ALEKSI KALLIO
JAVASCRIPT VISUALIZATION LIBRARIES: VISUALIZING BIOLOGICAL DATA
Bachelor of Science Thesis

Supervisor: Heikki Huttunen
Instructor: Matti Nykter
ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Biotechnology
KALLIO, ALEKSI: JavaScript visualization libraries: visualizing biological data
Computational Systems Biology
Bachelor of Science Thesis, 21 pages
December 2012
Supervisor: Heikki Huttunen
Instructor: Matti Nykter
Keywords: JavaScript, HTML5, canvas, SVG, data visualization, random forest, RF-ACE

Recent advances in next-generation sequencing technologies have dramatically decreased the cost and effort of DNA sequencing. Next-generation sequencing technologies also offer ways to measure gene expression more efficiently than older technologies. These new technologies have multiplied the amount of omics data by many tenfolds and given rise to bioinformatics. Huge heterogeneous datasets bring problems to finding the significant features and statistically important relationships between features. Usually these datasets also have many more variables than samples and to analyze these problematic heterogeneous datasets, specific data mining algorithms are needed. Therefore data mining has become an important part of any bioinformatic data-analysis.

After the data mining algorithms have been applied to the data there is still a lot of work to be done before any possible scientific breakthroughs can be achieved. The data mining algorithm can produce big outputs without any guides to what is scientifically important and what is not. To easily study this output it should be possible to further search and visualize subsets of the data mining findings. New web based tools combined with JavaScript, HTML5’s canvas feature, or scalable vector graphics (SVG) can produce simple and interactive visualizations quite easily.

There are dozens of JavaScript visualization libraries available for free non-commercial use. In this thesis we discuss six visualization libraries. We also take a brief look to analyzing biological data with Random Forests data mining algorithms.
PREFACE

This thesis is based on the work I did in the summer of 2012 to the Prostate cancer project at the Laboratory of Complex Biosystems Modeling in the Technical University of Tampere. I want to thank my instructor Matti Nykter for helping with my thesis and for providing a job for me, I want to thank my co-workers Jake Lin and Timo Erkkilä for providing help and suggestions for my thesis. I also want to thank Jäger (M.Sc.) for providing a good atmosphere for the last Bachelor thesis seminar meeting.

Tampere, December 16, 2012
Aleksi Kallio
CONTENTS

1 Introduction .......................................................... 1
2 Data mining in bioinformatics ....................................... 2
   2.1 Random forest ...................................................... 2
      2.1.1 Variable importance ....................................... 3
      2.1.2 Significant variable interaction ......................... 4
   2.2 RF-ACE .............................................................. 4
3 JavaScript .............................................................. 6
   3.1 History ............................................................. 6
   3.2 Main characteristics ............................................. 7
   3.3 Object-oriented programming .................................... 7
   3.4 Functionality ....................................................... 9
   3.5 Usage and libraries ............................................... 9
4 Graphics rendering methods ......................................... 10
   4.1 SVG ................................................................. 10
   4.2 Canvas .............................................................. 11
5 JavaScript visualization libraries ................................... 13
   5.1 SVG libraries ...................................................... 14
      5.1.1 Raphaël and gRaphaël ..................................... 14
      5.1.2 Data-Driven Documents (D3) .............................. 15
      5.1.3 HighCharts ................................................... 16
   5.2 Canvas libraries .................................................. 16
      5.2.1 Rgraph ........................................................ 16
      5.2.2 InfoVis ......................................................... 16
      5.2.3 CanvasXpress ............................................... 17
6 Comparison ............................................................ 18
References ............................................................... 22
# LIST OF SYMBOLS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Artificial Contrast Ensembles</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>D3</td>
<td>Data-Driven Documents, JavaScript visualization library</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>Omics</td>
<td>A collection of different biological terms ending in omic, e.g. genomics, transcriptomics, and proteomics</td>
</tr>
<tr>
<td>RE</td>
<td>Regulome Explorer</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest classification algorithm</td>
</tr>
<tr>
<td>RF-ACE</td>
<td>Random Forests with Artificial Contrast Ensembles</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

Recent advances in next-generation sequencing technologies have dramatically decreased the cost and effort of DNA sequencing. Next-generation sequencing technologies also offer ways to measure gene expression more efficiently than older microarray technologies although microarrays are still popular. [1] These next-generation sequencing technologies produce huge amounts of data, for example Applied Biosystems SOLiD sequencer produces 3-4 Gb of sequencing data for a single run [2].

