UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

OGGO PETERSEN MACHADO NETO

Análise de bibliotecas para geração de gráficos na WEB

Trabalho de Graduação.

Prof. Dr. Valter Roesler
Orientador

Porto Alegre, dezembro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Neto, Oggo Petersen Machado

Análise de bibliotecas para geração de gráficos na WEB / Oggo Petersen Machado Neto. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

75 f.: il.

Trabalho de Conclusão (bacharelado) – Universidade Federal do Rio Grande do Sul. BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO, Porto Alegre, BR–RS, 2013. Orientador: Valter Roesler.

1. Bibliotecas para geração de gráficos para a web. 2. Flot. 3. JqPlot. 4. Dygraphs. 5. Chart.js. 6. NVD3. 7. Dc.js. 8. JQuery Sparklines. I. Roesler, Valter. II. Título.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
BMP	Bitmap
BSD	Berkeley Software Distribution
CSS	Cascading Style Sheets
CSV	Comma-separated Value
DOM	Document Object Model
GIF	Graphics Interchange Format
GPL	General Public License
HTML	HyperText Markup Language
JPEG	Joint Photographic Experts Group
JSON	Javascript Object Notation
MIT	Massachusetts Institute of Technology
PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
RNP	Rede Nacional de Ensino e Pesquisa
SGBD	Sistema de Gerenciamento de Banco de Dados
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
XML	eXtensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

Analysis of chart generating libraries for web

ABSTRACT

This work presents a comparative analysis of chart generating libraries for web, considering quantitative characteristics, which are focused on performance and interactivity, and qualitatives, which are focused on quality and abundance of available documentation. This analysis aims to help the reader to choose which API better suits his needs through comparative tests. During this study a test environment was developed using a code with PHP, JavaScript, jQuery and AJAX. The points to be plotted were obtained from a MySQL database that has 1 million points for the time rendering tests, and also number of points supported per rendering. For the validation of this work, 7 chart generating libraries were chosen using a selection criteria among all the available API's, such as opensource code, short inactivity time and the usage of a versioning tool that is accessible to other people. Each of the chosen libraries were implemented internally in the project in order to make this analysis possible.

Keywords: chart generating libraries for web, Flot, jqPlot, dygraphs, Chart.js, NVD3, dc.js, jQuery Sparklines.

RESUMO

Este trabalho apresenta uma análise comparativa de bibliotecas para geração de gráficos para a web, considerando características quantitativas, focadas no desempenho e interatividade; qualitativas, focadas na qualidade e abundância de documentação. Essa análise visa auxiliar o leitor na escolha de qual biblioteca melhor se destina ao seu projeto por meio de testes comparativos. Neste estudo foram implementados ambientes de teste utilizando um código desenvolvido em PHP, JavaScript, jQuery e chamadas via AJAX. Os pontos a serem plotados foram obtidos a partir de um banco de dados MySQL, possuindo 1 milhão de pontos para testes de tempo de renderização e número de pontos suportados por plotagem. Para a validação deste trabalho, selecionou-se 7 (sete) bibliotecas geradoras de gráficos utilizando critérios de seleção, em que as bibliotecas devem ser de código aberto, não devem possuir um longo tempo de inatividade e devem utilizar alguma ferramenta de versionamento acessível para seus usuários. Então, as bibliotecas foram agregadas e implementadas internamente no projeto para tornar a análise possível.

Palavras-chave: Bibliotecas para geração de gráficos para a web, Flot, jqPlot, dygraphs, Chart.js, NVD3, dc.js, jQuery Sparklines.

1 INTRODUÇÃO

O reuso de software é o uso de conhecimento ou artefatos existentes para a construção de novos softwares (Frakes e Fox 1995). É uma prática amplamente utilizada por programadores para evitar o retrabalho, pois ao se utilizar algoritmos ou bibliotecas já existentes economizam-se recursos (como tempo e energia) em algo que já está funcional e também testado.

Alguns sistemas utilizam seções que apresentam gráficos (seja para monitoramento do sistema, seja para verificação de recursos do servidor, entre outros) e os projetistas necessitam saber qual a biblioteca geradora de gráficos que melhor se encaixe de acordo com as restrições do projeto. Este trabalho de graduação terá como objetivo a análise dessas bibliotecas, visto que os gráficos são uma parte importante dos projetos que os utilizam.

Ao longo da pesquisa por possíveis opções, é provável que se encontre algumas bibliotecas que atendem às demandas do projeto. Surge, então, uma necessidade de avaliar e filtrar os resultados, não sendo uma tarefa trivial selecionar o componente que melhor se destina ao sistema que se implementa. A principal motivação desse trabalho é, por meio de testes e análises, permitir que o leitor consiga escolher de forma rápida, a ferramenta que melhor atenda às suas necessidades.

1.1 Objetivo

Baseado no cenário descrito na seção anterior, o objetivo geral desse trabalho será realizar um levantamento de bibliotecas geradoras de gráficos para a web, efetuar uma análise comparativa entre elas utilizando métricas qualitativas e quantitativas, a fim de esclarecer as vantagens e desvantagens de cada uma.

1.2 Estrutura

A estrutura desse trabalho está organizada da seguinte forma:

No capítulo 2, **Conceitos Básicos**, são apresentados conceitos introdutórios para uma compreensão mais ampla desse trabalho, onde serão explicadas linguagens utilizadas pelas bibliotecas selecionadas, contendo também uma breve explicação dos tipos de gráficos gerados por elas.

No capítulo 3, **Ferramentas para Geração de Gráficos**, são apresentados os critérios para a seleção das bibliotecas (como tempo de inatividade e linguagens utilizadas), sendo apresentadas logo em seguida as 7 bibliotecas selecionadas baseadas nos critérios de seleção.

No capítulo 4, **Metodologias de Validação**, é citado e explicado as metodologias de validação selecionadas, dividindo-as em métricas qualitativas e quantitativas.

No capítulo 5, **Experimentos**, são apresentados os resultados das métricas selecionadas (explicadas no capítulo 4) para cada uma das bibliotecas, trazendo tabelas comparativas para melhor visualização.

Por fim, o capítulo 6 contém as **Conclusões** desse trabalho, descrevendo os principais pontos e as considerações finais sobre o estudo realizado. Em seguida são apresentadas as referências bibliográficas utilizadas.

2 CONCEITOS BÁSICOS

Nesta seção são abordadas as linguagens utilizadas pelas bibliotecas de geração de gráficos assim como as linguagens que o ambiente de validação implementado utiliza. Também serão explicados brevemente os tipos de gráficos que essas API's geram.

2.1 Linguagens

2.1.1 HTML5

O HTML5 (Hypertext Markup Language) conforme definido em (HTML5 2013) é uma linguagem utilizada para realizar a estruturação e apresentação de conteúdo para a World Wide Web e é uma tecnologia que foi originalmente proposta por Opera Software. Esta versão é a quinta da linguagem HTML. O HTML5 possui em sua essência melhorar a linguagem obtendo suporte à tipos mais recentes de conteúdo multimídia, mantendo sua simplicidade e facilidade de leitura. Essa linguagem ainda está em fase de desenvolvimento mas já existem navegadores que implementam algumas de suas funcionalidades.

Temos várias novas funções sintáticas nesta tecnologia. As bibliotecas analisadas fazem o uso das etiquetas (*tags*)[1] *canvas* e *SVG* que são explicadas a seguir.

2.1.1.1 canvas

O canvas (w3schools 2013) é um elemento que faz parte do HTML5 e permite a renderização dinâmica de formas em duas dimensões, três dimensões e imagens bitmap. Ele é uma área destinada para delimitar a renderização de gráficos onde todo o trabalho de criação e animação é realizado através de linguagens de programação dinâmicas (como o JavaScript). O canvas permite que o usuário desenhe formas mas não é possível ter acesso aos elementos já renderizados nesta tag (ao contrário do SVG, que será explicado a seguir). A Figura 2.1 demonstra um exemplo de utilização de canvas, juntamente com um script feito em JavaScript para criação do desenho.

[1] Uma tag é uma palavra-chave (rótulo) que informa ao navegador como a página web deve ser apresentada.

```
<html>
<body>

<canvas id="meuCanvas" width="200" height="100" style="border:4px solid #d3d3d3;">
Seu navegador não suporta a tag canvas do HTML5.</canvas>

<script>

var c=document.getElementById("meuCanvas");
var ctx=c.getContext("2d");

// Cria o gradiente
var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"green");
grd.addColorStop(1,"white");

// Preenche o gradiente
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);

</script>

</body>
</html>
```

Figura 2.1: Exemplo de figura gerada em tag canvas

Conforme demonstrado na Figura 2.1, existe uma tag canvas no documento HTML onde é delimitado o espaço a ser desenhado na tela do navegador e um script JavaScript para desenhar as figuras nesse elemento.

2.1.1.2 SVG

A tag SVG (w3schools 2013) significa Scalable Vector Graphics e é utilizada para definir gráficos utilizando vetores. Os gráficos são definidos em XML e, por serem baseados em vetores, não perdem qualidade quando efetuamos zoom ou redefinimos seu tamanho (ao contrário de imagens JPEG, por exemplo). A Figura 2.2 demonstra que o redimensionamento de imagens vetoriais não ocasiona perda de qualidade.

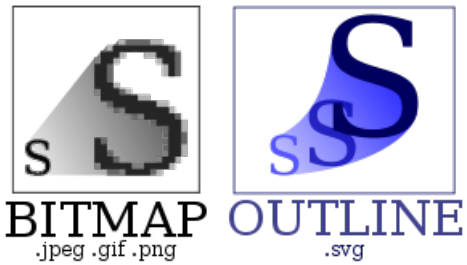


Figura 2.2: Exemplo de redimensionamento de imagem SVG

Ao contrário de outros tipos de formatos de imagens (como BMP e JPEG), podemos criar imagens SVG através de um editor de texto comum, essas gravuras podem ser impressas em alta qualidade utilizando qualquer resolução, ocupam um espaço menor (por serem representadas por um arquivo XML) mas esse fator não está diretamente relacionado com a velocidade de renderização de um SVG (o tempo aumenta conforme a complexidade do documento aumenta). Podemos também animar os objetos que se encontram na tag SVG.

A Figura 2.3 mostra um exemplo de imagem gerada a partir de um SVG, onde o código HTML se encontra à esquerda e a imagem SVG gerada está à direita.

```
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">
  <polygon points="100,10 40,180 190,60 10,60 160,180"
  style="fill:lime;stroke:black;stroke-width:5;fill-rule:evenodd;">
</svg>

</body>
</html>
```

Figura 2.3: Exemplo de figura gerada em tag SVG

2.1.1.3 Comparação de Canvas com SVG

A comparação entre as duas tags foi realizada por (Sucan 2010), que fez uma análise das vantagens e desvantagens das tags SVG e canvas, mostrando-as em uma tabela comparativa. A tabela 2.1 mostra algumas vantagens do canvas e do SVG.

Canvas	SVG
Uma superfície 2D para criação de imagens	Independente da resolução do navegador do usuário, o que faz do SVG uma boa escolha para diferentes tipos de resoluções de clientes
Bom desempenho (não importa a complexidade do que o canvas contém, somente o aumento da resolução irá resultar em uma perda de performance).	É possível controlar qualquer elemento dentro do SVG utilizando a interface DOM
É possível exportar o conteúdo do canvas em uma imagem no formato PNG ou JPG	O SVG é suportado na maioria dos navegadores atuais

Tabela 2.1: Vantagens das tags SVG e canvas

A tabela 2.2 apresenta uma desvantagem do canvas e do SVG.

Canvas	SVG
Não há a possibilidade de manipular elementos já renderizados	Conforme a complexidade do SVG aumenta, sua performance é degradada

Tabela 2.2: Desvantagens das tags SVG e canvas

Segundo Mihai Sucan, a utilização do canvas ou do SVG depende muito do objetivo do programador, não havendo uma melhor opção entre essas duas tecnologias. Ele recomenda o uso do canvas para edição de imagens de forma interativa, geração de gráficos, análise de imagens e renderização de jogos. Para o SVG, o autor recomenda o uso quando a aplicação necessite de total compatibilidade entre diferentes resoluções, interfaces de usuário interativas e animadas, geração de gráficos e edição de imagens vetoriais. Há também casos que se pode utilizar ambos, onde é possível renderizar gráficos utilizando o canvas e animá-las utilizando o SVG.

2.1.2 CSS

O CSS (Cascading Style Sheets) é uma linguagem de estilo que é utilizada para definir a forma que documentos escritos em linguagem de marcação serão apresentados. Sua ver-

são mais recente é a 3 (que ainda se encontra em desenvolvimento) e alguns navegadores já possuem suporte para algumas de suas funcionalidades. O CSS visa deixar a criação da apresentação de uma página mais fácil, onde os estilos criados dentro do arquivo definem como serão apresentados os elementos no cliente. A Figura 2.4 visa demonstrar como é a linguagem CSS, onde é utilizado um documento HTML para apresentação de conteúdo (no topo esquerdo há o código CSS, no topo direito está um segmento do arquivo HTML e na parte inferior da imagem o resultado final é apresentado).

Figura 2.4: Exemplo de uso do CSS para apresentação de um documento HTML

2.1.3 JavaScript

JavaScript é uma linguagem script multiplataforma e orientada a objetos (JavaScript 2013). Essa linguagem é leve, pequena e foi criada para ser utilizada em outros programas e aplicações, tais como navegadores web. Ela foi originalmente desenvolvida por Brendan Eich da Netscape com o nome Mocha, posteriormente seu nome foi alterado para LiveScript e finalmente JavaScript. Essa linguagem foi originalmente implementada para que navegadores interpretassem o código e o executassem no cliente, ao invés de executar o código no servidor. A seguir são citadas algumas de suas características:

- A tipagem do JavaScript é fraca e dinâmica, ou seja, suas variáveis podem ser interpretadas de formas diferentes dependendo do contexto em que se encontram.
- Possui funções de primeira classe, ou seja, funções podem ser passadas como argumentos e funções podem retornar outras funções como resultado.

A seguir há exemplo de código utilizando a linguagem JavaScript:

```
function fatorial(numero)
{
  // calcula o fatorial do numero enviado
  if (numero < 0)
    return null;

  else if(numero == 0 || numero == 1)
    return 1;

  else
  {
    var resultado = 1;
    while(numero > 1)
    {
      resultado = resultado * numero;
      numero --;
    }
    return resultado;
  }
}
```

A função apresentada anteriormente é um exemplo da sintaxe JavaScript e calcula o fatorial de um dado número.

2.1.4 jQuery

jQuery (jQuery 2013) é uma biblioteca para JavaScript e seu objetivo é simplificar a programação de scripts que interagem com a página HTML no lado do cliente. É uma biblioteca de código aberto, utiliza as licenças MIT (MIT 2013) ou GNU (GNU 2013). Sua sintaxe foi desenvolvida para tornar mais intuitiva a navegação no documento HTML, seleção de elementos no DOM [2], criar animações, manipular eventos (como cliques de mouse, etc...) e também desenvolver aplicações AJAX[3].

A Figura 2.5 apresenta algumas diferenças entre a linguagem JavaScript nativa e como se é possível expressar a mesma idéia utilizando a biblioteca jQuery.

[2]DOM (ou Document Object Model) é um meio que permite que programas e scripts acessem dinamicamente elementos, estruturas e estilos de um documento.

[3]AJAX (Asynchronous JavaScript and XML) é o uso de tecnologias como o JavaScript para buscar informações de forma assíncrona. Apesar do nome, podemos utilizar outras formas de passar dados, como o JSON e as requisições podem ser síncronas também.

```javascript
function funcaoExemplo()
{
    var obj=document.getElementById("id_do_elemento");
    obj.innerHTML="Texto a ser inserido";
}
onload=funcaoExemplo;
```

```javascript
function funcaoExemplo()
{
    $("#id_do_elemento").html("Texto a ser inserido");
}
$(document).ready(funcaoExemplo);
```

Figura 2.5: Exemplo de comparação entre JavaScript (esquerda) e jQuery (direita)

Como é possível notar na figura 2.5, a linguagem JavaScript requer um número maior de comandos do que o jQuery, onde, por exemplo, para se obter um elemento da página HTML que possui o identificador igual a *id_do_elemento* é necessário acessar o objeto *document*, em seguida chamar a função *getElementById* passando como argumento o identificador do elemento que é de interesse. No jQuery a mesma operação é feita utilizando o comando $("#id_do_elemento"), onde o símbolo "#" representa a função "getElementById". Com este exemplo é possível notar algumas das simplificações que o jQuery trouxe para os programadores web.

