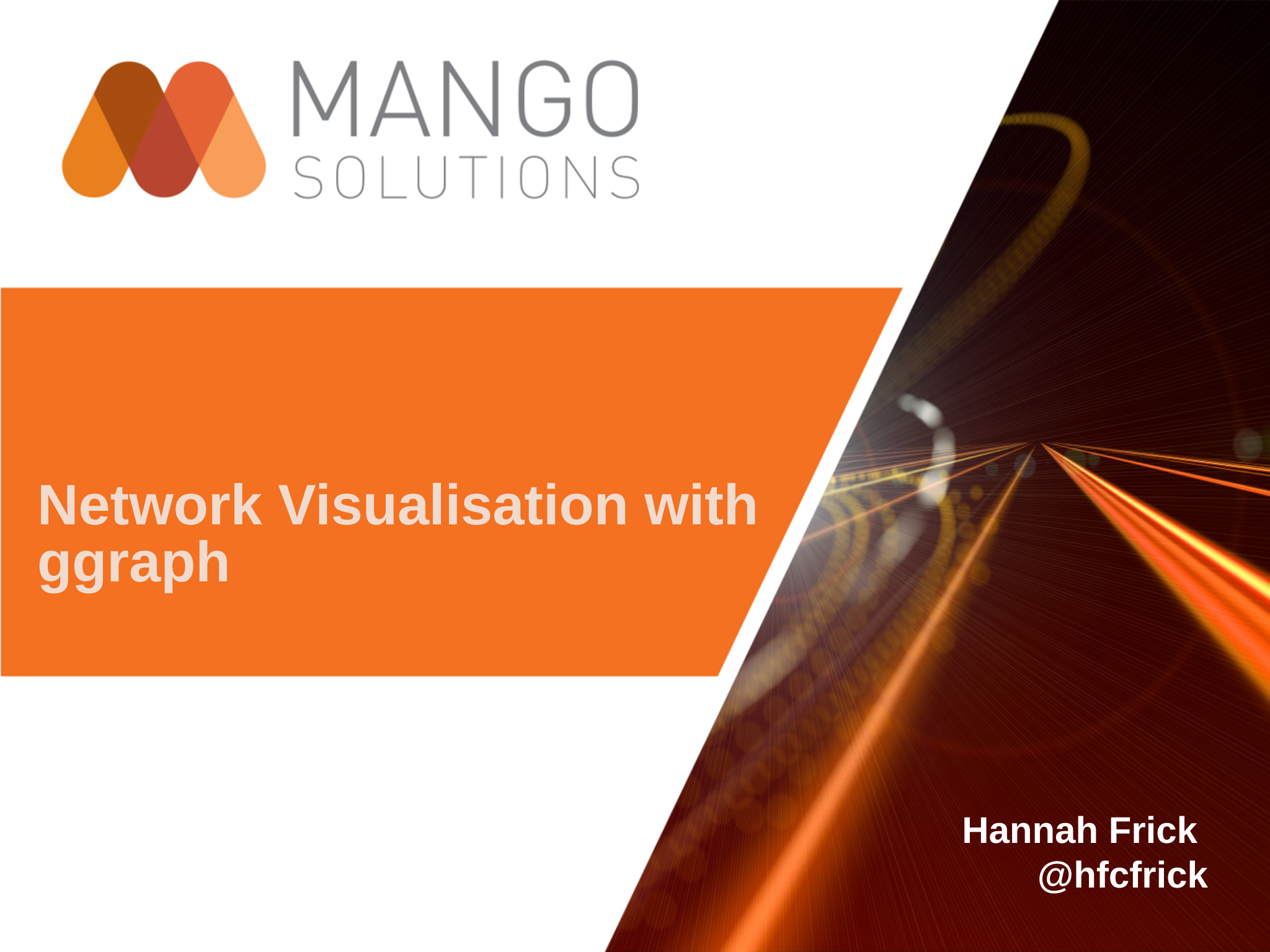


Network Visualisation with ggraph

Hannah Frick
@hfcfrick

Outline for the Tutorial

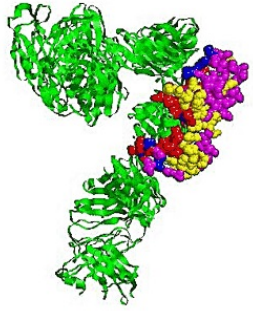
- Representation of networks with **igraph**
- Visualisation with **ggraph**
  - layers
  - nodes
  - edges

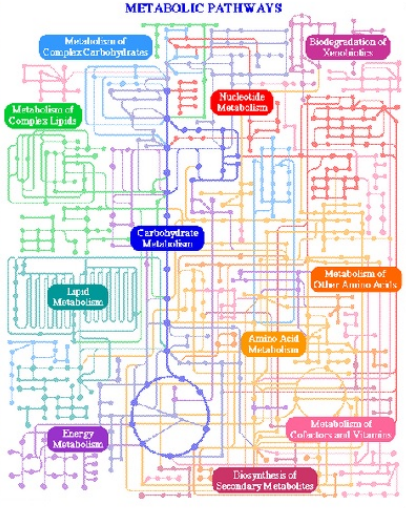
Based on Thomas Lin Pedersen's series of blog posts on **ggraph**

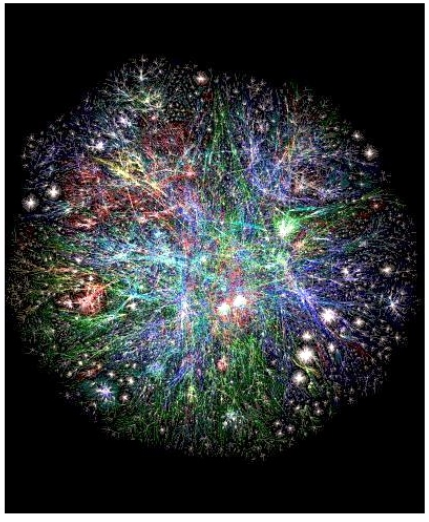
About me

- Statistician by training
- Author of **psychomix** and **trackeR** packages
- A Mango cat since ~ mid 2017
- R-Ladies: co-organiser for London chapter and part of the global Leadership Team
- Github: `hfrick`
- Twitter: `@hfcfrick`

Networks are everywhere



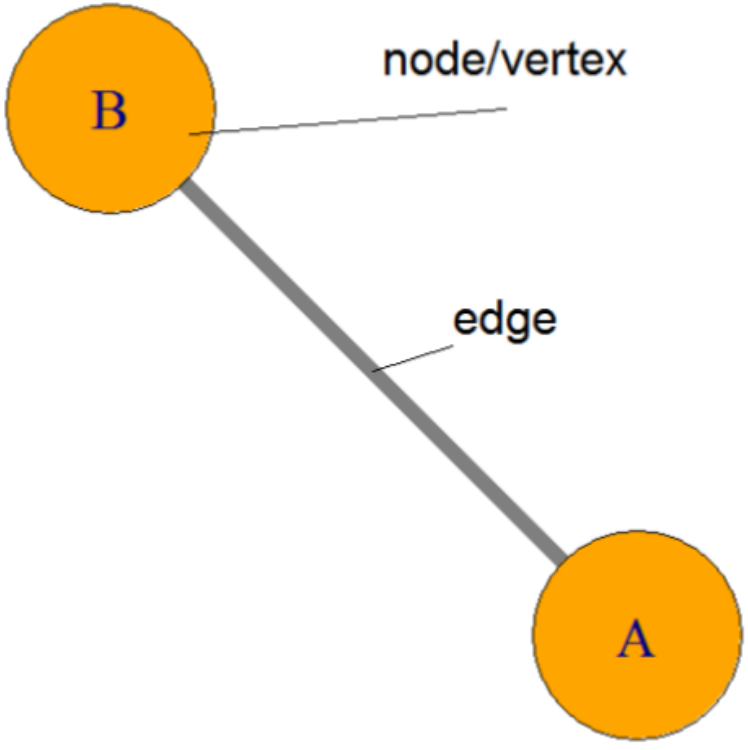




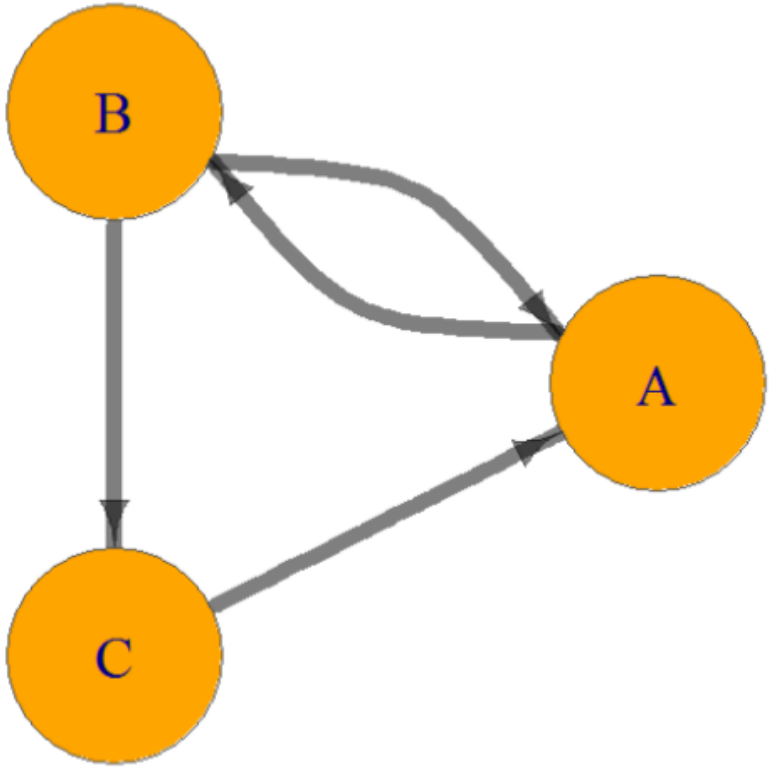
Representing Networks

Terminology

Nodes and edges



Directed



Matrix Representation

Code whether each possible edge exists or not.

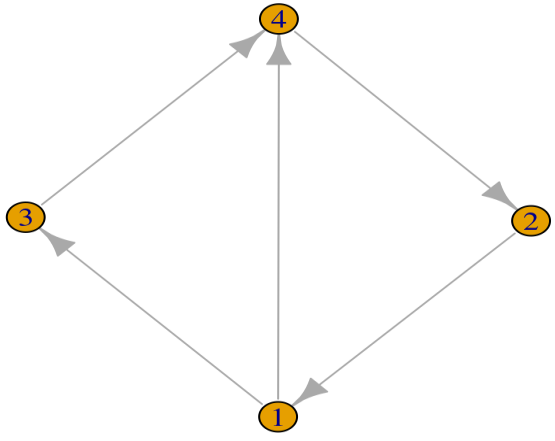
$$A_{ij} = \begin{cases} 0, & \text{no edge} \\ 1, & \text{edge exists} \end{cases}$$

For example

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Matrix Representation in R

```
library(igraph)
A <- rbind(c(0, 0, 1, 1), c(1, 0, 0, 0), c(0, 0, 0, 1), c(0, 1, 0, 0))
g <- graph_from_adjacency_matrix(A)
plot(g)
```



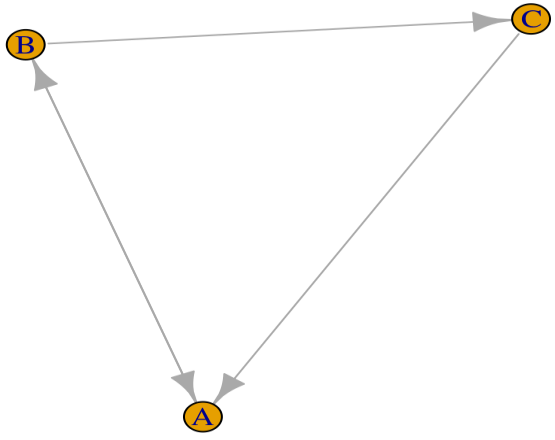
Edge List Representation

It is much more compact to just list the edges. We can use a two-column matrix

$$E = \begin{bmatrix} A & B \\ B & A \\ B & C \\ C & A \end{bmatrix}$$

Edge List in R

```
el <- rbind(c("A","B"), c("B","A"), c("B","C"), c("C","A"))
g <- graph_from_edgelist(el)
plot(g)
```



Creating igraph Objects

From an adjacency matrix or edge list

```
g <- graph_from_adjacency_matrix(A)
g <- graph_from_edgelist(el)
```

From a textual description

```
g <- graph_from_literal(A--B, B-+C, C-+A)
```

From a data frame

```
df <- as.data.frame(el)
g <- graph_from_data_frame(df)
```

Converting back

From a graph object we can output just as we read in:

Adjacency Matrix

```
A <- as_adjacency_matrix(g,
  sparse = FALSE)
A
```

```
##   A B C
## A 0 1 0
## B 1 0 1
## C 1 0 0
```

Edge List

```
el <- as_edgelist(g)
el
```

```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "B"  "A"
## [3,] "B"  "C"
## [4,] "C"  "A"
```

Import/Export

Networks from Data Frames

Usually we'll read in edge lists as `data.frame` objects. The csv file `dolphin_edges.csv` contains the edge list of the undirected social network of 62 dolphins in a community living off Doubtful Sound, New Zealand (Lusseau et al, 2003).