These new technologies have multiplied the amount of omics data by many tenfolds, and given rise to field of bioinformatics. Bioinformatics deals with omics data; including data from different omics technologies such as genomics, transcriptomics, proteomics, metabolomics, epigenomics and metagenomics. When all these data sources are combined the result is a huge heterogeneous dataset. This bioinformatic dataset brings problems to finding the significant features from the data. Usually these datasets have so called small \( n \) large \( p \) problem which means that in a dataset there are many more variables than samples, this problem is also known as curse of dimensionality. To analyze these datasets, specific data mining algorithms are needed. [3] Therefore data mining has become an important part of any bioinformatic data-analysis.

After the data mining algorithms have analyzed the data there is still a lot of work to be done before any possible scientific breakthroughs can be achieved. Because these datasets are so huge, the data mining algorithm outputs can also be big and the algorithm does not know what is important to the researcher and what is not. To easily study this output it should be possible to further search, subtract and visualize parts of it. A great way to store and visualize analyzed data is to use web accessible database and interactive visualization methods. Web technologies combined with new HTML5’s canvas feature or Scalable Vector Graphics (SVG) can produce simple and interactive visualizations quite easy. Using these modern databases also allows the data to be integrated across result sets, queried and filtered for different purposes. Another benefit of web accessibility is to enable sharing and collaboration with other researchers. In this thesis we concentrate more to the visualization of the data than data mining algorithms.

This thesis is structured as follows. First we talk briefly about data mining in bioinformatics. After that we turn into web based visualizations by discussing about JavaScript programming language, SVG and Canvas graphics. In chapter 5 we go into more details about a couple of different JavaScript visualization libraries. In chapter 6 we compare these libraries and draw conclusions about their strengths and weaknesses.
2 DATA MINING IN BIOINFORMATICS

Data mining has become an important part of any data analysis as the amount of information has multiplied tremendously and usually it is hard to draw any immediate or obvious statistically important relationships from the data set. Data mining means the process of uncovering interesting characteristics and patterns from a vast data set. Algorithms designed to do this extract useful information and transform it to understandable form. Data mining is particularly important in biological analysis as its datasets are heterogeneous, multi-dimensional and very large.

Data mining methods usually consist of machine learning algorithms that implement a classification of the data. Support vector machines and recently Random Forest (RF) have become popular machine learning algorithms for biological learning and classification research [3]. In this thesis we concentrate on the RF algorithm and a variation of it which is called Random Forests with Artificial Contrast Ensembles (RF-ACE).

2.1 Random forest

RF has some very important and beneficial data mining attributes: high prediction accuracy, can handle numerical and categorical variables, and has information about variable importance to classification. RF models are also non-parametric, hard to over-train, relatively robust and fast to train. [3]

A RF classifier is a collection of tree classifiers. For each tree a training set is selected by selecting a random training set, or bootstrap set, containing about two-thirds of the original data. The remaining samples are called out-of-bag (OOB) samples. A classification tree is constructed by selecting a random set of variables at each node. These variables are evaluated for their ability to split the data and the best variable is selected. This continues until each leaf is either pure, i.e. they contain samples from only one class or contain a certain sample count. Usually a tree is grown until the leaf nodes are pure, even if that results to a single sample in a leaf. [3] Despite that a single tree from the RF is a weak classifier because it is trained from a subset of data, the collection of all trees is a strong classifier [4]. In Figure 2.1 is an illustration of the general idea of random forest.
2. Data mining in bioinformatics

Figure 2.1. The general idea of random forest classification. To classify an input $x$, each tree casts a unweighted vote for the most popular class and $x$ is classified by majority vote rule [4].

Although RF is a strong classifier it also has other significant features for bioinformatics. In this thesis we will discuss more about RFs variable importance and significant variable interaction capabilitys. Other features are discussed in article by Touw et al. [3].

2.1.1 Variable importance

Variable importance score is an estimate of how important a variable is to the classification. This score can then be used to identify important variables e.g. biomarkers and genes or to filter out non-informative variables.

