graph Attributes

General Graph Attributes:

`bgcolor`: Color for the entire canvas. The default is to use `transparent`

`center`: Drawing is centered in the output canvas. This is a boolean value with a default of `FALSE`.

`fontcolor`: Color used for text, defaulting to `black`.

`fontname`: Font used for text. Default is Times Roman.

`fontpath`: Directory used to search for fonts

`fontsize`: Font size, in points, for text. This attribute accepts doubles for values with a default of `14.0` and a minimum value of `1.0`

`label`: Label for the graph. This is a string with some extra escape sequences which can be used. The substring `\N` is replaced by the name of the node, and the substring `\G` by the name of the graph. For graph or cluster attributes, `\G` is replaced by the name of the graph or cluster. For edge attributes, the substring `\N` is replaced by the name of the edge, and the substrings `\T` and `\H` by the names of the tail and head nodes. The default value for nodes is `\N` with everything else defaulting to

`labeljust`: Subgraphs inherit cluster behavior for labeljust. If `r`, the labels are right-justified within the bounding rectangle. If `l` they are left-justified. Otherwise the labels are centered. The default value is `c`.

`labelloc`: Top/bottom placement for graph labels. A value of `t` places the label at the top, `b` places them at the bottom. By default root graph labels go on the bottom and cluster labels go on the top.

`layers`: Specifies a linearly ordered list of layer names attached to the graph. Only those components belonging to the current layer appear. This attribute accepts a `layerList`, which is list of strings separated by the characters defined with the `layersep` attribute which defines layer names and implicitly numbered 1,2,etc. The default is which only uses one layer.

`layersep`: Separator characters used to split the layers into a list of names. The default is colons, tabs or spaces.

`margin`: Set X&Y margins of canvas in inches. Specified using the notation `val,val` where each val is a double.

`orientation`: If "[1L*]*", set graph orientation to landscape. Used only if `rotate` is not defined. Default is the empty string.

`page`: Width & height of output pages, in inches. If set and smaller then the size of the layout, a rectangular array of pages of the specified page size is overlaid on the layout, with origins aligned in the lower-left corner, thereby partitioning the layout into pages. The pages are then produced one at a time in 'pagedir' order. Specified using the notion `val,val` where each val is a double.

`pagedir`: If the `page` attribute is set and applicable, this attribute specifies the order in which the pages are emitted. This is limited to one of the 8 row or column major orders. One of `BL`, `BR`, `TR`, `RB`, `RT`, `LB`, or `LT` specifying the 8 row or column major orders for traversing a rectangular array (the first character corresponding to the major order and the second to the minor order). The default value is `BL`

`quantum`: If 'quantum' > 0.0, node label dimensions will be rounded to integral multiples of the quantum. This attribute is of type double, with a default and minimum value of `0.0`

`ranksep`: In dot, this gives the desired rank separation in inches. If the value contains `equally`, the centers of all the ranks are spaced equally apart. In twopi, specifies the separation of concentric circles. This value is of type double, with a minimum value of `0.02`. In dot, the default value is `0.5` and for twopi it is `1.0`.

`ratio`: Sets the aspect ratio for the drawing. If numeric, defines aspect ratio. If it is 'fill' and 'size' has been set, node positions are scaled such that the final drawing fills exactly the specified size. If 'compress' and the 'size' attribute is set and the graph can not be drawn on a single page, then the drawing is compressed to fit in the given size. If 'auto', the 'page' attribute is set and the graph cannot be drawn on a single page, then 'size' is set to an "ideal" value. The default for this attribute is "fill"

`rotate`: If 90, set drawing orientation to landscape. This attribute accepts integer values and defaults to `0`

`samplepoints`: If the input graph defines the 'vertices' attribute, and output is dot or xdot, this gives the number of points used to represent circles and ellipses. It plays the same role in neato, when adjusting the layout to avoid overlapping nodes. This attribute accepts integer values and defaults to `8`

`size`: Maximum width and height of drawing, in inches. Specified using the notation `val,val` where each val is a double. If not specified and there is a current output device, the graph is scaled to fit into that device. If `size` is not specified and there is no current output device, a plot region of the default size will be opened and the graph scaled to match that.

`start`: Parameter used to determine the initial layout of nodes. By default, nodes are randomly placed in a square whose sides have length (number of nodes). The same seed is always used for the random number generator, so the initial placement is repeatable. If 'start' converts to an integer, this is used as a seed value for the RNG. If 'start' is "regular", the nodes are placed regularly about a circle. Finally if 'start' is defined, and not one of those cases, the current time is used to pick a seed. The default is the empty string.

Dot Only Attributes

`clusterrank`: Mode used for handling clusters. If "local", a subgraph whose name begins with "cluster" is given special treatment. The subgraph is laid out separately, and then integrated as a unit into its parent graph. If the cluster has a 'label' parameter, this label is displayed w/ the cluster. Acceptable values are `local`, `global` or `none` - with `local` being the default.

`compound`: Allow edges between clusters. This is a boolean value with a default of `FALSE`

`concentrate`: Use edge concentrators. This is a boolean value with a default of `FALSE`

`mclimit`: Scale factor used to alter the 'MinQuit' and 'MaxIter' parameters used during crossing minimization. This is a double value with a default of `1.0`

`nodesep`: Minimum space between two adjacent nodes of the same rank, in inches. This is a double value with a default of `0.25` and a minimum of `0.02`

`ordering`: If `out` for a graph and n is a node in G, then edges n->* appear left to right in the order that they were defined.

`rankdir`: Determines if the layout is left-to-right or top-to-bottom. Acceptable values are `LR` and `TB` with the default being the latter.

`remincross`: If `TRUE` and there are multiple clusters, run cross minimization a second time. Default is `FALSE`

`searchsize`: During network simplex, maximum number of edges with negative cut values to search when looking for one with minimum cut value. This is an integer value with a default of `30`

`showboxes`: Debugging feature for postscript output, R only. Not currently implemented.

Neato Only Attributes

`Damping`: Factor damping force motions. On each iteration, a node's movement is limited to this factor of its potential motion. This is a double value with a default of `0.99` and a minimum value of `0.0`

`defaultdist`: Default distance between nodes in separate connected components. Only applicable if `pack=FALSE`. Defaults to `1+(avg. len)*sqrt(|V|)` and has a minimum value of `epsilon`.

`dim`: Set the number of dimensions used for the layout. This is an integer value with a default and minimum value of `2`.

`epsilon`: Terminating condition. If length squared of all energy gradients are less than epsilon, the algorithm stops. This is a double value with a default of `.0001 times the number of nodes`.

`maxiter`: Sets the number of iterations used. This is an integer value with the default of `MAXINT`.

`model`: If `circuit`, use circuit resistance model to compute dissimilarity values, otherwise use shortest path. Defaults to the empty string.

Twopi Only Attributes

`root`: Name of the node to use as the center of the layout. If not defined, will pick the most central node.

Not Dot Attributes

`normalize`: Normalize coordinates of final layout so that the first point is at the origin, then rotate so that the first edge is horizontal. This is a boolean value with a default of `FALSE`.

`overlap`: If `scale`, remove node overlaps by scaling. If `FALSE`, use Voronoi technique, otherwise leave overlaps. Default is the empty string.

`pack`: If "true" or non-negative integer - each connected component is laid out separately and then the graphs are packed tightly. If pack has an integral value, this is used as the size (in points) of a margin around each part; otherwise a default margin of 8 is used. If 'false', the entire graph is laid out together. For twopi layouts, this just sets the margin. Default is `FALSE`

`packmode`: Indicates the granularity and method used for packing. This will automatically turn on `pack`. Acceptable values are `node`, `clust` and `graph`, specifying the granularity of packing connected components when `pack=TRUE`. The default is `node`.

