

# RGraph2js: Usage from an R session

*Stephane Cano, Sylvain Gubian, Florian Martin*

November 23, 2025

## Contents

|        |                                            |    |
|--------|--------------------------------------------|----|
| 1      | Introduction . . . . .                     | 2  |
| 1.1    | Technology . . . . .                       | 3  |
| 1.2    | External third-party libraries . . . . .   | 3  |
| 1.3    | Input . . . . .                            | 3  |
| 1.4    | Output . . . . .                           | 4  |
| 2      | Examples . . . . .                         | 5  |
| 2.1    | Simple Example . . . . .                   | 5  |
| 2.2    | Visual appearance . . . . .                | 6  |
| 2.3    | Fixed node positons . . . . .              | 9  |
| 2.4    | Time data or categories . . . . .          | 10 |
| 2.5    | Rendering barplots inside nodes . . . . .  | 12 |
| 2.6    | Rendering starplots inside nodes . . . . . | 14 |
| 2.7    | Customizing the tooltip content . . . . .  | 16 |
| 2.8    | Use the DOT description language . . . . . | 18 |
| 2.9    | Use a graph class . . . . .                | 19 |
| 3      | Interactions . . . . .                     | 20 |
| 3.1    | Using the bottom panel buttons . . . . .   | 20 |
| 3.1.1  | Search . . . . .                           | 20 |
| 3.1.2  | About dialog . . . . .                     | 20 |
| 3.1.3  | Reload . . . . .                           | 20 |
| 3.1.4  | Layout settings . . . . .                  | 20 |
| 3.1.5  | Export . . . . .                           | 21 |
| 3.1.6  | Zoom . . . . .                             | 21 |
| 3.1.7  | Leading nodes . . . . .                    | 21 |
| 3.1.8  | Dragging nodes . . . . .                   | 22 |
| 3.1.9  | Node neighbors . . . . .                   | 22 |
| 3.1.10 | Tooltips . . . . .                         | 23 |
| 3.1.11 | Magnify . . . . .                          | 23 |
| 3.2    | Using the Mouse . . . . .                  | 23 |

# 1 Introduction

*RGraph2js* provides a powerful HTML visualizer to navigate and manipulate graphs/networks. This package has been designed to display results from in-house algorithms on biological networks [1], where it is required to associate a plot for each node [2]. The package is not limited to this specific usage since it is a general tool to visualize various types of networks. *RGraph2js* is highly customizable and offers a user-friendly interface.

Included features are:

- Interactive visualization tool (pan, zoom)
- Customizable appearance
- Customizable graph layout
- Different node connection types support
- Tooltips support
- Node dragging
- Export as a Scalable Vector Graphics (SVG<sup>1</sup>) image
- Barplots and starplots displayable inside the nodes
- Compatibility with most platforms and browsers
- The generated interactive graph can be easily shared

*RGraph2js* takes the description of a graph/network as input and generates an HTML page the user can open in any recent web browser with SVG (Scalable Vector Graphics) rendering support to visualize it and interact with it.

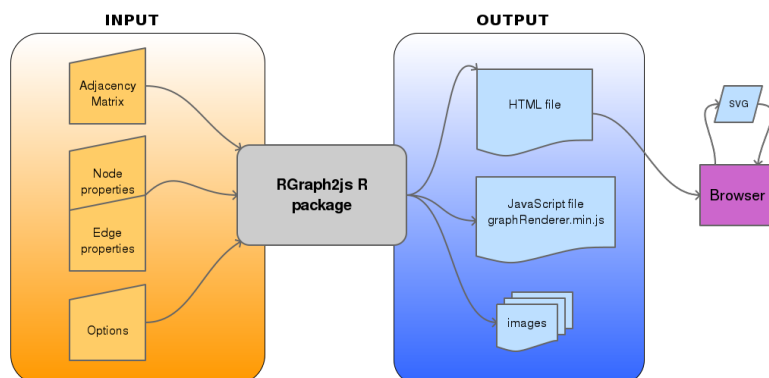


Figure 1: Overview

<sup>1</sup>[https://en.wikipedia.org/wiki/Scalable\\_Vector\\_Graphics](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics)

## RGraph2js: Usage from an R session

### 1.1 Technology

The *D3js*[3] (Data-Driven Documents) JavaScript library is used to render graphs/networks. *Raphael*[5] is another JavaScript library used to render specific in-nodes plots like starplots. *JQuery*[4] and *JQueryUI*[6] are used for the graphical interface and the user interactions. *qTip2*[7], a *JQuery* plugin, is used to render advanced tooltips. A SVG (Scalable Vector Graphics) capable browser is required since both *D3js* and *Raphael* generate SVG code.

*Comment: An Internet connection is required in order to use external third-party JavaScript libraries, further information is given in the next section*

### 1.2 External third-party libraries

*D3js*, *JQuery*, *JQueryUI*, *qTip2* and *Raphael* are used via *CDNJS*, the links are:

<http://cdnjs.cloudflare.com/ajax/libs/jquery/1.11.0/jquery.min.js>

<http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.10.3/jquery-ui.min.js>

<http://cdnjs.cloudflare.com/ajax/libs/qttip2/2.2.0/basic/jquery.qtip.min.js>

<http://cdnjs.cloudflare.com/ajax/libs/qttip2/2.2.0/basic/imagesloaded.pkg.min.js>

<http://cdnjs.cloudflare.com/ajax/libs/d3/3.5.6/d3.min.js>

<http://cdnjs.cloudflare.com/ajax/libs/raphael/2.1.4/raphael-min.js>

<http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.10.3/css/base/minified/jquery-ui.min.css>

<http://cdnjs.cloudflare.com/ajax/libs/qttip2/2.2.0/basic/jquery.qtip.min.css>

The above URLs are declared in the function `RGraph2js::getDefaultToolParameters()`.