2.1.5 Tecnologias e linguagens utilizadas para teste de bibliotecas

Abaixo serão apresentadas as linguagens e tecnologias utilizadas para os experimentos das bibliotecas selecionadas.

2.1.5.1 PHP

A linguagem PHP (PHP 2013), que significa PHP: Hypertext Preprocessor, é uma linguagem interpretada e livre usada para desenvolvimento de sistemas web no lado do servidor, capaz de gerar conteúdo dinâmico na *World Wide Web*. O código é interpretado no lado do servidor, onde é gerado um documento HTML que é enviado ao cliente com o resultado final (ou seja, a página HTML). O PHP surgiu em meados de 1994 e foi criado por Rasmus Lerdorf sob o nome *Personal Home Page Tools*, passando por diversas versões posteriormente (versão atual é a 5.5.0). Suas principais características são:

- É uma linguagem de programação veloz e robusta;
- O usuário pode programar código PHP de forma estruturada ou orientada a objetos;
- É portável, ou seja, o mesmo código pode ser executado em diferentes sistemas;
- Possui tipagem dinâmica;
- A sintaxe do PHP é semelhante às linguagens C/C++ e o Perl;
- O PHP é Código Aberto (Open Source 2013);
- O PHP é executado no servidor (o cliente manda a requisição e o servidor responde com um arquivo HTML)

A seguir temos um exemplo de um código simples escrito em PHP.

```
<?php
// Funcao que soma dois numeros e retorna a soma
function somaDoisValores($a, $b) {
  return $a + $b;
}
// Mostra na tela o valor '3'
echo somaDoisValores(1,2)
?>
```

O código apresentado anteriormente possui uma função que realiza a soma de dois valores e a chamada para esta função (que irá imprimir na tela do usuário o valor 3).

2.1.5.2 MySQL

O MySQL (MySQL 2013) é o banco de dados relacional mais popular da atualidade (segundo o próprio site do MySQL), desenvolvido, distribuído e mantido pela Oracle Corporation. O MySQL é um sistema de gerenciamento de banco de dados (SGBD), seus bancos de dados são relacionais, o código do MySQL é aberto e está disponível sob a licença GPL ou pode-se comprar uma versão comercial como alternativa à licença gratuita.

2.2 Tipos de gráficos

Para uma melhor compreensão dos gráficos que as bibliotecas selecionadas geram, esta seção contém uma breve descrição de cada tipo de gráfico.

2.2.1 Gráfico de linhas

Este tipo de gráfico apresenta suas informações como uma série de pontos conectados por uma linha, ele é usualmente aplicado para a visualização de dados sob intervalos de tempo. Como exemplo é apresentada a Figura 2.6 contento uma tabela com informações de velocidade (metros por segundo) à medida que o tempo passa acompanhada do gráfico de linha correspondente (dados hipotéticos).

Tempo (s)	Velocidade (m/s)
0	0
1	3
2	7
3	12
4	20
5	30
6	45

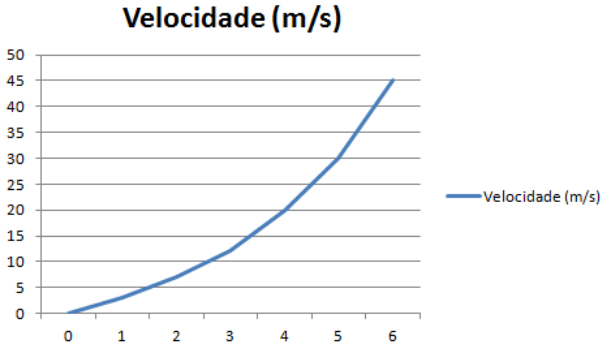


Figura 2.6: Exemplo de gráfico de linhas

2.2.2 Gráfico de barras

O gráfico de barras contém barras retangulares com tamanhos proporcionais aos valores que representam, onde as barras podem ser plotadas verticalmente ou horizon-

talmente. Ele é utilizado normalmente para comparação entre diferentes categorias, onde um eixo representa as categorias que estão sendo comparadas e o outro normalmente contém valores discretos. A Figura 2.7 apresenta como exemplo o consumo de energia de uma família durante um ano (dados hipotéticos).

Mês	Consumo em kWh
Janeiro	380
Fevereiro	300
Março	280
Abril	290
Maio	270
Junho	260
Julho	370
Agosto	310
Setembro	305
Outubro	315
Novembro	330
Dezembro	390

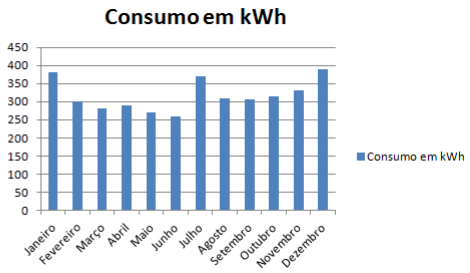


Figura 2.7: Exemplo de gráfico de barras

2.2.3 Gráfico de área

Um gráfico de área acompanha uma ou mais séries de dados graficamente e é especialmente útil para expressar alterações de valores entre categorias de dados. Este tipo de gráfico friza a mudança no decorrer do tempo e são usados para obter o valor total ao longo de uma tendência. A Figura 2.8 apresenta um exemplo de gráfico de área, onde é apresentada a velocidade de um veículo hipotético e o tempo (dados hipotéticos), a partir do cálculo da área deste gráfico é possível obter a distância total percorrida.

Tempo (s)	Velocidade m/s
0	0
100	10
200	20
300	30
400	40
500	50
600	60

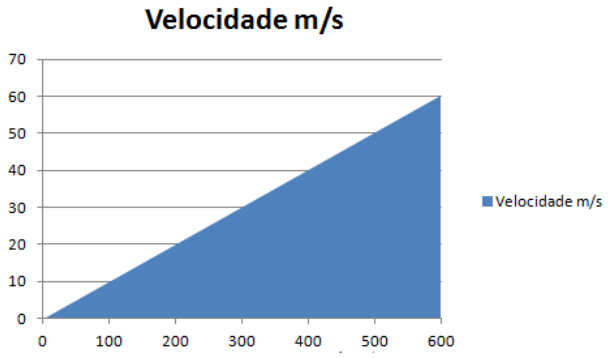


Figura 2.8: Exemplo de gráfico de área

2.2.4 Gráfico de dispersão

Um gráfico de dispersão apresenta coordenadas numéricas nos eixos X e Y, sendo uma representação gráfica para dados bivariados quantitativos [4] em que cada par de dados (x,y) é representado por um ponto de coordenadas (x,y) em um sistema de coordenadas. Na Figura 2.9 é apresentado um exemplo de gráfico de dispersão, onde os dados retirados para o exemplo abaixo se encontram em (Martins 2011).

[4]Dados bivariados são o resultado de uma observação de duas variáveis sobre o mesmo indivíduo de amostra. Como exemplo, a observação da altura e o peso de um mesmo indivíduo geram pares de dados.

Idade da mulher	19	24	20	28	26	25	27	23	32	31	35	32	34	40
Idade do marido	28	29	27	26	31	24	39	33	37	34	35	42	40	41

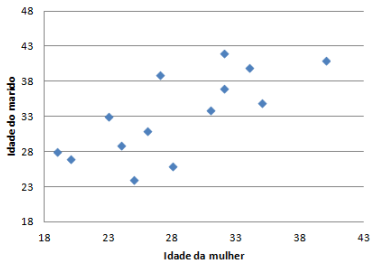


Figura 2.9: Exemplo de gráfico de dispersão

2.2.5 Gráfico de bolhas

O gráfico de bolhas é uma variação do gráfico de dispersão, porém ele utiliza mais uma informação para indicar uma terceira categoria de dados, que reflete no tamanho da bolha plotada no gráfico. A Figura 2.10 apresenta um exemplo de gráfico de bolhas.

Número de produtos	Venda (em R$)	Participação de
14	12.200,00	15%
20	60.200,00	33%
18	24.400,00	10%
22	32.000,00	42%

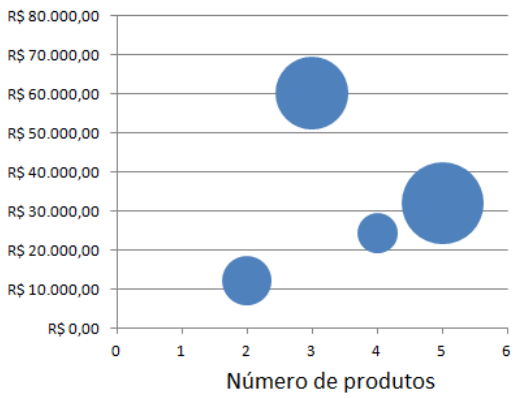


Figura 2.10: Exemplo de gráfico de bolhas

2.2.6 Gráfico de ações

O gráfico de ações é mais utilizado para ilustrar flutuações de preços de ações, mas pode ser utilizado para fins científicos. É possível identificar valores de abertura, fechamento, preço máximo e mínimo para o desenho dos símbolos que compõem o gráfico. Quando o candelabro (blocos que compõem o gráfico, conhecido também por *candlestick*) é claro (vazado) significa que ele é positivo, ou seja, a abertura ocorre no valor mínimo do corpo real do candelabro e o fechamento na máxima do corpo real do candelabro, a sombra superior (linha vertical acima do corpo do *candlestick*) possui o valor máximo e a sombra inferior, o valor mínimo. Quando o candelabro é preenchido o chamamos de negativo, onde o fechamento ocorre no mínimo do corpo real e a abertura no valor máximo do corpo do *candlestick*. Na Figura 2.11 temos um exemplo deste tipo de gráfico utilizando dados hipotéticos.

	Abertura	Máximo	Mínimo	Fechamento
Março	R$ 138,70	R$ 139,68	R$ 135,18	R$ 135,40
Abril	R$ 143,46	R$ 144,66	R$ 139,79	R$ 140,02
Maio	R$ 140,67	R$ 143,56	R$ 132,88	R$ 142,44
Junho	R$ 136,01	R$ 139,50	R$ 134,53	R$ 139,48
Julho	R$ 143,82	R$ 144,56	R$ 136,04	R$ 136,97
Agosto	R$ 136,47	R$ 146,40	R$ 136,00	R$ 144,67
Setembro	R$ 124,76	R$ 135,90	R$ 124,55	R$ 135,81
Outubro	R$ 123,73	R$ 129,31	R$ 121,57	R$ 122,50

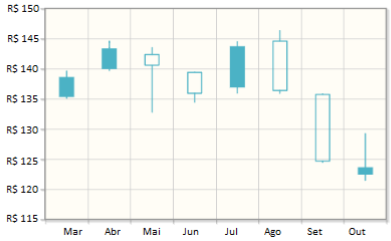


Figura 2.11: Exemplo de gráfico de ações

2.2.7 Gráfico Mekko

Os gráficos Mekko (também conhecidos por Marimekko) são amplamente utilizados na área de marketing. São colunas empilhadas com largura variável, indicando uma terceira dimensão de dados (normalmente uma divisão de segmento de mercado). É possível representar 3 dimensões de dados de marketing fazendo o uso deste tipo de gráfico e a Figura 2.12 apresenta um exemplo de uso do mesmo, utilizando dados hipotéticos.

	Camisetas	Adesivos	Sapatos	Luvas	Brinquedos
Celebridade A	R$ 8 Milhões	R$ 14 Milhões	R$ 6 Milhões	R$ 16 Milhões	R$ 12 Milhões
Celebridade B	R$ 15 Milhões	R$ 6 Milhões	R$ 9 Milhões	R$ 13 milhões	R$ 6 Milhões
Celebridade C	R$ 4 Milhões	R$ 2 milhões	R$ 7 Milhões	R$ 9 Milhões	R$ 7 Milhões

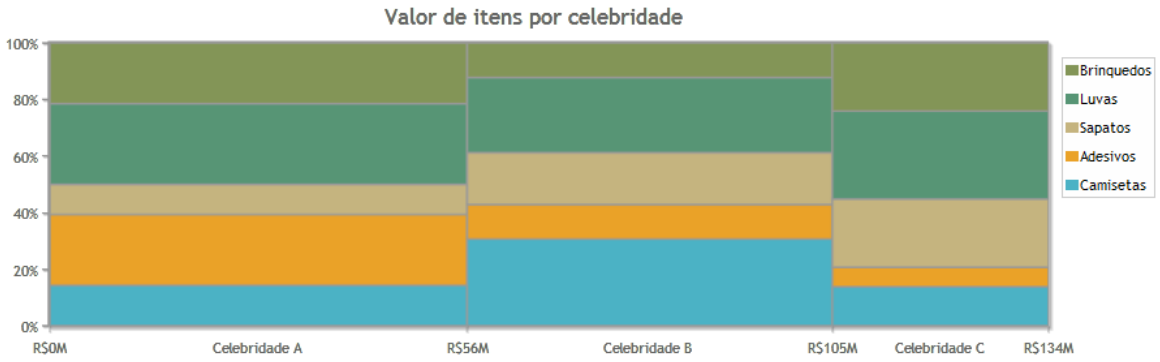


Figura 2.12: Exemplo de gráfico mekko

2.2.8 Gráfico medidor

O gráfico medidor são indicadores que utilizam um valor. São utilizados em painéis de gerenciamento, monitores em tempo real e relatórios. Na Figura 2.13 há um exemplo de medidor, mostrando a frequência utilizada por uma placa de vídeo.

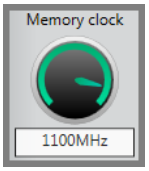


Figura 2.13: Exemplo de gráfico medidor

2.2.9 Gráfico de pizza

Um gráfico de pizza ilustra a relação entre as partes e um todo, onde os dados de valor são exibidos como porcentagem de um todo e as categorias são representadas por fatias individuais. A Figura 2.14 apresenta um exemplo deste tipo de gráfico.

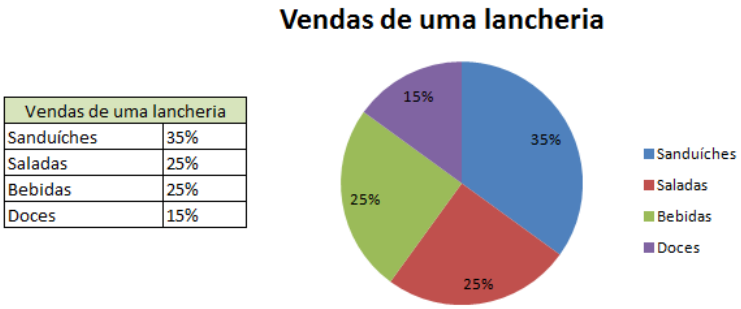


Figura 2.14: Exemplo de gráfico de pizza

2.2.10 Gráfico de rosca

O gráfico de rosca é similar ao gráfico de pizza, porém pode conter mais de uma série. A Figura 2.15 mostra um exemplo de gráfico de rosca utilizando dados hipotéticos.

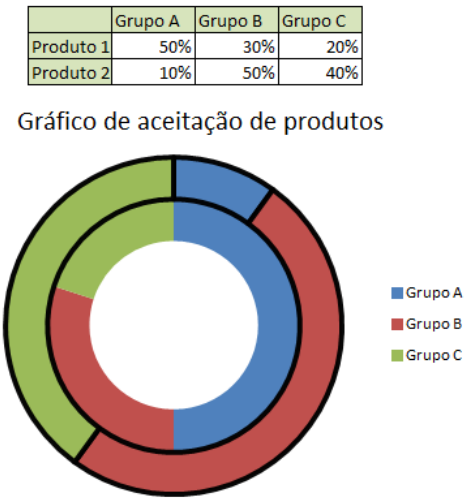


Figura 2.15: Exemplo de gráfico de rosca contendo duas séries de dados

2.2.11 Gráfico de pirâmide etária

O gráfico de pirâmide etária é um tipo de gráfico que mostra como grupos de diferentes faixas etárias em uma dada população. É um gráfico que contém dois conjuntos de barras

representando o sexo masculino e feminino. A Figura 2.16 apresenta um exemplo deste tipo de gráfico.