`sep`: Fraction to increase polygons for purposes of determining overlap. This is a double value with a default of `0.01`.

`splines`: Draw edges as splines. This is a boolean value with a default of `TRUE`.

`voro_margin`: Factor to scale up drawing to allow margin for expansion in Voronoi technique. This is a double value with a default of `0.05` and a minimum value of `0.0`

Output Based

`resolution`: Number of pixels per inch on a display, used for SVG output. The default for this attribute is `0.96` and it accepts values of type double.

`stylesheet`: A URL or pathname specifying a XML style sheet, for SVG output.

`truecolor`: Output relies on a truecolor color model, used with bitmap outputs. This is a boolean value with a default of `FALSE`

Misc

`URL`: Hyperlink incorporated into the output. For whole graphs, used as part of an image map. Not currently implemented

`comment`: A device dependent commentary. Not currently implemented.

`nslimit`: Number of iterations in network simplex applications. Used in computing node x coordinates

`nslimit1`: Same as `nslimit` but for ranking nodes

`outputorder`: Specify order in which nodes and edges are drawn. R only

Edge Attributes

General Edge Attributes

`arrowhead`: Shape of the arrowhead. Currently somewhat limited in what can be rendered in R as opposed to what is available via Graphviz in that only `open` and `none` are allowed. The `open` is used by default for directed edges and `none` for undirected edges. R only(?)

`arrowsize`: Multiplicative scale factor for arrowheads, type double. R only (?). This attribute accepts values of type double with a default of `1.0` and minimum of `1.0`.

`arrowtail`: Style of arrow on the tail node of an edge, see `arrowhead`. For directed edges that with bidirectional arrows, `open` is used. R only(?)

`color`: The color of the edge. Defaults to `black`.

`decorate`: If TRUE, attach edge label to edge by a 2-segment polyline, underlining the label, then going to the closest point of the spline. Default is `FALSE`. Currently unimplemented.

`dir`: Edge type drawing, which ends should get the arrowhead. R only(?). For directed graphs, this defaults to `forward` and undirected graphs default to `both`. Other possible values are `both` (arrows in both directions) and `back` (Arrow on the tail only)

`fontcolor`: The color used for text. The default value is `black` in R

`fontname`: Font used for text. Defaults to Times Roman. Currently unimplemented.

`fontsize`: Font size, used for text. Defaults to `14.0` with a minimum value of `1.0`

`headclip`: Head of the edge is clipped to the boundary of the head node, otherwise it goes to the center of the node. This is a boolean value with the default of `TRUE`. Currently unimplemented

`headlabel`: Label for the head of the edge. See `label` in `Graph Attributes` for a description of additional escape sequences. Currently unimplemented.

`headport`: Where on the node to aim the edges, uses `center`, `n`, `s`, `e`, `nw`, `nw`, `se`, and `sw`. The default is `center`.

`label`: The edge label. See `label` in `Graph Attributes` for a description of additional escape sequences.

`labelangle`: Angle in degrees that the label is rotated, as a double. Default is `-25.0` with a minimum value of `-180.0`. Currently unimplemented.

`labeldistance`: Multiplicative scaling factor adjusting the distance that the label is from the node. This is a double value with a default of `1.0` and a minimum value of `0.0`.

`layer`: Specifies the layers that this edge is present, type of `layerRange`. This is specified as a string in the format `layerID` or `layerIDslayerIDslayerID...`, where `s` is a character from the `layersep` attribute. The `layerID` attribute can be `all`, a decimal integer or a layer name. Defaults to the empty string.

`style`: Set line style for the edge. R only. Can be one of `dashed`, `dotted`, `solid`, `invis` and `bold`. Defaults to `solid`.

`tailclip`: Same as `headclip` except for the tail of the edge. Defaults to `TRUE`. Currently unimplemented.

`taillabel`: Same as `headlabel` except for the tail of the edge. Defaults to the empty string. Currently unimplemented.

`weight`: The weight of the edge. This attribute is of type double with a default of `0.75` and a minimum value of `0.01`.

Dot Only Attributes

`constraint`: If FALSE, edge is not used in ranking nodes. Default is `TRUE`.

`lhead`: Logical head of an edge. If `compound` is TRUE, if `lhead` is defined and is the name of a cluster containing the real head, then the edge is clipped to the boundary of the cluster.

`ltail`: Same as `lhead` but for the tail of the edge

`minlen`: Minimum edge length (rank difference between head and tail). This is an integer value with a default of `1` and a minimum of `0`.

`samehead`: Edges with the same head and `samehead` value are aimed at the same point on the head node.

`sametail`: Same as `samehead` but for the tail of the edge.

Neato Only Attributes

`len`: Preferred edge length, in inches. This attribute accepts double values with a default of `1.0`.

Misc

`URL`: Hyperlink incorporated into the output. Not currently supported

`headURL`: URL for the head of the edge. Not currently supported

`headtooltip`: If there's a headURL, annotation for a tooltip. R only, not currently supported

`tailURL`: Same as `headURL` but for the tail of an edge

`tailtooltip`: Same as `headtooltip` but for the head of an edge

`tooltip`: Same as `headtooltip` but for the edge in general

`showboxes`: Debugging feature for postscript. Not currently supported

`comment`: Device dependent comment inserted into the output. Not currently supported

Node Attributes

General Node Attributes

`color`: Basic drawing color for the node, corresponding to the outside edge of the node. The interior of the node is specified with the `fillcolor` attribute. Defaults to `black`.

`distortion`: Distortion factor for `shape=polygon`, positive values cause top to be larger then bottom, negative is opposite. This is a double value with a default of `0.0` and a minimum value of `-100.0`

`fillcolor`: Background color of the node. This defaults to `black`.

`fixedsize`: Use only `width` and `height` attributes, do not expand for the width of the label. This defaults to `TRUE`.

`fontcolor`: Color used for text. This defaults to `black`.

`fontname`: Font used for text. The default of this is Times Roman.

`fontsize`: Size of font for the text. This defaults to `14.0` with a minimum size of `1.0`.

`height`: Height of the node, in inches. This attribute accepts values as doubles with a default of `0.5` and a minimum value of `0.02`.

`label`: Label for the node. See `label` in `Graph Attributes` for an explanation of extra escape sequences.

`layer`: Layers in which the node is present. See `layer` in `Edge Attributes` for an explanation of acceptable inputs.

`peripheries`: Set number of peripheries used in polygonal shapes and cluster boundaries. Note that user-defined shapes are treated as a form box shape, so the default peripheries value is 1 and the user-defined shape will be drawn in a bounding rectangle. Setting peripheries=0 will turn this off. Also, 1 is the maximum peripheries value for clusters. Not currently implemented.

`pos`: Position of the node (For neato layouts, this is the initial position of the node). Specified using the notion `val,val` where each val is a double.

`regular`: Force the polygon to be regular. Defaults to `FALSE`.

`shape`: The shape of the node. Current acceptable values are `circle`, `rectangle`, `rect`, `box` and `ellipse`. The `circle` shape is the default. Note that `box`, `rect` and `rectangle` all correspond to the same actual shape.

`sides`: Number of sides if `shape=polygon`. This is an integer value with a default value of `4` and a minimum value of `0`.