Networks from Data Frames

```
dolphinEdges <- read.csv("data/dolphin_edges.csv")
head(dolphinEdges, n = 4)
```

```
##   From      To
## 1  CCL  Double
## 2 DN16 Feather
## 3 DN21 Feather
## 4 Beak    Fish
```

```
dolphin <- graph_from_data_frame(dolphinEdges, directed = TRUE)
```

Node Lists

It is also possible to load information about the nodes at the same time.

```
dolphinVertices <- read.csv("data/dolphin_vertices.csv")
head(dolphinVertices, n = 4)
```

```
##          Name Gender
## 1        Beak   Male
## 2 Beescratch   Male
## 3      Bumper   Male
## 4         CCL Female
```

```
dolphin <- graph_from_data_frame(dolphinEdges, directed = TRUE,
                                 vertices = dolphinVertices)
```

Exporting to a Data Frame

To export an igraph object back to a data.frame we use the `as_data_frame` function. This function can output the edge list, the node list or both depending on the `what` argument.

```
dolphinDFs <- as_data_frame(dolphin, what = "both")
str(dolphinDFs) # Both data frames in a list
```

```
## List of 2
##  $ vertices:'data.frame':    62 obs. of  2 variables:
##   ..$ name  : chr [1:62] "Beak" "Beescratch" "Bumper" "CCL" ...
##   ..$ Gender: chr [1:62] "Male" "Male" "Male" "Female" ...
##  $ edges   :'data.frame':    159 obs. of  2 variables:
##   ..$ from: chr [1:159] "CCL" "DN16" "DN21" "Beak" ...
##   ..$ to  : chr [1:159] "Double" "Feather" "Feather" "Fish" ...
```

Other Formats

`igraph` allows import and export from a number of common formats.

This is done using the `read_graph` and `write_graph` functions. A common open format is `graphml`.

```
write_graph(dolphin, "dolphin.graphml", format = "graphml")
```

Other Formats

Format	Description
`edgelist`	A simple text file with one edge per line
`pajek`	Pajek is a popular Windows program for network analysis
`gml`	Graph Modelling Language is a common text based open format
`graphml`	Graph Markup Language is an XML based open format
`dot`	Format used by GraphViz

Visualisation

Visualisation in R

- **igraph**: large collection of layouts; base graphics
- **ggraph**: leverages **igraph** for **ggplot2**-based graphics
- interactive JS visualisations: **visNetwork**, **networkD3**, **threejs**
- …

ggraph Package

> ***ggraph*** *is an extension to the* ***ggplot2*** *API to support relational data such as networks and trees.*

- A *grammar of graphics* approach with geoms, aesthetics, etc
- Ability to quickly exchange plot components
- Central components: layouts, nodes, edges
- Additional ggplot goodies: theme, facetting

Layout

A layout is the specification on how nodes should be placed on a plane.

- Spatial position of the nodes spans all layers of the plot, hence define outside of geoms or stats
- `ggraph()` is the equivalent to `ggplot()` and `layout` is an argument to `ggraph()`

Available Layouts

Dendrogram objects

- `auto`
- `dendrogram`
- `even`

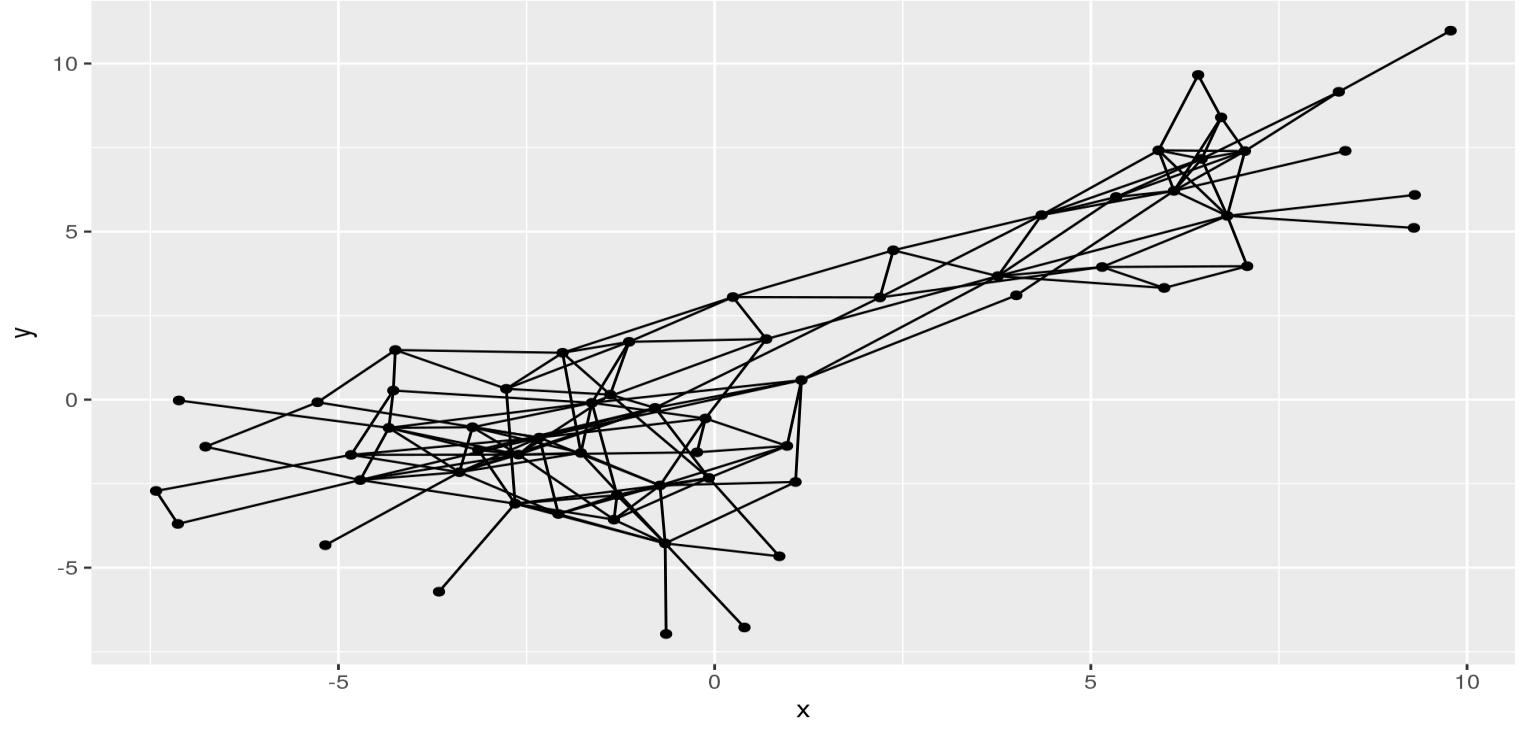
igraph objects

- `auto`
- `igraph`
- `dendrogram`
- `manual`
- `linear`
- `treemap`
- `circlepack`
- `partition`
- `hive`

Default Layout

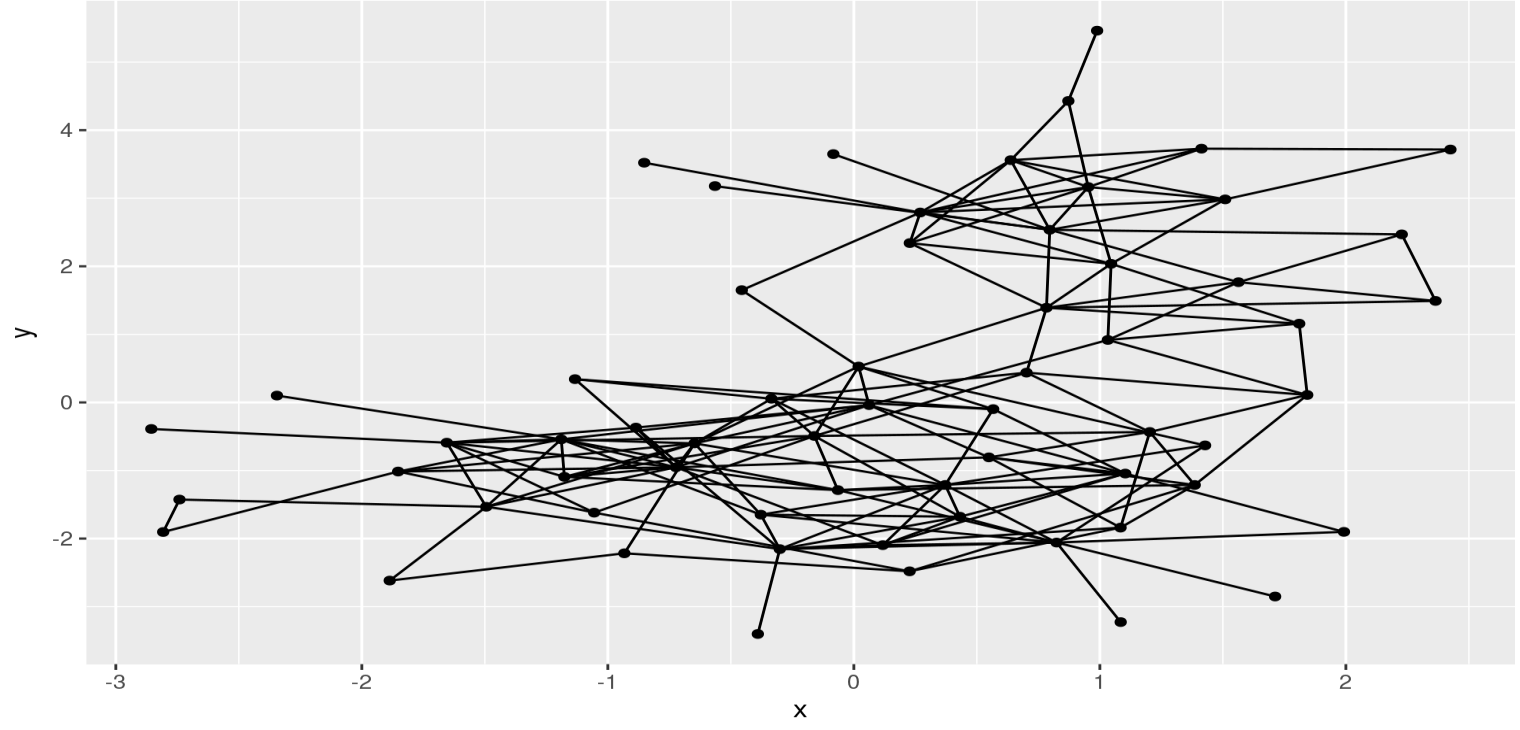
```
ggraph(dolphin) + geom_node_point() + geom_edge_link()
```

```
## Using `nicely` as default layout
```

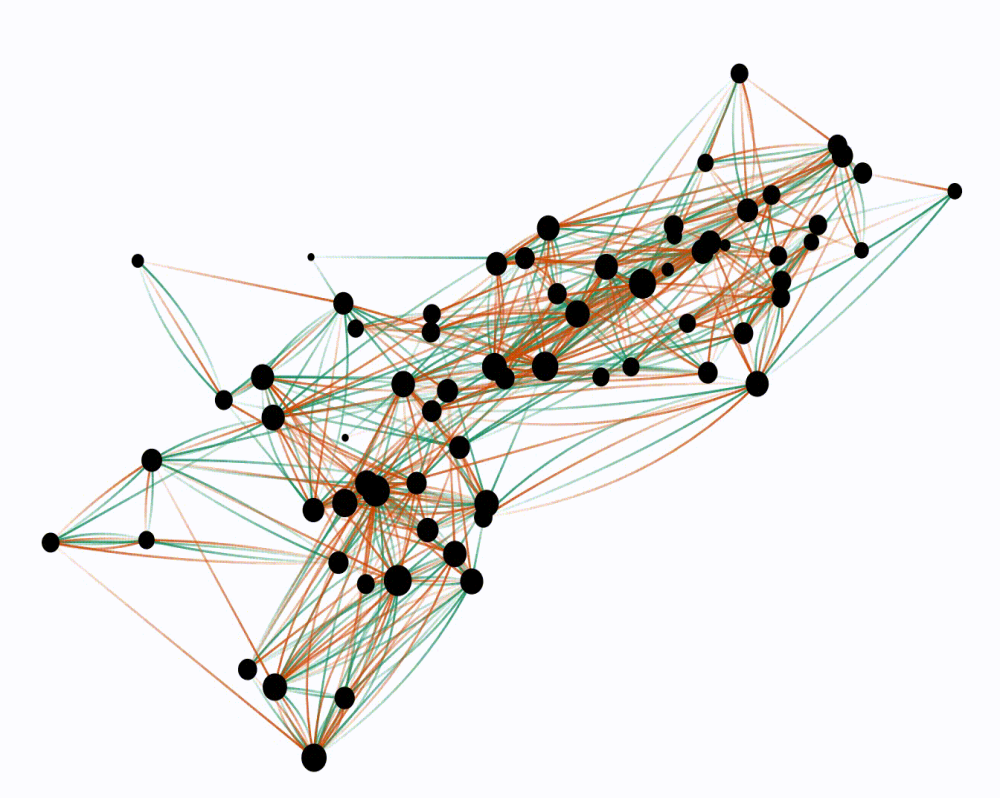


Layout

```
# specify igraph algorithm directly
ggraph(dolphin, layout = "kk") + geom_node_point() + geom_edge_link()
```