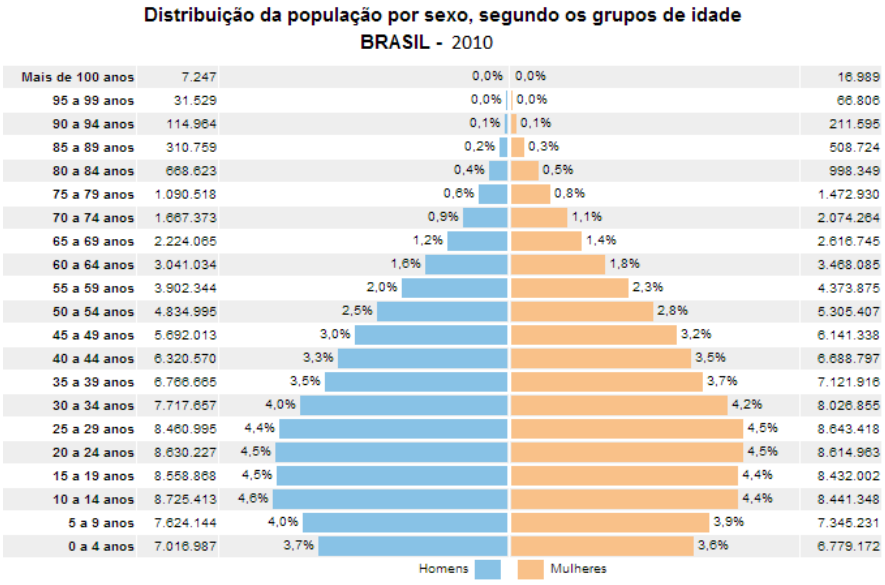


Figura 2.16: Exemplo de gráfico de pirâmide etária retirado de (IBGE 2010)

2.2.12 Gráfico de marcadores

O gráfico de marcador é uma variação do gráfico de barra e foi desenvolvido para substituir níveis de medidores do painel. Ele é geralmente utilizado para comparar uma medida primária com uma ou mais medidas de um contexto de faixas qualitativas de desempenho (como satisfatório, bom e ruim). O valor atual é representado pela barra escura, há um gradiente de cores que acompanham este gráfico e representam o desempenho (de ruim para bom) e a linha vertical representa o objetivo. A Figura 2.17 apresenta um gráfico de marcadores para uma melhor compreensão.

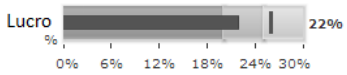


Figura 2.17: Exemplo de gráfico de marcador (retirado de (Bullet Chart - Fusion Charts 2013))

2.2.13 Gráfico de Radar

O gráfico de radar serve para comparar valores agregados de várias séries de dados. A Figura 2.18 apresenta um exemplo para este tipo de gráfico, onde é possível comparar dois tipos de carros baseando-se em certas características.

	Velocidade (km/h)	Eficiência	Confiabilidade	Conforto	Segurança
Carro A	60	80	90	70	80
Carro B	130	40	60	40	65

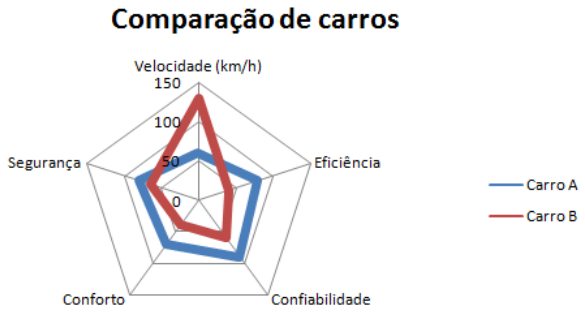
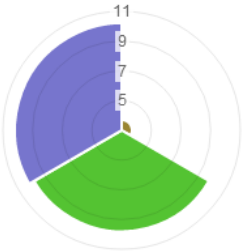


Figura 2.18: Exemplo de gráfico de radar

2.2.14 Gráfico de área polar

Um gráfico polar exibe uma série como um conjunto de pontos agrupados por categoria em um círculo de 360 graus, tendo seus valores representados pelo comprimento do ponto e quanto mais distante o ponto está do centro do círculo maior é o seu valor. A Figura 2.19 apresenta um exemplo para este tipo de gráfico.



Valor 1	Valor 2	Valor 3
3,7	9,8	10,2

Figura 2.19: Exemplo de gráfico de área polar

2.2.15 Gráfico de cascata

O gráfico de cascata é uma forma de visualizar os dados permitindo determinar o efeito cumulativo de valores negativos e positivos introduzidos de forma sequencial em um valor inicial. A Figura 2.20 apresenta um exemplo para este tipo de gráfico.

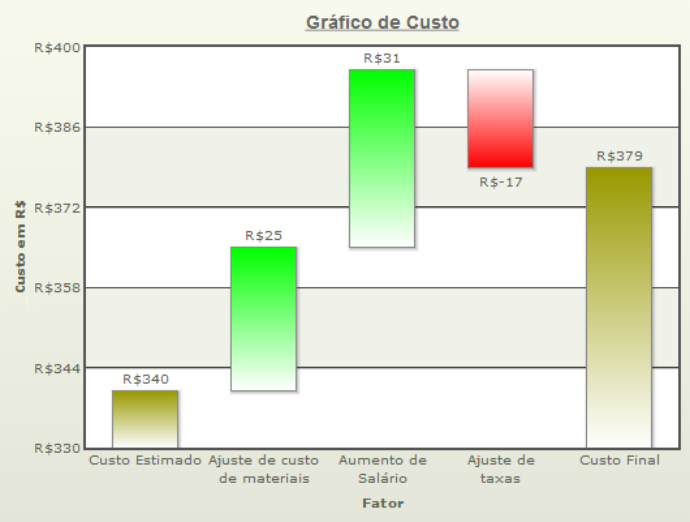


Figura 2.20: Exemplo de gráfico de cascata (Exemplo retirado de (Charts 2013))

2.2.16 Diagrama de caixa

O diagrama de caixa, utilizado na estatística descritiva, é uma ferramenta utilizada para localizar e analisar a variação de uma variável dentre diferentes grupos de dados, identificando onde estão localizados 50% dos valores mais prováveis, a mediana (que é, após a ordenação dos valores do conjunto, o valor que divide a metade inferior da metade superior da amostra) e os valores externos ao grupo de dados. Para renderizar este tipo de gráfico é necessário calcular a mediana (que será a linha que secciona o corpo da caixa), os quartis (onde o quartil inferior é o número que separa 25% dos menores valores da amostra e o quartil superior é o número que separa 25% dos maiores valores da amostra) que definirão a altura da caixa, a estimativa do Whisker (que definirá a altura da linha que secciona a caixa, entrando pela sua base e saindo pelo seu topo), e os valores atípicos (ou outliers) que são os valores muito diferentes do conjunto. A Figura 2.21 apresenta um exemplo para este tipo de gráfico.

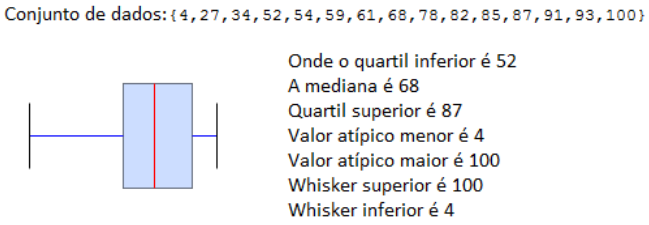


Figura 2.21: Exemplo de Diagrama de caixa

3 BIBLIOTECAS GERADORAS DE GRÁFICOS

Nesta seção serão apresentadas as ferramentas selecionadas para comparação. Em primeiro momento será apresentada na seção 3.1 uma tabela com bibliotecas geradoras de gráficos para a web, apresentando em cada coluna da tabela critérios que foram utilizados para filtrar esses resultados. Na seção 3.2 serão apresentados os critérios de seleção para a validação do trabalho, e finalmente serão apresentadas as bibliotecas selecionadas na seção 3.3. Para bibliotecas que possuem extensões criadas por terceiros, será levado em conta apenas o que a biblioteca suporta em sua versão original (ou seja, o código que está disponível no site dos seus respectivos desenvolvedores).

3.1 Bibliotecas disponíveis

Há uma grande quantidade de bibliotecas para se analisar, sendo necessário utilizar alguns critérios de seleção que serão explicados mais adiante. Na tabela 3.1 são apresentadas as bibliotecas disponíveis atualmente para geração de gráficos para a web (e somente plotagem de gráficos, sem nenhuma funcionalidade a mais), onde cada linha representa uma biblioteca e cada coluna representa características das mesmas. As colunas apresentadas na tabela representam os critérios de filtro para cada um dos componentes, critérios que serão explicados na seção 3.2. Abaixo são explicadas cada uma das colunas da tabela 3.1:

- Linguagem: linguagem que cada biblioteca utiliza;
- Licença: sob qual licença está disponibilizada;
- Tempo inativo: tempo que o projeto não recebeu atualizações;
- Código em repositório: se o código do componente está disponível em algum repositório (como, por exemplo, o Github);
- Analisada: se a biblioteca será analisada neste trabalho ou não.

Nome	Linguagem	Licença	Tempo inativo	Código em repositório	Analisada
Highcharts	Javascript / HTML5	Licença própria	menor que 5 meses	Sim	Não
JScharts	JavaScript / HTML5	Licença própria	Não disponível	Não	Não
Protovis	JavaScript / HTML5	Licença BSD	maior que 5 meses	Sim	Não
PlotKit	JavaScript / HTML5	Licença BSD e Apache	maior que 5 meses	Sim	Não
MilkChart	JavaScript / HTML5	Licença MIT	maior que 5 meses	Sim	Não
jqPlot	JavaScript / HTML5	Licença MIT ou GPL	menor que 5 meses	Sim	Sim
JavaScript InfoVis Toolkit	JavaScript / HTML5	Licença MIT	maior que 5 meses	Sim	Não
GoogleCharts	JavaScript / HTML5	Licença Apache	Não disponível	Não	Não
FusionCharts	JavaScript / HTML5	Licença própria da FusionCharts	Não disponível	Não	Não
Flot	JavaScript / HTML5	Licença MIT	menor que 5 meses	Sim	Sim
dygraphs	JavaScript / HTML5	Licença MIT	menor que 5 meses	Sim	Sim
jQuery Sparklines	JavaScript / HTML5	Licença BSD	menor que 5 meses	Sim	Sim
moochart	JavaScript / HTML5	Licença MIT	maior que 5 meses	Sim	Não
TufteGraph	JavaScript / HTML5	Licença MIT	maior que 5 meses	Sim	Não
jQuery Visualize	JavaScript / HTML5	Licença MIT e GPL	maior que 5 meses	Sim	Não
NVD3	JavaScript / HTML5	Licença Apache	menor que 5 meses	Sim	Sim
dc.js	JavaScript / HTML5	Licença Apache	menor que 5 meses	Sim	Sim
Chart.js	JavaScript / HTML5	Licença MIT	menor que 5 meses	Sim	Sim

Tabela 3.1: Tabela apresentando as diversas bibliotecas disponíveis

3.2 Critérios para seleção de bibliotecas

A seleção de cada uma das bibliotecas levou em conta os seguintes critérios:

3.2.1 Código Aberto

É importante que cada biblioteca tenha seu código aberto para que os utilizadores possam analisá-lo e realizar contribuições (ou modificações específicas conforme a necessidade de cada um). Para os componentes selecionados para este trabalho foram selecionadas somente licenças de código aberto, sendo descartadas outros tipos. Para uma melhor compreensão sobre licenças de código aberto recomenda-se a leitura de (Chapman 2010), disponível no link `http://www.smashingmagazine.com/2010/03/24/a-short-guide-to-open-source-and-similar-licenses/`.

3.2.2 Continuidade

Para todas as bibliotecas pesquisadas há um fator importante: a continuidade do projeto. É necessário que elas possuam comunidades ativas e o suporte constante, incluindo modificações de código (como correção de problemas e possíveis refatorações) e inserção de novas funcionalidades, assim seus usuários sabem que a vida útil destes componentes é longa. Foram descartadas bibliotecas cujo tempo sem atualizações excedia 5 meses.

3.2.3 Disponibilidade de código

O código de cada componente selecionado deve estar disponível na internet a partir de sistemas de versionamento (como o GitHub) onde várias pessoas podem obtê-los e editá-los de forma livre conforme suas necessidades específicas.

3.2.4 Linguagem

A linguagem selecionada para essas bibliotecas deve executar sua renderização no navegador do cliente e não no servidor. Esse critério visa economizar recursos do servidor.

3.2.5 Compatibilidade

Os componentes selecionados devem ser compatíveis com os seguintes navegadores:

- Firefox;
- Chrome;
- Internet Explorer.

De acordo com (Browser Statistics 2013) estes navegadores são os 3 mais utilizados na atualidade.

3.3 Bibliotecas Selecionadas

Para estudo, foram selecionadas as ferramentas:

- Flot;
- jqPlot;

- dygraphs;
- Chart.js;
- NVD3;
- dc.js;
- jQuery Sparklines;

Vale ressaltar que para o programador utilizar as bibliotecas citadas nesse trabalho é necessário que ele entenda alguns conceitos de programação. Serão explicadas a seguir as ferramentas citadas anteriormente.

3.3.1 Flot

Flot foi um projeto iniciado em meados de 2007 por Ole Laursen, sendo mantido atualmente por David Schnur. É uma biblioteca desenvolvida utilizando a linguagem JavaScript juntamente com a API jQuery, fazendo o uso do canvas (elemento do HTML5) e disponibilizada sob a licença MIT (MIT 2013). O Flot em sua versão pura suporta 2 tipos de gráficos (existem várias extensões que ampliam esta quantidade), o de barras e o de linha, conforme apresentado na Figura 3.1.

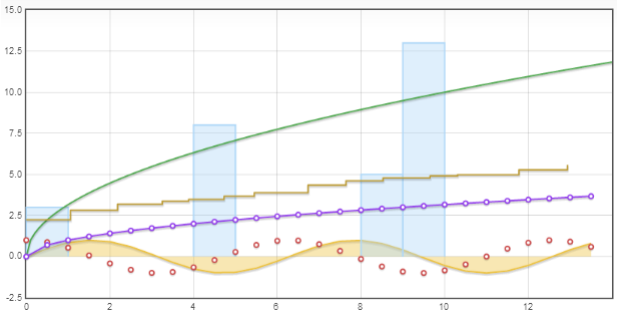


Figura 3.1: Exemplo de gráfico gerado pelo Flot

Como é possível notar na Figura 3.1, essa API suporta gráficos de linhas e barras, podendo, inclusive, reunir esses diferentes tipos em uma única plotagem, utilizando a mesma formatação para os dados.

Na Figura 3.2 é apresentado um exemplo de código juntamente com o resultado obtido.

```
$(function() {

    // dados a serem plotados
    var data = [ ["Janeiro", 10], ["Fevereiro", 8], ["Março", 4], ["Abril", 13], ["Maio", 17], ["Junho", 9] ];

    // chamada da biblioteca flot realizando a plotagem do gráfico
    $.plot("#placeholder", [ data ], {
        series: {
            // identifica que o gráfico é de barras com algumas opções - como largura de cada barra
            bars: {
                show: true,
                barWidth: 0.6,
                align: "center"
            }
        },
        // identifica algumas opções do eixo x
        xaxis: {
            mode: "categories",
            tickLength: 0
        }
    });

});
```

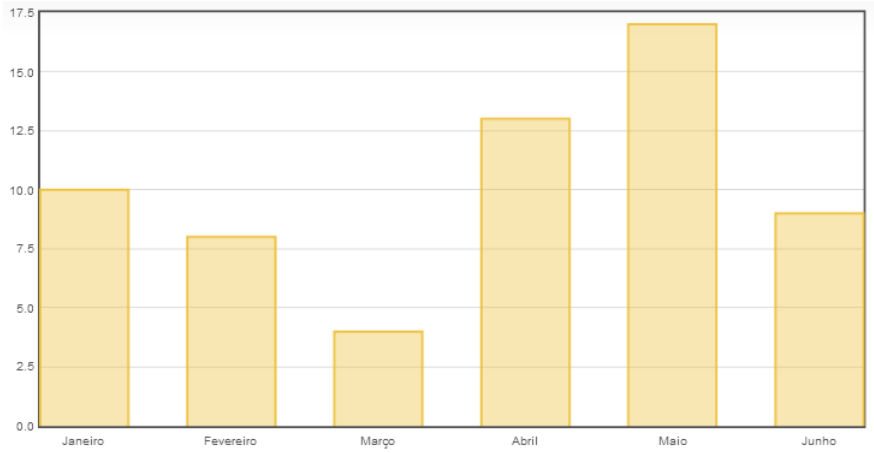


Figura 3.2: Gráfico em barras gerado pela API Flot (na parte inferior) e seu respectivo código (na parte superior)

No código apresentado na Figura 3.2, após a criação da variável *data* contendo os dados a serem utilizados pela plotagem, então é realizada uma chamada (denotada por $.plot) à API recebendo o identificador do elemento div da página (aqui chamado de "placeholder"), os dados (variável *data*), algumas opções para especificar qual o tipo de gráfico a ser plotado e opções de customização para o eixo x (como o modo e também os intervalos do eixo).