There are two frequently used types of importance score measurement. The mean decrease in classification and GDI measure. The mean decrease in classification is based on permutation of the OOB samples. The classification accuracy of the OOB samples is determined for each tree both with and without random permutation of the values of the variables. The accuracy after permutation is subtracted from the accuracy before the permutation and averaged over all trees to give the mean decrease in classification importance value. [3]

The GDI measure is calculated with the Gini impurity decrease of every node that contains the variable. [5] Gini impurity decrease is defined as follows:

$$giniDecr(n) = i(n) - p_L \cdot i(n_L) - p_R \cdot i(n_R)$$ (2.1)
In which $i(n)$ calculates the Gini Index of node $n$, $n_L$ and $n_R$ refer to child nodes of $n$ and $p_L$ and $p_R$ correspond to the fraction of objects in node $n$ that end up in these nodes. Gini Index is an estimation of the probability of misclassifying an object at a given node. [5] In eq. (2.1) is measured the importance of a single node, to calculate the importance of a variable we need to sum the gini impurity decreases from all the nodes in the forest that contain the variable. This gives us a GDI measure of a variable.

### 2.1.2 Significant variable interaction

Variable interaction means that two variables interact together in a sample classification at least once. These variables can be found by defining a decision rule that explains how samples are classified. A decision rule can be obtained from a tree by defining a path from root to leaf. The entire set of decision rules can then be generated by defining all unique paths through the tree. If two features occur in at least one decision rule, they are considered to interact. [5]

Significant variable interactions can be found by combining variable importance scores to variable interaction. One way to combine these is to extend GDI measure to two interacting variables $X$ and $Y$ as proposed by de Ruiter in [5]. RF-ACE introduces an alternative method which we will talk about in section 2.2. It is good to note that because the variable interaction definition is inclusive it produces a lot of interactions, so we need to use some filtering criteria to determinate the truly significant variables and their interactions.

### 2.2 RF-ACE

RF-ACE is a machine learning algorithm that combines Random Forest with Artificial Contrast Ensembles (ACE). In addition to its classification features, RF-ACE can also do association prediction with variable importance score.

The variable importance score can produce a relative feature ranking but it does not separate relevant features from irrelevant. Trees also tend to split on variables with more distinct values but those might at the same time be completely irrelevant variables. ACE is a method, proposed by Tuv et al. in [6], to find truly informative variables from the data set through random forest. In ACE a set of artificial variables with the same distribution as the original variables but no relevancy to the target are generated and added to data set. This can be done by e.g. randomly permuting variables values. Now any stable feature ranking system should favor a original variable over an artificial variable. This means that a true relevant variable is expected to have higher importance score than an artificially generated variable. A variable can then be said to be truly important if its importance score is statistically significantly higher than the artificial variables importance score. [6]
Unlike in normal RF algorithm in RF-ACE the variable importance scores are translated to p-values which allows for statistically assess differences between associations. This also allows comparison between RF-ACE runs as the normal RF importance score is run iteration specific. RF-ACE can do association prediction by doing a statistical T-test between the target feature and a possible association feature. This creates an association network which structure and statistical aspects can be visualized as we will see in chapter 5. Doing a T-test to tens of thousand genes is computationally very time consuming but RF-ACE scales well to concurrent computer clusters which is important for powerful data mining algorithm. [7]
3 JAVASCRIPT

After data mining is done the result data set can still be massive and to make the analysis of it easier; the findings must be visualized. Nowadays it is useful to do the visualization in the web, which enables other scientists to access and contribute to it more easily. JavaScript is a programming language used in the web pages, so it is a natural choice to visualize data in the web. A majority of modern websites use JavaScript and all modern web browsers on desktop, game consoles, tablets and smart phones have JavaScript interpreters. Thus, JavaScript can be said to be one of the most ubiquitous programming languages in history. [8]

The newest version of JavaScript is actually called ECMAScript. Technically JavaScript refers to language implemented by Netscape and Mozilla Foundation but in practice everyone and also this thesis calls the language JavaScript. JavaScript also has nothing to do with Java other than some syntactical resemblance. [9]