`skew`: Skew factor for `shape=polygon`. Positive values skew the polygon to the right, negative to the left. This is a double value with a default value of `0.0` and a minimum value of `-100.0`.

`style`: Set style for the node boundary. R only. Can be one of `dashed`, `dotted`, `solid`, `invis` and `bold`. Defaults to `solid`.

`width`: Width of the node, in inches. Default is `0.75` with a minimum value of `0.01`

Dot Only Attributes

`group`: If the end points of an edge belong to the same group, ie they have the same group attributes, parameters are set to avoid crossings and keep the edges straight

Neato Only Attributes

`pin`: If TRUE and node has a `pos` attribute on input, neato prevents the node from moving from the input position. The default for this attribute is `FALSE`.

Misc

`tooltip`: Annotated tooltip if URL exists. R only. Currently unsupported

`toplabel`: Label near the top of nodes of shape M*. Currently unsupported

`URL`: Hyperlink incorporated into the output. Not currently supported

`bottomlabel`: Same as `toplabel` but for the bottom of the node

`comment`: Device dependent comment inserted into the output. Not currently supported

`shapefill`: If non-empty, if output is ps or svg and shape is 'espf', taken as a filename containing a device-dependent description of a node's shape. If non-empty for bitmap output (gif, jpg, etc), and shape set to 'custom', taken as the URL for a file containing the bitmap image for the node. For files on the local machine, the URL begins with "file://". For remote files, graphviz must have been configured to use a command such as curl to retrieve the files remotely. Currently unsupported

`z`: Provides z coordinates for the node in a 3D system. Currently unsupported

details

Different attributes are appropriate for different specific graph layout algorithms. Graphviz supports three different layout algorithms, dot, neato and twopi.

There is some tension between attributes that graphviz supports and those that we can support at the R level. Please let us know if there are situations that are not being handled appropriately.

All attributes are passed down to graphviz. However they can be later modified for rendering in R.

Author(s)

Jeff Gentry

References

http://www.graphviz.org/pub/scm/graphviz2/doc/info/attrs.html

See Also

plot.graph, agopen, GraphvizLayouts

GraphvizLayouts *Graphviz Layout Methods*

Description

The following describes the different layout methods that can be used within Rgraphviz. Each layout method has its own particular advantages and disadvantages and can have its own quirks. Currently Rgraphviz supports three different layout methods: dot, twopi and neato.

Details

Portions of the layout descriptions were taken from documents provided at http://www.research.att.com/sw/graphviz. The specific documents are listed in the references section of this page.

The dot layout

The dot algorithm produces a ranked layout of a graph honoring edge directions. It is particularly appropriate for displaying hierarchies or directed acyclic graphs. The basic layout scheme is attributed to Sugiyama et al. The specific algorithm used by dot follows the steps described by Gansner et al.

dot draws a graph in four main phases. Knowing this helps you to understand what kind of layouts dot makes and how you can control them. The layout procedure used by dot relies on the graph being acyclic. Thus, the first step is to break any cycles which occur in the input graph by reversing the internal direction of certain cyclic edges. The next step assigns nodes to discrete ranks or levels. In a top-to-bottom drawing, ranks determine Y coordinates. Edges that span more than one rank are broken into chains of virtual nodes and unit-length edges. The third step orders nodes within ranks to avoid crossings. The fourth step sets X coordnates of nodes to keep edges short, and the final step routes edge splines.

In dot, higher edge weights have the effect of causing edges to be shorter and straighter.

Fine-tuning should be approached cautiously. dot works best when it can makes a layout without much help or interference in its placement of individual nodes and edges. Layouts can be adjusted somewhat by increasing the weight of certain edges, and sometimes even by rearranging the order of nodes and edges in the file. But this can backfire because the layouts are not necessarily stable with respect to changes in the input graph. One last adjustment can invalidate all previous changes and make a very bad drawing.

The neato layout

`neato` is a program that makes layouts of undirected graphs following the filter model of `dot`. Its layout heuristic creates virtual physical models and runs an iterative solver to find low energy configurations. An ideal spring is placed between every pair of nodes such that its length is set to the shortest path distance between the endpoints. The springs push the nodes so their geometric distance in the layout approximates their path distance in the graph.

In `neato`, the edge weight is the strength of the corresponding spring.

As with `dot`, fine-tuning should be approached cautiously, as often small changes can have a drastic effect and create a poor looking layout.

The twopi layout

The radial layout algorithm represented by `twopi` is conceptually the simplest. It takes a node specified as the center of the layout and the root of the generated spanning tree. The remaining nodes are placed on a series of concentric circles about the center, the circle used corresponding to the graph-theoretic distance from the node to the center. Thus, for example, all of the neighbors of the center node are placed on the first circle around the center. The algorithm allocates angular slices to each branch of the induced spanning tree to guarantee enough space for the tree on each ring. It should be obvious from the description that the basic version of the twopi algorithm relies on the graph being connected.

Of great importance to the quality of the layout is the selection of an appropriate center node. By default, the `twopi` will randomly pick one of the nodes that are furthest from a leaf node, where a leaf node is a node of degree `1`. The `root` attribute can be used to manually select a central node for the layout, and users are encouraged to use this attribute to select a node which provides a good quality layout. It often might not be obvious what that node will be, as it will vary from graph to graph, so some experimentation might be required.

As with `dot` and `neato`, fine-tuning should be approached cautiously, as often small changes can have a drastic effect and create a poor looking layout. The `root` node of the layout, as mentioned before, can have a profound effect on the outcome of the layout and care should be taken to select an appropriate one.

The circo layout

The `circo` layout method draws graphs using a circular layout (see Six and Tollis, GD '99 and ALENEX '99, and Kaufmann and Wiese, GD '02.) The tool identifies biconnected components and draws the nodes of the component on a circle. The block-cutpoint tree is then laid out using a recursive radial algorithm. Edge crossings within a circle are minimized by placing as many edges on the circle's perimeter as possible. In particular, if the component is outerplanar, the component will have a planar layout.

If a node belongs to multiple non-trivial biconnected components, the layout puts the node in one of them. By default, this is the first non-trivial component found in the search from the root component.

The fdp layout

The `fdp` layout draws undirected graphs using a spring model similar to `neato`. It relies on a force-directed approach in the spirit of Fruchterman and Reingold. The `fdp` model uses springs only between nodes connected with an edge, and an electrical repulsive force between all pairs of nodes. Also, it achieves a layout by minimizing the forces rather than the energy of the system.

Author(s)

Jeff Gentry

References

http://www.research.att.com/sw/tools/graphviz/dotguide.pdf, http://www.research.att.com/sw/tools/graphviz/neatoguide.pdf, http://www.research.att.com/sw/tools/graphviz/libguide.pdf