3.3.2 jqPlot

O jqPlot é uma biblioteca desenvolvida e mantida por Chris Leonello e utiliza o elemento canvas, a linguagem JavaScript e o jQuery para a plotagem dos gráficos. O jqPlot em sua versão atual suporta 11 gráficos apresentados na Figura 3.3 e está disponível sob a licença MIT (MIT 2013) ou GPL (GPL 2013).

Figura 3.3: Tipos de gráficos suportatos pela biblioteca jqPlot

Para uma melhor compreensão em como utilizar o jqPlot, a Figura 3.4 apresenta um código exemplificando sua utilização juntamente com o resultado final.

```
var grafico = $.jqplot (
    // O primeiro argumento contém o identificador do elemento div do html que
    // receberá o gráfico
    'id_do_elemento_div',
    // O segundo argumento contém os dados a serem plotados
    [[3,7,9,1,4,6,8,2,5]],
    // O último argumento contém opções de customização do gráfico
    {
        // Titulo do gráfico
        title: 'Titulo do grafico',
        // Opções de renderização para eixos
        axesDefaults: {
          labelRenderer: $.jqplot.CanvasAxisLabelRenderer
        },
        // Um objeto axes contem as opções para todos os eixos
        axes: {
          // Opções para o eixos X
          xaxis: {
            // Título do eixo X
            label: "Eixo X"
          },
          yaxis: {
            // Título do eixo Y
            label: "Eixo Y"
          }
    }
});
```

Figura 3.4: Exemplo de código e resultado final obtido com jqPlot

Como é possível perceber na Figura 3.4 é necessário somente declarar uma variável que receberá o resultado da função $.jqplot. Nota-se que há três argumentos na função jqplot, são eles:

- O primeiro argumento é o identificador do elemento div da página HTML que receberá o gráfico a ser plotado;
- O segundo argumento possui um vetor de dados que será utilizado na plotagem;
- O terceiro e último argumento contém um objeto que possui opções de customização do gráfico;

3.3.3 dygraphs

O dygraphs é uma biblioteca desenvolvida utilizando a linguagem JavaScript e renderiza seus gráficos utilizando o canvas do HTML5. Ele foi desenvolvido para suportar uma quantidade grande de pontos e opera recebendo um vetor, um arquivo do tipo CSV [1], um arquivo de texto, uma função que retorne os dados ou uma url (que é posteriormente é interpretada como um arquivo CSV). Ele está disponibilizado sob a licença MIT (MIT 2013) e atualmente contém somente gráficos de linhas. A Figura 3.5 mostra um exemplo de gráfico obtido com o dygraphs.

[1] CSV (comma-separated value ou valores separados por vírgula) armazena dados em tabela na forma de texto.

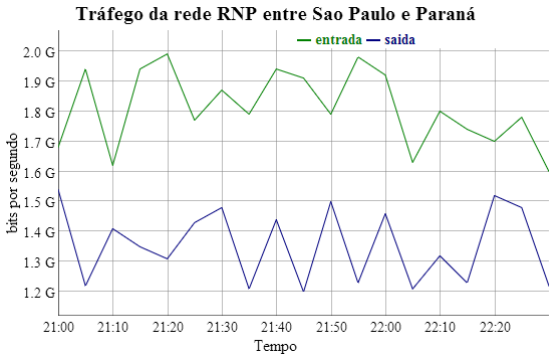


Figura 3.5: Exemplo de gráfico gerado pelo dygraph

Para melhor compreensão em como utilizar essa biblioteca é apresentada a Figura 3.6 contendo o código javascript.

```
// Variável grafico recebe recebe um objeto do tipo Dygraph
var grafico = new Dygraph(
        // Obtém o elemento div que contem o identificador "grafico"
        document.getElementById("grafico"),
        // Dados a serem plotados, representando X e Y, respectivamente
        [
            [1,2],
            [2,7],
            [3,5],
            [4,15],
            [5,3]
        ],
        // Opções de customização para o gráfico
        {
            // Título do gráfico
            title: 'Exemplo de gráfico gerado no Dygraphs',
            // Rótulo do eixo X
            xlabel: 'Eixo X',
            // Rótulo do eixo Y
            ylabel: 'Eixo Y'
        }
);
```

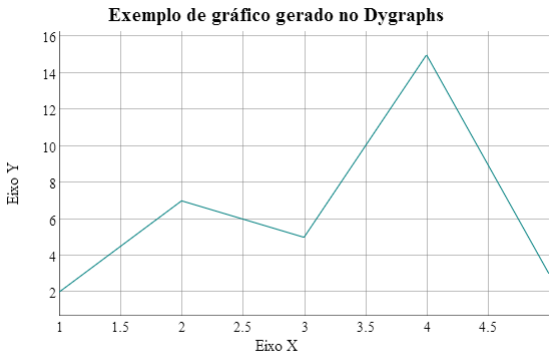


Figura 3.6: Exemplo de código para renderizar um gráfico utilizando o dygraphs

Como se pode notar na Figura 3.6 é criada uma variável chamada *"grafico"* que recebe um objeto do tipo *Dygraph*. No método construtor desse objeto seleciona-se o elemento *div* da página HTML que receberá o gráfico informando seu identificador para a função *document.getElementById*, são passados em forma de vetor os dados a serem plotados (nesse exemplo os dados estão como um vetor, mas pode-se utilizar outros tipos de formatos conforme explicado anteriormente) e por fim as opções de customização do gráfico. O próprio método construtor encarrega-se de renderizar o gráfico na página HTML.

3.3.4 Chart.js

Chart.js é uma biblioteca desenvolvida por Nick Dowine sob a linguagem JavaScript, renderiza os gráficos utilizando o elemento canvas do HTML5, está disponibilizada sob a licença MIT (MIT 2013). A Figura 3.7 mostra os 6 tipos de gráficos disponíveis em Chart.js.

É uma biblioteca extremamente leve e não possui dependências (diferentemente de outras bibliotecas analisadas nesse trabalho). Para melhor compreensão da utilização dessa biblioteca é apresentada na Figura 3.8 um código exemplo demonstrando também o resultado final na página do cliente.

Figura 3.7: Exemplo de tipos de gráficos oferecidos por Chart.js

```
// Opções para o gráfico
var options =  {
    // Mostrar pontos no gráfico
    pointDot: true,
    // Animar o gráfico quando carregá-lo pela primeira vez
    animation: true
};
// Dados para plotagem
var data = {
    // Etiquetas para o eixo X para seus respectivos valores de pontos
    labels : ["Jan","Fev","Mar","Abr","Mai","Jun","Jul"],
    datasets : [
            {
                // Cor para a primeira linha
                fillColor : "rgba(220,220,220,0.5)",
                strokeColor : "rgba(220,220,220,1)",
                pointColor : "rgba(220,220,220,1)",
                // Cor do contorno dos pontos
                pointStrokeColor : "#fff",
                // Dados para plotagem
                data : [65,59,90,81,56,55,40]
            },
            {
                // Cor para a segunda linha
                fillColor : "rgba(151,187,205,0.5)",
                strokeColor : "rgba(151,187,205,1)",
                pointColor : "rgba(151,187,205,1)",
                // Cor do contorno dos pontos
                pointStrokeColor : "#fff",
                // Dados para plotagem
                data : [28,48,40,19,96,27,100]
            }
        ]
};
// Obtem o identificador do canvas e coloca o contexto como duas dimensões
var ctx = document.getElementById("chart").getContext("2d");
// A partir do canvas selecionado chama a função de plotagem de linha da biblioteca
// passando os dados e as opções selecionadas
var myNewChart = new Chart(ctx).Line(data,options);
```

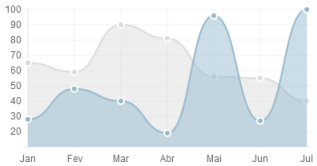


Figura 3.8: Exemplo de código para renderizar um gráfico de linhas utilizando o Chart.js

Analisando-se a Figura 3.8 é possível notar que a plotagem do gráfico pode receber alguns parâmetros: como opções (não obrigatório) que possuem atributos de customização do gráfico, os dados (que possuem as etiquetas para cada valor correspondente do eixo Y), há um conjunto de dados que contém características de cada linha (como cor e transparência) assim como os dados que serão utilizados para a plotagem. Por fim é criada uma varíavel denotada por *"ctx"* que recebe o identificador do elemento canvas contido no html (denotado aqui por *"chart"*), a variável *"ctx"* é passada como argumento para a função *"Line"* da biblioteca Chart.js que realiza a plotagem do gráfico no elemento canvas.

3.3.5 NVD3

O NVD3 (NVD3 2013) é uma biblioteca contruída utilizando a biblioteca D3.js [2] para geração de gráficos na web e está sob a licença Apache Versão 2.0 (Apache License V 2.0 2004). Essa biblioteca utiliza a tag SVG para a plotagem de seus gráficos e conta com 8 tipos de gráficos apresentados na Figura 3.9.

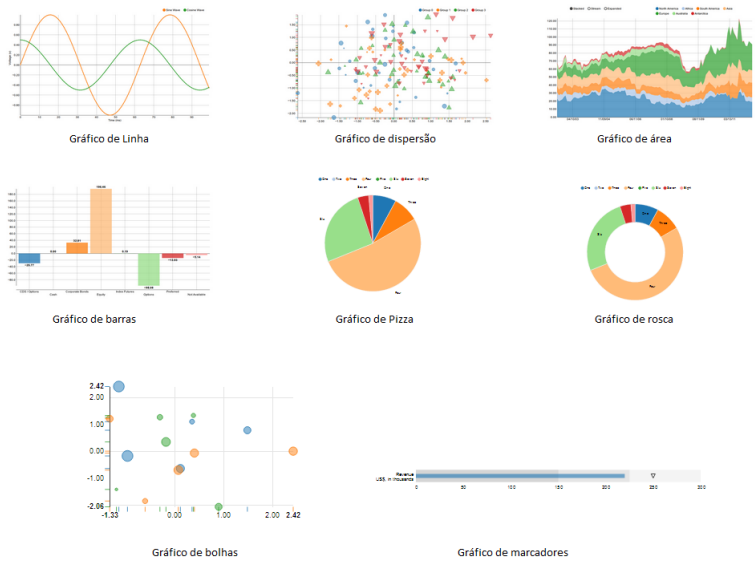


Figura 3.9: Exemplo de tipos de gráficos oferecidos pelo NVD3

Vale a pena mencionar também que a biblioteca possui suporte para a combinação de alguns tipos de gráficos (por exemplo, gráfico de linhas em conjunto com um gráfico de barras).

A Figura 3.10 apresenta um exemplo de código juntamente com o seu resultado final.

```
// Aqui obtemos os dados de seno e cosseno para plotagem de exemplo
dados = gerarSeno();

// Aqui definimos como será as duas séries de dados para a plotagem correta
var seriesPlotagem =  [
    {
      // Define se a linha possuirá uma área
      area: true,
      // Valores no formato [ {x: valor, y: valor}, { x: valor, y: valor}, ... ]
      values: dados.seno,
      // Define qual o nome da série
      key: "Funcao seno",
      // Cor da linha no gráfico
      color: "#ff7f0e"
    }
];

// Define que a variável chart receberá um gráfico de linhas
var chart = nv.models.lineChart();

// Aqui é definido o formato de dado que será apresentado nos intervalos do gráfico
// Por exemplo: ",.1f" retorna números com 1 casa após a virgula
chart.xAxis
    .tickFormat(d3.format(',.1f'));

// Aqui são colocadas opções para o eixo Y, como seu nome e também o formato
// de dados que serão apresentados neste eixo (",.2f" apresenta duas casas após a virgula)
chart.yAxis
    .axisLabel('Voltage (v)')
    .tickFormat(d3.format(',.2f'));

// Aqui é informado à biblioteca d3 qual o elemento svg que deve receber a plotagem
// e os dados a serem usados, realizando a plotagem logo em seguida
d3.select('#chart1 svg')
  .datum(seriesPlotagem)
  .transition().duration(500)
  .call(chart);
```

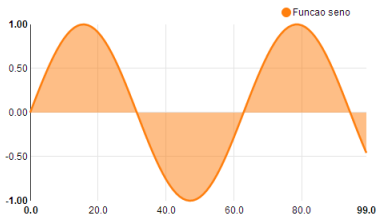


Figura 3.10: Exemplo de código para renderização de gráfico de linhas utilizando o NVD3

[2] D3.js (D3.js 2012) é uma biblioteca para criação de gráficos utilizando o elemento SVG.

A Figura 3.10 apresenta à sua esquerda um código de exemplo que gera valores utilizando a função seno (na chamada *"gerarSeno"*) e em seguida é criada uma variável chamada *"seriesPlotagem"* que conterá algumas informações citadas abaixo:

- **area**: define se a série irá ter uma área;
- **values**: possui um vetor contendo objetos com atributos *x* e *y* que mapeiam x e y, respectivamente;
- **key**: atributo que contém o nome da série em questão;
- **color**: contém a cor que a série deverá receber na plotagem do gráfico.

Logo após é inicializada uma varíavel *chart* que será utilizada para realizar a plotagem de um gráfico de linhas, onde são definidas algumas opções de seus eixos (como nome e formato dos dados). Em seguida é feita uma chamada à biblioteca gráfica *D3* que se encarregará de renderizar o objeto *chart* na tela.

3.3.6 dc.js

O dc.js é uma biblioteca escrita utilizando JavaScript, criado também utilizando a biblioteca gráfica D3.js (D3.js 2012) e utiliza a tag SVG para renderização, assim como o NVD3, explicado anteriormente. Essa biblioteca está sob a licença Apache Versão 2.0 (Apache License V 2.0 2004) e possui integração com o Crossfilter (Crossfilter 2012), que é uma série de funções para JavaScript destinadas para exploração de dados multidimensionais de forma rápida e otimizada, tornando a renderização dos gráficos mais eficiente.

O dc.js possui 6 tipos de gráficos, apresentados na Figura 3.11.

Figura 3.11: Exemplo de tipos de gráficos oferecidos pelo jQuery Sparklines

Para uma melhor compreensão dessa biblioteca é apresentado na Figura 3.12 um exemplo de código com o seu respectivo resultado.

```
//dados
var data = [
  {
      date: '02-01-2013',
      val: 13
  },
  {
      date: '02-25-2013',
      val: 99
  },
  {
      date: '03-01-2013',
      val: 15
  },
  {
      date: '04-01-2013',
      val: 9
  },
  {
      date: '05-01-2013',
      val: 22
  },
  {
      date: '06-01-2013',
      val: 1
  },
  {
      date: '07-01-2013',
      val: 6
  }

];
// obtem objeto crossfilter a partir dos dados
var cf = crossfilter(data);
// seta a dimensão como a data
var dim = cf.dimension(function(d){ return new Date(d.date); });
// agrupa os dados de cada mês
var group = dim.group().reduceSum(function(d){return d.val;});
// cria um novo gráfico de linhas
var lineChart = dc.lineChart('#chart')
        // configura a largura e altura do gráfico
        .width(850)
        .height(250)
        // configura as margens
        .margins({top: 10, right: 50, bottom: 30, left: 40})
        // configura a dimensão
        .dimension(dim)
        // configura o grupo
        .group(group)
        // opção para formatação de data no eixo X
        .x(d3.time.scale()
                .domain([new Date(data[0].date),
                        new Date(data[data.length-1].date)]))
        // renderiza linhas horizontais para facilitar a leitura
        .renderHorizontalGridLines(true)
        // renderiza linhas verticais para facilitar a leitura
        .renderVerticalGridLines(true)
        // por fim, renderiza o gráfico
        .render();
```

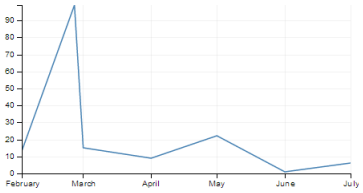


Figura 3.12: Exemplo de código para a renderização de um gráfico de linhas utilizando o dc.js

Como é possível notar no código apresentado na Figura 3.12 primeiramente temos os dados que serão utilizados por essa biblioteca (denotados pela variável *data*, que é um vetor de objetos). Após a declaração dos dados, criamos um objeto do tipo *crossfilter* passando esses dados como argumento (pois como foi explicado anteriormente, essa biblioteca utiliza o *crossfilters* para a manipulação de seus dados), logo em seguida é setada a dimensão [3] e o grupo (onde informamos, através da função *reduceSum*, qual o campo que estaremos utilizando para o eixo Y). Após todas essas configurações informamos à biblioteca que desejamos plotar um gráfico de linhas, enviando o identificador da *div* que estará recebendo o elemento *svg* e algumas opções de customização. Por fim, o gráfico é renderizado a partir da função *render*.