### 1.3 Input

The graph/network is defined with a signed and weighted adjacency matrix or with the following R objects from the *graph* package:

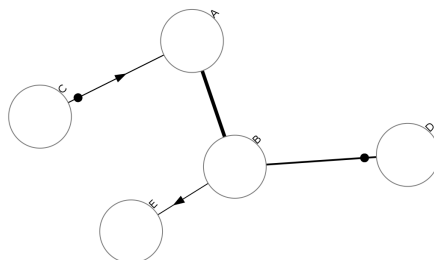
- `graphAM`
- `graphBAM`
- `graphNEL`
- `clusterGraph`

Considering the matrix `a35`:

```
> v <- c(0, 4, 1,
+       1, 0, 0,
+       -1, 0, 0,
+       0, -2, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

... its graphical representation would be as follows:

## RGraph2js: Usage from an R session



**Figure 2:** Graph representation

|   | A | B | C  | D  | E |
|---|---|---|----|----|---|
| A | 0 | 1 | -1 | 0  | 0 |
| B | 4 | 0 | 0  | -2 | 1 |
| C | 1 | 0 | 0  | 0  | 0 |

**Table 1:** 3x5 Signed Weighted Adjacency Matrix

Reading the adjacency matrix by rows, gives the following links/edges:

Line 1: [A  $\rightarrow$  B], [A  $\bullet$  C]

Line 2: [B  $\rightarrow$  A], [B  $\bullet$  D], [B  $\rightarrow$  E]

Line 3: [C  $\rightarrow$  A]

In the adjacency matrix, a value of

- 0 means "no connection"
- 1 " $\rightarrow$ " "arrow, directional connection"
- -1 " $\bullet$ " "dot, directional connection"

*Comment: Any bidirectional connection of the same type implies an undirected link marked as "-"*

[A  $\rightarrow$  B] and [B  $\rightarrow$  A]  $\Rightarrow$  [A - B]

*Comment: Any loop connection, when a node connects with itself, will not be graphically represented*

*Comment: Edges weights can be directly specified in the adjacency matrix as real numbers*

## 1.4 Output

The result files will be made available in a temporary folder or in a specified folder of your choice. The folder will contain:

- A folder for the images
- The main HTML file
- A JavaScript library

## RGraph2js: Usage from an R session

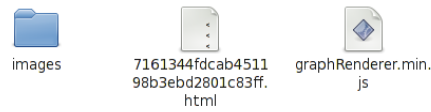


Figure 3: Output folder content

## 2 Examples

### 2.1 Simple Example

This example will show the basics, we will generate a simple network given an adjacency matrix.

Define the adjacency matrix `a1515`:

```
> library(RGraph2js)
> v <- c(1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
+       1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,
+       1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
+       0,1,0,1,1,0,0,0,0,0,0,1,0,0,0,
+       0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,
+       0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,
+       0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,
+       0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
+       0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,
+       0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
+       0,0,0,1,0,0,0,0,0,0,0,1,1,1,0,
+       0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
+       0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
+       0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,
+       0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
+ )
> a1515 <- matrix(v, 15, 15)
> colnames(a1515) <- LETTERS[1:15]
> rownames(a1515) <- LETTERS[1:15]
```

Define the output destination folder `outputDir` and generate the graph with the function `graph2js()`

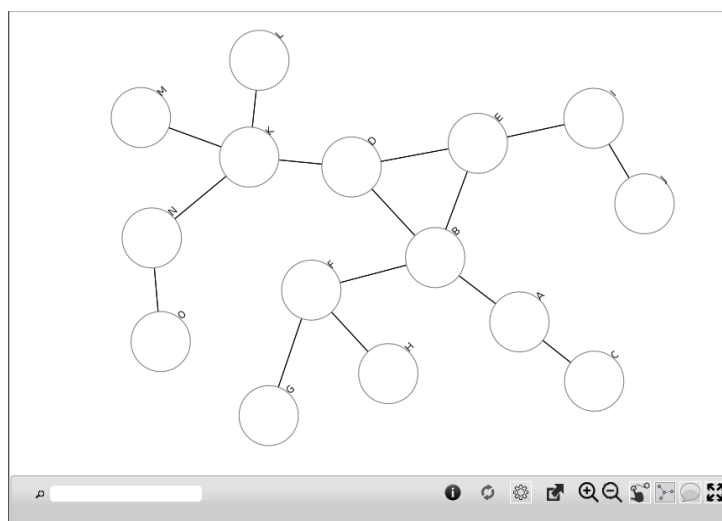
```
> outputDir <- file.path(tempdir(), "RGraph2js_simpleExample")
> g <- graph2js(a1515, outputDir=outputDir)
```

Open the `outputDir` in your browser and click on the html file. You should be able to see something similar to this:

## RGraph2js: Usage from an R session

|   | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| E | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| J | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| K | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| M | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| O | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**Table 2:** 15x15 Adjacency matrix



**Figure 4:** Simple example

## 2.2 Visual appearance

In the previous example, we only specified the adjacency matrix. This example will show how to customize the visual appearance of both nodes and links.

The properties of edges (links) can be specified globally or for each edge. `edgesGlobal` below is applied to all edges, where `edgesProp` is only applied to the edges D to E, D to B and B to E.