See Also

GraphvizAttributes, plot.graph, agopen

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
if (interactive()) {
  op <- par()
  on.exit(par=op)
  par(ask=TRUE)
  plot(g1, "dot")
  plot(g1, "neato")
  plot(g1, "twopi")
}
```

Ragraph-class *Class "Ragraph": A class to handle libgraph representations*

Description

Class Ragraph is used to handle libgraph representations of R graph objects.

Objects from the Class

Objects can be created by calls to the function agopen.

Slots

agraph: Object of class "externalptr": A C based structure containing the libgraph information

laidout: Object of class "logical": Whether or not this graph has been laid out or not.

layoutType: Object of class "character": The layout method used for this object

edgemode: Object of class "character": The edgemode for this graph - "directed" or "undirected"

AgNode: Object of class "list": A list of AgNode objects.

AgEdge: Object of class "list": A list of AgEdge objects.

boundBox: Object of class "boundBox": Defines the bounding box of the plot.

bg: Object of class "character": The background color.

fg: Object of class "character": The foreground color.

Methods

show `signature(object = "Ragraph")`: A brief summary of the contents

agraph `signature(object = "Ragraph")`: Returns the external libgraph pointer

laidout `signature(object = "Ragraph")`: Returns the `laidout` slot

boundBox `signature(object = "Ragraph")`: Returns the bounding box.

AgEdge `signature(object = "Ragraph")`: Returns the edge list.

AgNode `signature(object = "Ragraph")`: Returns the node list.

edgemode `signature(object = "Ragraph")`: Retrieves the edgemode of this object.

layoutType `signature(object = "Ragraph")`: Retrieves the method used for the layout of this graph.

edgeNames `signature(object = "Ragraph")`: Returns a vector of the names of the edges in this graph.

graphDataDefaults `signature(self= "Ragraph")`: Gets default attributes of the given graph.

graphDataDefaults<- `signature(self= "Ragraph", attr="vector", value="vector")`: Sets default attributes of the given graph.

graphData `signature(self= "Ragraph", attr="vector")`: Gets attributes of the given graph.

graphData<- `signature(self= "Ragraph", attr="vector", value="vector")`: Sets attributes of the given graph.

clusterData `signature(self= "Ragraph", cluster="numeric", attr="vector")`: Gets attributes of a cluster for the given graph.

clusterData<- `signature(self= "Ragraph", cluster="numeric", attr="vector", value="vector")`: Sets attributes of a cluster for the given graph.

edgeDataDefaults `signature(self= "Ragraph",attr="missing")`: Gets default attributes of the given edge.

edgeDataDefaults<- `signature(self= "Ragraph", attr="vector", value="vector")`: Sets default attributes of the given edge.

edgeData `signature(self= "Ragraph", from="vector", to="vector", attr="vector")`: Gets attributes of the given edge.

edgeData<- `signature(self= "Ragraph", from="vector", to="vector", attr="vector", value="vector")`: Sets attributes of the given edge.

nodeDataDefaults `signature(self= "Ragraph",attr="missing")`: Gets default attributes of the given node.

nodeDataDefaults<- `signature(self= "Ragraph", attr="vector", value="vector")`: Sets default attributes of the given node.

nodeData `signature(self= "Ragraph", n="vector", attr="vector")`: Gets attributes of the given node.

nodeData<- `signature(self= "Ragraph", n="vector", attr="vector", value="vector")`: Sets attributes of the given node.

getNodeXY `signature(object = "Ragraph")`: Returns a two element list, the first element contains a numerical vector with the 'x' positions of every node in this graph, and the second element contains a numerical vector with the 'y' positions for every node in the graph.

getNodeHeight `signature(object = "Ragraph")`: Returns a vector with the heights of every node in the graph

getNodeLW `signature(object = "Ragraph")`: Returns a vector with the left width of every node in the graph.

getNodeRW `signature(object = "Ragraph")`: Returns a vector with the right width of every node in the graph.

Author(s)

Jeff Gentry and Li Long <li.long@isb-sib.ch>

See Also

`agopen`

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,"foo")
z

## The various methods in action

## These methods are all used to obtain positional information about nodes
getNodeXY(z)
getNodeHeight(z)
getNodeLW(z)
getNodeRW(z)

## Retrieve information about the edges in the graph
edgeNames(z)
edgemode(z)

## These get information about the layout
laidout(z)
layoutType(z)
boundBox(z)

## Used to retrieve the entire list of edges or nodes
AgEdge(z)
AgNode(z)
```

`agopen` *A function to obtain a libgraph object*

Description

This function will read in a `graph` object and create a `Ragraph` object, returning it for use in other functions. The graph represented by the `Ragraph` can be laidout in various formats.

Usage

```
agopen(graph, name, nodes, edges, kind = NULL, layout = TRUE,
       layoutType=c("dot","neato","twopi","circo","fdp"),
       attrs=list(),
       nodeAttrs=list(), edgeAttrs=list(),
       subGList=list(), edgeMode=edgemode(graph),
       recipEdges=c("combined", "distinct"))
```

Arguments

graph	An object of class graphNEL
nodes	A list of pNode objects
edges	A list of pEdge objects
name	The name of the graph
kind	The type of graph
layout	Whether to layout the graph or not
layoutType	Defines the layout engine. Defaults to dot
attrs	A list of graphviz attributes
nodeAttrs	A list of specific node attributes
edgeAttrs	A list of specific edge attributes
subGList	A list describing subgraphs for the graph parameter
edgeMode	Whether the graph is directed or undirected
recipEdges	How to handle reciprocated edges, defaults to combined

Details

graph is from the package graph-class.

The user can specify either the graph parameter and/or a combination of nodes and edges. If either of the latter parameters are not specified then graph must be passed in, and is used in the functions buildNodeList and buildEdgeList (as appropriate - if nodes is passed in but edges is not, only buildEdgeList is called) which are default transformer functions to generate the pNode and pEdge lists for layout.

The edgeMode argument specifies whether the graph is to be laid out with directed or undirected edges. This parameter defaults to the edgemode of the graph argument - note that if graph was not passed in then edgeMode must be supplied.

The kind parameter works as follows:

NULL: Determine the direction of the graph from the graph object. This is the default and the recommended method.

AGRAPH: An undirected graph

AGDIGRAPH: A directed graph

AGRAPHSTRICT: A strict undirected graph

AGDIGRAPHSTRICT: A strict directed graph

Strict graphs do not allow self arcs or multi-edges.

If `layout` is set to `TRUE`, then the `libgraph` routines are used to compute the layout locations for the graph. Otherwise the graph is returned without layout information.

The `subGList` parameter is a list describing any subgraphs, where each element represents a subgraph and is itself a list with up to three elements. The first element, `graph` is required and contains the actual `graph` object for the subgraph. The second element, `cluster` is a logical value indicating if this is a `cluster` or a `subgraph` (a value of `TRUE` indicates a cluster, which is also the default value if this element is not specified). In Graphviz, subgraphs are more of an organizational mechanism, whereas clusters are laid out separately from the main graph and then later inserted. The last element of the list, `attrs` is used if there are any attributes for this subgraph. This is a named vector where the names are the attributes and the elements are the values for those attributes.

For a description of `attrs`, `nodeAttrs` and `edgeAttrs`, see the `Ragraph` man page.

The `recipEdges` argument can be used to specify how to handle reciprocal edges. The default value, `combined` will combine any reciprocated edges into a single edge (and if the graph is directed, will by default place an arrowhead on both ends of the edge), while the other option is `distinct` which will draw to separate edges. Note that in the case of an undirected graph, every edge of a `graphNEL` is going to be reciprocal due to implementation issues.