3.1 History

Back in 1993 when Mosaic debuted, the first popular web browser, web became available to everyone and across different platforms. Developers needed a programming language that would work in browser despite the operating system. Hypertext Markup Language (HTML), which only defines how web page should look like, was not sufficient to this. One possibility was Sun’s Java that produced byte code and ran in virtual machine so it did not matter what platform it was run on. But Java was a rich and complex language aimed for professional programmers so Netscape and others also wanted a lightweight scripting language that would appeal to non-professional programmers. [10]

In April 1995 Netscape hired Brendan Eich to create and produce a prototype of lightweight programming language that would run in Netscape’s browser. Eich had only 10 days to complete this task. Eich began his work: he took basic syntax from C language, simplified object model that combined C’s structs, SmallTalks patterns, LISP’s symmetry between data and code and Self’s prototyped object inheritance. Eich got the job done and JavaScript had enough good parts to survive to this day. One of the most important parts was the runtime support for different events on web page, e.g. mouse click on specific item. At the beginning of 21st century JavaScript had been in browsers for a decade but it had not been used to anything spectacular, other than to scrolling messages or flashing images. But then Microsoft added a XMLHTTP-request to Internet Explorer and this started Ajax revolution and JavaScript’s dominance. [10]
XMLHTTP-request allows the browser to ask data from the server and execute piece
of JavaScript code when this data arrives. This allowed dynamic behavior to web pages
as JavaScript could change the HTML document without reloading the whole web page.
Web pages like Google Mail and Maps are all based to JavaScript changing the HTML
document.

3.2 Main characteristics

JavaScript is a scripting language, which implements multiple programming language
paradigms as discussed in section 3.1: Scripting, object-oriented, imperative and func-
tional. Syntactically JavaScript resembles C, C++ and Java, it has same syntax for if and
loop statements and although it is not mandatory, JavaScript statements end in a semicolon
(;
).

JavaScript being a scripting language means that it is interpreted instead of compiled.
This leads to faster development and also allows JavaScript code to be generated at run-
time. For example different functions can be generated depending on user input.

In JavaScript there is no built-in input-output-functionality (I/O) which is a bit un-
usual. Instead the runtime environment, e.g. browser, provides the I/O functionality by a
console.

3.3 Object-oriented programming

JavaScript is a heavily object-oriented language. JavaScript has numbers, strings, booleans
and object variables. Objects are mutable keyed collections so arrays can be represented
also as objects. But numbers, strings and booleans are also object-like so that they have
methods but they are immutable meaning their state can not be changed after creation.
Functions and regular expressions are also object-like in JavaScript. [11]

An object is a container of key value pairs. A key can be any string even an empty
string. Value can be any value even other object or a function. This makes objects very
versatile and able to represent very complex data structures. [11] Program 3.1 shows an
example of JavaScript object. Often JavaScript functions use objects to pass parameters,
these objects follow the structure shown in Program 3.1 which is called JavaScript Object
Notation (JSON).
3. JavaScript

```javascript
var storage = {
  name = "John",
  age=20,
  friends = {
    name="Alice",
    age=20
  }
};
```

Program 3.1. Example of JavaScript object. Object is curly braces surrounding zero or more key value pairs.

In JavaScript object inheritance is called prototype-oriented. Unlike in classical programming languages (e.g. Java or C++) where objects are instances of classes and classes can inherit from other classes, in JavaScript objects inherit directly from other objects. This means that in JavaScript first an object is made with all the properties and methods that are needed, then more objects are made from the old object so that the new objects inherit all the properties and methods of the old object. In Program 3.2 is first made a useful object (myMammal) then more instances are made from that object (myCat) and the new instance is customized by giving it new functions. [11]

```javascript
var myMammal = {
  name : 'Herb the Mammal',
  get_name : function ( ) {
    return this.name;
  },
  says : function ( ) {
    return this.saying || ''; 
  }
};

var myCat = Object.create(myMammal);
myCat.name = 'Henrietta';
myCat.saying = 'meow';
myCat.purr = function (n) {
  var i, s = ''; 
  for (i = 0; i < n; i += 1) {
    if (s) { s += '-'; }
    s += 'r'; 
  }
  return s;
};
myCat.get_name = function ( ) {
  return this.says ( ) + ' ' + this.name + ' ' + this.says( ); 
};
```