Layouts



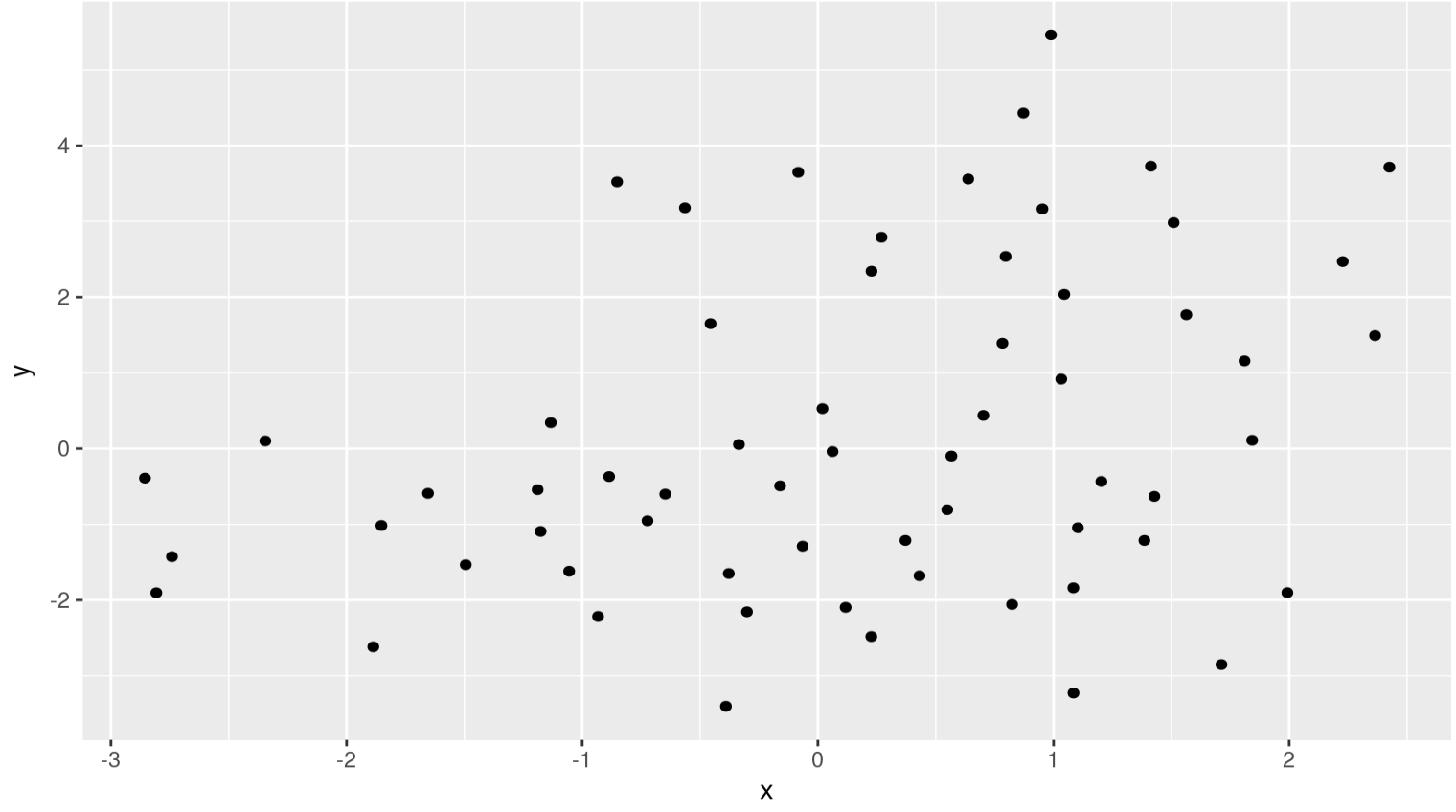
Nodes

The layout specifies *where* nodes are drawn, now we specify *how* they are drawn.

The layout methods return a data frame with coordinates, so regular **ggplot2** geoms work.

Nodes

```
ggraph(dolphin, layout = 'kk') + geom_point(aes(x = x, y = y))
```



Nodes

ggraph provides various `geom_node_*()`

ggraph	corresponds to
`geom_node_point()`	`geom_point()`
`geom_node_text()`	`geom_text()`
`geom_node_label()`	`geom_label()`
`geom_node_tile()`	`geom_tile()`
`geom_node_circle()`	`ggforce::geom_circle()`
`geom_node_arc_bar()`	`ggforce::geom_arc_bar()`

Nodes

Why use **ggraph**'s family of `geom_node_*()` functions?

- readability: clear that this layer adds the nodes
- saves typing: defaults play nicely with the layout data frame
- extra functionality: filter aesthetic

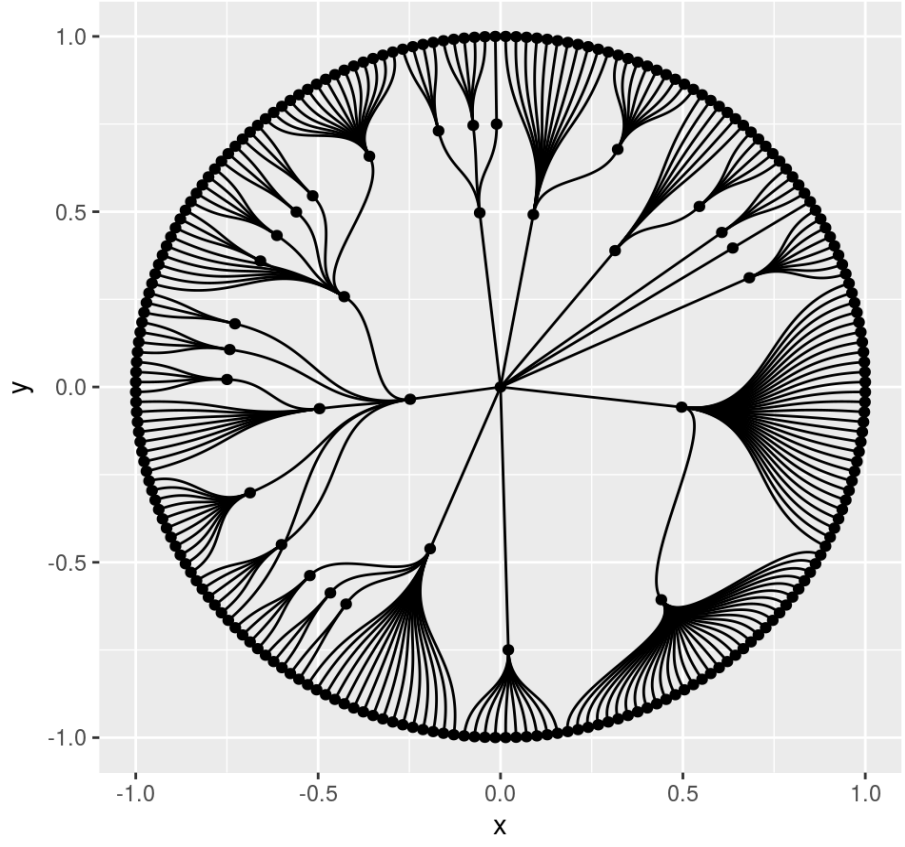
Nodes

Illustration of filter aesthetic

```
gr <- graph_from_data_frame(flare$edges,
                            vertices = flare$vertices)
p <- ggraph(gr, layout = 'dendrogram', circular = TRUE) +
  geom_edge_diagonal() +
  coord_fixed()
```

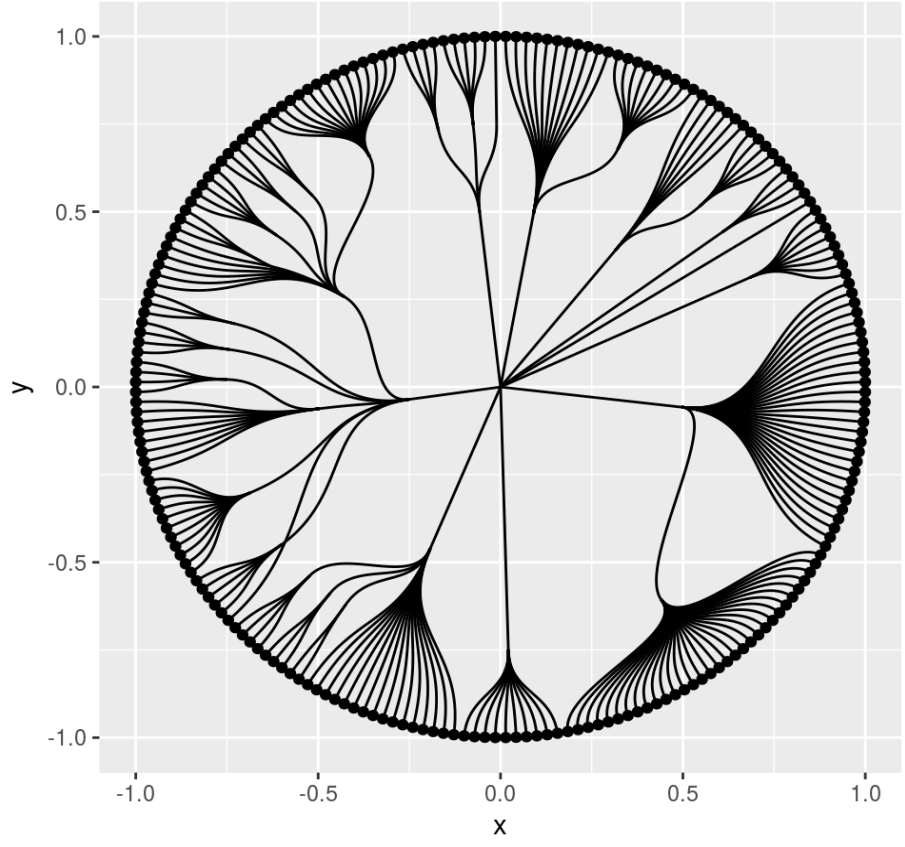
Nodes

```
p + geom_node_point()
```



Nodes

```
p + geom_node_point(aes(filter = leaf))
```



Edges

An edge is an abstract concept denoting a relationship between two entities.

Most often drawn as a (straight) line between nodes but sometimes also represented by, e.g., position.

Family of `geom_edge_*()` functions available: some for all layouts, some only for specific layouts.

Iris Example

```
hierarchy <- as.dendrogram(hclust(dist(iris[, 1:4])))

# Classify nodes based on agreement between children
hierarchy <- tree_apply(hierarchy, function(node, children, ...) {
    if (is.leaf(node)) {
        attr(node, 'Class') <- as.character(
          iris[as.integer(attr(node, 'label')),5])
    } else {
        classes <- unique(sapply(children, attr, which = 'Class'))
        if (length(classes) == 1 && !anyNA(classes)) {
            attr(node, 'Class') <- classes
        } else {
            attr(node, 'Class') <- NA
        }
    }
    attr(node, 'nodePar') <- list(class = attr(node, 'Class'))
    node
}, direction = 'up')
```