[3]Dimensão: Neste escopo, a dimensão é a forma como queremos seccionar os dados.

3.3.7 jQuery Sparklines

O jQuery Sparklines é uma biblioteca criada por Gareth Watts e está disponível sob a licença BSD (BSD 2005), foi desenvolvida utilizando o JavaScript aliado ao jQuery e plota seus gráficos no elemento canvas do HTML5. Ela atualmente suporta 5 tipos de gráficos apresentados na Figura 3.14.

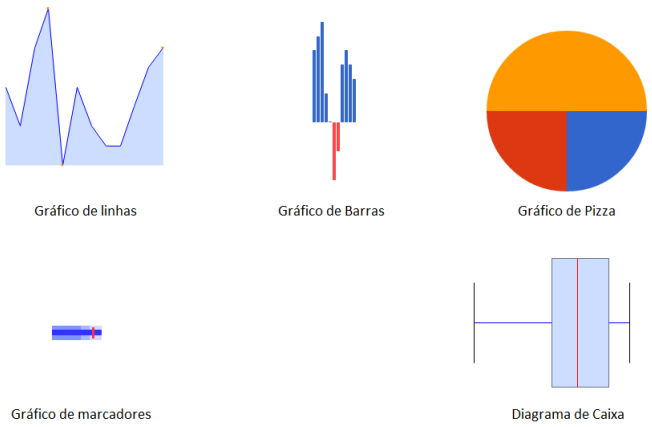
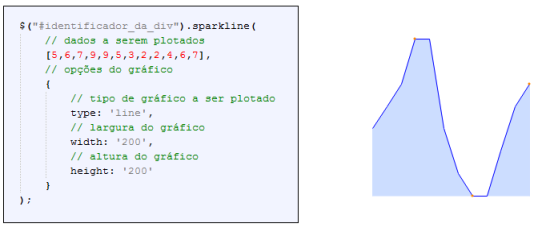


Figura 3.13: Exemplo de tipos de gráficos oferecidos pelo jQuery Sparklines

Essa biblioteca utiliza *sparklines*, ou seja, pequenos gráficos para visualizações simplificadas de dados, não apresentando valores tanto nos eixos x ou y (é necessário que o ponteiro do mouse esteja em cima do ponto plotado para saber quais são os seus devidos valores). A Figura 3.14 apresenta um exemplo de código com o seu devido resultado.



```
$("#identificador_da_div").sparkline(
    // dados a serem plotados
    [5,6,7,9,9,5,3,2,2,4,6,7],
    // opções do gráfico
    {
        // tipo de gráfico a ser plotado
        type: 'line',
        // largura do gráfico
        width: '200',
        // altura do gráfico
        height: '200'
    }
);
```

Figura 3.14: Exemplo de código para a renderização de um gráfico de linhas utilizando o jQuery Sparklines

Como é possível perceber em 3.14 o código necessário para a geração de um gráfico nessa biblioteca é simples. É preciso informar o identificador do elemento *div* que receberá o *canvas* com o gráfico realizando a chamada *sparkline*, enviando como argumento os dados a serem plotados e um objeto contendo algumas opções, como o tipo de gráfico, a largura e a altura (esses dois últimos não sendo suportados por todos os tipos de gráficos dessa biblioteca).

4 METODOLOGIAS DE VALIDAÇÃO

Para a validação deste trabalho é necessária a discussão de algumas metodologias que avaliem as características de cada biblioteca selecionada. Essas características, porém, nem sempre se apresentam de forma quantitativa (como tempo de renderização, tamanho total do componente, quantidade de pontos suportatos por plotagem e número de eventos suportados), sendo necessário também que se leve em conta fatores qualitativos (como a presença (ou ausência) de documentação e tutoriais) assim como o tempo que um programador levaria para utilizar essas bibliotecas (esse critério sendo mais subjetivo).

Neste capítulo serão abordadas as métricas selecionadas para avaliação das bibliotecas, sendo separadas em métricas quantitativas e qualitativas.

4.1 Métricas quantitativas

4.1.1 Quantidade de pontos suportados por plotagem

Algumas vezes há a necessidade de plotar uma grande quantidade de pontos em um gráfico sem perder informação, pois a perda de informação pode acarretar problemas de precisão ou até mesmo em transtornos legais (como, por exemplo, vários usuários de um banco desejam saber quais foram as suas movimentações financeiras durante um longo período de tempo, se algum dado for perdido a situação será crítica). Para esse tipo de escopo um banco de dados simulando a rede de São Paulo da RNP foi criado (mais de 1 milhão de pontos representando o uso de um período de 10 anos com amostragem de 5 minutos para realizar os testes de plotagem).

Para essa métrica serão levados em conta gráficos de linha e gráficos de barras (quando as bibliotecas disponibilizarem os mesmos). A largura do gráfico a ser plotado será de 1500 pixels, pois de acordo com (Screen Resolution Survey 2013) a maioria dos usuários atualmente utilizam resoluções maiores que 1024x768. Nesse tipo de teste o objetivo não é a interpretação do gráfico, sim a quantidade de pontos que são suportados em cada biblioteca analisada.

4.1.2 Tempo de renderização

Essa métrica visa verificar o tempo que cada biblioteca selecionada leva para apresentar seus gráficos ao usuário. O cálculo começará no momento em que a biblioteca recebe os dados para a plotagem até sua apresentação na tela, onde serão testados gráficos de linhas e barras com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 pontos (caso a biblioteca suporte até essa quantidade de dados). Cada teste será executado um total de 30 vezes e, logo após, será calculado o tempo médio para cada caso e será incluído o desvio padrão para as 30 execuções de cada teste.

4.1.3 Tamanho total da biblioteca

Esta métrica visa verificar o tamanho total da biblioteca para utilização (serão contabilizadas também possíveis dependências caso sejam utilizadas). É necessário ter esse conhecimento para que seja possível estimar o tamanho total do projeto caso a API venha a fazer parte dele.

4.1.4 Eventos de mouse suportados

Esta métrica visa analisar quantos eventos são suportados pelas bibliotecas selecionadas (esses eventos serão explicados na seção 5.1.6). Esta métrica é importante, pois assim é possível saber o quão interativo são os gráficos de cada componente com o usuário. Para essa métrica são levados em conta apenas eventos de mouse.

4.1.5 Tempo de implementação

Esta métrica tem por objetivo obter o tempo preciso para agregação da biblioteca em um sistema. Nessa métrica é levado em conta o tempo que o autor levou para utilizar a API no projeto e está relacionada diretamente com a complexidade encontrada na implementação. Nota-se que esta medida seria mais precisa caso fossem utilizados vários programadores para a implementação de cada uma das bibliotecas, porém devido à falta de tempo e de candidatos somente um programador fará as implementações (o autor). O autor possui 5 anos de experiência com aplicações web e fará um levantamento do tempo que foi necessário para que fosse possível utilizar cada uma das API's selecionadas nesse trabalho.

4.2 Métricas qualitativas

4.2.1 Documentação

Esta métrica apresenta os tipos de documentação que cada biblioteca possui, como a existência de uma documentação oficial, comunidades ativas e exemplos de código.

4.3 Ambiente de validação

Para realizar os testes deste trabalho, criou-se um ambiente de testes explicado em detalhes nesta seção.

4.3.1 Banco de dados

O MySQL foi utilizado para tornar disponíveis os dados para geração de gráficos pela aplicação, onde foi utilizado um banco de dados com mais de 1 milhão de entradas geradas através de um script PHP simulando a rede RNP [1] em São Paulo (onde são simulados tráfegos de entrada e saída em bits por segundo, utilizando amostragem de 5 minutos com intervalo de 10 anos). A seguir é apresentado o algoritmo simplificado para geração de valores de tráfegos de entrada e saída e inserção no banco de dados.

```
Conectar ao banco de dados
horarioFinal = 01/01/2010
```

[1] RNP ou Rede Nacional de Ensino e Pesquisa é a primeira rede de acesso à internet do Brasil e integra mais de 800 instituições de ensino e pesquisa do país. Seu objetivo é melhorar a infraestrutura de redes em níveis nacional, metropolitano e local (redes de campus) (RNP 2013).

```
horario = 01/01/2000

enquanto horario <= horarioFinal:

  Caso horario esteja entre 24hrs ate 3hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(1.2,1.6)
    Saida = valor_ponto_flutuante_aleatorio_entre(1.2,1.6)
  Caso horario esteja entre 4hrs ate 7hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(0.8,1.6)
    Saida = valor_ponto_flutuante_aleatorio_entre(0.6,1.2)
  Caso horario esteja entre 8hrs ate 8hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(0.8,1.8)
    Saida = valor_ponto_flutuante_aleatorio_entre(0.6,1.2)
  Caso horario esteja entre 9hrs ate 12hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(1.8,2.5)
    Saida = valor_ponto_flutuante_aleatorio_entre(1.2,2.0)
  Caso horario esteja entre 13hrs ate 14hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(2.5,2.8)
    Saida = valor_ponto_flutuante_aleatorio_entre(2.0,2.4)
  Caso horario esteja entre 15hrs ate 16hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(2.8,3.5)
    Saida = valor_ponto_flutuante_aleatorio_entre(2.3,2.7)
  Caso horario esteja entre 17hrs ate 18hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(2.5,2.8)
    Saida = valor_ponto_flutuante_aleatorio_entre(2.0,2.3)
  Caso horario esteja entre 19hrs ate 20hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(2.5,2.0)
    Saida = valor_ponto_flutuante_aleatorio_entre(2.0,2.3)
  Caso horario esteja entre 21hrs ate 23hrs:
    Entrada = valor_ponto_flutuante_aleatorio_entre(2.0,1.8)
    Saida = valor_ponto_flutuante_aleatorio_entre(1.4,2.3)

  Insira no banco de dados com valores de Entrada, Saida e horario
  horario = horario + 5 minutos
```

O banco de dados criado objetiva justificar os testes de número de pontos máximo suportado por plotagens, com dados que tenham algum significado e reflitam uma situação real.

4.3.2 Arquitetura do código produzido para testes

O projeto para testar as bibliotecas de geração de gráficos utiliza as linguagens e tecnologias descritas previamente. A Figura 4.1 apresenta um diagrama descrevendo o funcionamento do software criado.

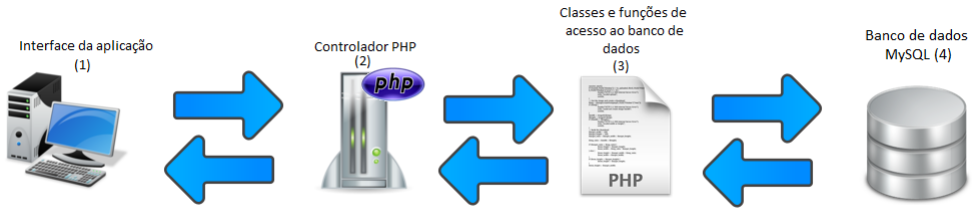


Figura 4.1: Arquitetura simplificada do projeto para teste de bibliotecas

O usuário acessa a aplicação em (1) e seleciona o número de pontos que ele deseja e o tipo de gráfico a ser plotado. Após o clique é feita uma requisição AJAX enviando os parâmetros em POST para o servidor, diretamente para o controlador PHP (2) que acessa a classe de seleção de dados (3) que cria uma consulta ao banco MySQL (4), enviando em seguida os pontos para o controlador que repassa os dados em JSON [2] até a função JavaScript que iniciou a requisição. O gráfico então é plotado na interface (1), mostrando os resultados na página do cliente.

Para a seleção dos dados do banco de dados criou-se uma classe chamada *Database*, que no seu método construtor efetua uma conexão com o banco de dados MySQL. A classe possui uma série de funções de seleção de dados específicas para cada tipo de biblioteca, pois cada uma possui um método diferente de manipular os dados que recebe. Houve também a necessidade de criar consultas que exportam os resultados para um arquivo auxiliar, seguindo restrições de algumas API's utilizadas neste trabalho. Para o correto funcionamento de cada um dos componentes agregados no ambiente de testes, criou-se um controlador PHP, um arquivo com funções auxiliares e de disparo de plotagem criado em JavaScript e jQuery e um arquivo HTML, onde esses arquivos são únicos para cada biblioteca analisada.

Para se conhecer melhor o fluxo do programa, a Figura 4.2 apresenta as telas implementadas no ambiente de testes. O menu à esquerda é a tela principal do programa, onde o usuário seleciona a biblioteca a ser testada (pode-se notar que a biblioteca Chart.js se encontra selecionada no exemplo). As telas apresentadas à direita da mesma figura apresentam o painel de testes para cada uma das API's analisadas neste trabalho. Pode-se perceber na Figura 4.2 que em algumas bibliotecas implementou-se mais gráficos do que o necessário para a realização dos testes. Esse fato deve-se à uma maior investigação dentro da documentação disponível para se ter uma avaliação mais abrangente sobre a biblioteca em análise.

[2]JSON (JavaScript Object Notation) é uma formatação leve para troca de dados baseado em texto (JSON 2013).

Figura 4.2: Execução da aplicação implementada para esse trabalho

5 EXPERIMENTOS

Nesta seção serão apresentados testes de bibliotecas selecionadas para esse trabalho. Os testes onde se fez necessário executar o programa foram feitos em uma máquina com um processador AMD FX 6100 @ 3.3Ghz, memória de 4 GB e sistema operacional Windows 7.

5.1 Métricas Qualitativas

5.1.1 Quantidade de pontos suportados por plotagem

Os testes realizados nesta seção não tem por objetivo que o leitor realize a interpretação dos gráficos, sendo relevante somente o valor máximo de dados que cada biblioteca suporta para cada tipo de gráfico analisado neste trabalho. Todos os gráficos utilizam os mesmos dados do banco explicados na seção 4.3.1.

5.1.1.1 Chart.js

O valor máximo de pontos plotáveis dentro de um espaço de 1500 pixels de largura para gráficos de linha é 1371 conforme demonstrado na Figura 5.1.

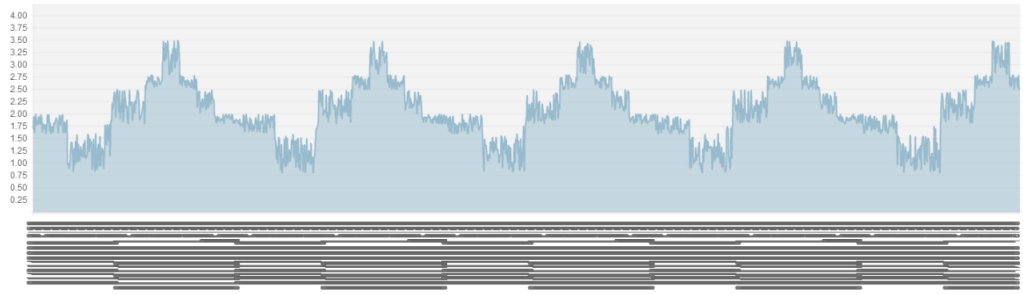


Figura 5.1: Gráfico de linhas produzido pelo Chart.js com 1371 pontos em espaço de 1500 pixels de largura

Para o gráfico de barras, temos um máximo de 1370 pontos conforme mostra a Figura 5.2.

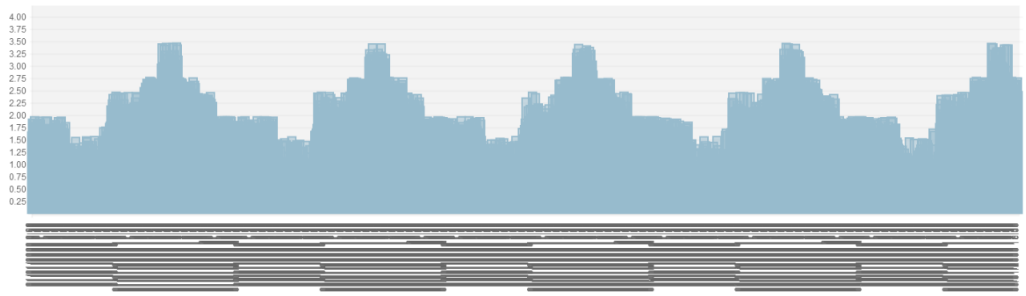


Figura 5.2: Gráfico de barras produzido pelo Chart.js com 1370 pontos em espaço de 1500 pixels de largura

Acima de 1371 pontos para gráfico de linhas e acima de 1370 pontos para gráfico de barras temos o erro apresentado na Figura 5.3.

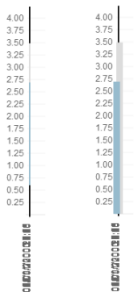


Figura 5.3: Exemplo de erro de renderização do Chart.js (à esquerda gráfico de linhas, à direita o de barras)

Como se pode perceber a biblioteca (em sua versão atual) não permite a simplificação de pontos no eixo X, tornando impossível a leitura das informações apresentadas no gráfico.