Value

An object of class `Ragraph`

Author(s)

Jeff Gentry

References

http://www.research.att.com/sw/tools/graphviz/

See Also

`graphLayout`, `Ragraph`, `plot`

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,name="foo")
z
z <- agopen(g1,name="foo",layoutType="neato")
```

agopenSimple *A function to obtain a Ragraph object*

Description

This function will read in a graphNEL object and create a Ragraph object, returning it for use in other functions. The graph represented by the Ragraph can be laidout in various formats.

Usage

```
agopenSimple(graph, name, kind = NULL, edgeMode=edgemode(graph),
       subGList=list(), recipEdges=c("combined", "distinct"))
```

Arguments

graph	An object of class graphNEL
name	The name of the Ragraph
kind	The type of graph
subGList	A list describing subgraphs for the graph parameter
edgeMode	Whether the graph is directed or undirected
recipEdges	How to handle reciprocated edges, defaults to combined, TODO: use this

Details

graph is from the package graph-class.

The edgeMode argument specifies whether the graph is to be laid out with directed or undirected edges. This parameter defaults to the edgemode of the graph argument.

The kind parameter works as follows:

NULL: Determine the direction of the graph from the graph object. This is the default and the recommended method.

AGRAPH: An undirected graph

AGDIGRAPH: A directed graph

AGRAPHSTRICT: A strict undirected graph

AGDIGRAPHSTRICT: A strict directed graph

Strict graphs do not allow self arcs or multi-edges.

The subGList parameter is a list describing any subgraphs, where each element represents a subgraph and is itself a list with up to three elements. The first element, graph is required and contains the actual graph object for the subgraph. The second element, cluster is a logical value indicating if this is a cluster or a subgraph (a value of TRUE indicates a cluster, which is also the default value if this element is not specified). In Graphviz, subgraphs are more of an organizational mechanism, whereas clusters are laid out separately from the main graph and then later inserted. The last element of the list, attrs is used if there are any attributes for this subgraph. This is a named vector where the names are the attributes and the elements are the values for those attributes.

The recipEdges argument can be used to specify how to handle reciprocal edges. The default value, combined will combine any reciprocated edges into a single edge (and if the graph is

directed, will by default place an arrowhead on both ends of the edge), while the other option is `distinct` which will draw to separate edges. Note that in the case of an undirected graph, every edge of a `graphNEL` is going to be reciprocal due to implementation issues.

Value

An object of class `Ragraph`

Author(s)

Li Long <li.long@isb-sib.ch>

References

http://www.research.att.com/sw/tools/graphviz/

See Also

`graphLayout`, `Ragraph`, `plot`

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopenSimple(g1,name="foo")
if(graphvizVersion() >= "2.10") {
  ## This example will only run with Graphviz >= 2.10
  plot(z, "twopi")
}
```

`agwrite` *A function to write a Ragraph object to a file*

Description

This function will take a `Ragraph` object and write it out in DOT format to a file.

Usage

```
agwrite(graph, filename)
```

Arguments

`graph`	An object of class `Ragraph`
`filename`	The output filename

Details

This function is a wrapper to the agwrite() call in Graphviz.

Author(s)

Jeff Gentry

See Also

agopen, agread

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,"foo",layout=FALSE)
agwrite(z,tempfile())
```

---

boundingBox-class *Class "boundingBox": A class to describe the bounding box of a Ragraph*

Description

The boundingBox class is used to describe the dimensons of the bounding box for a laid out Ragraph

Objects from the Class

Objects can be created by calls of the form new("boundingBox", ...).

Slots

botLeft: Object of class "xyPoint" Defines the bottom left point of the bounding box

upRight: Object of class "xyPoint" Defines the upper right point of the bounding box

Methods

botLeft Retrieve the botLeft slot

upRight Retrieve the upRight slot

Author(s)

Jeff Gentry

See Also

Ragraph, graph-class

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,"foo")
x <- boundBox(z)
x

botLeft(x)
upRight(x)
```

---

buildNodeList *A function to build lists of node and edge objects*

Description

These functions can be used to generate lists of pNode and pEdge objects from an object of class graph. These lists can then be sent to Graphviz to initialize and layout the graph for plotting.

Usage

```
buildNodeList(graph, nodeAttrs = list(), subGList=list(), defAttrs=list())
buildEdgeList(graph, recipEdges=c("combined", "distinct"),
              edgeAttrs = list(), subGList=list(), defAttrs=list())
```

Arguments

graph	An object of class graph
nodeAttrs	A list of attributes for specific nodes
edgeAttrs	A list of attributes for specific edges
subGList	A list of any subgraphs to be used in Graphviz
recipEdges	How to deal with reciprocated edges
defAttrs	A list of attributes used to specify defaults.

Details

These functions will take either the nodes or the edges of the specified graph and generate a list of either pNode or pEdge objects.

The recipEdges argument can be used to specify how to handle reciprocal edges. The default value, combined will combine any reciprocated edges into a single edge (and if the graph is directed, will by default place an arrowhead on both ends of the edge), while the other option is distinct which will draw to separate edges. Note that in the case of an undirected graph, every edge of a graphNEL is going to be reciprocal due to implementation issues.

The nodeAttrs and edgeAttrs attribute lists are to be used for cases where one wants to set an attribute on a node or an edge that is not the default. In both cases, these are lists with the names of the elements corresponding to a particular attribute and the elements containing a named vector - the names of the vector are names of either node or edge objects and the values in the vector are the values for this attribute.

Note that with the edgeAttrs list, the name of the edges are in a particular format where an edge between x and y is named x~y. Note that even in an undirected graph that x~y is not the same as y~x - the name must be in the same order that the edge was defined as having.

The subGraph argument can be used to specify a list of subgraphs that one wants to use for this plot. The buildXXXList functions will determine if a particular node or edge is in one of the subgraphs and note that in the object.

The defAttrs list is a list used to specify any default values that one wishes to use. The element names corresponde to the attribute and the value is the default for that particular attribute.

If there is no default specified in defAttrs for an attribute declared in nodeAttrs or edgeAttrs, then the latter must have a value for every node or edge in the graph. Otherwise, if a default is supplied, that value is used for any node or edge not explicitly defined for a particular attribute.

Value

A list of class pNode or pEdge objects.

Author(s)

Jeff Gentry

See Also

agopen, plot.graph, pNode,pEdge

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)

z <- buildEdgeList(g1)
x <- buildNodeList(g1)
```

---

clusterData-methods

Get and set attributes for a cluster of an Ragraph object

Description

Attributes of a graph can be accessed using clusterData. There's no default attributes for clusters. The attributes must be defined using graphDataDefaults.

Usage

```
clusterData(self, cluster, attr)
clusterData(self, cluster, attr) <- value
```

Arguments

self	A Ragraph-class instance
cluster	cluster number
attr	A character vector of length one specifying the name of a cluster attribute
value	A character vector to store as the attribute value

Author(s)

Li Long <li.long@isb-sib.ch>

Examples

```
library(graph)
library(Rgraphviz)

g1_gz <- gzfile(system.file("GXL/graphExample-01.gxl.gz",package="graph"), open="rb")
g11_gz <- gzfile(system.file("GXL/graphExample-11.gxl.gz",package="graph"), open="rb")
g1 <- fromGXL(g1_gz)
g11 <- fromGXL(g11_gz)
g1_11 <- join(g1, g11)
sgl <- vector(mode="list", length=2)
sgl[[1]] <- list(graph=g1, cluster=TRUE)
sgl[[2]] <- list(graph=g11, cluster=TRUE)
ng <- agopenSimple(g1_11, "tmpsg", subGList=sgl)
clusterData(ng, 1, c("bgcolor")) <- c("blue")
clusterData(ng, 2, c("bgcolor")) <- c("red")
toFile(ng, layoutType="dot", filename="g1_11_dot.ps", fileType="ps")
```

---

getDefaultAttrs *Functions to generate and check global attribute lists*

Description

The getDefaultAttrs function can be used to generate a default global attribute list for Graphviz. The checkAttrs function can be used to verify that such a list is valid for use.

Usage