```
> edgesGlobal <- list(width=2, color="#0000ff")
> edgesProp <- data.frame(from=c("D", "D", "B"),
+                           to=c("E", "B", "E"),
+                           width=c(5, 5, 5))
```

## RGraph2js: Usage from an R session

We can also define `edgesProp` by starting with its default value returned by the function `getEdgesDataFrame()`. The first column is an automatically generated unique identifier, followed by the columns *from*, *to* and *type* which are automatically set according to the given adjacency matrix `a1515`. *color*, *width*, *link* and *tooltip* have default values we can customize.

- *color* : edge color formatted as hex RGB
- *width* : edge tickness
- *link* : URL associated with the edge
- *tooltip* : tooltip content with HTML support

```
> getEdgesDataFrame(a1515)
```

|                                   | from | to | type | width | color   | link | tooltip |
|-----------------------------------|------|----|------|-------|---------|------|---------|
| 44849e10a384ca97bce1cfd10bfbba49  | A    | A  | --   | 1     | #000000 |      |         |
| 7729a246cea8bb6f94f6babde4580443  | A    | B  | --   | 1     | #000000 |      |         |
| 3582f418dccb4116d85bf4ebbc8a43dc  | A    | C  | --   | 1     | #000000 |      |         |
| 5f3376f6c060c458f0bc83dde6011158  | B    | A  | --   | 1     | #000000 |      |         |
| 39ad8dba0347ec71d5010c01b91161b9  | B    | B  | --   | 1     | #000000 |      |         |
| 8435b69634225e372b5389b2a146aef8  | B    | D  | --   | 1     | #000000 |      |         |
| 399b9dcabd8470e1989d539694cd2aab  | B    | E  | --   | 1     | #000000 |      |         |
| bad5ddd9aacc6cd33e51ad4cde7e3991  | B    | F  | --   | 1     | #000000 |      |         |
| d2b15200cf583e212151f01d8e740438  | C    | A  | --   | 1     | #000000 |      |         |
| 1a7cf664997bae2b944e1fb8ece4abc8  | D    | B  | --   | 1     | #000000 |      |         |
| 62435036701687344dd7825f2394fa49  | D    | D  | --   | 1     | #000000 |      |         |
| 05f671eada96e9ccebdcf011c913c613  | D    | E  | --   | 1     | #000000 |      |         |
| f3bf8a0333bff715b99e18115adde968  | D    | K  | --   | 1     | #000000 |      |         |
| 5b049077fe3f9d2b1a06ee88d8b0499a  | E    | B  | --   | 1     | #000000 |      |         |
| 34a81d610eeed20331ea7e096ac1e2de  | E    | D  | --   | 1     | #000000 |      |         |
| 71f31c02b22bb9212b41f702d1c8f876  | E    | E  | --   | 1     | #000000 |      |         |
| 5cde14c8755685a7c65147fb3702019e  | E    | I  | --   | 1     | #000000 |      |         |
| f43411fdb5c7e9a9430af4f2755ff3dd  | F    | B  | --   | 1     | #000000 |      |         |
| dd23b527644ed7642f181cce72cf392f  | F    | G  | --   | 1     | #000000 |      |         |
| 6a0a2e5e59a224255ad4b434cfbaf1fb  | F    | H  | --   | 1     | #000000 |      |         |
| 6744176738581d44180b8640d435f534  | G    | F  | --   | 1     | #000000 |      |         |
| f77f3018d1792ebe0bcaf81118219379  | G    | G  | --   | 1     | #000000 |      |         |
| b1b7ca5d0f5f33195e39c8cbc98f801a  | H    | F  | --   | 1     | #000000 |      |         |
| e06c9b4d801061c9d6b215218be309ba  | I    | E  | --   | 1     | #000000 |      |         |
| 32030f5488046b2337498431b81b323d  | I    | I  | --   | 1     | #000000 |      |         |
| 72879760c86d63ad8dbf5a49c0168fc4  | I    | J  | --   | 1     | #000000 |      |         |
| 85766046803131f1923feceeeea944dc  | J    | I  | --   | 1     | #000000 |      |         |
| a2ab2975600c198d7f163adc009237a7  | K    | D  | --   | 1     | #000000 |      |         |
| 7450b393b94b7dffdff2c3b7dc147a111 | K    | L  | --   | 1     | #000000 |      |         |
| 3af9e129a12b06895c61c607851b5478  | K    | M  | --   | 1     | #000000 |      |         |
| d73b8adba780a7497023f42ff74d1534  | K    | N  | --   | 1     | #000000 |      |         |
| dbcedb68807cb9cbf465da7a91aa8d83  | L    | K  | --   | 1     | #000000 |      |         |
| 687ab195fbe14395b5ac83354e28f219  | M    | K  | --   | 1     | #000000 |      |         |
| 9689338efd576a8c53c80f92767d1d6a  | N    | K  | --   | 1     | #000000 |      |         |
| d05e499428d1837a3fa08369b31cc0c3  | N    | O  | --   | 1     | #000000 |      |         |
| 93798bf5ed37d6f329832e7d00d7f287  | O    | N  | --   | 1     | #000000 |      |         |
| 4ecf57c0fcc911300358f7502e906824  | O    | O  | --   | 1     | #000000 |      |         |

Similarly, node properties can be global or specific.

## RGraph2js: Usage from an R session

```
> nodesGlobal <- list(color="#ebebeb")
> nodesProp <- data.frame(shape=c("triangle", "lozenge", "rect"),
+                           color=c("#ff0000", "#0000ff", "#ffff00"))
> rownames(nodesProp) <- c("C", "E", "G")
```

Since `nodesProp` holds node specific properties, row names are mandatory. We can call the `getNodeDataFrame()` to define `nodesProp`. The returned data frame contains default values for each node.

- *color* : color of the node in hex RGB format
- *shape* : the shape to use ("rect", "circle", "lozenge", "triangle")
- *link* : URL associated with the node
- *tooltip* : tooltip content with HTML support

```
> getNodeDataFrame(A=a1515, nGlobal=nodesGlobal, nProp=nodesProp)
```

|   | width | color   | shape    | link | tooltip |
|---|-------|---------|----------|------|---------|
| A | 1     | #ebebeb | circle   |      |         |
| B | 1     | #ebebeb | circle   |      |         |
| C | 1     | #ff0000 | triangle |      |         |
| D | 1     | #ebebeb | circle   |      |         |
| E | 1     | #0000ff | lozenge  |      |         |
| F | 1     | #ebebeb | circle   |      |         |
| G | 1     | #ffff00 | rect     |      |         |
| H | 1     | #ebebeb | circle   |      |         |
| I | 1     | #ebebeb | circle   |      |         |
| J | 1     | #ebebeb | circle   |      |         |
| K | 1     | #ebebeb | circle   |      |         |
| L | 1     | #ebebeb | circle   |      |         |
| M | 1     | #ebebeb | circle   |      |         |
| N | 1     | #ebebeb | circle   |      |         |
| O | 1     | #ebebeb | circle   |      |         |

Call the `graph2js()` function as before and specify both nodes and edges properties.

