# How To use the *hyperdraw* package

October 17, 2016

## 1 Introduction

The *hyperdraw* package provides a particular style of visualization for hypergraphs.

A hypergraph differs from a normal graph in that each edge of a hypergraph may connect more than two nodes together (in a normal graph, an edge may only connect two nodes), which can present some difficulties for drawing the edges of a hypergraph.

This package provides an algorithm for rendering hypergraphs.

## 2 Getting Started

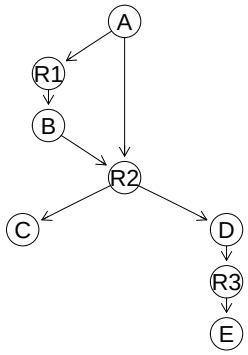
Visualizing any sort of graph involves three steps:

1. create a representation of the graph.
2. arrange the graph layout.
3. render the graph layout.

The *hyperdraw* package requires a specific representation of a hypergraph, implemented by the *graphBPH* class, which is based on the *graphNEL* class from the *graph* package. The hypergraph is specified as a vector of node names, plus a list of edges for each node, *but* (unlike a standard *graphNEL* graph) the nodes are divided into two sets - normal nodes and **edge-nodes** - *and* all of the edges must connect a normal node to an edge-node. The edges of the graph must also be "directed".

Here is a simple example of a *graphNEL* object that satisfies these criteria (an *Rgraphviz* rendering of the graph is shown to the right) ...

```
> nodes <- c(LETTERS[1:5], paste("R", 1:3, sep=""))
> testgnel <- new("graphNEL",
+                 nodes=nodes,
+                 edgeL=list(
+                   A=list(edges=c("R1", "R2")),
+                   B=list(edges="R2"),
+                   C=list(),
+                   D=list(edges="R3"),
+                   E=list(),
+                   R1=list(edges="B"),
+                   R2=list(edges=c("C", "D")),
+                   R3=list(edges="E")),
+                 edgemode="directed")
>
```



The following code converts this into a *graphBPH* object where the normal nodes are the nodes A, B, C, D, and E and the edge-nodes are R1, R2, and R3. The `edgeNodePattern` argument is a regular expression that is used to identify edge-nodes in the graph.

```
> testbph <- graphBPH(testgnel, edgeNodePattern="^R")
>
```

This represents a hypergraph with an edge that connects A to B, an edge that connects A *and* B to C *and* D, and an edge that connects D to E. The edge-nodes represent the edges of the hpyergraph. This is referred to as a "bipartite" representation of the hypergraph.

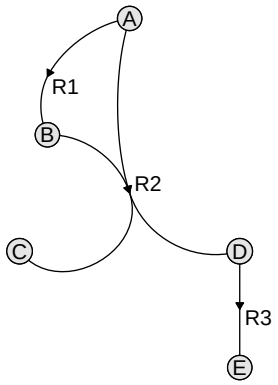
It is also possible to produce a *graphBPH* object from a *Hypergraph* object (from the *hypergraph* package). The following code demonstrates the *Hypergraph* version of the simple *graphBPH* example that we are working with (note that the hyperedges of the *Hypergraph* must all be *DirectedHyperedge* objects).

```
> library(hypergraph)

> dh1 <- DirectedHyperedge("A", "B", "R1")
> dh2 <- DirectedHyperedge(c("A", "B"), c("C", "D"), "R2")
> dh3 <- DirectedHyperedge("D", "E", "R3")
> hg <- Hypergraph(LETTERS[1:5], list(dh1, dh2, dh3))
> hgbph <- graphBPH(hg)
```

The layout and rendering of the hypergraph can be performed in a single step as follows (result shown beside the code) ...

```
> plot(testbph)
>
```

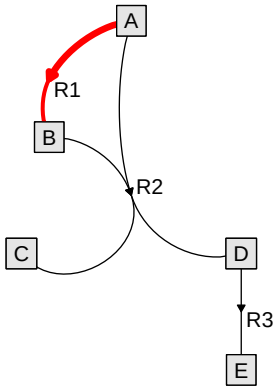



Alternatively, the graph can be laid out as a separate step, as in the following code ...

```
> testrabph <- graphLayout(testbph)
>
```

... which is useful if you want to specify details about how the graph is to be rendered before the rendering step, as in the following code (some edges are made thicker and red and the nodes are made bigger and box-shaped) ...

```
> edgeDataDefaults(testrabph, "lwd") <- 1
> edgeData(testrabph, c("A", "R1"),
+                     c("R1", "B"), "lwd") <- c("5", "3")
> edgeDataDefaults(testrabph, "color") <- "black"
> edgeData(testrabph, c("A", "R1"),
+                     c("R1", "B"), "color") <- "red"
> nodeDataDefaults(testrabph, "margin") <- 'unit(3, "mm")'
> nodeDataDefaults(testrabph, "shape") <- "box"
> plot(testrabph)
```




This two-step approach demonstrates the second class that is provided by the *hyperdraw* package: the *RagraphBPH* class for a laid out hypergraph.

# 3 Attributes affecting rendering

As demonstrated in the previous example, it is possible to set attributes on edges, or nodes, or even the whole graph, in order to modify the default appearance of the rendered graph.

The functions `nodeData()`, `edgeData()`, and `graphData()` can be used to set these attributes.

Some attributes are provided by the *Rgraphviz* package (e.g., the `edgeDataDefaults()` function shows the default edge attributes).

The following extra attributes are supported by the *hyperdraw* package:

**arrowLoc** A whole-graph attribute that controls where arrows are drawn on the hypergraph edges. "middle" draws arrows at the edge nodes (in the middle of the curves), "end" draws arrows at the end of edges, "start" draws arrows at the start of edges, "both" draws arrows at both the start and the end of edges, and "none" produces no arrows.

**margin** A node attribute that controls the amount of space around the label within a node. This should be a character value that evaluates to a grid `unit` object.

**lwd** An edge attribute that controls the width of the edge.

In addition, *hyperdraw* currently only supports the values "circle", "box", and "plain" for the `shape` attribute.

# 4 The algorithm

The algorithm for rendering a hypergraph that is implemented in this package is roughly as follows:

- represent the hypergraph in bipartite form as a *graphNEL* object.
- use the graphviz software (via the *Rgraphviz* package) to lay out the *graphNEL* object, *but do not use Rgraphviz to render the lay out.*
- render the normal nodes at the locations given by the graphviz layout.
- for each edge-node, determine the average angle at which the graph edges enter or exit the node.

- for each edge (between a normal node and an edge-node), generate a bezier curve from the location of the normal node to the location of the edge-node, with the constraint that, where the curve is incident on the edge-node, it must be tangent to the average angle for that edge-node. This produces a single smooth curve between any two normal nodes (via an edge-node). (The angle of the curve at the normal node depends on the difference between the average angle at the edge-node and the angle between the normal node and the edge-node.)
- trim the bezier curves so that they start or end at the boundary of the normal nodes.
- render the bezier curves, with an arrow on the end of any curve that ends at an edge-node.
- render a label "beside" each edge-node.

# 5 Issues

The calculation and trimming of bezier curves is currently quite slow. There is also no automatic scaling of text done at this point.