```
getDefaultAttrs(curAttrs = list(), layoutType = c("dot", "neato",
                  "twopi","circo","fdp"))
checkAttrs(attrs)
```

Arguments

curAttrs	Any attributes currently defined
layoutType	The layout method being used
attrs	An attribute list of graphviz attributes

Details

The `getDefaultAttrs` function generates a four element list (elements being "graph", "cluster", "node" and "edge"). Contained in each is another list where the element names correspond to attributes and the value is the value for that attribute. This list can be used to set global attributes in Graphviz, and the exact list returned by `getDefaultAttrs` represents the values that are used as basic defaults.

The `checkAttrs` function can be used to verify that a global attribute list is properly formed.

Author(s)

Jeff Gentry

See Also

`agopen`, `plot.graph`

Examples

```
z <- getDefaultAttrs()
checkAttrs(z)
```

---

graphData-methods *Get and set attributes of an Ragraph object*

Description

Attributes of a graph can be accessed using `graphData`. The attributes could be defined using `graphDataDefaults`.

Usage

```
graphData(self, attr)
graphData(self, attr) <- value
```

Arguments

`self`	A `Ragraph-class` instance
`attr`	A `character` vector of length one specifying the name of a graph attribute
`value`	A `character` vector to store as the attribute values

Author(s)

Li Long <li.long@isb-sib.ch>

graphDataDefaults-methods *Get and set default attributes for an Ragraph*

Description

Get/Set default values for attributes associated with a graph.

Usage

```
graphDataDefaults(self)
graphDataDefaults(self, attr) <- value
```

Arguments

self	A Ragraph-class instance
attr	A character value giving the name of the attribute
value	A character value as the default value for the specified attribute

Author(s)

Li Long <li.long@isb-sib.ch>

graphLayout *A function to layout graph locations*

Description

This function will take an object of class Ragraph and will perform a libgraph layout on the graph locations.

Usage

```
graphLayout(graph, layoutType=graph@layoutType)
```

Arguments

graph	An object of type Ragraph
layoutType	layout algorithm to use

Details

If the graph has already been laid out, this function merely returns its parameter. Otherwise, it will perform a libgraph layout and retrieve the appropriate location information.

Value

A laid out object of type Ragraph.

Author(s)

Jeff Gentry

See Also

agopen

Examples

```
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
z <- agopen(g1,"foo",layout=FALSE)
x <- z
a <- graphLayout(z)
```

---

graphvizVersion *A function to determine graphviz library version*

Description

This function will query the graphviz libraries that the package was built against and report what version of Graphviz is being used.

Usage

```
graphvizVersion()
```

Value

A list with two elements, each of class `numeric_version`. The first element named `installed_version` represents the version of Graphviz that is being used by the package. The second element named `build_version` represents the version of Graphviz that was used to build the package. A mismatch between these two versions may indicate problems.

Author(s)

Jeff Gentry, modified by Kasper Daniel Hansen

Examples

```
graphvizVersion()
```

`layoutGraph` *A function to compute layouts of graph objects*

Description

This is a wrapper to layout graph objects using arbitrary layout engines. The default engine (and so far the only implemented engine) is ATT's Graphviz.

Usage

```
layoutGraph(x, layoutFun = layoutGraphviz, ...)
```

Arguments

`x`	A graph object
`layoutFun`	A function that performs the graph layout and returns a graph object with all necessary rendering information
`...`	Further arguments that are passed to `layoutFun`

Details

Layout of a graph and its rendering are two separate processes. `layoutGraph` provides an API to use an arbitrary algorithm for the layout. This is archived by abstraction of the layout process into a separate function (`layoutFun`) with well-defined inputs and outputs. The only requirements on the `layoutFun` are to accept a graph object as input and to return a valid graph object with all the necessary rendering information stored in its `renderInfo` slot. This information comprises

for nodes:

nodeX, nodeY the locations of the nodes, in the coordinate system defined by `bbox` (see below).

lWidth, rWidth the width components of the nodes, `lWidth+rWidth=total width`.

height the heights of the nodes.

labelX, labelY node label locations.

labelJust the justification of the node labels.

label node label text.

shape the node shape. Valid values are `box`, `rectangle`, `ellipse`, `plaintext`, `circle` and `triangle`.

for edges:

splines representation of the edge splines as a list of `BezierCurve` objects.

labelX, labelY edge label locations.

label edge label text.

arrowhead, arrowtail some of Graphviz's arrow shapes are supported. Currently they are: `open`, `normal`, `dot`, `odot`, `box`, `obox`, `tee`, `diamond`, `odiamond` and `none`. In addition, a user-defined function can be passed which needs to be able to deal with 4 arguments: A list of xy coordinates for the center of the arrowhead, and the graphical parameters `col`, `lwd` and `lty`.

direction The edge direction. The special value `both` is used when reciprocrated edges are to be collapsed.

To indicate that this information has been added to the graph, the graph plotting function should also set the laidout flag in the `graphData` slot to `TRUE` and add the bounding box information (i.e., the coordinate system in which the graph is laid out) in the format of a two-by-two matrix as item `bbox` in the `graphData` slot.

AT&T's `Graphviz` is the default layout algorithm to use when `layoutGraph` is called without a specific `layoutFun` function. See `agopen` for details about how to tweak `Graphviz` and the valid arguments that can be passed on through `....` The most common ones to set in this context might be `layoutType`, which controls the type of layout to compute and the `nodeAttrs` and `edgeAttrs` arguments, which control the fine-tuning of nodes and edges.

Value

An object inheriting from class `graph`

Note

Please note that the layout needs to be recomputed whenever attributes are changed which are bound to affect the position of nodes or edges. This is for instance the case for the `arrowhead` and `arrowtail` parameters.

Author(s)

Florian Hahne, Deepayan Sarkar

See Also

`renderGraph`, `graph.par`, `nodeRenderInfo`, `edgeRenderInfo`, `agopen`,

Examples

```
library(graph)
set.seed(123)
V <- letters[1:5]
M <- 1:2
g1 <- randomGraph(V, M, 0.5)
edgemode(g1) <- "directed"
x <- layoutGraph(g1)
renderGraph(x)

## one of Graphviz's additional layout algorithms
x <- layoutGraph(g1, layoutType="neato")
renderGraph(x)

## some tweaks to Graphviz's node and edge attributes,
## including a user-defined arrowhead and node shape functions.
myArrows <- function(x, ...)
{
for(i in 1:4)
points(x,cex=i)
}

myNode <- function(x, col, fill, ...)
```

```
symbols(x=mean(x[,1]), y=mean(x[,2]), thermometers=cbind(.5, 1,
runif(1)), inches=0.5,
fg=col, bg=fill, add=TRUE)

eAtt <- list(arrowhead=c("a~b"=myArrows, "b~d"="odiamond", "d~e"="tee"))
nAtt <- list(shape=c(d="box", c="ellipse", a=myNode))
edgemode(g1) <- "directed"
x <- layoutGraph(g1, edgeAttrs=eAtt, nodeAttrs=nAtt, layoutType="neato")
renderGraph(x)
```

`makeNodeAttrs` *make a list of character vectors that can be used as a value for the*

Description

make a list of character vectors that can be used as a value for the nodeAttrs argument in agopen

Usage

```
makeNodeAttrs(g, label=nodes(g), shape="ellipse", fillcolor="#e0e0e0", ...)
```

Arguments

`g`	graph
`label`	character of length either `1` or `numnodes(g)`. If the length is `1`, the value is recycled.
`shape`	character of length either `1` or `numnodes(g)`
`fillcolor`	character of length either `1` or `numnodes(g)`
`...`	further named arguments that are character vectors of length either 1 or numNodes(g)

Details

This function is trivial but convenient.

Value

A list of named character vectors, each of which with length `numNodes(g)`.

Author(s)

Wolfgang Huber <huber@ebi.ac.uk>

Examples