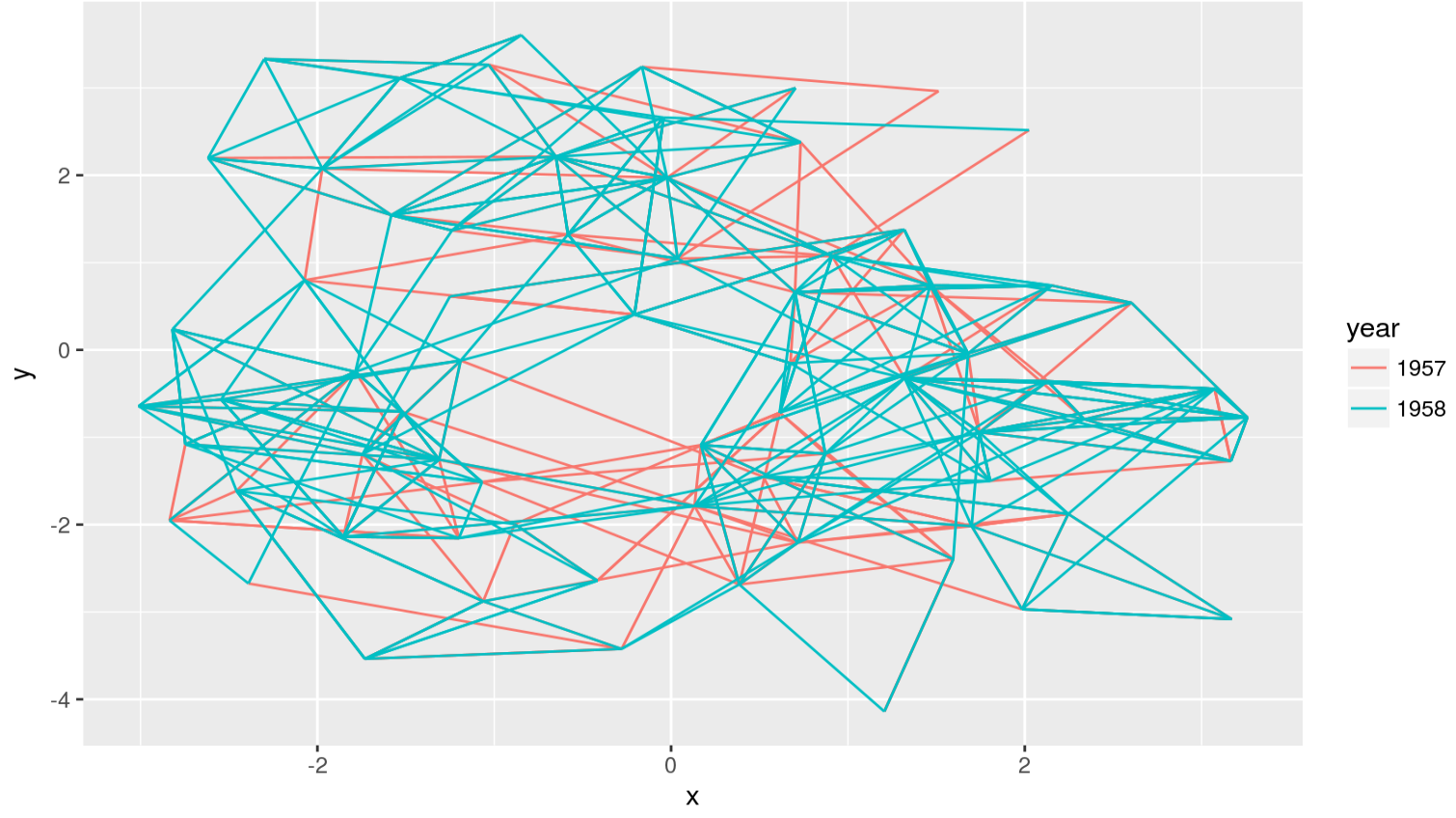
Highschool Example

```
hairball <- graph_from_data_frame(highschool)

# Classify nodes based on popularity gain
pop1957 <- degree(delete_edges(hairball,
                               which(E(hairball)$year == 1958)),
                  mode = 'in')
pop1958 <- degree(delete_edges(hairball,
                               which(E(hairball)$year == 1957)),
                  mode = 'in')
V(hairball)$pop_devel <- ifelse(pop1957 < pop1958, 'increased',
                                ifelse(pop1957 > pop1958, 'decreased',
                                       'unchanged'))
V(hairball)$popularity <- pmax(pop1957, pop1958)
E(hairball)$year <- as.character(E(hairball)$year)
```

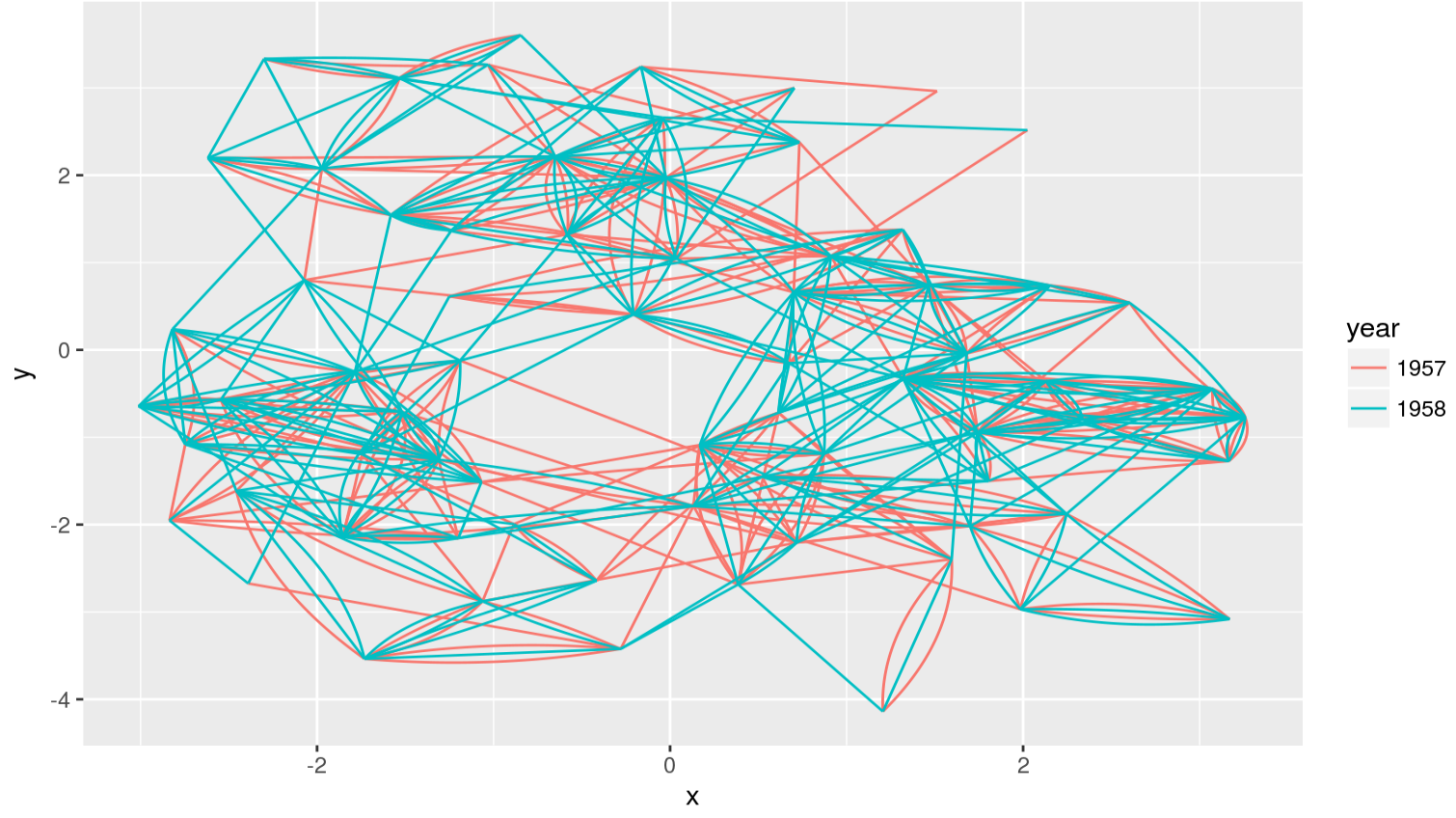
Link

```
ggraph(hairball, layout = 'kk') + geom_edge_link(aes(colour = year))
```



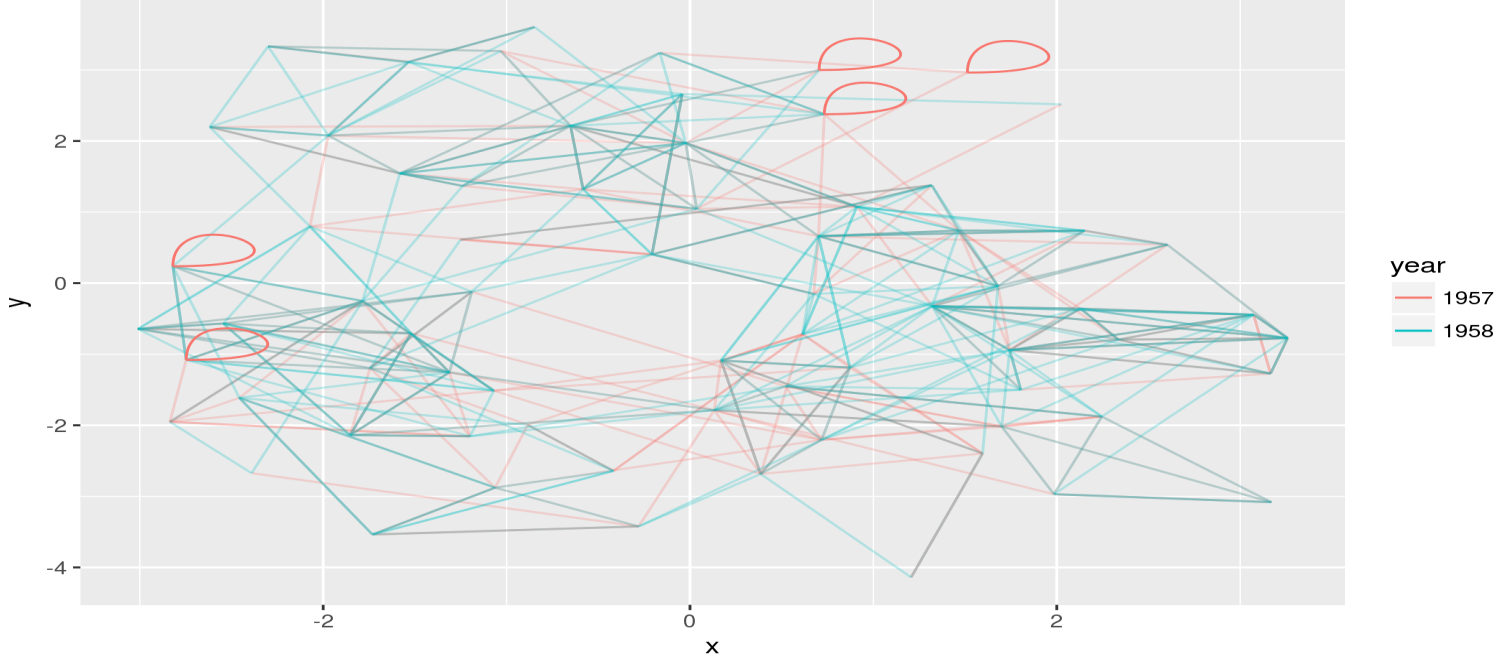
Fan

```
ggraph(hairball, layout = 'kk') + geom_edge_fan(aes(colour = year))
```



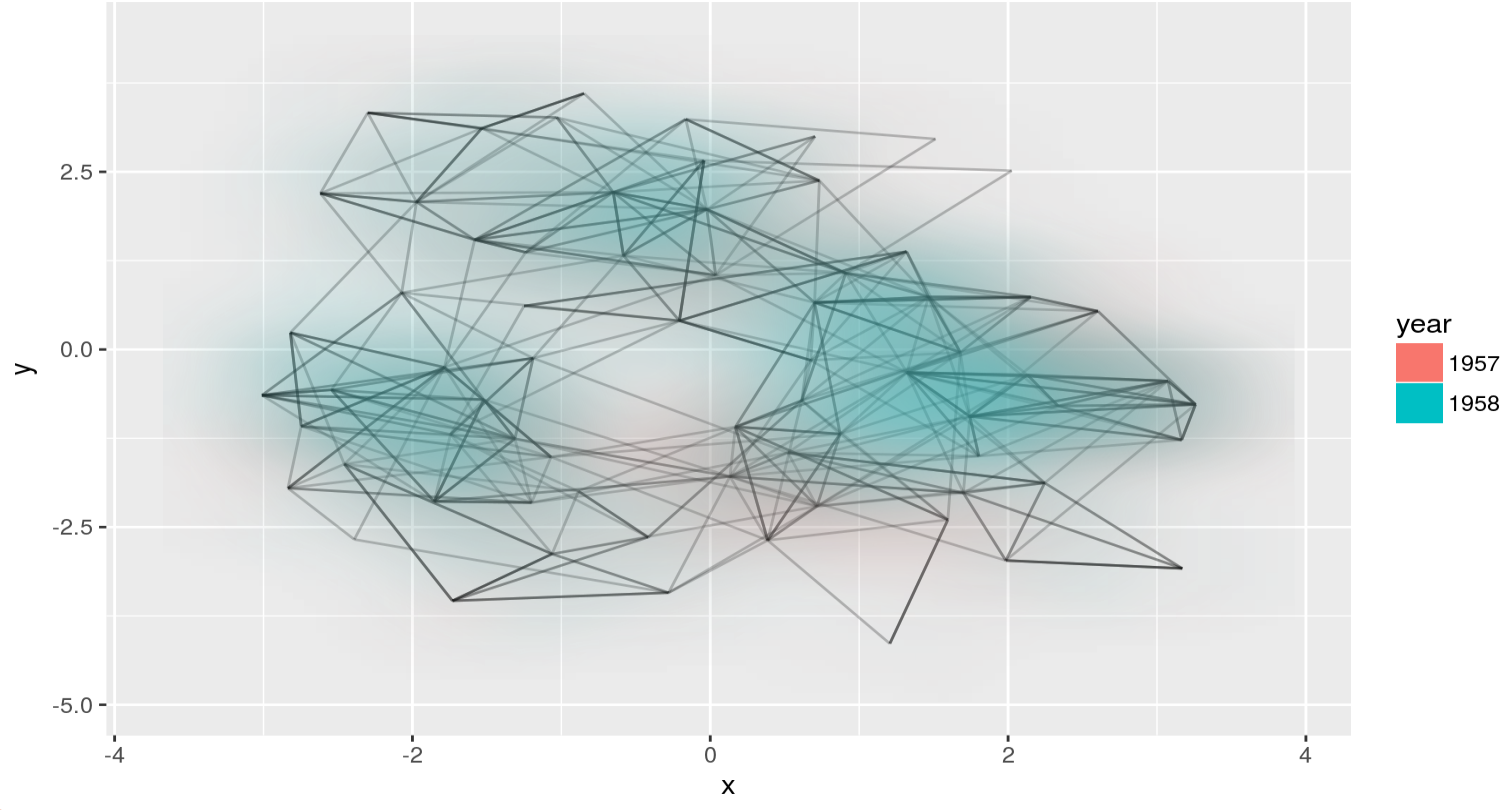
Loops

```
loopy_hairball <- add_edges(hairball, rep(1:5, each = 2),
                            year = rep('1957', 5))
ggraph(loopy_hairball, layout = 'kk') +
  geom_edge_link(aes(colour = year), alpha = 0.25) +
  geom_edge_loop(aes(colour = year))
```