Program 3.2. Example of prototypal inheritance from [11].
3. JavaScript

3.4 Functionality

Functions in JavaScript are similar to Java or C++ functions in a way that they take in arguments, are enclosed with curly braces, and can return values. On the other hand functions in JavaScript are like objects so they can be stored in variables and objects. Functions can be passed as arguments or returned by other functions. Functions can also have methods and inherit other functions. This makes functions first-class objects and thus JavaScript a functional programming language. [11]

3.5 Usage and libraries

As discussed in the beginning of chapter 3, JavaScript is the language of the web. JavaScript is usually embedded in HTML to interact with the Document Object Model (DOM) of the page. JavaScript can for example be used to:

- fetch new data from a server via AJAX and update parts of the page without reloading the whole page
- animate page elements, change their color, fade them in or out, move them, etc.
- generate interactive content e.g. games, video or audio, and
- validate user input of a web form.

These are just some examples of JavaScript usage. JavaScript can also be used for server side programming but that isn’t very common. JavaScript being built in 10 days means it has some limitations and drawbacks. Luckily JavaScript is a object-oriented language so it allows developers to add their own libraries to the language which can hide some of the undesirable parts. A simple Google search "JavaScript library" turns out over 81 million results. In chapter 5 we will take a closer look at six visualization libraries which are designed to ease the making of interactive graphs and visualizations on the web page.
4 GRAPHICS RENDERING METHODS

To understand some differences between visualization libraries we must first discuss how graphics are drawn to a web page. JavaScript visualization libraries can be categorized to two groups by how they draw graphics to a web page. Visualization libraries can either append Scalable Vector Graphics (SVG) elements directly to the web pages DOM or draw inside a HTML5 canvas tag.

4.1 SVG

Scalable Vector Graphics is a XML like dialect to describe vector graphics. In SVG, picture consist of a list of paths, basic shapes and text, unlike normal pixel image that consists of colorized pixels. It’s up to the web browser to rasterize the vector image from the XML description. SVG pictures can be very compact compared to a pixel image because only visible shapes needs to be specified and not the whole pixel plane. [12] In Figure 4.1 is an example of basic SVG image which consists of a rectangular object and a path object. The Figure 4.1 was produced by using Program 4.1. In Program 4.1 the object and attribute oriented approach of SVG is plainly visible.

Figure 4.1. Image containing a rectangular object and a path object.
4. Graphics rendering methods

Program 4.1. SVG code used to produce Figure 4.1. On line 2 is defined a rectangular with width, height and style. On line 3 is defined the path for the triangular.

SVG elements behave like normal DOM elements so for example events like mouse-over and mouse-click can be bound to them. This leads to fast event handling and element selection. Normal element behavior also means that all the elements must be in systems memory so if the visualization is complex it reserves a lot of memory.

4.2 Canvas

Canvas is a new feature in HTML5 which is the newest iteration of HTML. Canvas tag \(<\text{canvas}>\) defines a bitmapped area on the web page which can be manipulated through JavaScript. Canvas consists basically from a picture that is re-rendered on every frame from JavaScript code. This differs from SVG where all the objects of the image are stored in the memory and rendered according to their attributes, e.g. width, height, color. [13] With canvas it is also possible to generate 3D visualizations which are difficult to do with SVG. Canvas is compatible with all the modern browsers excluding Internet Explorers before version 8. In Program 4.2 is example JavaScript code to control canvas area. The code produces same picture as Figure 4.1.

Program 4.2. JavaScript code used to produce Figure 4.1. On lines 3-5 is defined the rectangular and on lines 7-12 is defined the triangle path with stroke and fill.

Canvas being a still image means that binding events directly to elements in the picture is impossible. To mimic this behavior canvas has to track mouse coordinates and compare
4. Graphics rendering methods

them to element coordinates, this can be time consuming and complex if there are a lot of elements in the visualization. On the other hand canvas is just a still image so it requires much less systems memory than SVG in complex visualizations with many elements.
5 JAVASCRIPT VISUALIZATION LIBRARIES

JavaScript visualization libraries are JavaScript libraries that are dedicated to different kinds of visualizations. The most popular ways of visualization are through SVG or canvas. To fully analyze data mining output, algorithm specific visualizations are needed. In Figure 5.1 is shown a visualization from a tool called Regulome Explorer (RE), which is developed in collaboration between Institute of Systems Biology and Technical University of Tampere. With RE it is possible to explore RF-ACE outputs using influential circos-like [14] graphs interactively. In Figure 5.1 can be seen associations between different genomic features throughout the genome and genes positions in the genome. The visualization has interactive user controlled tooltips, filtering feature that redraws the visualization with new associations based on users search parameters, and other data analyzing features.