```
library(graph)
g = randomEGraph(letters[1:10], p=0.2)
makeNodeAttrs(g)
```

pEdge-class *Class "pEdge": A class to represent an edge*

Description

This class is used to represent all necessary information to plot an edge in Graphviz

Details

The `attrs` slot is a named list, where the names correspond to attributes and the values in the list correspond to the value for that element's attribute.

The `subG` slot describes which subgraph this edge is a part of. A value of `0` implies that the edge is not a member of any subgraph.

Objects from the Class

Objects can be created by calls of the form `new("pEdge", ...)`.

Slots

`from`**:** Object of class `"character"`: The name of the node on the tail of this edge.

`to`**:** Object of class `"character"`: The name of the node on the head of this edge.

`attrs`**:** Object of class `"list"`: A list of attributes specific to this edge.

`subG`**:** Object of class `"integer"`: Which subgraph this edge is a part of.

Methods

from `signature(object = "pEdge")`: Retrieves the `from` slot of this edge

to `signature(object = "pEdge")`: Retrieves the `to` slot of this edge

Author(s)

R. Gentleman and Jeff Gentry

See Also

`pNode`, `agopen`, `buildEdgeList`

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)

z <- buildEdgeList(g1)
vv <- z[[1]]   ## Object of type pEdge

vv
from(vv)
to(vv)
```

pNode-class *Class "pNode": A class to plot nodes*

Description

This class is used to transfer information to Graphviz that is necessary to represent and plot a node.

Details

The attrs slot is a named list, where the names correspond to attributes and the values in the list correspond to the value for that element's attribute.

The subG slot describes which subgraph this node is a part of. A value of 0 implies that the node is not a member of any subgraph.

Objects from the Class

Objects can be created by calls of the form new("pNode", ...).

Slots

name: Object of class "character": The name of the node, used to reference the node.

attrs: Object of class "list": A list of attributes specific to this node.

subG: Object of class "integer": Which subgraph this node is a part of.

Methods

name signature(object = "pNode"): Retrieves the name slot of the object.

Author(s)

R. Gentleman and Jeff Gentry

See Also

pEdge, agopen, buildNodeList

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)

z <- buildNodeList(g1)
z[[1]]   ## Object of type pNode

name(z[[1]])
```

pieGlyph *A function to plot pie graphs as a glyph*

Description

This function allows the user to plot a pie graph at a specified x/y location in a plotting region.

Usage

```
pieGlyph(x, xpos, ypos, labels = names(x), edges = 200, radius = 0.8, density =
```

Arguments

xpos	The x location of the glyph
ypos	The Y location of the glyph
x	a vector of positive quantities. The values in x are displayed as the areas of pie slices.
labels	a vector of character strings giving names for the slices. For empty or NA labels, no pointing line is drawn either.
edges	the circular outline of the pie is approximated by a polygon with this many edges.
radius	the pie is drawn centered in a square box whose sides range from -1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.
density	the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise).
col	a vector of colors to be used in filling or shading the slices. If missing a set of 6 pastel colours is used, unless density is specified when par("fg") is used.
border, lty	(possibly vectors) arguments passed to polygon which draws each slice.
main	an overall title for the plot.
...	graphical parameters can be given as arguments to pie. They will affect the main title and labels only.

Author(s)

R. Gentleman, F. Sim

See Also

pie

Examples

```
plot(1:10, col="white")
pieGlyph(1:20, 5, 5)
```

plot-methods *Plot a graph object - methods*

Description

A plot method for graph objects.

Usage

```
'graph': plot(x, y, ..., subGList=list(), attrs=list(), nodeAttrs=list(),
edgeAttrs=list(), xlab="", ylab="", main=NULL, sub=sub, recipEdges=c("combined",
"distinct"))

'Ragraph': plot(x, y, ..., xlab="", ylab="", main=NULL, sub=sub, drawNode=drawA
nodeAttrs=list(), edgeAttrs=list())
```

arguments

x The graph object to plot

y The layout method to use: One of dot, neato, twopi, circo, and fdp. The default is dot

subGList A list of subgraphs taken from the primary graph object to be plotted. If provided, these subgraphs will be clustered visually. If not provided, no clusters will be used.

attrs A list of Graphviz attributes to be sent to the layout engine

nodeAttrs A list of attributes for specific nodes

edgeAttrs A list of attributes for specific edges

xlab Label for the x axis of the plot

ylab Label for the y axis of the plot

main Main label for the plot

sub Subtitle for the plot

drawNode Function to draw the nodes. The default is drawAgNode

recipEdges Determines how to draw reciprocating edges. See agopen

... General commands to be sent to plot

details

The first plot method in the usage section corresponds to the graph class in the graph package. It will convert the graph object into an object of class Ragraph by using Graphviz to perform the layout. Then it will call the plot method for Ragraph. The plot.graph method is nothing more then a wrapper that calls agopen and plot.Ragraph.

The second plot method in the usage section is for the Ragraph class, which describes a Graphviz structure of a graph. This method will extract necessary information from the object and use it to plot the graph.

Users can specify arbitrary drawing functions for the nodes of the Ragraph with the drawNode argument, although caution is urged. The default drawing function for all nodes is drawAgNode, which will draw a basic circle, ellipse or rectangle according to the layout specifications for each node. If supplying a custom function, users are encouraged to look at the code of this function for a more clear picture of the information required to properly draw a node. Users can specify either

one custom function to be used for all nodes or a list (where length is equal to the nubmer of nodes) of functions where the Nth element in the list provides the drawing function for the Nth node, and every function will take four parameters - the first is an object of class AgNode representing the node itself and the second is an object of class xyPoint representing the upper right corner of the Graphviz plotting region (where the lower left is 0,0). The third parameter, attrs is a node attribute list and represents post-layout attribute changes where the user wants to override values present in the layout. The last argument, radConv is a divisor to the radius and is used to convert from Graphviz units to R plotting units. Outside of the first argument, the rest of these (particularly radConv which generally shouldn't be specifically set) do not need to be set by the user, but any drawing function must have them as parameters.

The attrs list requires a particular format. It must be of length 3 with names graph, node, and edge. Each of these elements themselves are lists - such that an element of graph corresponds to a graph element. A full listing of attributes and their possible settings is available at http://www.research.att.com/~erg/graphviz/info/attrs.html. All attribute values should be entered as character strings (even if the requested value is to be otherwise).

The nodeAttrs list is used to specify attributes on a particular node, instead of for all nodes. The format of this list is such that the elements correspond to attributes (the name of the element is used to note which attribute) and each element contains a named vector. The names of the vector denote which nodes are having this attribute set and the values in the vector specify the value.

The edgeAttrs list is identical in format to nodeAttrs. However, the name of the edges is in a particular format where an edge between x and y is named x~y. Note that even in an undirected graph that x~y is not the same as y~x - the name must be in the same order that the edge was defined as having.

value

The Ragraph object used for plotting

author

Jeff Gentry

seealso

graphNEL-class, graph-class

Examples

```
#  WHY DOES THIS NOT WORK IN CHECK?
#  V <- letters[1:10]
#  M <- 1:4
#  g1 <- randomGraph(V, M, .2)
#  plot(g1)
```

---

removedEdges *A Function To List Removed Edges*

Description

This function can be used to retrieve a numerical vector which will describe which edges in a graph would be removed if recipEdges is set to combined during plotting.

Usage

```
removedEdges(graph)
```

Arguments

graph	An object of class graph or Ragraph, the graph to perform this operation on

Details

This function will simply detect which (if any) edges in a graph would be removed during combination of reciprocated edges.

Value

A numerical vector, where the values correspond to removed edges.

Author(s)

Jeff Gentry

See Also

edgeNames, agopen, buildEdgeList

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)
removedEdges(g1)
```

---

renderGraph *Render a laid out graph object*

Description

This method uses the renderInfo slot of a graph object to render it on a plotting device. The graph must have been laid out using the layoutGraph function before.