5.1.1.2 Flot

O valor máximo de pontos plotáveis dentro de um espaço de 1500 pixels de largura para o gráfico de linha é de 100.000 pontos, apresentado na Figura 5.4.

Para o gráfico de barras o valor máximo é de 500.000 pontos (muito superior ao de linhas), apresentado na Figura 5.5.

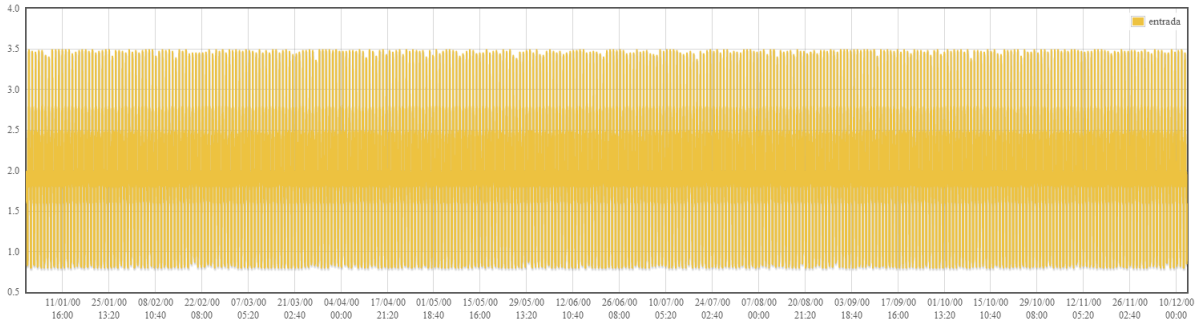


Figura 5.4: Gráfico de linhas produzido pelo Flot com 100.000 pontos em um espaço de 1500 pixels de largura

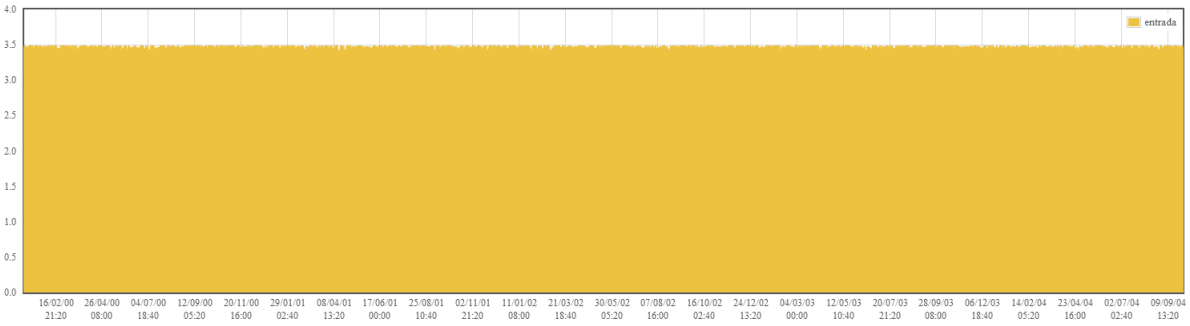


Figura 5.5: Gráfico de barras produzido pelo Flot com 500.000 pontos em um espaço de 1500 pixels de largura

A renderização do gráfico trava para valores maiores nos dois casos.

5.1.1.3 jqPlot

A biblioteca jqPlot suporta até 125.000 pontos no gráfico de linhas conforme a Figura 5.6.

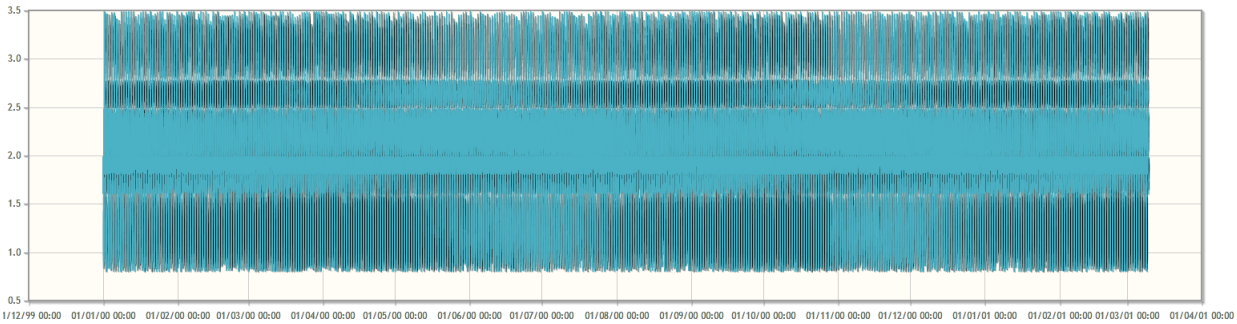


Figura 5.6: Gráfico de linhas produzido pelo jqPlot com 125.000 pontos.

Para o gráfico de barras são suportados até 125.000 pontos, esse exemplo é demonstrado na Figura 5.7.

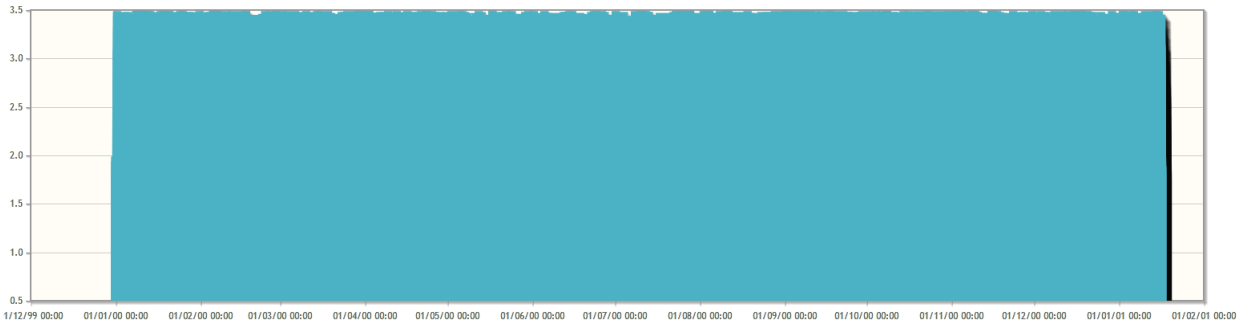


Figura 5.7: Gráfico de barras produzido pelo jqPlot com 125.000 pontos.

É possível notar que o intervalo não está corretamente configurado nos dois gráficos, porém, um intervalo menor causa a aglomeração do texto no eixo x conforme apresentado na Figura 5.8. Essa margem tanto à esquerda quanto à direita não interfere no teste de número de pontos total que o gráfico suporta, visto que ele causava problemas quanto ao tamanho do vetor no qual ele operava e não havia problemas relacionados com o espaço para a renderização (tanto do gráfico de linhas como o de barras).

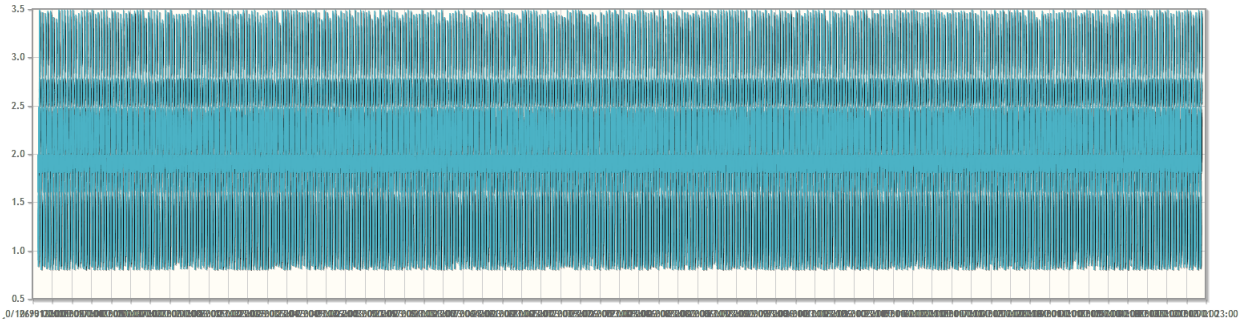


Figura 5.8: Erro obtido no eixo X ao definir um intervalo menor para a prevenção de margens indesejadas.

5.1.1.4 dygraphs

Para o dygraphs foi possível utilizar todos os 1.051.944 pontos disponíveis no banco, plotando ainda duas séries (tráfego de entrada e tráfego de saída em bits por segundo) sem nenhuma complicação (perda de desempenho ou lentidão). No gráfico plotado também é possível efetuar o zoom para a leitura ficar mais fácil. A Figura 5.9 apresenta o resultado do teste.

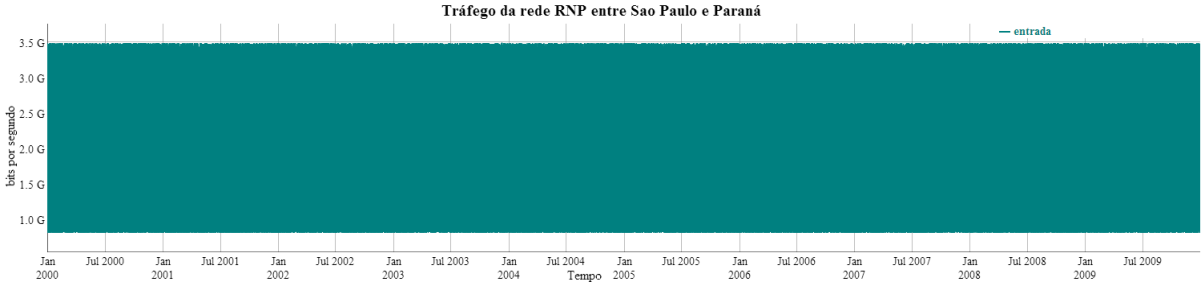


Figura 5.9: Gráfico de linhas produzido pelo dygraphs com 1.051.944 pontos, utilizando duas séries.

5.1.1.5 NVD3

A biblioteca NVD3 suporta em torno de 150.000 pontos em gráfico de linhas (embora não seja recomendado que se use um número tão alto pois há uma perda de desempenho considerável) conforme a Figura 5.10.

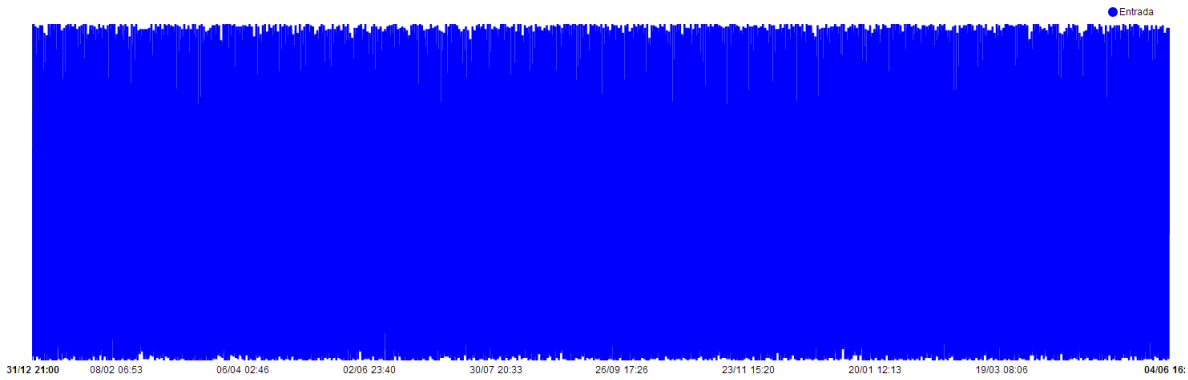


Figura 5.10: Gráfico de linhas produzido pela NVD3 com 150.000 pontos.

Foi possível plotar até 40.000 pontos para o gráfico de barras, o resultado se encontra na Figura 5.11.

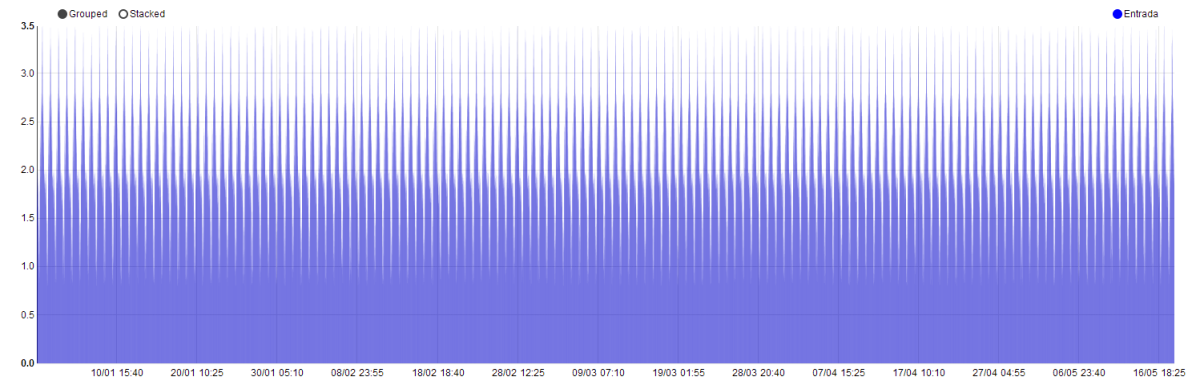


Figura 5.11: Gráfico de linhas produzido pela NVD3 com 40.000 pontos.

Como se pode notar essa biblioteca suporta uma quantidade grande de pontos a serem plotados, mas, como ela utiliza a tag *SVG* para a renderização dos gráficos, o desempenho fica prejudicado. A tentativa de utilização de valores acima dos utilizados aqui geram erro no navegador.

5.1.1.6 dc.js

Para o gráfico de linhas foi possível gerar 500.000 pontos (apresentado na Figura 5.12), porém, uma tag *SVG* com esta quantidade de elementos torna a interação do gráfico com o usuário inviável (extrema lentidão). Para testes com valores maiores ocorre o travamento do navegador.

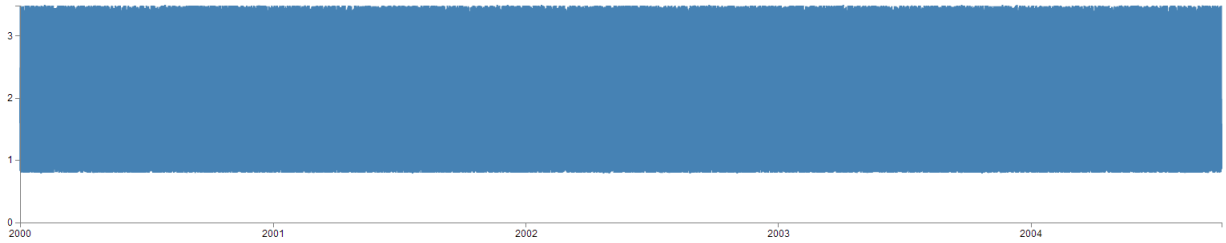


Figura 5.12: Gráfico de linhas produzido pelo dc.js com 500.000 pontos.

Para o gráfico de barras o máximo obtido foram 90.000 pontos conforme a Figura 5.13 demonstra, qualquer valor acima de 90.000 pontos causa o congelamento do navegador.

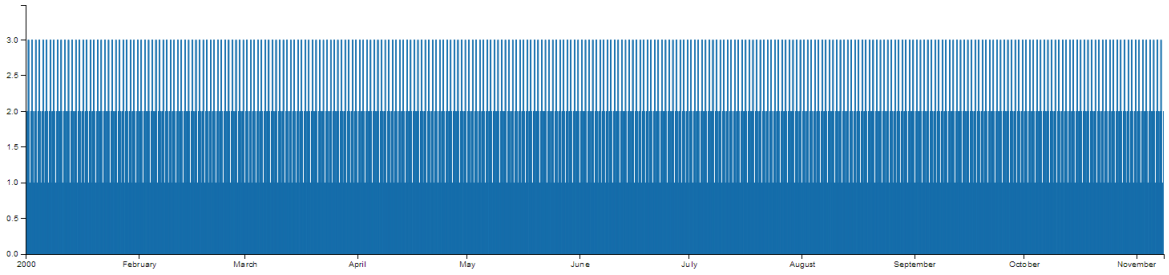


Figura 5.13: Gráfico de barras produzido pelo dc.js com 90.000 pontos.

5.1.1.7 jQuery Sparklines

Para a biblioteca jquery Sparklines foi somente possível realizar os testes para o gráfico de linhas, já que o gráfico de barras não possui a opção de regular a largura do *canvas*. Foi possível renderizar 125.000 pontos para o gráfico de linhas utilizando esta biblioteca, conforme apresentado na Figura 5.14.