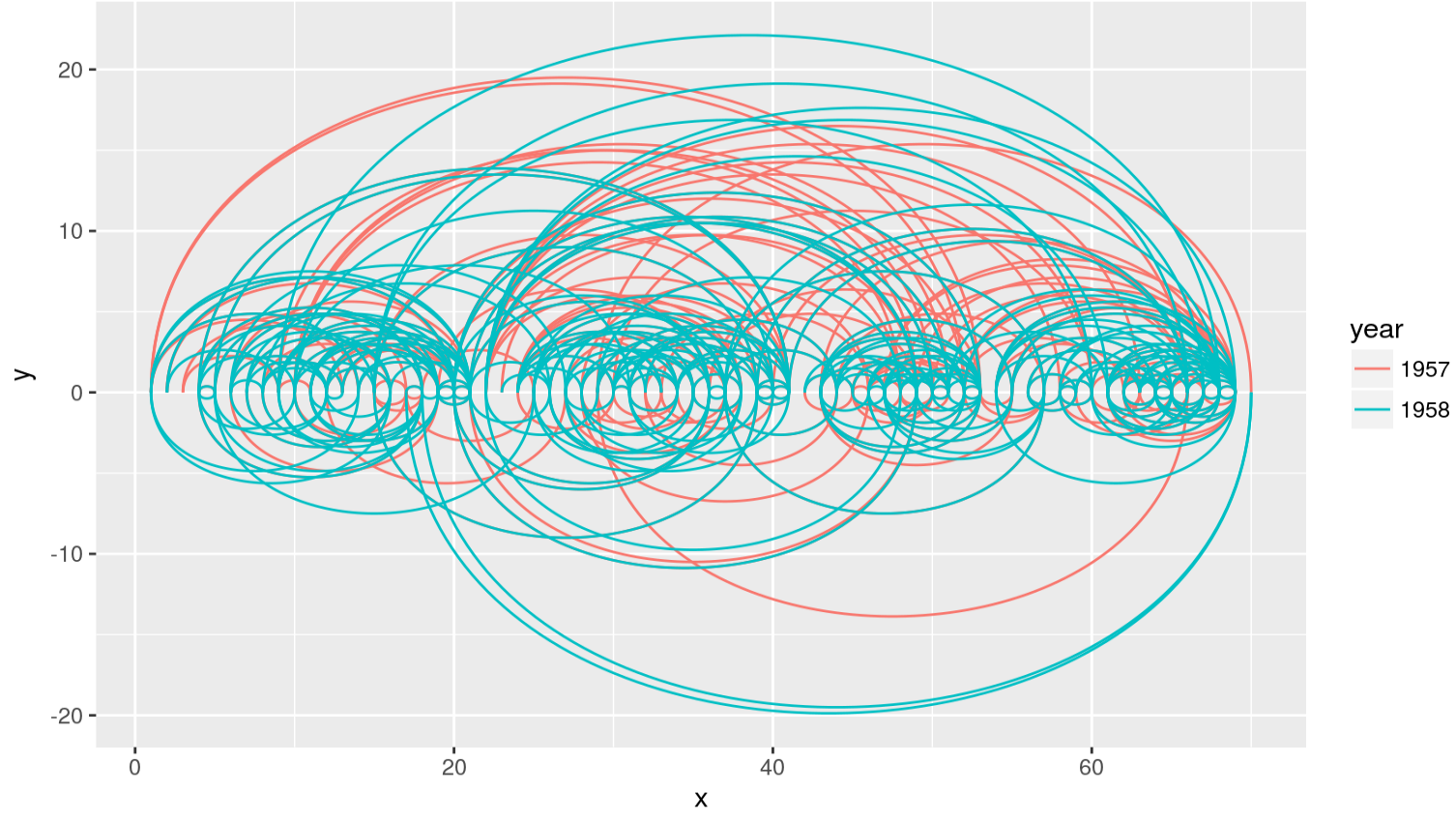
Density

```
ggraph(hairball, layout = 'kk') + geom_edge_link(alpha = 0.25) +
    geom_edge_density(aes(fill = year))
```



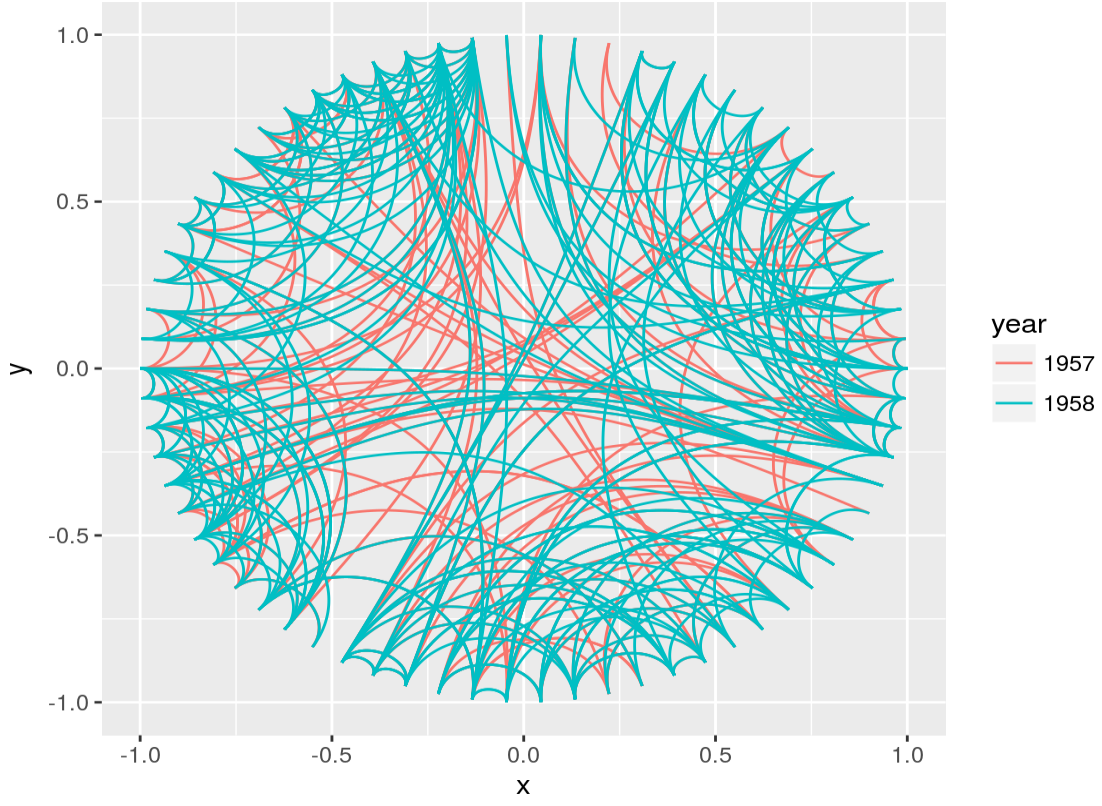
Arcs

```
ggraph(hairball, layout = 'linear') + geom_edge_arc(aes(colour = year))
```



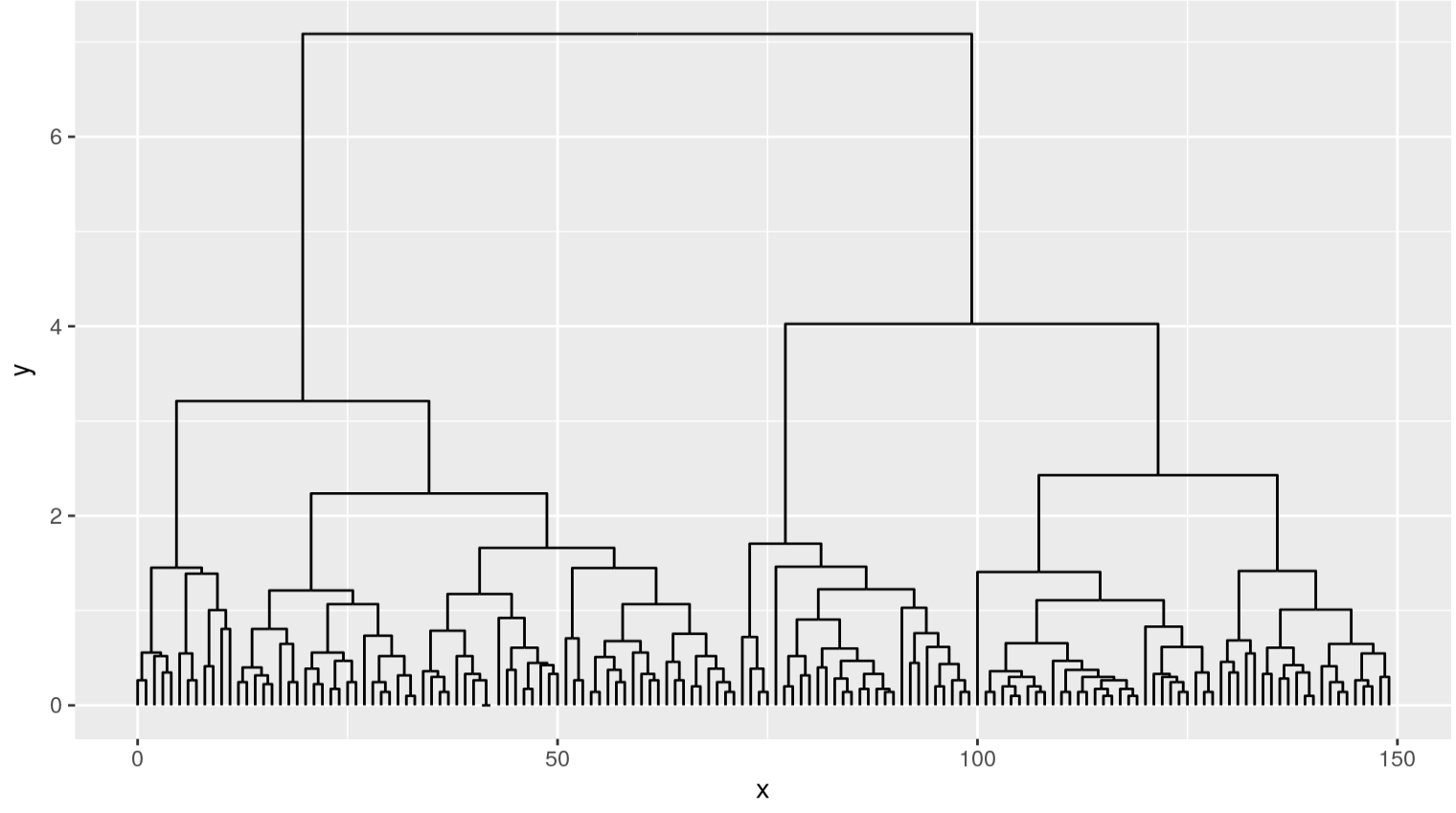
Arcs

```
ggraph(hairball, layout = 'linear', circular = TRUE) +
    geom_edge_arc(aes(colour = year)) + coord_fixed()
```