**Figure 5.1.** A Circvis visualization from RE depicting associations between genomic features. Blue nodes represent gene expression features where orange nodes are copy number aberrations, edges represent RF-ACE associations based on filtering parameters.
It takes a lot of time to produce this kind of specialized visualization but it is also possible to extract useful information from more simple and ready-made visualizations. A simple internet search tells us that there are dozens of different JavaScript visualization libraries. In this thesis we take a closer look to six of them.

Using all these libraries is fairly simple. In Program 5.1 is the basic syntax to use all these libraries except D3 and Raphaël which are a bit more low level API's. Usually the options object holds information about data, styling and user events of the visualization. The structure of data and options JSON objects varies a bit from library to library. All these libraries are open-sourced and free for non-commercial usage.

```
1 var name = new library_name.visualization_type(data, options)
2 var name = new library_name(DOM_target, data, options); // visualization type can also be inside options
3 var name = new library_name(DOM_target, options); // data can be inside options
```

*Program 5.1. General notation to use these JavaScript libraries (except D3 and Raphaël)*

## 5.1 SVG libraries

SVG libraries are visualization libraries that use SVG as the image rendering method. SVG produces XML like description from the image that the web browser then renders.

### 5.1.1 Raphaël and gRaphaël

Raphaël is a small library for drawing SVG vector graphics and gRaphaël is a charting library built on top of Raphaël. Raphaël offers a low level API to draw visualizations. In Raphaël developer first defines an area to which he/she will draw. After that Raphaël offers functions to draw basic SVG shapes e.g. circles, rectangles and paths. It is then up to developer to position and scale every object in the visualization. Raphaël doesn’t even try to deliver a full visualization library, instead it creates an API that the developer can use as he/she wishes. This allows the API to be used to other things than data visualizations. Newest version of Raphaël can be downloaded from http://raphaeljs.com/

gRaphaël is built on top of Raphaël by same developers and it extends Raphaël’s API. gRaphaël’s idea is to make creation of data visualizations with Raphaël easier. It is used by first defining a Raphaël drawing area and then using gRaphaël’s functions to do the visualization, that work much like depicted in Program 5.1. gRaphaël supports basic pie, bar and line plots. Newest version of gRaphaël can be downloaded from http://g.raphaeljs.com/

Raphaël and gRaphaël both have fairly limited documentation and the examples don’t have the code on display. But once the developer learns to use Raphaël it can be used to make good and informative visualizations.
5. Data-Driven Documents (D3)

D3 doesn't explicitly specify what method to use for visualizations but it encourages to use SVG. D3 offers a low level API to draw visualizations similar to Raphaël. Unlike Raphaël which is designed to draw vector graphics in general, D3 is designed to draw data visualizations. In D3 as the name says developer can use data to drive the visualization creation. In D3 data can be bound to the drawing area and then used to draw the visualization. For example if we draw a simple histogram visualization, in Raphaël we have to use some kind of for or while loop structure to loop through the data points and then draw rectangles from them. In D3 we can bind the data to the document and then use special enter function to draw a rectangular from all the data points without any loop structure. In Program 5.2 is an example of simple histogram with both D3 and Raphaël. Even though Raphaël is more simple in creating simple charts, D3 is far more easier to update with new data and create complex visualizations with many elements.

```javascript
//D3
var bars = d3.select("body").selectAll(".bar")
.data(data) //data is bound
.enter()  //enter() returns new data points
.append("rect") //a rectangular is drawn from every data point
.attr("y", function(d,i){return 100;})
.attr("x", function(d,i){return d*20;})
.attr("width", function(d,i){15;})
.attr("height", function(d,i){data*50;})

//Raphaël
var bars = Raphael("body",800,800) //Raphaël drawing area
for (var i = 0; i < data.length; i++){
   bars.rect(i*20, 100, 15, data[i]*50) //x, y, width, height
}
```