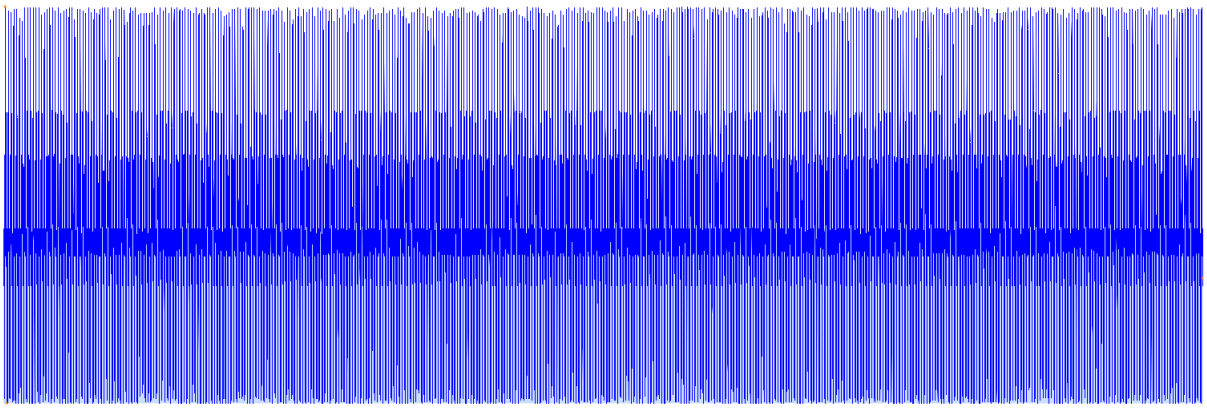


Figura 5.14: Gráfico de linhas produzido pelo jquerySparklines com 125.000 pontos.

5.1.2 Gráfico comparativo para máximo pontos de plotagem

A Figura 5.15 apresenta um breve gráfico comparando os resultados obtidos nesta seção para uma visualização geral sobre os testes com o gráfico de linhas.

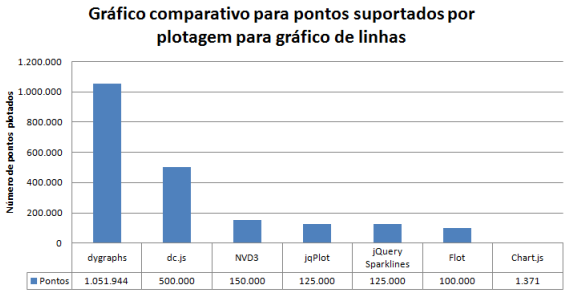


Figura 5.15: Gráfico comparativo para número de pontos máximo por plotagem para cada biblioteca utilizando o gráfico de linhas.

A Figura 5.16 apresenta um comparativo para as bibliotecas utilizando o gráfico de barras.

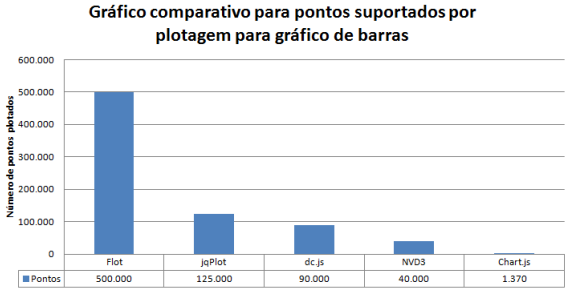


Figura 5.16: Gráfico comparativo para número de pontos máximo por plotagem para cada biblioteca utilizando o gráfico de barras.

5.1.3 Tempo de Renderização

5.1.3.1 Flot

Para a renderização do gráfico de linhas para a biblioteca Flot foram obtidos os resultados apresentados na tabela 5.1.

Número de pontos	Tempo (ms)	Desvio Padrão
10	22,77	7,27
100	22,03	4,75
1.000	28,07	2,95
10.000	123,87	5
100.000	1.093,3	19,86

Tabela 5.1: Tempo de renderização para o gráfico de linhas na biblioteca Flot

Os resultados utilizando o gráfico em barras são apresentados na tabela 5.2.

Número de pontos	Tempo (ms)	Desvio Padrão
10	21,1	5,15
100	24,23	4,18
1.000	150,73	3,71
10.000	1.429,53	21
100.000	14.097,2	275,59

Tabela 5.2: Tempo de renderização para o gráfico de barras na biblioteca Flot

5.1.3.2 jqPlot

Os resultados para o gráfico de linhas na biblioteca jqPlot encontram-se na tabela 5.3.

Número de pontos	Tempo (ms)	Desvio Padrão
10	24,47	9,05
100	54,3	8,34
1.000	334,1	13,03
10.000	3.139,37	53,02
100.000	34.629,9	1721,7

Tabela 5.3: Tempo de renderização para o gráfico de linhas na biblioteca jqPlot

Os resultados para os testes realizados no gráfico de barras encontram-se na tabela 5.4.

Número de pontos	Tempo (ms)	Desvio Padrão
10	26,53	10,65
100	53,7	10,17
1.000	331,7	13,33
10.000	3.108,57	53,72
100.000	30.591,3	404,36

Tabela 5.4: Tempo de renderização para o gráfico de barras na biblioteca jqPlot

5.1.3.3 dygraphs

Para o dygraphs foram testados até um milhão de pontos e os resultados dos testes encontram-se na tabela 5.5.

Número de pontos	Tempo (ms)	Desvio Padrão
10	12,2	10,37
100	12,27	7,03
1.000	16,9	6,55
10.000	65,47	5,46
100.000	549,73	59,73
1.000.000	1.891,93	62,86

Tabela 5.5: Tempo de renderização para o gráfico de linhas na biblioteca dygraphs

5.1.3.4 Chart.js

Para o gráfico de linhas na biblioteca Chart.js obteve-se os resultados apontados na tabela 5.6. É possível notar que o tempo foi descrescendo conforme o número de pontos aumentava, isso se deve a um problema que faz com que a biblioteca realize o redimensionamento do elemento *canvas* conforme o número de pontos, diminuindo-o de tamanho conforme o número de pontos para a plotagem era acrescido.

Número de pontos	Tempo (ms)	Desvio Padrão
10	23,17	4,4
100	10,83	2,21
1.000	7,57	1,05

Tabela 5.6: Tempo de renderização para o gráfico de linhas na biblioteca Chart.js

Para o gráfico de barras obteve-se o resultado apresentado na tabela 5.7.

Número de pontos	Tempo (ms)	Desvio Padrão
10	21,03	5,33
100	10,03	2,39
1.000	8,07	1,36

Tabela 5.7: Tempo de renderização para o gráfico de barras na biblioteca Chart.js

O mesmo problema ocorre para o gráfico de barras, conforme o número de pontos é acrescido o espaço total de renderização do gráfico é diminuído (mesmo possuindo uma área de largura total de 1500 pixels), ocasionando o tempo menor.

5.1.3.5 NVD3

A tabela 5.8 apresenta os resultados obtidos dos testes para o gráfico de linhas.

Número de pontos	Tempo (ms)	Desvio Padrão
10	30,7	15,9
100	51,13	17,12
1.000	231,43	11,57
10.000	2.084,93	157,67
100.000	20.557,2	1.203,29

Tabela 5.8: Tempo de renderização para o gráfico de linhas na biblioteca NVD3

A tabela 5.9 apresenta os resultado dos testes para o gráfico de barras.

Número de pontos	Tempo (ms)	Desvio Padrão
10	29,7	12,87
100	78,87	17,48
1.000	559,73	62,04
10.000	6.499,4	336,03

Tabela 5.9: Tempo de renderização para o gráfico de barras na biblioteca NVD3

5.1.3.6 dc.js

Para o gráfico de linhas obteve-se os resultados apresentados na tabela 5.10.

Número de pontos	Tempo (ms)	Desvio Padrão
10	44,2	10,22
100	57,87	8,83
1.000	104,9	11,53
10.000	632,03	133,1
100.000	5.665,87	869,69

Tabela 5.10: Tempo de renderização para o gráfico de linhas na biblioteca dc.js

Utilizando o mesmo teste no gráfico de barras obteve-se os resultados apresentados na tabela 5.11.

Número de pontos	Tempo (ms)	Desvio Padrão
10	49,23	5,31
100	68,7	7,17
1.000	343,5	101,32
10.000	2.154,1	340,01

Tabela 5.11: Tempo de renderização para o gráfico de barras na biblioteca dc.js

5.1.3.7 jQuery Sparklines

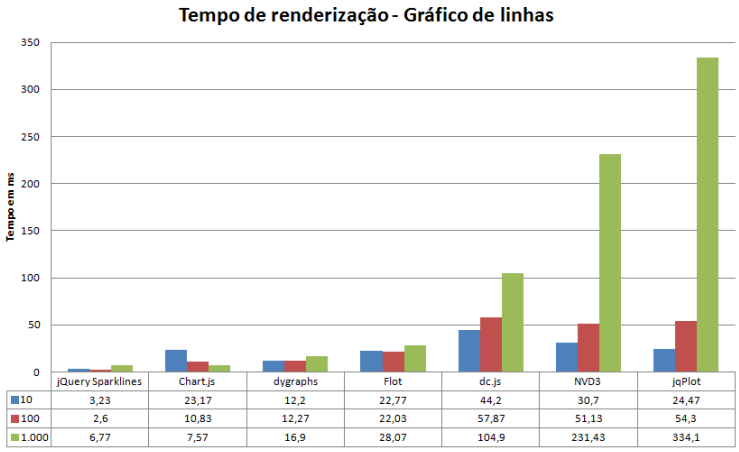
Os resultados para o gráfico de linhas no jQuery Sparklines são apresentados na tabela 5.12.

Número de pontos	Tempo (ms)	Desvio Padrão
10	3,23	2,87
100	2,6	0,49
1.000	6,77	1,89
10.000	39,33	3,64
100.000	400,97	30,57

Tabela 5.12: Tempo de renderização para o gráfico de linhas na biblioteca jQuery Sparklines

5.1.4 Gráfico comparativo para tempo de renderização

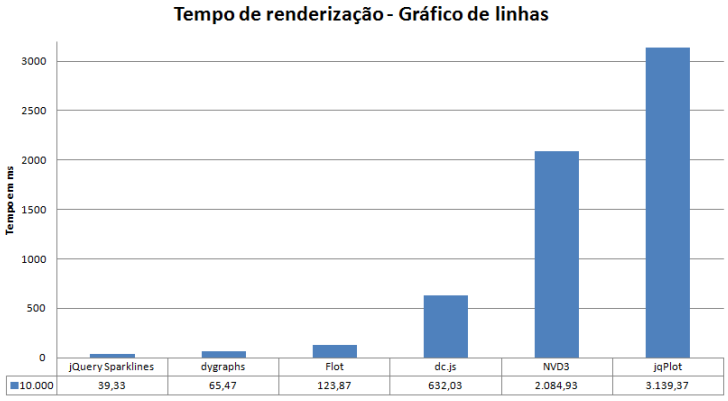
A Figura 5.17 apresenta o gráfico comparativo para tempo de renderização de gráficos de linhas no intervalo de 10 até 1.000 pontos.



	jQuery Sparklines	Chart.js	dygraphs	Flot	dc.js	NVD3	jqPlot
10	3,23	23,17	12,2	22,77	44,2	30,7	24,47
100	2,6	10,83	12,27	22,03	57,87	51,13	54,3
1.000	6,77	7,57	16,9	28,07	104,9	231,43	334,1

Figura 5.17: Gráfico comparativo para tempo de renderização de gráficos de linhas com 10 até 1.000 pontos

A Figura 5.18 apresenta o gráfico comparativo para tempo de renderização de gráficos de linhas com 10.000 pontos.



	jQuery Sparklines	dygraphs	Flot	dc.js	NVD3	jqPlot
10.000	39,33	65,47	123,87	632,03	2.084,93	3.139,37

Figura 5.18: Gráfico comparativo para tempo de renderização de gráficos de linhas para 10.000 pontos

A Figura 5.19 apresenta o gráfico comparativo para tempo de renderização de gráficos de linhas no intervalo de 100.000 até 1.000.000 de pontos.

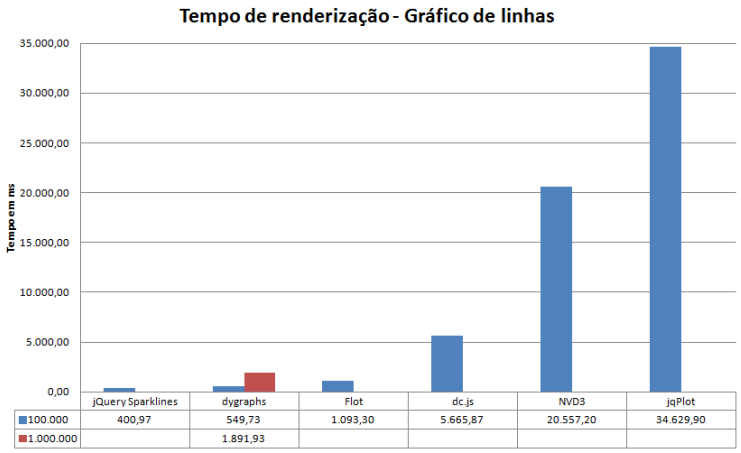


Figura 5.19: Gráfico comparativo para tempo de renderização de gráficos de linhas com 100.000 até 1.000.000 de pontos

A Figura 5.20 apresenta o gráfico comparativo para tempo de renderização de gráficos de barras no intervalo de 10 até 1.000 pontos.

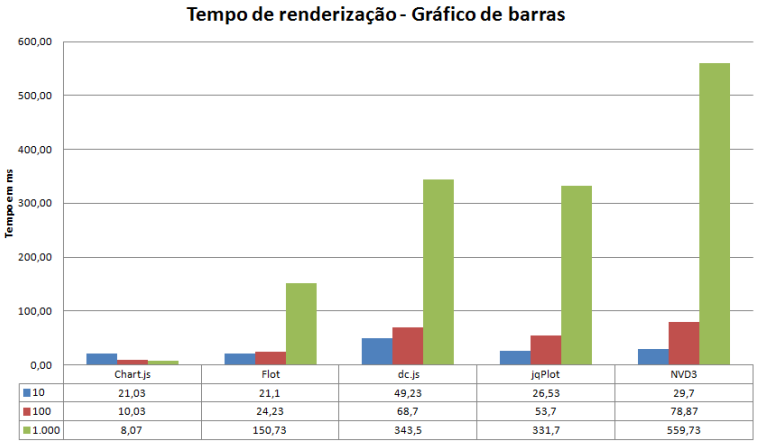


Figura 5.20: Gráfico comparativo para tempo de renderização de gráficos de barras com 10 até 1.000 pontos

A Figura 5.21 apresenta o gráfico comparativo para tempo de renderização de gráficos de barras no intervalo de 10.000 até 100.000 pontos.

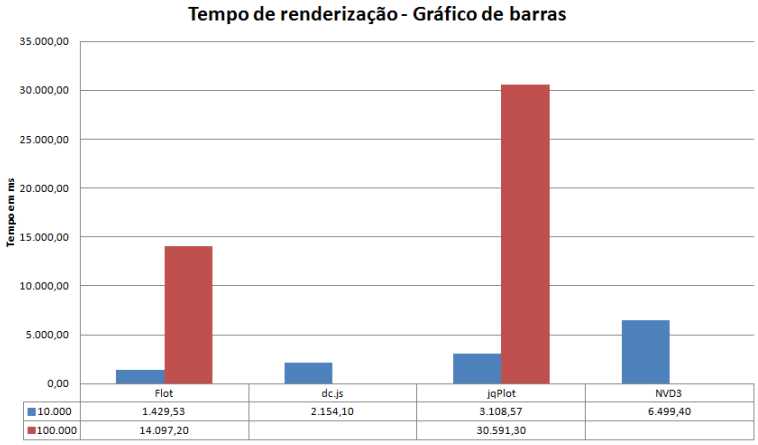


Figura 5.21: Gráfico comparativo para tempo de renderização de gráficos de barras com 10.000 até 100.000 pontos

5.1.5 Tamanho total da biblioteca

A tabela 5.13 apresenta o tamanho de cada uma das bibliotecas analisadas.