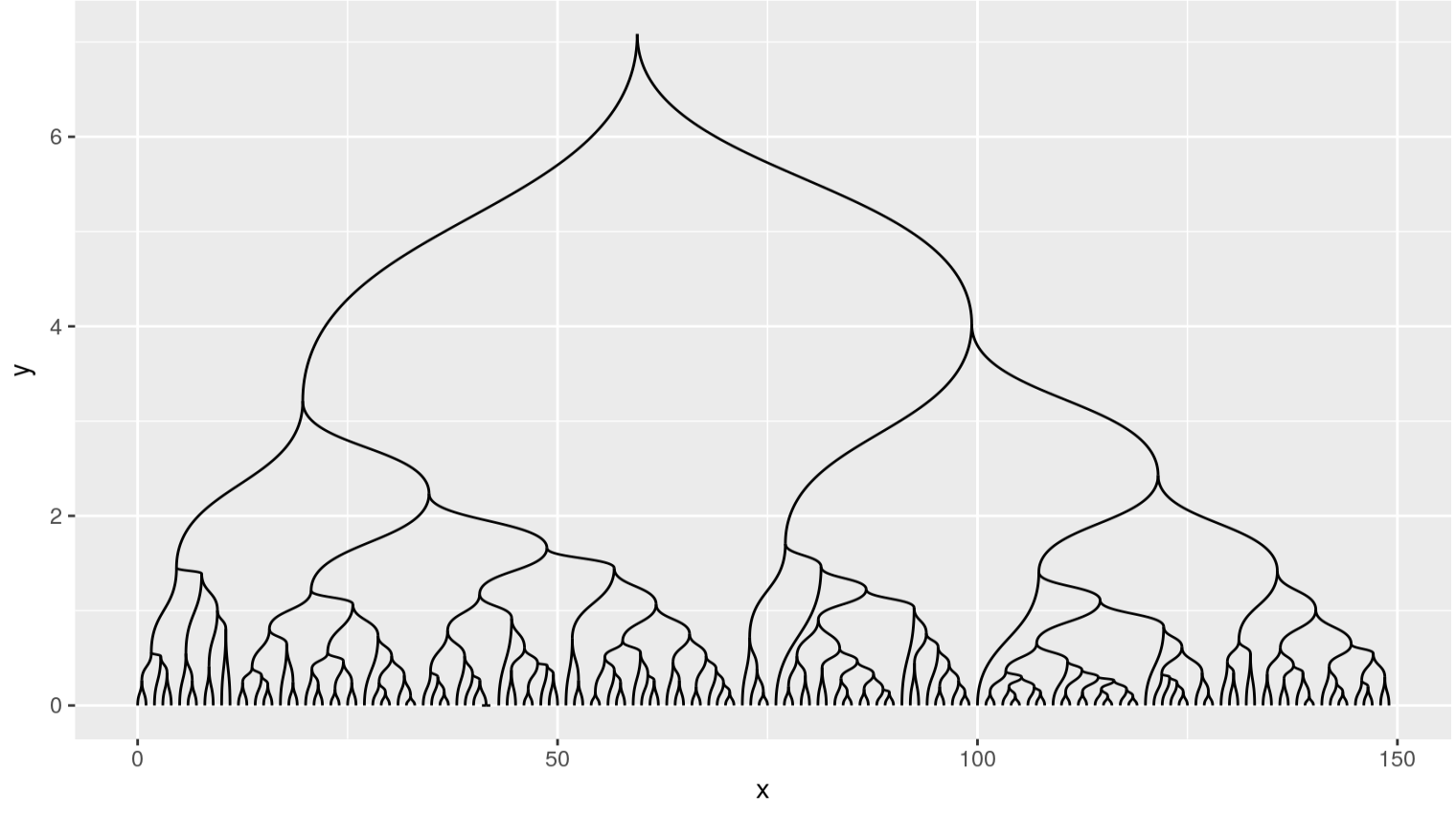
Elbow

```
ggraph(hierarchy, layout = 'dendrogram') + geom_edge_elbow()
```



Diagonals

```
ggraph(hierarchy, layout = 'dendrogram') + geom_edge_diagonal()
```



Three Types of Edge Geoms

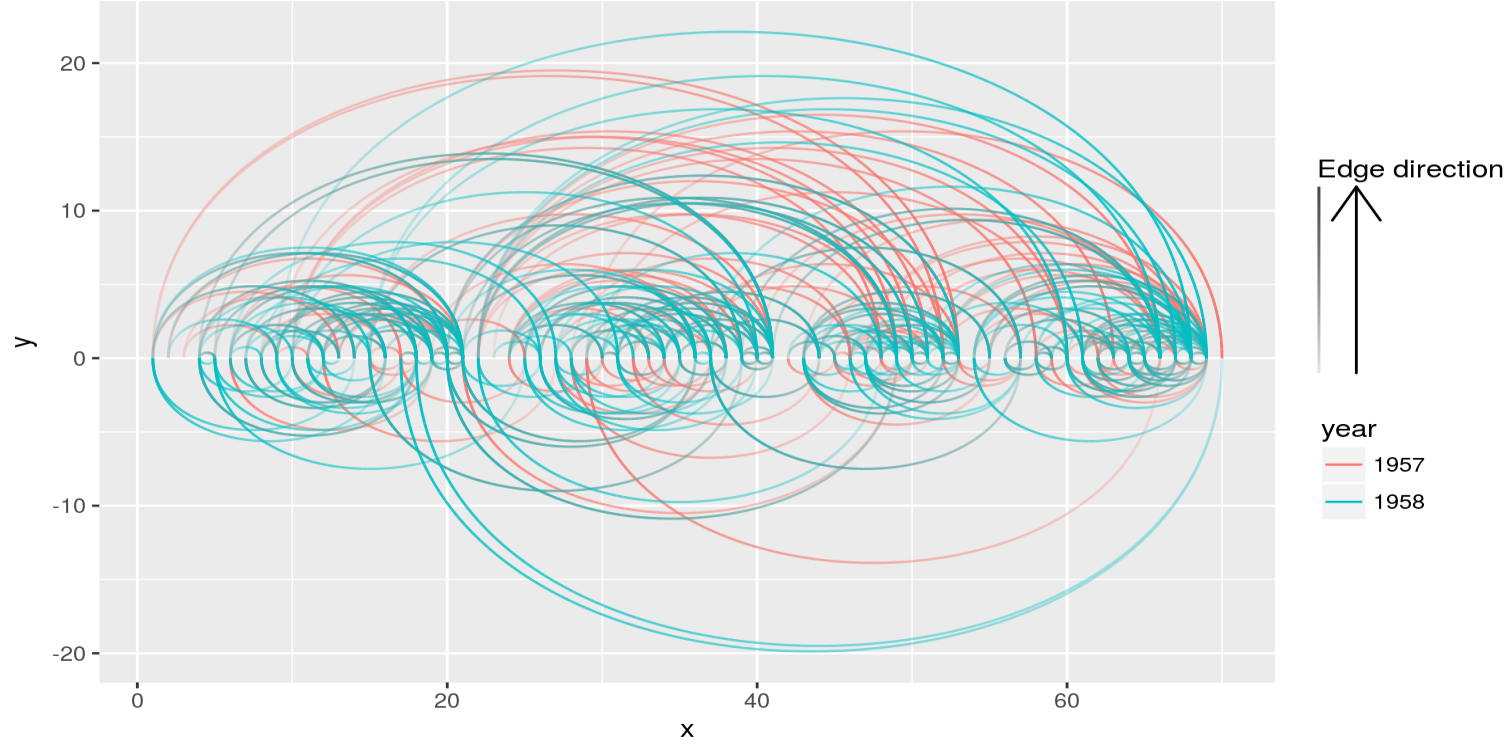
- base variant, e.g., `geom_edge_link()`
- suffix `2`, e.g., `geom_edge_link2()`
- suffix `0`, e.g., `geom_edge_link0()`

Base Variant

- Drawn by calculating a number of points along the edge path and draw a line between these
- Control the level of detail for curved edges
- Can be used for a gradient along the edge

Base Variant

```
ggraph(hairball, layout = 'linear') +
    geom_edge_arc(aes(colour = year, alpha = ..index..)) +
    scale_edge_alpha('Edge direction', guide = 'edge_direction')
```

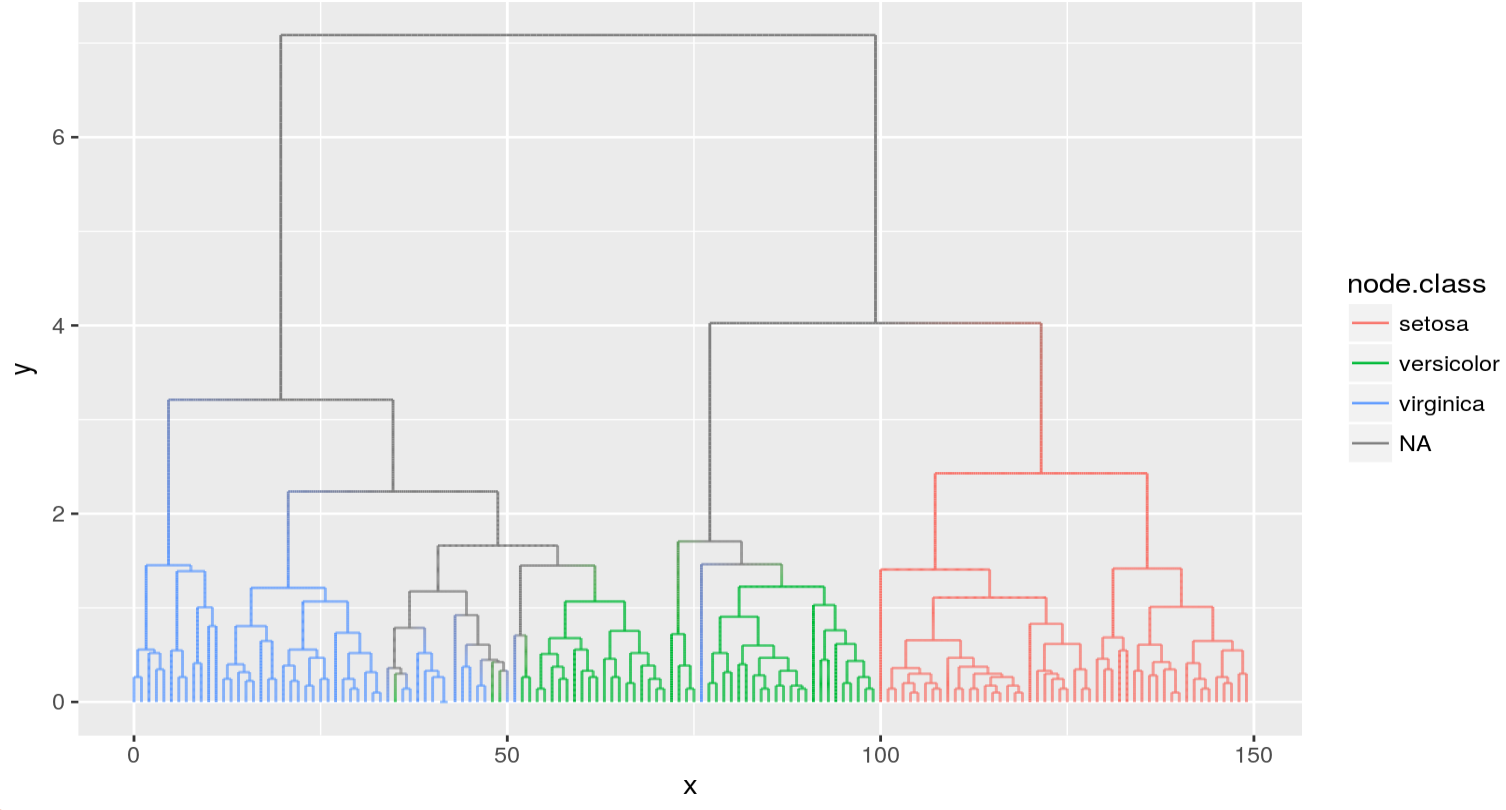


2-Variant

- Similar to base variant
- Difference: possible to map node attributes to the edge, aesthetics are then interpolated along the edge