**Program 5.2.** Histogram with D3 and Raphaël

In D3, unlike in the other libraries, styling of visualizations is done with CSS by defining classes and ids for elements. This allows the styling to be done and modified completely separated from visualization code. D3 also has a lot of functions that make drawing visualizations easier, for example data preprocessing to histogram bins, different scales that allow scaling the data attributes to certain scale e.g. histogram bar heights and many other. Newest version of D3 can be downloaded from http://d3js.org/.
5. JavaScript visualization libraries

5.1.3 HighCharts

HighCharts is a SVG visualization library that supports in addition to basic chart types also polar charts, spiderweb and wind rose to name a few. Highcharts web page has good documentation and examples with code. HighCharts is a very extensive charting library and it has a lot of options but it is still easy to use by syntax shown in Program 5.1. A nice feature in HighCharts is to have by default dedicated print and download buttons for visualization since SVG and Canvas elements can not be downloaded straight from a web page as an image.

Newest version of HighCharts can be downloaded from http://www.highcharts.com/. HighCharts is licensed to IBM, NASA, HP, BMW and other companies from the Global Fortune 500 list [15].

5.2 Canvas libraries

Canvas libraries are visualization libraries that use HTML5 canvas as the image renderer. In canvas the image is rendered from JavaScript code one frame at the time.

5.2.1 Rgraph

Rgraph supports over twenty different chart types and uses HTML5 canvas tag to draw them. RGraphs website has good documentation on basic things and even some how to guides on some not so trivial features. Although there are many basic examples, the actual code is not shown so the developer needs to look directly to source files to see the code. Also a full API-documentation is missing.

Rgraph also offers a drawing API to draw rectangles, circles, images and two kinds of markers. This drawing API can be used to draw additional shapes to visualizations. However this API isn’t as extensive as D3’s or Raphaëls APIs and it is not meant to be used exclusively. Newest version can be downloaded from http://www.rgraph.net/.

5.2.2 InfoVis

With InfoVis the developer can do interactive special purpose visualizations. InfoVis mainly specializes on different kinds on graph based visualizations, for example different kinds of trees and force directed graphs but it can also be used to draw more basic bar and pie charts. Current release 2.0.1 is from 05.05.2011. It is fairly old but the library works great as it is, so there is no reason to not use it. InfoVis documentation and examples with code can be found at http://philogb.github.com/jit/
5.2.3 CanvasXpress

CanvasXpress can create all the basic charts and also a box plot, genome browser, cellular pathways and 3D charts which are fairly rare features. CanvasXpress was developed to show genomic data so it has many useful features for biological data, e.g. tissue and sample annotations in data structure.

This diverse data structure also has its drawbacks as the data structure that is given to CanvasXpress initializer is fairly complicated. Data structure can be conceptually represented as a matrix with samples as columns and variables as rows. This is complicated to represent with JSON structure and even a simple histogram visualization needs this redundant annotation data schema. Data structure also varies a bit from visualization type to visualization type.

There is a thorough documentation and many good examples with code on web page http://www.canvasxpress.org/.
6 COMPARISON

For this thesis a multivariate data set from the Prostate cancer project at the Laboratory of Complex Biosystems Modeling in the Technical University of Tampere was analyzed with RF-ACE and then six different JavaScript libraries, introduced in chapter 5, were tried out to visualize statistical attributes of the data set. The RF-ACE algorithm produced about 8.8 million prostate cancer associations and the top one thousand associations, by p-value, were visualized. In Figure 6.1 is four visualizations from http://intianjora.cs.tut.fi/kallioa/kandi/ website that was made for this thesis.

Figure 6.1. Example of visualizations from http://intianjora.cs.tut.fi/kallioa/kandi/. In a) is a pie chart made with D3, it is visualizing the distribution of genomical features in the data set. In b) is the same visualization made with Infovis. In c) is a scatter chart correlation vs p-value between different associations made with HighCharts, the shape and color of the marker tells the type of the association. In d) is a histogram from the p-value distribution in the data set made with CanvasXpress.
6. Comparison

Each library was used to create histogram, pie chart, scatter plot, and force directed graph as it was possible. The visualizations can be seen at http://intianjora.cs.tut.fi/kallioa/kandi/. In this chapter we try to summarize and compare features from each library.