```
> outputDir <- file.path(tempdir(), "RGraph2js_visualAppearance")
> g <- graph2js(a1515,
+               nodesGlobal=nodesGlobal, edgesGlobal=edgesGlobal,
+               nodesProp=nodesProp, edgesProp=edgesProp,
+               outputDir=outputDir, file="index.html")
```

Going further, several options can be changed via the `opts` parameter of the function `graph2js()`. `opts` defaults to the value returned by the function `getDefaultOptions()`.

Please check out the manual for further details.

## RGraph2js: Usage from an R session

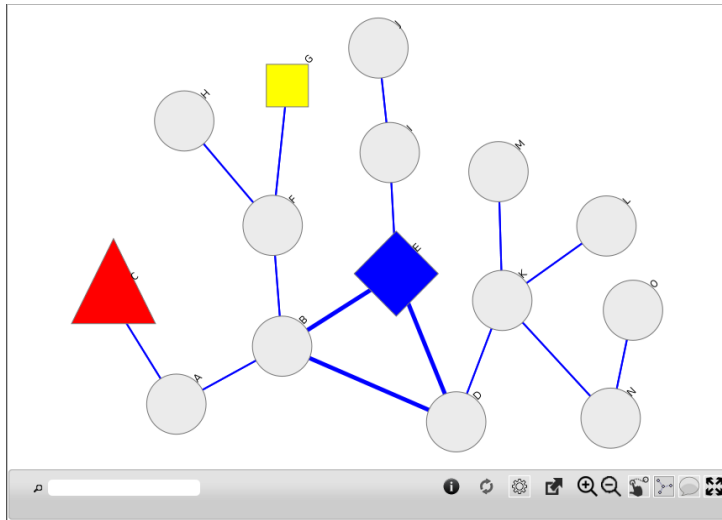


Figure 5: Appearance customized

### 2.3 Fixed node positions

We start from a simple adjacency matrix:

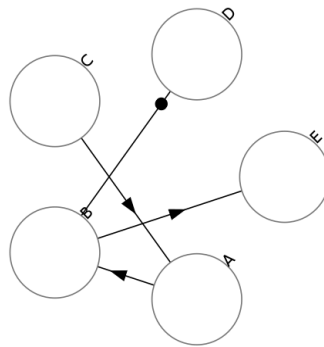
```
> v <- c(0, 0, 1,
+       1, 0, 0,
+       0, 0, 0,
+       0, -1, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

Then, we specify node coordinates via the node properties. `x` and `y` represent the Cartesian coordinates, and `fixed` means they are immutable.

```
> r <- 100
> sector <- 2*pi/5
> n.prop <- data.frame(
+   x=c(r*cos(1*sector), r*cos(2*sector), r*cos(3*sector),
+       r*cos(4*sector), r*cos(5*sector)),
+   y=c(r*sin(1*sector), r*sin(2*sector), r*sin(3*sector),
+       r*sin(4*sector), r*sin(5*sector)),
+   fixed=c(TRUE,TRUE,TRUE,TRUE,TRUE)
+ )
> rownames(n.prop) <- c("A","B","C","D","E")
```

Now, we render the graphics.

```
> outputDir <- file.path(tempdir(), "RGraph2js_fixedNodes")
> g <- graph2js(a35, nodesProp=n.prop, outputDir=outputDir)
```



**Figure 6:** Fixed Node Coordinates Graph

## 2.4 Time data or categories

*RGraph2js* implements a time-step functionality where, at each step, a different set of nodes can be highlighted with thicker borders and different colors. Each time-step is specified by an index starting at 1 and the highlighted nodes and their colors are specified by a dataframe as explained below.

Take for instance the following graph definition:

```
> v <- c(0, 0, 1,
+       1, 0, 0,
+       0, 0, 0,
+       0, -1, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

We specify 4 time-steps in the dataframe below using 2 prefixes:

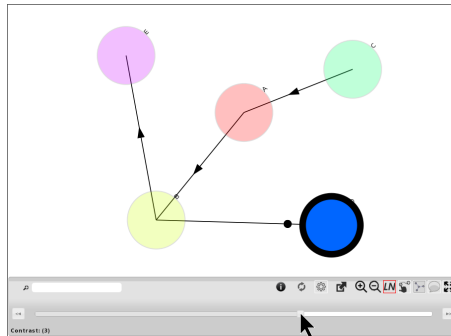
- `leading.nodes.index` specifies the nodes to highlight with thicker border
- `highlight.index` specifies the colors for those leading nodes.

```
> numnodes <- 5
> nodesProp <- data.frame(leading.nodes.1=rbinom(numnodes, 1, 1/2),
+                          leading.nodes.2=rbinom(numnodes, 1, 1/2),
+                          leading.nodes.3=rbinom(numnodes, 1, 1/2),
+                          leading.nodes.4=rbinom(numnodes, 1, 1/2),
+                          highlight.1=rainbow(numnodes),
+                          highlight.2=rainbow(numnodes),
+                          highlight.3=rainbow(numnodes),
+                          highlight.4=rainbow(numnodes))
> rownames(nodesProp) <- LETTERS[1:5]
```

Rendering the Graph leads to:

## RGraph2js: Usage from an R session