2-Variant

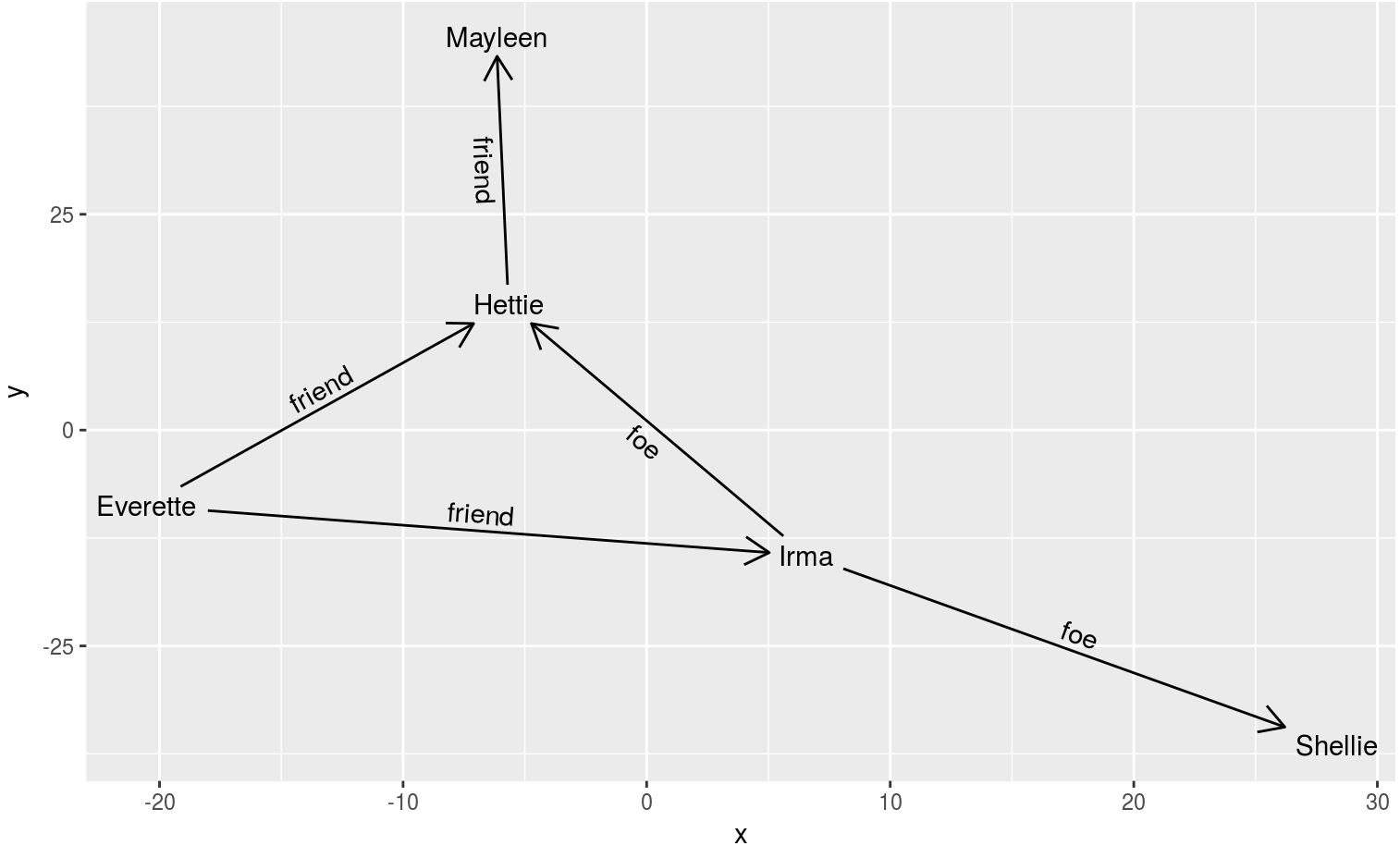
```
ggraph(hierarchy, layout = 'dendrogram') +
    geom_edge_elbow2(aes(colour = node.class))
```



0-variant

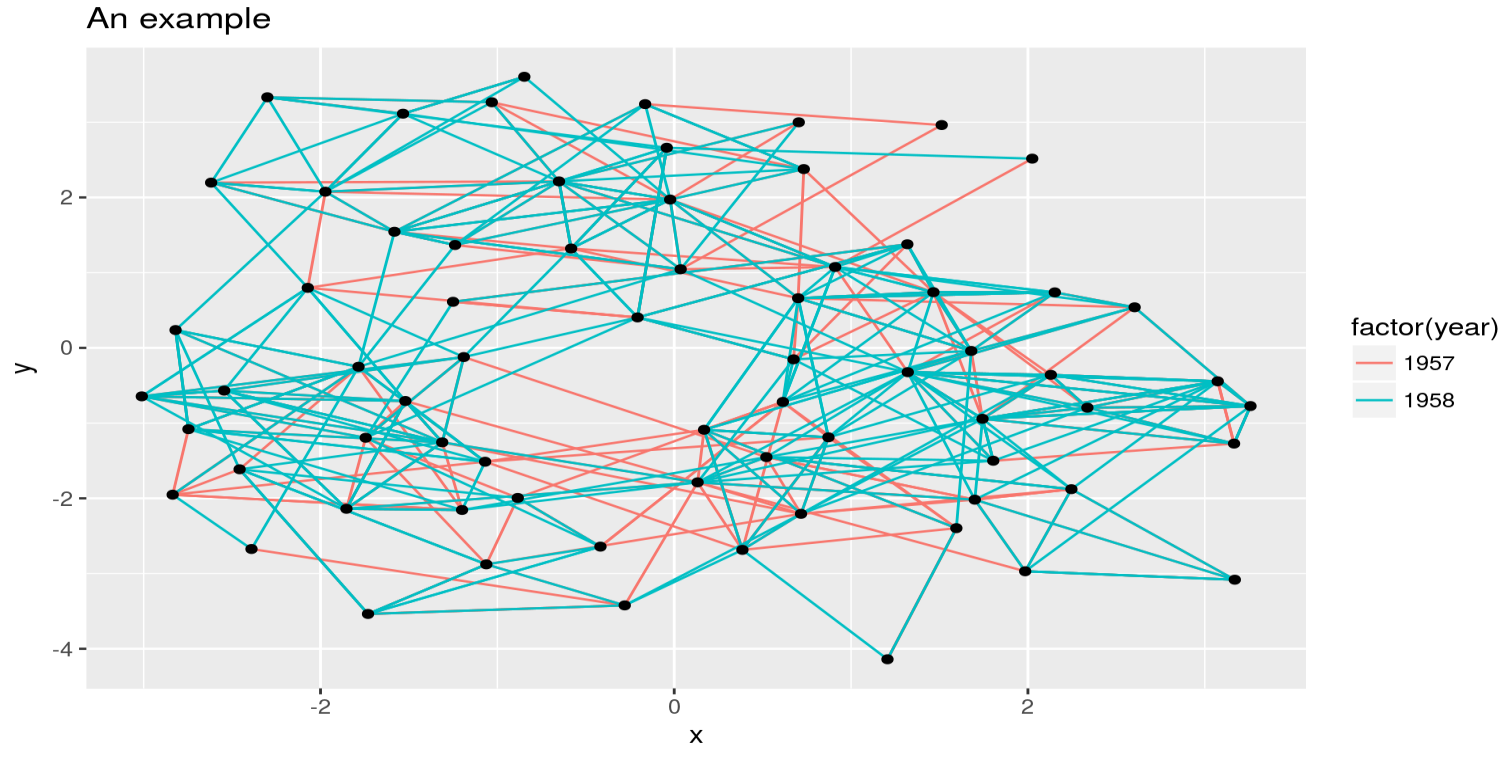
- “tack sharp resolution in the curves”
- no gradients along the edge

Decorating Edges



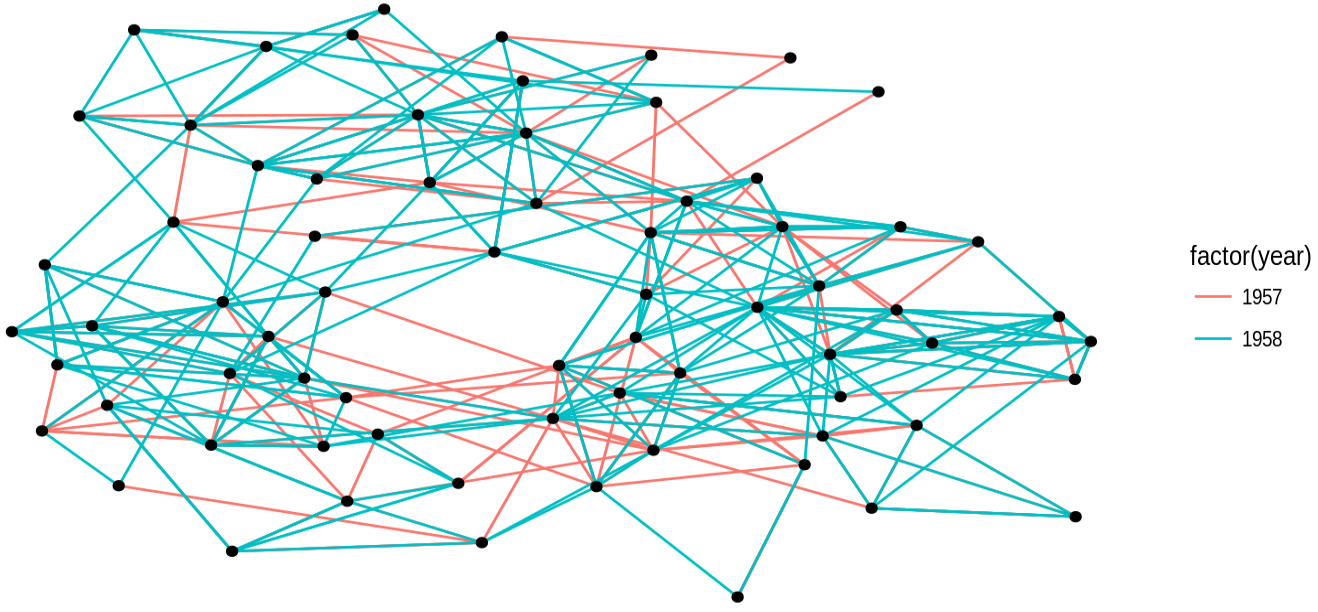
Theme

```
graph <- graph_from_data_frame(highschool)
(p <- ggraph(graph, layout = 'kk') + geom_edge_link(
  aes(colour = factor(year))) + geom_node_point() + ggtitle('An example'))
```



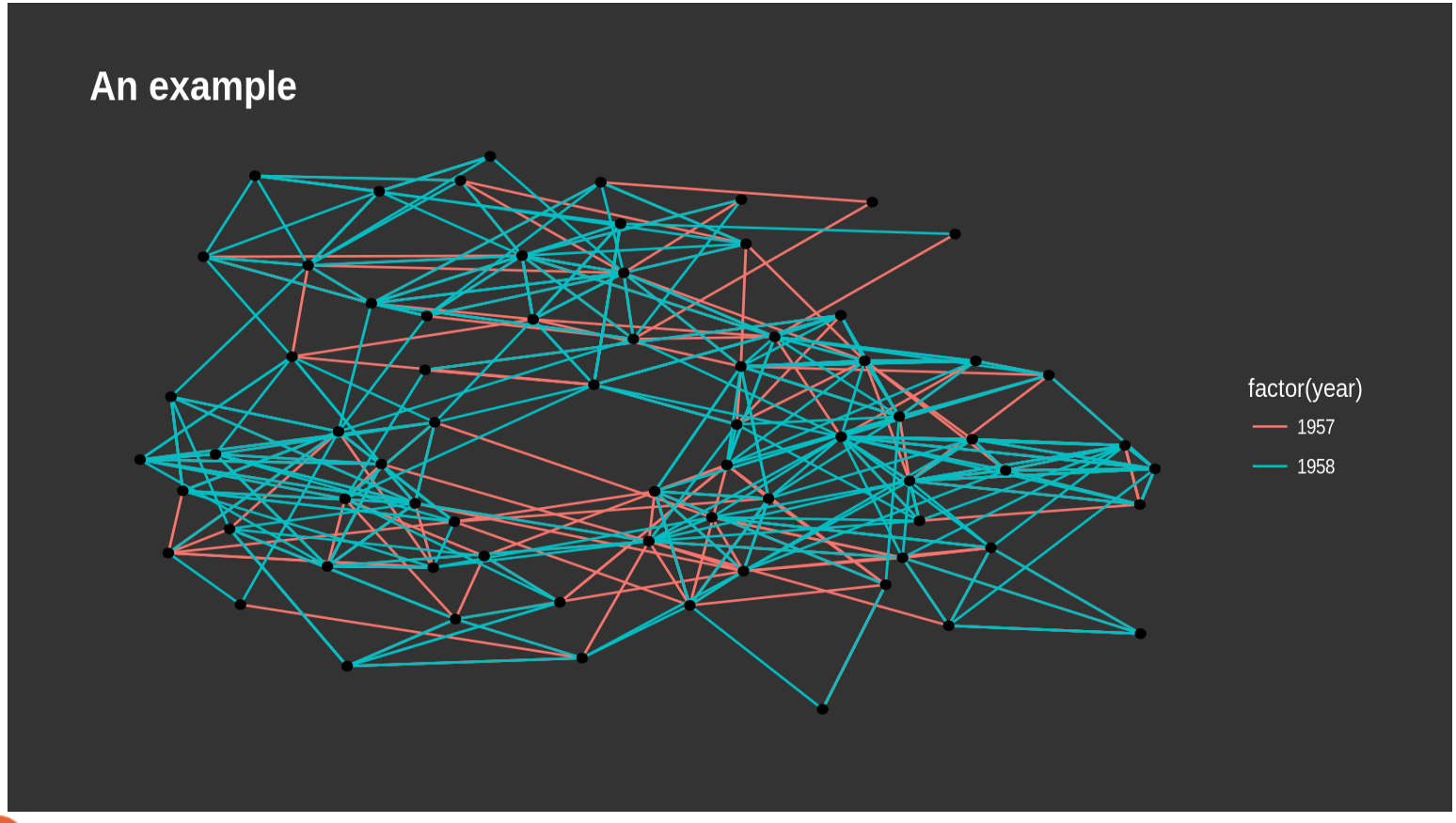
Theme

```
p + theme_graph()
```



Theme

```
p + theme_graph(background = 'grey20', text_colour = 'white')
```



Facetting

- Easy facetting is one of the great advantages of ggplot2
- Concept of plotting small multiples is useful also for graphs, e.g., to avoid hairballs and overplotting
- **ggraph** has its own facetting functions: `facet_nodes()`, `facet_edges()` and `facet_graph()`