Biblioteca	Tamanho
Flot	537 KiloBytes
jqPlot	288,2 KiloBytes
dygraphs	136,1 KiloBytes
Chart.js	20 KiloBytes
NVD3	1,5 MegaBytes
jQuery Sparklines	43 KiloBytes
dc.js	223,1 KiloBytes

Tabela 5.13: Tamanho total de cada biblioteca

5.1.6 Eventos de mouse suportados

Para uma melhor compreensão dos eventos de mouse analisados nesta seção, segue uma breve explicação conforme (jQuery 2013):

- click: evento disparado quando ocorre um clique de mouse em cima de algum elemento (*mousedown* e *mouseup* combinados);
- *dblclick*: evento disparado quando ocorrem dois cliques (clicks) de mouse em cima de algum elemento em um tempo curto.
- *focusout*: evento disparado quando um determinado elemento perde o foco do mouse;
- *hover*: este evento ocorre quando o ponteiro do mouse acessa e deixa a área do elemento que possui este evento (*mouseenter* e *mouseleave* combinados);
- *mousedown*: evento disparado quando o botão esquerdo do mouse é pressionado no elemento que possui este evento;
- *mouseenter*: evento disparado quando o ponteiro do mouse acessa a àrea do elemento que possui este evento;

- *mouseleave*: evento disparado quando o mouse deixa a área do elemento que possui este evento;
- *mousemove*: evento disparado quando o mouse se move na área do elemento que possui este evento;
- *mouseover*: evento disparado quando o ponteiro do mouse está em cima do elemento que possui este evento (o evento é disparado à medida que o mouse se move neste elemento);
- *mouseout*: evento disparado quando o ponteiro do mouse deixa de estar no elemento que possui este evento;
- *mouseup*: evento disparado quando o botão esquerdo do mouse é liberado no elemento que possui este evento;

A tabela 5.14 apresenta os eventos suportados por cada uma das bibliotecas selecionadas para análise.

Evento	Flot	jQuery Sparklines	Chart.js	jqPlot	dygraphs	NVD3	dc.js
click	Sim	Sim	Não	Sim	Sim	Sim	Sim
dblclick	Não	Não	Não	Sim	Sim	Sim	Sim
focusout	Não	Não	Não	Não	Não	Sim	Sim
hover	Sim	Não	Não	Não	Não	Sim	Sim
mousedown	Sim	Não	Não	Sim	Sim	Sim	Sim
mouseenter	Sim	Sim	Não	Sim	Não	Sim	Sim
mouseleave	Sim	Sim	Não	Sim	Não	Sim	Sim
mousemove	Sim	Sim	Não	Sim	Sim	Sim	Sim
mouseover	Sim	Sim	Não	Sim	Sim	Sim	Sim
mouseout	Sim	Não	Não	Sim	Sim	Sim	Sim
mouseup	Sim	Não	Não	Sim	Sim	Sim	Sim

Tabela 5.14: Eventos suportados por biblioteca

5.1.7 Tempo de implementação

Esta seção reflete a opinião do autor sobre a implementação de cada uma das bibliotecas analisadas.

5.1.7.1 Flot

Flot é uma biblioteca de simples aprendizado para programadores que possuem um conhecimento básico de jQuery. É necessário apenas o cuidado de formatar as séries de dados conforme a especificação da biblioteca e criar um HTML que tenha um elemento *div* para receber o canvas gerado pelo Flot, o restante é cuidado pela própria API de forma automática (sendo possível também configurar os intervalos dos eixos caso o resultado não seja o esperado). Para agregar esta biblioteca ao projeto foram necessárias 6 horas.

5.1.7.2 jqPlot

Para o jqPlot foram necessárias 9 horas, pois há a necessidade de tomar cuidado com os tipos diferentes de formatação de dados para os variados tipos de gráficos (fez-se

necessária a implementação de diferentes funções para os diferentes tipos de formatos). Para esta biblioteca especifica-se o elemento *div* da página HTML que deverá receber o gráfico e há a necessidade de um conhecimento prévio em jQuery para facilitar o uso desta API.

5.1.7.3 dygraphs

Para o dygraphs foram necessárias 6 horas, pois envolveu a necessidade de criar uma consulta para o banco de dados que exportasse os dados selecionados em um arquivo do tipo *CSV* e a implementação de rotinas de manipulação de arquivos. O dygraphs não utiliza jQuery, o que torna a implementação mais facilitada no lado do cliente, onde é somente necessário criar o elemento *div* na página HTML e informar à biblioteca para renderizar seu gráfico neste espaço.

5.1.7.4 Chart.js

Para o Chart.js foram necessárias apenas 3 horas, pois a biblioteca é extremamente simples e de fácil implementação. É necessário apenas cuidar como os diferentes tipos de gráficos manipulam as séries que recebem.

5.1.7.5 NVD3

A biblioteca NVD3 é uma biblioteca que oferece uma maior quantidade de opções de customização, porém possui uma complexidade maior devido à este fato. Não é uma atividade simples aprender a utilizar esta biblioteca, fato agravado pela qualidade da documentação também. Utilizou-se 12 horas no aprendizado e implementação desta biblioteca.

5.1.7.6 dc.js

Esta foi a biblioteca mais difícil de se aprender, devido à sua documentação insuficiente e alta complexidade de utilização. Para o dc.js utilizou-se 14 horas.

5.1.7.7 jQuery Sparklines

Esta biblioteca é extremamente simples e intuitiva, embora não ofereça muitas opções de customização e seja focada apenas para a utilização de gráficos pequenos, não sendo indicada para apresentar gráficos complexos. Utilizou-se 1 hora no aprendizado e implementação desta biblioteca.

5.1.8 Tabela comparativa para tempo de implementação

A tabela 5.15

Biblioteca	Tempo
Flot	6 horas
jqPlot	9 horas
dygraphs	6 horas
Chart.js	3 horas
NVD3	12 horas
dc.js	14 horas
jQuery Sparklnes	1 hora

Tabela 5.15: Tempo de implementação por biblioteca

5.2 Métricas Qualitativas

5.2.1 Documentação

5.2.1.1 Flot

- **Documentação**: Essa biblioteca possui uma documentação oficial dentro da página GitHub do projeto;
- **Comunidades ativas**: Essa biblioteca possui comunidades ativas no GitHub;
- **Exemplos de código**: Essa biblioteca possui exemplos de código, eles se encontram dentro do projeto ao se efetuar o download.

5.2.1.2 jqPlot

- **Documentação**: Essa biblioteca possui uma documentação oficial na página do projeto;
- **Comunidades ativas**: Essa biblioteca possui comunidades ativas no GitHub e no google groups;
- **Exemplos de código**: Essa biblioteca possui exemplos de código ao se efetuar o download do projeto (pode-se encontrar vários exemplos, assim como na página do projeto).

5.2.1.3 dygraphs

- **Documentação**: Essa biblioteca possui uma documentação oficial na página do projeto;
- **Comunidades ativas**: Essa biblioteca possui comunidades ativas no google groups, Stack Overflow e GitHub;
- **Exemplos de código**: Essa biblioteca possui exemplos de código na página do projeto.

5.2.1.4 Chart.js

- **Documentação**: Essa biblioteca possui uma documentação oficial dentro da página do projeto;
- **Comunidades ativas**: Essa biblioteca possui uma comunidade ativa no Github;
- **Exemplos de código**: Essa biblioteca possui exemplos de código na página do projeto.

5.2.1.5 NVD3

- **Documentação**: Essa biblioteca não possui documentação oficial, o que dificulta o seu aprendizado.
- **Comunidades ativas**: Essa biblioteca possui comunidades ativas no GitHub e no Stack Overflow;

- **Exemplos de código**: Essa biblioteca possui exemplos de código na página do projeto (exemplos antigos) e também possui alguns exemplos dentro do projeto ao se efetuar o download.

Como essa biblioteca apresenta várias opções de customização para cada tipo de gráfico, assim como cada gráfico manipula de forma diferente o formato de dados que recebe, a documentação desta biblioteca da forma como se apresenta atualmente dificulta o aprendizado.

5.2.1.6 dc.js

- **Documentação**: Essa biblioteca não possui documentação oficial;
- **Comunidades ativas**: Essa biblioteca possui comunidades ativas no GitHub e no Stack Overflow;
- **Exemplos de código**: Essa biblioteca possui alguns exemplos de código que são apresentados no site do projeto.

O próprio autor dessa biblioteca reconhece que o seu forte não é a documentação, os usuários dela devem retirar suas dúvidas em fóruns (Stack Overflow, por exemplo) ou dentro da própria página no GitHub do projeto, tornando o aprendizado dessa biblioteca extremamente oneroso devido à falta de uma documentação oficial.

5.2.1.7 jQuery Sparklines

- **Documentação**: Essa biblioteca possui uma documentação oficial no site da biblioteca;
- **Comunidades ativas**: Essa biblioteca possui uma comunidade ativa no GitHub;
- **Exemplos de código**: Essa biblioteca possui exemplos de código no site oficial do projeto.

6 CONCLUSÕES

Nesse trabalho, foi apresentado um estudo comparativo de bibliotecas geradoras de gráficos para a web. Esse estudo foi motivado pela necessidade de descobrir qual a melhor biblioteca que se destina a um projeto que é implementado, pois não é uma tarefa trivial realizar a escolha sem se conhecer as API's que são oferecidas para uso atualmente. Em relação ao objetivo, pode-se dizer que este trabalho realizou esse estudo comparativo com sucesso, realizando testes para cada uma das bibliotecas selecionadas para análise e auxiliando o leitor a escolher o componente que melhor se destina às suas necessidades.

Foi possível perceber através dos testes efetuados que as bibliotecas selecionadas nessa análise possuem características próprias e atendem à diferentes exigências de projeto. Para os testes de número máximo de pontos por plotagem e tempo de renderização, foi possível perceber que a biblioteca mais indicada é o dygraphs (plotando uma grande quantidade de pontos sem perda de desempenho). Caso o projeto possua restrições de tamanho indica-se bibliotecas que sejam menores: como o Chart.js ou o jQuery Sparklines. Se uma das demandas do projeto for a utilização de vários tipos de gráficos, recomenda-se o jqPlot (que possui 11 tipos de gráficos para utilização). Sistemas que necessitem de gráficos com um aspecto mais profissional e que tenham uma alta interatividade com o usuário recomenda-se o uso do NVD3 ou do dc.js (porém, como essas duas bibliotecas utilizam a tag *SVG*, não recomenda-se o seu uso para plotagens com um grande número de pontos). Para projetos que carecem de tempo de implementação recomenda-se bibliotecas cujo tempo de aprendizado foi menor: como o Chart.js, o jQuery Sparklines, o Flot ou o dygraphs.

Pode-se concluir que cada componente possui seus pontos fortes e seus pontos fracos, dependendo da necessidade de cada projeto. Este trabalho procurou realizar os testes mais significativos para que fosse possível fornecer uma visão mais ampla das API's selecionadas para análise para uma possível agregação futura ao projeto do leitor.

REFERÊNCIAS

[Apache License V 2.0 2004]APACHE License V 2.0. 2004. Available from Internet: <http://www.apache.org/licenses/LICENSE-2.0.html>.

[Browser Statistics 2013]BROWSER Statistics. 2013. Available from Internet: <http://www.w3schools.com/browsers/browsers_stats.asp>.

[BSD 2005]BSD. 2005. Available from Internet: <http://www.linfo.org/bsdlicense.html>.

[Bullet Chart - Fusion Charts 2013]BULLET Chart - Fusion Charts. 2013. Available from Internet: <http://www.fusioncharts.com/demos/gallery/#bullet-graphs>.

[Chapman 2010]CHAPMAN, C. *A Short Guide To Open-Source And Similar Licenses*. 2010. Available from Internet: <http://www.smashingmagazine.com/2010/03/24/a-short-guide-to-open-source-and-similar-licenses/>.

[Charts 2013]CHARTS, F. 2013. Available from Internet: <http://www.fusioncharts.com/demos/gallery/#waterfall-chart>.

[Crossfilter 2012]CROSSFILTER. 2012. Available from Internet: <square.github.io/crossfilter/>.

[D3.js 2012]D3.JS. 2012. Available from Internet: <d3js.org>.

[Frakes e Fox 1995]FRAKES, W.; FOX, C. Sixteen questions about software reuse. 1995.

[GNU 2013]GNU. 2013. Available from Internet: <http://creativecommons.org/>.

[GPL 2013]GPL. 2013. Available from Internet: <http://www.gnu.org/copyleft/gpl.html>.

[HTML5 2013]HTML5. 2013. Available from Internet: <www.w3.org/TR/html5>.

[IBGE 2010]IBGE. *Distribuição da população por sexo, segundo grupos de idade no Brasil*. 2010. Available from Internet: <http://censo2010.ibge.gov.br/sinopse/webservice/frm_piramide.php>.

[JavaScript 2013]JAVASCRIPT. 2013. Available from Internet: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

[jQuery 2013]JQUERY. 2013. Available from Internet: <http://api.jquery.com/category/events/mouse-events/>.

[jQuery 2013]JQUERY. 2013. Available from Internet: <http://jquery.com/>.

[JSON 2013]JSON. 2013. Available from Internet: <http://json.org/>.

[Martins 2011]MARTINS, M. E. G. p. 111–112, 2011. Available from Internet: <http://area.dgidc.min-edu.pt/materiais_NPMEB/matematicaOTD_Final.pdf>.

[MIT 2013]MIT. 2013. Available from Internet: <http://opensource.org/licenses/mit-license.php>.

[MySQL 2013]MYSQL. 2013. Available from Internet: <http://www.mysql.com/>.

[NVD3 2013]NVD3. 2013. Available from Internet: <http://nvd3.org/>.

[Open Source 2013]OPEN Source. 2013. Available from Internet: <http://opensource.org/>.

[PHP 2013]PHP. 2013. Available from Internet: <http://php.net/>.

[RNP 2013]RNP. 2013. Available from Internet: <http://www.rnp.br/>.

[Screen Resolution Survey 2013]SCREEN Resolution Survey. 2013. Available from Internet: <www.w3schools.com/browsers/browsers_display.asp>.

[Sucan 2010]SUCAN, M. *SVG or Canvas*. 2010. Available from Internet: <http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/>.

[w3schools 2013]W3SCHOOLS. 2013. Available from Internet: <www.w3schools.com>.

ANEXO A LISTA DE BIBLIOTECAS

Na tabela A.1 é apresentada cada biblioteca com a seu respectivo endereço eletrônico para acesso.

Biblioteca	Site
Highcharts	`http://www.highcharts.com`
JScharts	`http://www.jscharts.com`
Protovis	`http://mbostock.github.io/protovis`
PlotKit	`http://www.liquidx.net/plotkit`
MilkChart	`http://mootools.net/forge/p/milkchart`
jqPlot	`http://www.jqplot.com`
JavaScript InfoVis Toolkit	`http://philogb.github.io/jit`
GoogleCharts	`https://developers.google.com/chart`
FusionCharts	`http://www.fusioncharts.com`
Flot	`http://www.flotcharts.org`
dygraphs	`http://dygraphs.com`
jQuery Sparklines	`http://omnipotent.net/jquery.sparkline`
moochart	`http://moochart.coneri.se`
TufteGraph	`http://xaviershay.github.io/tufte-graph`
jQuery Visualize	`https://github.com/filamentgroup/jQuery-Visualize`
NVD3	`http://nvd3.org`
dc.js	`http://nickqizhu.github.io/dc.js`
Chart.js	`http://www.chartjs.org`

Tabela A.1: Tabela apresentando bibliotecas e seus respectivos sites

ANEXO B ESCOLHA DE BIBLIOTECAS

Baseado nas métricas avaliadas neste trabalho é possível prever qual biblioteca utilizar em alguns casos particulares. Os casos foram criados pelo autor e refletem, após análise de casos, as combinações mais relevantes após investigação. Os resultados são apresentados na tabela B.1.

Restrições de projeto	Bibliotecas recomendadas
Tempo de implementação restrito e a biblioteca deve possuir uma documentação oficial	jQuery Sparklines, Chart.js, Flot ou dygraphs
Tempo baixo de renderização, grande quantidade de pontos e suporte à interatividade com mouse	dygraphs, Flot ou jQuery Sparklines
Tempo baixo de renderização, grande quantidade de pontos, renderiza mais de um tipo de gráfico e possui suporte à interatividade com mouse	Flot ou jQuery Sparklines
Tamanho de projeto restrito e suporte à interatividade com mouse	dygraphs, Flot, jQuery Sparklines, dc.js ou jqPlot
Bibliotecas com suporte à todos os eventos de mouse e suportam pelo menos gráficos de linhas, barras e pizza	NVD3 ou dc.js
Biblioteca possui pelo menos gráficos de linhas, barras e pizza, tempo baixo de renderização e suporta interatividade com mouse	jQuery Sparklines

Tabela B.1: Tabela apresentando restrições de projeto e bibliotecas recomendadas