```
> outputDir <- file.path(tempdir(), "RGraph2js_timeData")  
> g <- graph2js(a35,  
+             nodesProp=nodesProp,  
+             outputDir=outputDir)
```



**Figure 7:** 4 Different states

Clicking on the **LN** button will expand a new panel at the bottom containing a slider to navigate accross the time steps.

### 2.5 Rendering barplots inside nodes

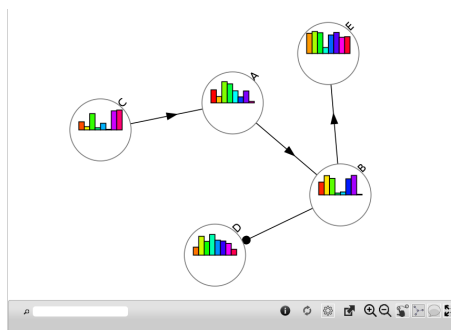
```
> v <- c(0, 0, 1,
+       1, 0, 0,
+       0, 0, 0,
+       0, -1, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

The `innerValues` parameter allows us to specify a barplot for each node and `innerColors` represent the bar colors. The order in both parameters is important for the barplots rendering.

```
> numnodes <- 5
> innerValues <- matrix(runif(numnodes * 8), numnodes, 8)
> rownames(innerValues) <- LETTERS[1:5]
> innerColors <- matrix(rainbow(numnodes * 8), numnodes, 8)
> rownames(innerColors) <- LETTERS[1:5]
```

```
> outputDir <- file.path(tempdir(), "RGraph2js_barplots")
> g <- graph2js(a35,
+              innerValues=innerValues,
+              innerColors=innerColors,
+              outputDir=outputDir)
```

After rendering, here is the result:

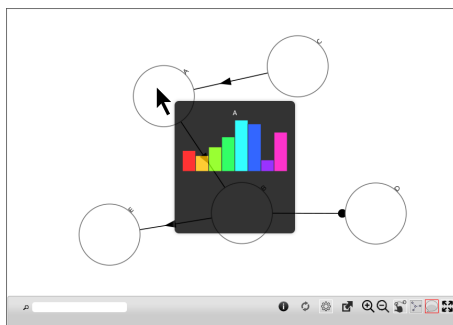


**Figure 8:** Barplots inside nodes

An alternate solution would be to display the barplot inside the node tooltips only, as shown below:

```
> opts <- getDefaultOptions()
> opts$displayBarPlotsInsideNodes <- FALSE
> opts$barplotInNodeTooltips <- TRUE
> g <- graph2js(a35,
+              opts=opts,
+              innerValues=innerValues,
+              innerColors=innerColors,
+              outputDir=outputDir)
```

## RGraph2js: Usage from an R session



**Figure 9:** Barplots in tooltips only

## 2.6 Rendering starplots inside nodes

```
> v <- c(0, 0, 1,
+       1, 0, 0,
+       0, 0, 0,
+       0, -1, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

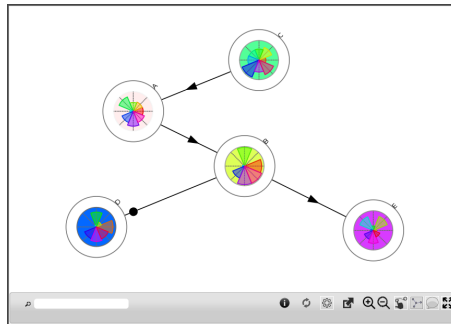
Define a starplot for each node. Each starplot has several parameters: the value (which determines its radius), the opacity, the color, the label, a URL and a tooltip. Additionally, a color and opacity can be specified for the starplot background.

```
> numnodes <- 5
> starplotValues <- matrix(runif(numnodes * 8), numnodes, 8)
> rownames(starplotValues) <- LETTERS[1:5]
> starplotColors <- matrix(rainbow(numnodes * 8), numnodes, 8)
> rownames(starplotColors) <- LETTERS[1:5]
> labels <- c("Sector1", "Sector2", "Sector3", "Sector4",
+            "Sector5", "Sector6", "Sector7", "Sector8")
> starplotLabels <- matrix(labels, numnodes, 8)
> rownames(starplotLabels) <- LETTERS[1:5]
> starplotTooltips <- matrix(labels, numnodes, 8)
> rownames(starplotTooltips) <- LETTERS[1:5]
> # add a url link for each sector
> urls <- c("http://d3js.org/", "http://jquery.com/",
+          "http://jqueryui.com/", "http://qtip2.com/",
+          "http://raphaeljs.com/", "http://www.bioconductor.org/",
+          "http://cran.r-project.org", "http://journal.r-project.org")
> starplotUrlLinks <- matrix(urls, numnodes, 8)
> rownames(starplotUrlLinks) <- LETTERS[1:5]
> starplotCircleFillColor <- matrix(rainbow(numnodes), numnodes, 1)
> rownames(starplotCircleFillColor) <- LETTERS[1:5]
> starplotCircleFillOpacity <- matrix(runif(numnodes,0,1), numnodes, 1)
> rownames(starplotCircleFillOpacity) <- LETTERS[1:5]
```

Render the Graph:

```
> outputDir <- file.path(tempdir(), "RGraph2js_starplots")
> output.filename <- "test.html"
> g <- graph2js(A=a35,
+               starplotColors=starplotColors,
+               starplotLabels=starplotLabels,
+               starplotValues=starplotValues,
+               starplotTooltips=starplotTooltips,
+               starplotUrlLinks=starplotUrlLinks,
+               starplotCircleFillColor=starplotCircleFillColor,
+               starplotCircleFillOpacity=starplotCircleFillOpacity,
+               outputDir=outputDir,
+               filename=output.filename)
```

## RGraph2js: Usage from an R session



**Figure 10:** Starplots

*Comment: Moving the mouse over the sectors will display a tooltip showing the sector name or label*

*Comment: Clicking on a sector will open the associated URL*

## 2.7 Customizing the tooltip content