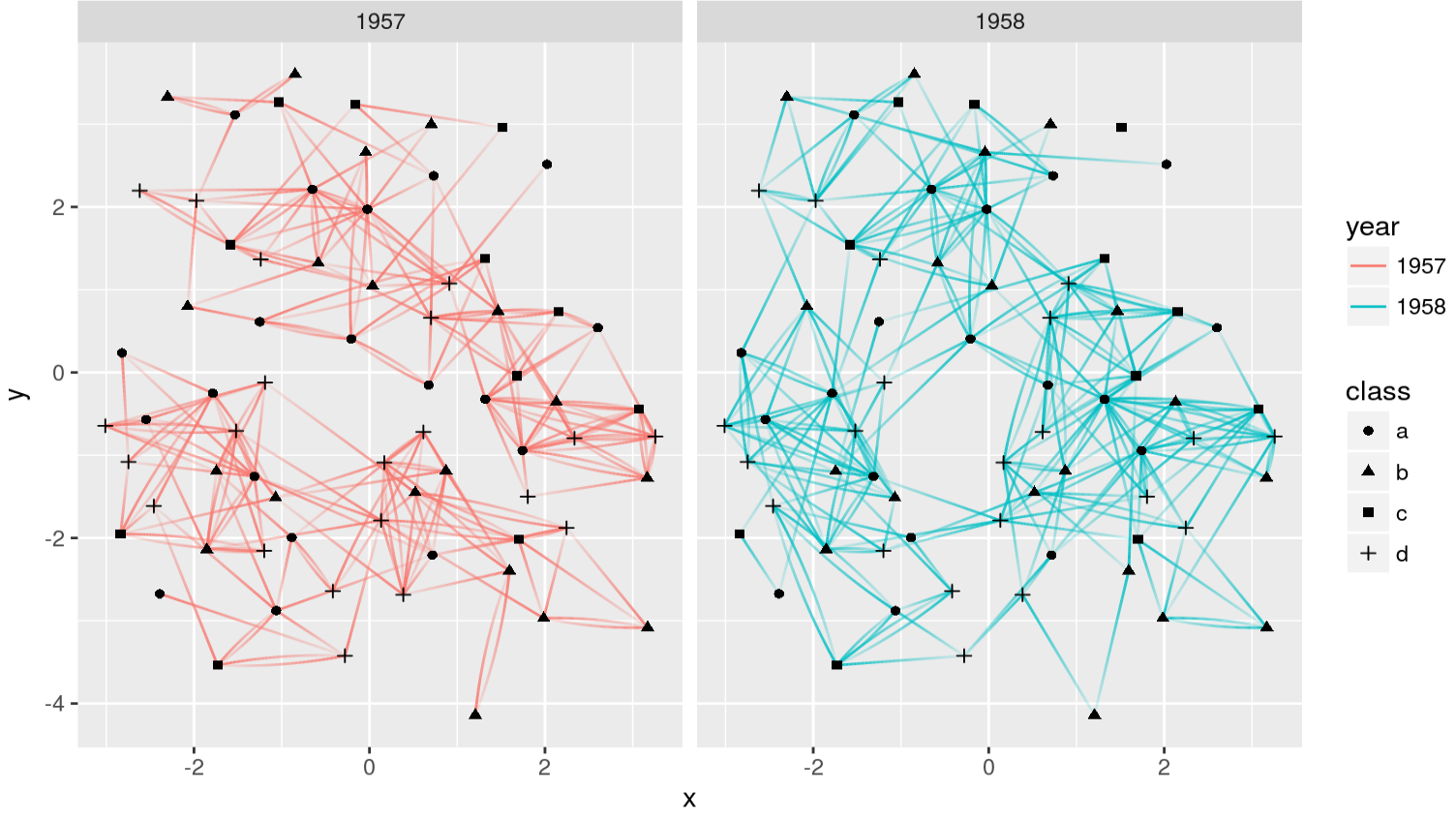
Facet Example

```
# Assign each node to a random class
V(graph)$class <- sample(letters[1:4], gorder(graph), TRUE)
# Make year a character
E(graph)$year <- as.character(E(graph)$year)

p <- ggraph(graph, layout = 'kk') +
    geom_edge_fan(aes(alpha = ..index.., colour = year)) +
    geom_node_point(aes(shape = class)) +
    scale_edge_alpha(guide = 'none')
```

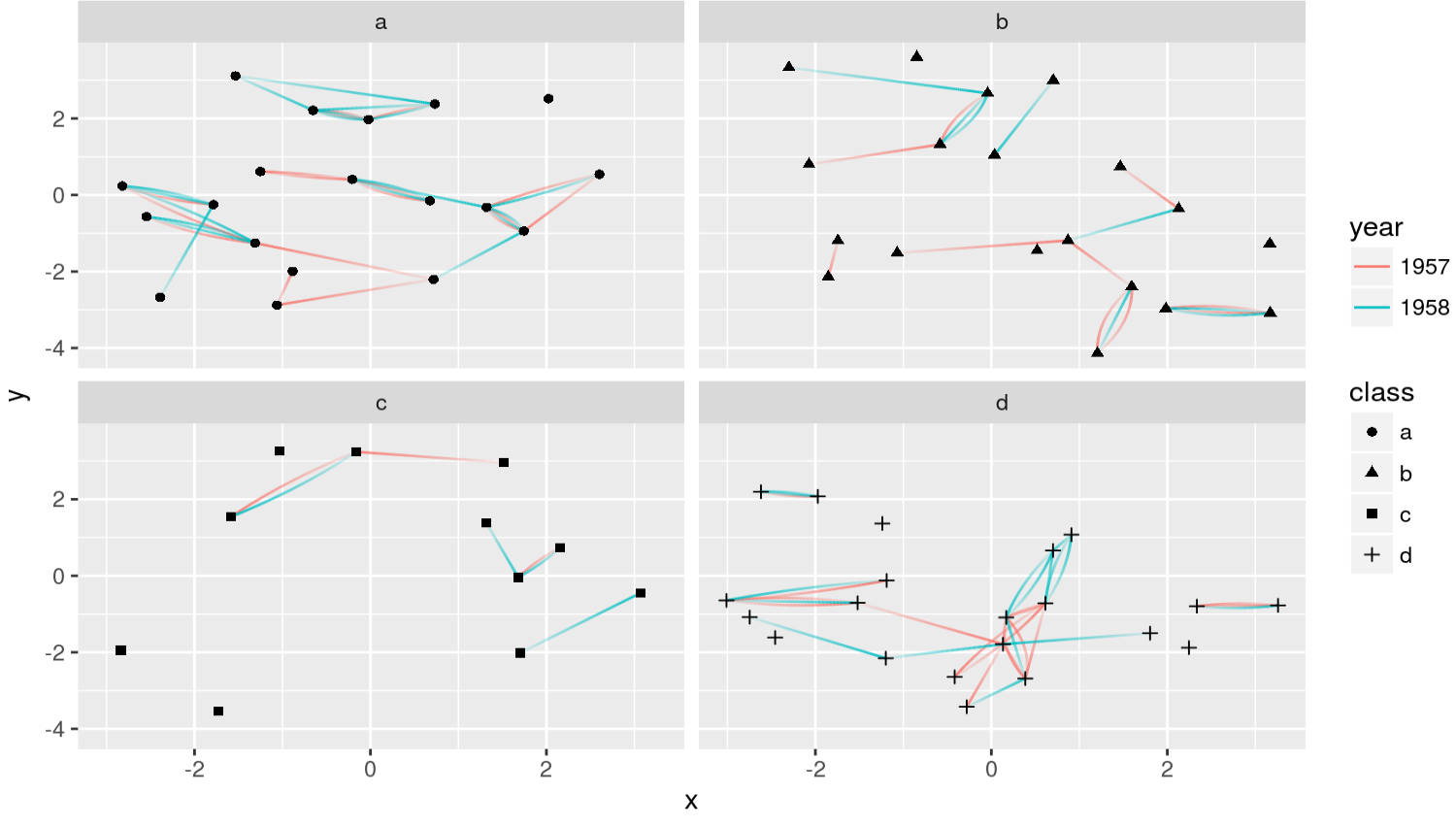
Facetting Edges

```
p + facet_edges(~year)
```



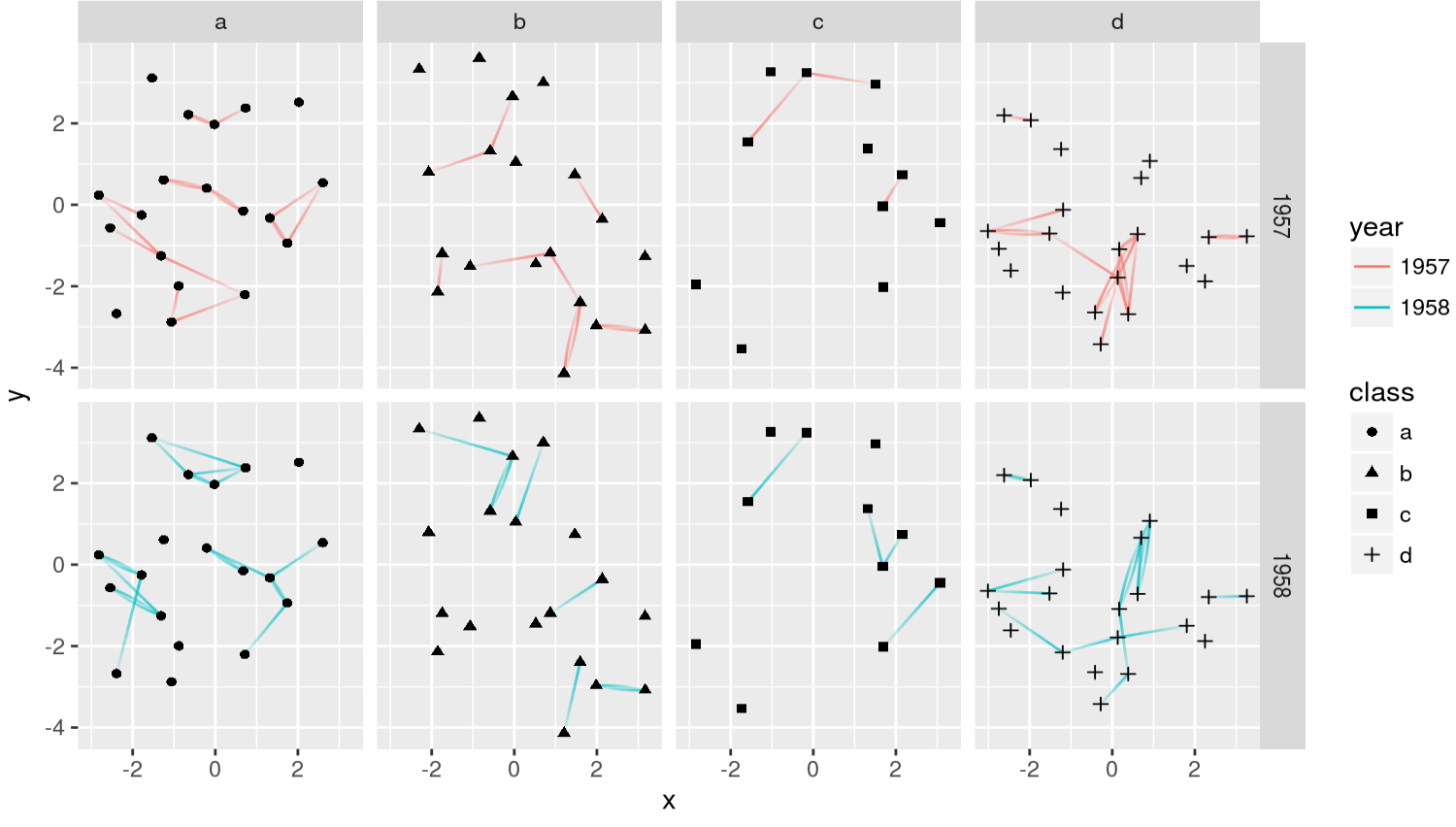
Facetting Nodes

```
p + facet_nodes(~class)
```



Facetting Nodes and Edges

```
p + facet_graph(year ~ class)
```



What next?

- Stay around for the evening event!
- Full course on network analysis including a statistical approach. Contact training@mango-solutions.com.
- Other courses: Introduction to R, shiny apps, package development, visualisation, python and more.
- EARL conference:
  - London: September 11-13
  - US Roadshow (Seattle, Houston, Boston): November 7/9/13