Usage

```
## S4 method for signature 'graph'
renderGraph(x, ..., drawNodes="renderNodes", drawEdges=renderEdges, graph.pars=l
```

Arguments

x	An object derived from class graph
drawNodes	A function that is used for the node rendering. The details of its API are still undecided, so far the input to the function is the (laid out) graph object. Defaults to renderNodes, which is not exported in the name space (type Rgraphviz:::renderNodes to see the function definition). This default function knows how to draw node shapes of type box, rectangle, ellipse, plaintext, circle and triangle. In addition, an arbitrary user-defined function can be passed on to draw the node. This function needs to be able to deal with the following arguments: a two-by-two matrix of the bounding box for the respective node, and labelX, labelY, fill, col, lwd, lty, textCol, style, label and fontsize, which are all defined by the layout algorithm or are graphical nodeRenderInfo parameters.
drawEdges	A function that is used for the edge rendering. Defaults to renderEdges. Currently, this function can draw different types of arrowheads: open, normal, dot, odot, box, obox, tee, diamond, odiamond and none. In addition, a user-defined function can be passed as arrowhead or arrowtail parameters which needs to be able to deal with 4 arguments: A list of xy coordinates for the center of the arrowhead, and the graphical parameters col, lwd and lty.
graph.pars	A list of rendering parameters to use as defaults. Parameters that have been explicitely set using nodeRenderInfo, edgeRenderInfo or graphRenderInfo take precendence. If not explicitely supplied, the value of a call to graph.par is used. This design allows to set session-wide defaults.
...	further arguments

Details

This method can render graph objects that have previously been laid out using the function layoutGraph. The details for user defined node drawing remain to be decided.

Value

An object derived from class graph with information about the coordinates of the nodes in the coordinate system of the plotting device added to the renderInfo slot.

Author(s)

Florian Hahne

See Also

layoutGraph, link[graph:renderInfo-class]{nodeRenderInfo}, link[graph:renderInfo-class]{edgeRenderInfo}, link[graph:renderInfo-class]{graphRenderInfo},

Examples

```
library(graph)
set.seed(123)
V <- letters[1:5]
M <- 1:2
g1 <- randomGraph(V, M, 0.5)
```

```
edgemode(g1) <- "directed"
x <- layoutGraph(g1)
renderGraph(x)

## one of Graphviz's additional layout algorithms
x <- layoutGraph(g1, layoutType="neato")
renderGraph(x)

## some tweaks to Graphviz's node and edge attributes,
## including a user-defined arrowhead and node shape functions.
myArrows <- function(x, ...)
{
for(i in 1:4)
points(x,cex=i)
}

myNode <- function(x, col, fill, ...)
symbols(x=mean(x[,1]), y=mean(x[,2]), thermometers=cbind(.5, 1,
runif(1)), inches=0.5,
fg=col, bg=fill, add=TRUE)

eAtt <- list(arrowhead=c("a~b"=myArrows, "b~d"="odiamond", "d~e"="tee"))
nAtt <- list(shape=c(d="box", c="ellipse", a=myNode))
edgemode(g1) <- "directed"
x <- layoutGraph(g1, edgeAttrs=eAtt, nodeAttrs=nAtt, layoutType="neato")
renderGraph(x)
```

`toDot-methods` *A Generic For Converting Objects To Dot*

Description

This generic is used to convert objects of varying classes to the Dot language. Currently, only the `graph` class is supported.

Usage

```
toDot(graph, filename, ...)
```

Arguments

`graph`	The `graph` to output to Dot
`filename`	The name of the file to output to.
`...`	Any arguments to pass on to `agopen`

details

The method defined for `graph` objects is a convenience wrapper around `agopen` and `agwrite` in that order. It will take an object of class `graph` (or one of its subclasses), call `agopen` (any extra arguments besides the graph and the `name` parameter should be passed in via `...`) and then write the resulting information via `agwrite` in the file specified by `filename`.

author

Jeff Gentry

seealso

agopen, agwrite, graph-class

Examples

```
set.seed(123)
V <- letters[1:10]
M <- 1:4
g1 <- randomGraph(V, M, .2)

nAttrs <- list()
eAttrs <- list()
nAttrs$label <- c(a="lab1", b="lab2", g="lab3", d="lab4")
eAttrs$label <- c("a~h"="test", "c~h"="test2")
nAttrs$color <- c(a="red", b="red", g="green", d="blue")
eAttrs$color <- c("a~d"="blue", "c~h"="purple")

toDot(g1, tempfile(), nodeAttrs=nAttrs, edgeAttrs=eAttrs)
```

---

toFile *Render a graph in a file with given format*

Description

Render a graph in a file with given format

Usage

```
toFile(graph,
          layoutType=c("dot","neato","twopi","circo","fdp"),
          filename,
          fileType=c("canon", "dot", "xdot", "dia", "fig",
                     "gd", "gd2", "gif", "hpgl", "imap", "cmapx",
                     "ismap", "mif", "mp", "pcl", "pdf", "pic",
                     "plain", "plain-ext", "png", "ps", "ps2",
                     "svg", "svgz", "vrml", "vtx", "wbmp"))
```

Arguments

graph	an instance of the Ragraph class
layoutType	which layout algorithm to use
filename	output file name
fileType	output file type

Details

This function takes a given Ragraph, does the chosen layout, then renders the output to an external file. Users could view the output file with corresponding viewer.

Value

`toFile` returns NULL after writing to a file.

Author(s)

Li Long <li.long@isb-sib.ch>

References

Rgraphviz by E. Ganssner, S. North, www.graphviz.org

Examples

```
library("graph")
library("Rgraphviz")

g1_gz <- gzfile(system.file("GXL/graphExample-01.gxl.gz",package="graph"), open="rb")
g1 <- fromGXL(g1_gz)
ag <- agopen(g1, name="test")

toFile(ag, layoutType="dot", filename="g1_dot.svg", fileType="svg")
toFile(ag, layoutType="neato", filename="g1_neato.ps", fileType="ps")
toFile(ag, layoutType="twopi", filename="g1_twopi.svg", fileType="svg")
toFile(ag, layoutType="circo", filename="g1_circo.png", fileType="png")
```

---

xyPoint-class *Class "xyPoint": A class to represent a X/Y coordinate.*

Description

This class is used to describe a coordinate in 2-dimensional (X/Y) space

Objects from the Class

Objects can be created by calls of the form `new("xyPoint", ...)`.

Slots

`x`**:** Object of class `"numeric"` The x coordinate

`y`**:** Object of class `"numeric"` The y coordinate

Methods

getX Returns the value stored in the `x` slot

getY Returns the value stored in the `y` slot

getPoints Returns a vector of two numerical values representing the `x` and `y` positions

show Display information about the object in a concise fashion

Author(s)

Jeff Gentry

Examples

```
z <- new("xyPoint", x=150, y=30)
z
getPoints(z)

getX(z)
getY(z)
```

Index