```
> v <- c(0, 0, 1,
+       1, 0, 0,
+       0, 0, 0,
+       0, -1, 0,
+       0, 1, 0)
> a35 <- matrix(v, 3, 5)
> colnames(a35) <- LETTERS[1:5]
> rownames(a35) <- LETTERS[1:3]
```

The content of the tooltip can be defined with the `nodesProperties` parameter which fully supports HTML content. *Comment: We can even add images like any other HTML content*

```
> numnodes <- 5
> someHtmlContent <- c(paste0("<table class=\"gridtable\">",
+                             "<tr><th>Header 1</th><th>Header 2</th><th>",
+                             "Header 3</th></tr><tr><td>Text 1,1</td><td>",
+                             "Text 1,2</td><td>Text 1,3</td></tr><tr><td>",
+                             "Text 2,1</td><td>Text 2,2</td><td>Text 2,3",
+                             "</td></tr></table>"),
+                       "This is another <i>content</i>",
+                       "Yet another <font style=\"color:#00ff00;\">one</font>",
+                       paste0("<table>",
+                               "<tr><th>Header 1</th><th>Header 2</th><th>",
+                               "Header 3</th></tr><tr><td>Text 1,1</td><td>",
+                               "Text 1,2</td><td>Text 1,3</td></tr><tr><td>",
+                               "Text 2,1</td><td>Text 2,2</td><td>Text 2,3",
+                               "</td></tr></table>"),
+                       "<h1>Header 1</h1><h2>Header 2</h2>")
> n.prop <- data.frame(tooltip=someHtmlContent)
> rownames(n.prop) <- LETTERS[1:5]
```

Since we specified a custom style *gridtable*, we can define it that way:

```
> userCssStyles <- "
+ <style type=\"text/css\">
+ table.gridtable {
+   font-family: verdana,arial,sans-serif;
+   font-size:11px;
+   color:#333333;
+   border-width: 1px;
+   border-color: #666666;
+   border-collapse: collapse;
+ }
+ table.gridtable th {
+   border-width: 1px;
+   padding: 8px;
+   border-style: solid;
+   border-color: #666666;
+   background-color: #dedede;
+ }
```

## RGraph2js: Usage from an R session

```
+ table.gridtable td {  
+   border-width: 1px;  
+   padding: 8px;  
+   border-style: solid;  
+   border-color: #666666;  
+   background-color: #ffffff;  
+ }  
+ </style>  
+ "
```

Render the Graph and provide custom styles with the `userCssStyles` parameter:

```
> outputDir <- file.path(tempdir(), "RGraph2js_tooltipContent")  
> g <- graph2js(a35,  
+   opts=opts,  
+   nodesProp=n.prop,  
+   userCssStyles=userCssStyles,  
+   outputDir=outputDir)
```

The 5 tooltips will be rendered as follows:

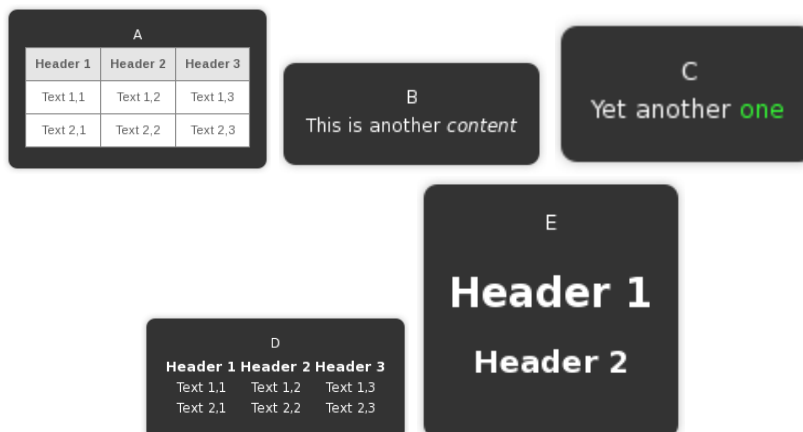


Figure 11: Custom Tooltips

### 2.8 Use the DOT description language

This example requires the [sna](#)[8] package which allows us to easily get an adjacency matrix from a DOT <sup>2</sup>[9] file.

```
> library(sna)
> extdata.path <- file.path(path.package(package="RGraph2js"), "extdata")
> dot.file.path <- file.path(extdata.path, "nohosts.dot")
> adj.mat <- read.dot(dot.file.path)
```

Since the graph is rather large, we can save computing resources by displaying the graph every 100 iterations only, with the option `displayNetworkEveryNLayoutIterations`. Setting it at "zero" would mean to display the graph upon completion only.

```
> opts <- getDefaultOptions()
> opts$displayNetworkEveryNLayoutIterations <- 100
> opts$displayNodeLabels <- FALSE
> opts$layout_forceCharge <- -2400
> nodesGlobal <- list(color="#5544ff")
> outputDir <- file.path(tempdir(), "RGraph2js_dot")
> g <- graph2js(A=adj.mat,
+               nodesGlobal=nodesGlobal,
+               opts=opts,
+               outputDir=outputDir)
```

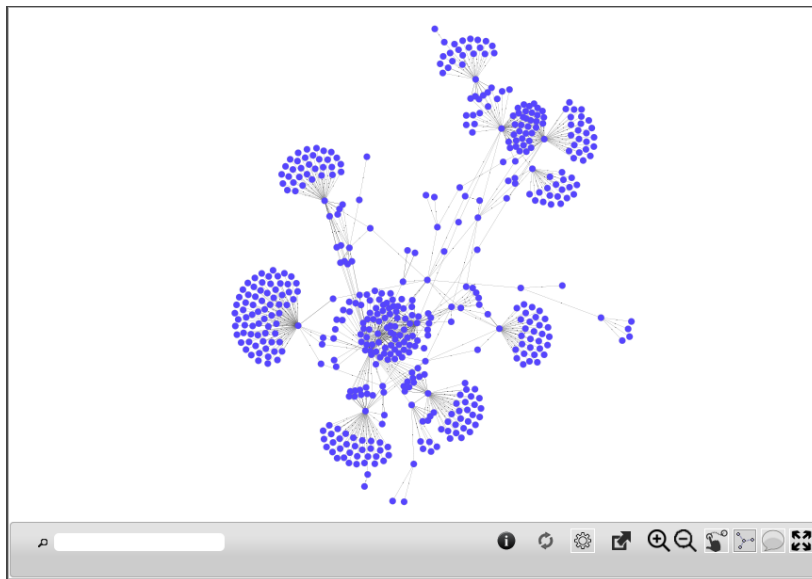


Figure 12: Generate a network from a DOT file

<sup>2</sup><http://www.graphviz.org/doc/info/lang.html>

### 2.9 Use a graph class

Instead of specifying an adjacency matrix, you can pass a [graph](#) class.

Here is an example with a `graphNEL` object `gnel`:

```
> library(graph)
> nodes <- c("A", "B", "C", "D", "E")
> edges <- list(
+   A=list(edges=c("A", "B"), weights=c(2, 2)),
+   B=list(edges=c("A", "E"), weights=c(0.25, 0.25)),
+   C=list(edges=c("A", "D"), weights=c(4, 4)),
+   D=list(edges=c("E"), weights=c(6)),
+   E=list(edges=c("A", "B"), weights=c(1, 1))
+ )
> gnel <- new("graphNEL", nodes=nodes, edgeL=edges, edgemode="directed")
```

The following shows how to graphically represent edges weights with the [Rgraphviz](#) package. As you can see, some extra steps are required.

```
> ew <- as.character(unlist(edgeWeights(gnel)))
> ew <- ew[setdiff(seq(along = ew), removedEdges(gnel))]
> names(ew) <- edgeNames(gnel)
> eAttrs <- list()
> eAttrs$label <- ew
> plot(gnel,
+   attrs=list(
+     edge=list(arrowsize=0.5)
+   ),
+   edgeAttrs=eAttrs)
```

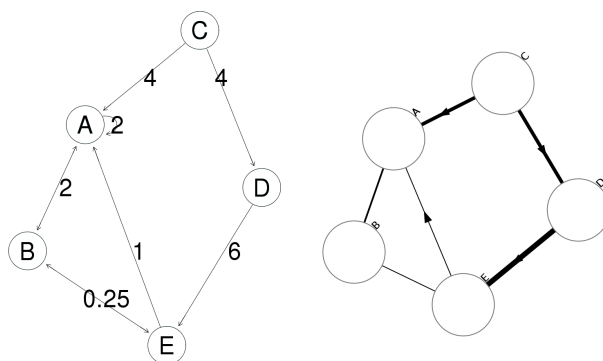
Now, with [RGraph2js](#), edges weights are translated into edges width by default. This default behaviour can be redefined by specifying edges properties.