In addition of categorizing these libraries to canvas and SVG libraries, these can also be categorized to low level and high level API libraries. D3 and Raphaël are more lower level APIs than others since they require the developer to start visualization from basic rectangles, circles and lines and then specifying events to them. Whereas gRaphaël, Rgraph, Infovis, CanvasXpress and HighCharts offer a much higher level APIs where developer specifies options for visualization and the library takes care of all the different elements in the visualization. Both approaches have advantages and disadvantages. Higher level APIs are easier and faster to learn and use but if they do not have a certain visualization or feature, it is hard to implement to them afterwards. Some examples of special custom visualizations can be found at D3’s website (https://github.com/mbostock/d3/wiki/Gallery). Most of those are very hard or almost impossible to implement with higher level APIs.

Even though D3 and Raphaël are both low level libraries, they are still very different. Raphaël can be thought more as an API to draw SVG with JavaScript than a fully featured data visualization library, whereas D3 is a fully featured data visualization library with data driven elements, data preprocessing, scaling and other functions. This abundance of features means that D3 is hard to fully learn, but it helps that D3 has a good documentation and examples. Raphaël being a bit hard to learn compared to its features makes gRaphaël a good library to draw basic visualizations based on Raphaël and SVG. Even though gRaphaël only supports pie, bar and line charts those have enough features for simple visualizations when D3’s additional features are not needed. gRaphaëls and Raphaël negative features are that they are a bit hard to approach since they do not have a good documentation nor examples with code.

If gRaphaëls features are not enough and D3 seems too hard to learn HighCharts and Rgraph offer good libraries. Their most significant difference is the used render. HighCharts uses SVG and Rgraph uses canvas. Both offer lots of different features and fine documentations with examples even though HighCharts documentation is displayed more clearly.

For a bit more special visualizations CanvasXpress and Infovis are good choices. CanvasXpress is good for biological data because it has ready-made visualizations for genome browser, box plots and genomal pathways. Even though the data structure of CanvasXpress is complicated, its documentation and examples are good. CanvasXpress is also a good pick for more traditional visualizations as it is easy to use, after developer understands the data structure, and offers a lot of options to customize visualizations. CanvasXpress is also the only one to provide 3D-visualizations. Infovis is a good choice for different kinds of tree and graph visualizations. It is also possible to draw bar and pie charts but since InfoVis does not have scatter charts, other libraries are better choice for
these basic charts. In table 6.1 is a conclusion from each libraries pros and cons and what it is best for.

**Table 6.1. Summary about visualization libraries pros and cons**

<table>
<thead>
<tr>
<th>Library</th>
<th>Pros</th>
<th>Cons</th>
<th>Best for</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raphaël</td>
<td>• versatile</td>
<td>• hard to learn</td>
<td>• simple visualizations</td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>• usable to other things than visualizations</td>
<td>• bad documentation</td>
<td>• other vector graphics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• hard to learn</td>
<td>• low level API</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• bad documentation</td>
<td>• simple</td>
<td>• simple charts</td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>• low level API</td>
<td>• bad documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gRaphaël</td>
<td>• simple</td>
<td>• simple</td>
<td></td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>• can expand Raphaëls functions</td>
<td>• bad documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>• versatile</td>
<td>• low level API</td>
<td>• all kinds of visualizations</td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>• many functions to ease creation of visualizations</td>
<td>• hard to learn</td>
<td>• custom/special visualizations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• data driven visualizations</td>
<td>• requires a lot of code</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• styling with CSS</td>
<td>• low level API</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• good documentation</td>
<td>• hard to learn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HighCharts</td>
<td>• good examples</td>
<td>• not expandable</td>
<td>• Chart visualizations</td>
<td>SVG</td>
</tr>
<tr>
<td></td>
<td>• light low level drawing API</td>
<td>• Designed only for chart visualizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• self-explanatory JSON options structure</td>
<td>• not expandable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• default style</td>
<td>• Designed only for chart visualizations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As seen in the table 6.1, all these six libraries are good for some use cases and none of them is the best for everything even though D3 comes close, though it has a steep learning curve. It is also possible to combine these different libraries on the same web application. An example is the Regulome Explorer teams work on improving the application by integrating D3 for genomic associations and HighCharts for scatter plots. Choosing the library depends more on the specific visualization requirements and developers expertise than the library itself. There is not any actual performance differences between libraries as the performance depends more from the used browser and renderer.
REFERENCES