```
> outputDir <- file.path(tempdir(), "RGraph2js_graphNELExample")
> g <- graph2js(A=gnel, outputDir=outputDir)
```

Please note the following limitations:

- links/edges representations are only translatable into "→" or "—"
- as mentioned earlier, loop connections are not rendered

## RGraph2js: Usage from an R session



**Figure 13:** Comparison of the original `graphNEL` (left) and the `RGraph2js` output (right)

## 3 Interactions

### 3.1 Using the bottom panel buttons

All buttons are described in the next sections.



**Figure 14:** Buttons bottom panel

#### 3.1.1 Search

The search field performs an incremental search on all node labels, highlighting matches with a tick red border. Clearing the search field cancels the search and resets the display.



**Figure 15:** Search field

#### 3.1.2 About dialog

📘 Gives information about the software and its version

#### 3.1.3 Reload

🔄 Re-compute the layout

#### 3.1.4 Layout settings

⚙️ Toggle the sub-panel to customize the layout engine

The parameters the user can control with sliders are:

- Charge

## RGraph2js: Usage from an R session

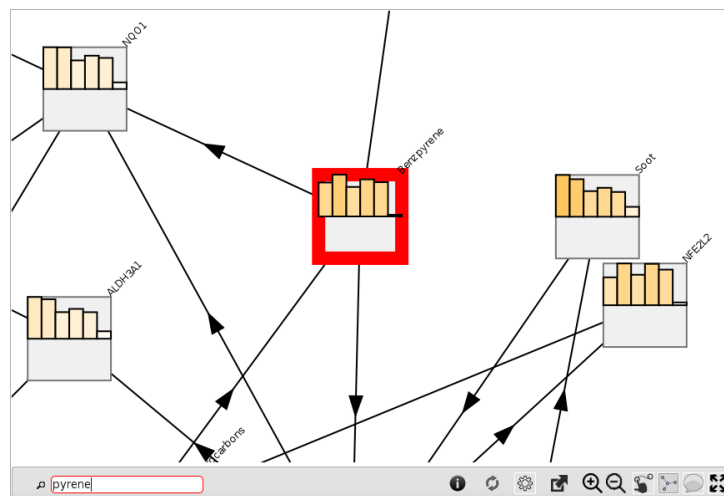


Figure 16: Search feature in action



Figure 17: Layout Settings

- Link distance

More details about the force layout can be found on the D3js wiki <sup>3</sup>.

### 3.1.5 Export

↳ Lets you export the graph and save it as an SVG image

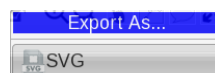


Figure 18: "Export As" popup menu

### 3.1.6 Zoom

🔍🔍 Zoom in/out without using the mouse wheel

### 3.1.7 Leading nodes

**LN** Expand a new panel at the bottom containing a slider to navigate accross the time steps. Please note this button is present only when such data exist.

<sup>3</sup><https://github.com/mbostock/d3/wiki/Force-Layout>

## RGraph2js: Usage from an R session



Figure 19: Layout Settings

### 3.1.8 Dragging nodes

 Toggle the nodes dragging feature

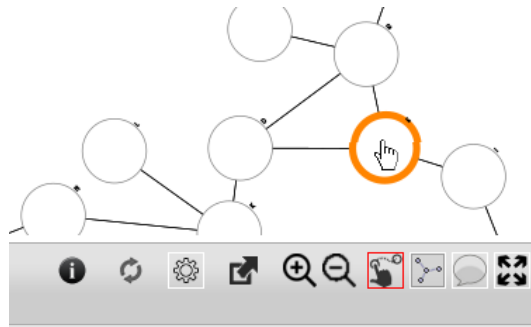


Figure 20: Dragging a node

### 3.1.9 Node neighbors

 Enable the highlight of the neighbors when hovering a node

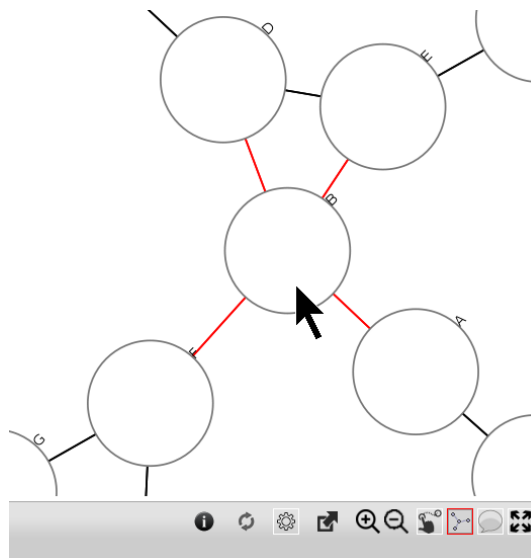


Figure 21: Highlighting of the neighbors

## RGraph2js: Usage from an R session

### 3.1.10 Tooltips

Toggle the display of Tooltips when the mouse hovers a node or an edge

Below is an example of a node tooltip containing the node name with a barplot

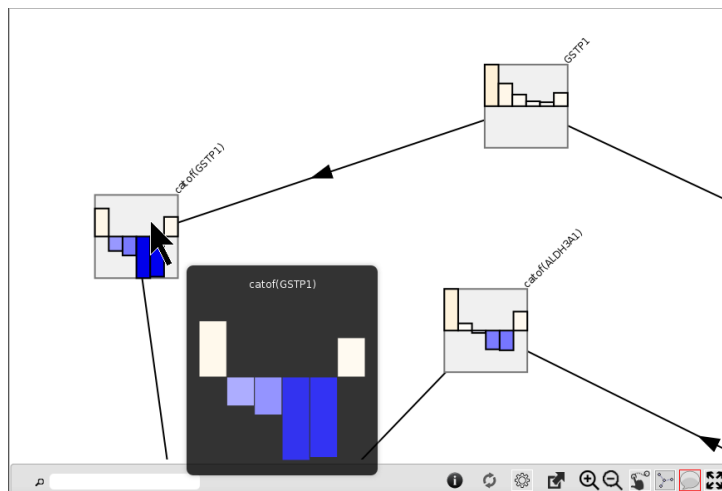


Figure 22: Tooltips

### 3.1.11 Magnify

Magnify the view area to fit to the browser current window size

## 3.2 Using the Mouse

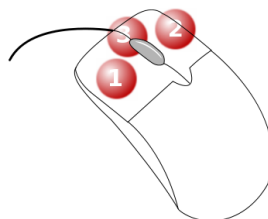


Figure 23: Mouse Buttons

Button (1) is used to drag the whole graph in the drawing area and to drag nodes when the corresponding mode  is activated. Double-clicking performs a zoom in.

Button (2) opens a popup menu when clicking a node.

Button (3), the mouse wheel allows to zoom in and out.

## RGraph2js: Usage from an R session

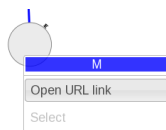


Figure 24: Node Popup Menu

## References

- [1] Florian Martin, Alain Sewer, Marja Talikka, Yang Xiang, Julia Hoeng and Manuel C Peitsch, *Quantification of biological network perturbations for mechanistic insight and diagnostics using two-layer causal models*, BMC Bioinformatics 2014, 15:238. URL <http://www.biomedcentral.com/1471-2105/15/238>
- [2] Poussin Carine, Laurent Alexandra, Peitsch Manuel C., Hoeng Julia and De Leon Hector *Systems Biology Reveals Cigarette Smoke-Induced Concentration-Dependent Direct and Indirect Mechanisms That Promote Monocyte-Endothelial Cell Adhesion.*, Toxicological Sciences Journal, 2015.  
Abstract:  
<http://toxsci.oxfordjournals.org/content/early/2015/08/20/toxsci.kfv137.abstract>  
Eprint:  
<http://toxsci.oxfordjournals.org/content/early/2015/08/20/toxsci.kfv137.full.pdf+html>
- [3] Michael Bostock, Vadim Ogievetsky and Jeffrey Heer, *D3: Data-Driven Documents*, IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011. URL <http://vis.stanford.edu/papers/d3>
- [4] *jQuery*, URL <http://jquery.org/>
- [5] *Raphael*, URL <http://dmitrybaranovskiy.github.io/raphael/>
- [6] *jQueryUI*, URL <http://jqueryui.com/>
- [7] *qTip2*, URL <http://qtip2.com/>
- [8] Mark S. Handcock, David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris, *statnet: Software tools for the Statistical Modeling of Network Data*. URL <http://statnetproject.org>
- [9] Emden R. Gansner and Stephen C. North, *An open graph visualization system and its applications to software engineering.*, SOFTWARE - PRACTICE AND EXPERIENCE Journal, Volume 30 Number 11 pages 1203-1233, 2000. URL <http://www.graphviz.org>
- [10] R. Gentleman and Elizabeth Whalen and W. Huber and S. Falcon, *graph: A package to handle graph data structures*, URL [graph](http://igraph.org/)