webGraphEd, an open source Graph Drawing Editor for the web

Josep Ciberta
Universitat Oberta de Catalunya
jciberta@xtec.cat

Robert Clarisó Viladrosa
Universitat Oberta de Catalunya
rclariso@uoc.edu

David Bañeres Besora
Universitat Oberta de Catalunya
dbaneres@uoc.edu

June 12, 2014

Abstract

webGraphEd is an open source software for graph visualization and manipulation. It is especially designed to work for the web platform through a web browser. The web application has been written in JavaScript and compacted later, which makes it a very lightweight software. There is no need of additional software, and the only requirement is to have an HTML 5 compliant browser. webGraphEd works with scalable vector graphics (SVG), which it makes possible to create lossless graph drawings. These graphs can be done with several predefined layouts: radial, trees and force-directed.

Keywords. Visualization tool, graph drawing, web editor, automatic layout, open source.

1 Introduction

Nowadays, the information treatment has reached a considerable large amount of data that another approach rather than the traditional one must be used. Even more if complex structures are involved. The need of modelling information and simplicity for extracting patterns, has led to the use of *graph drawings*. Thus, readability and expressiveness of this data is considerably increased.

Graphs are mathematical abstractions that represent entities and the relationships between them. It is formed by nodes and links. Nodes represent different elements, whereas links represent relations between elements. Typically, graph drawings are depicted by a set of dots (representing vertices) and solid lines (representing edges). Graph theory is a subject of study in discrete mathematics.

Layout consists in calculating the position of different objects in a space according to various constraints. In graph drawing, it is the process of calculating the position of the nodes and how the links between them will be drawn. However, when designing a graph, a variety of aesthetic criteria are also taken into account (readability, planarity, cross minimization, space...).

Graph drawings are largely used as a form of representation in a wide range of fields. Its utility lies in its simplicity showing a knowledge that, otherwise, would be difficult to illustrate. As the use of graph drawings becomes more popular, their size also grows. Thus, many different methods have been developed that can draw graphs automatically, most of them fully customizable, even aesthetically. Along with this development, several programming libraries have emerged containing implementations of these algorithms. Furthermore, they also incorporate other techniques in order to explore graphs, make searches...

Application fields in graph visualization have been spread out as well, increasing the number of areas progressively. Moreover, the emergence of Internet's web applications has caused in this field a great interest for scientific community.

The objective of this paper is to present an open source software for graph visualization and manipulation called *webGraphEd*. In Section 2, a review of the main layouts and graph drawing types is depicted. The state of the art of the main web libraries and applications libraries and applications is presented in section 3. In Section 4 the new web tool and its features are described. The architecture is described in section 5 and the implementation in section 6. Section 7 shows the results of the conducted research. Finally, section 8 gives the conclusions and future work.

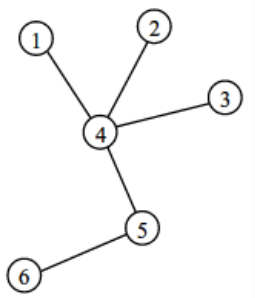

Figure 1: Tree[1]

2 Main Layouts and Graph Drawing Types

Several different graph drawing designs have appeared in order to satisfy the growing demand. These new designs require, sometimes, new layout methods. Several classifications have been done in this area [1] [2]. Most of these layout types are related to some particular algorithm, especially those that produce automatic layouts. In this section, a brief summary of the main layout types is presented (table 1).

Type	Subtype
Tree layout	H-Tree
	HV-Tree
Radial layout	Radial view
	Balloon view
Tree map	
Hierarchical	
Orthogonal	
Force-directed	
3D layout	Radial view
	Cone tree
Hyperbolic Layout	

Table 1: Main graph layout types

One of the most common graph drawings are trees consisting in a node (called root node) which is connected to other nodes, and these others are connected to other new nodes and so successively without creating cycles, that is a graph where any vertex is connected to anyone else by one simple path. Although not always the representation is as expected (figure 1), it matches with the definition: simple undirected graphs that are connected and has no cycles. In these cases, the layout is critical to appreciate several features (root node, children...). Some of the layout techniques are described in [3]. A binary tree (figure 2) is a special case of tree in which each node has at most two child nodes (denoted as left and right children).

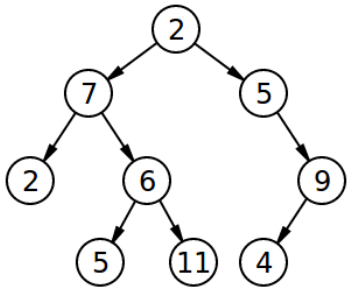

Figure 2: Binary tree[2]

Layout techniques are described by Wetherell and Shannon [4], also by Reingold and Tilford [5]. Successive refinements have been made by many other authors. Even, new representation forms have brought

[1] http://upload.wikimedia.org/wikipedia/commons/2/24/Tree_graph.svg

[2] http://upload.wikimedia.org/wikipedia/commons/f/f7/Binary_tree.svg

out: H-Tree [6] (figure 3) and HV-Tree layout [7] (figure 4).

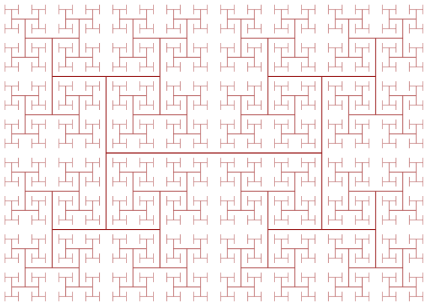

Figure 3: H-Tree layout[3]

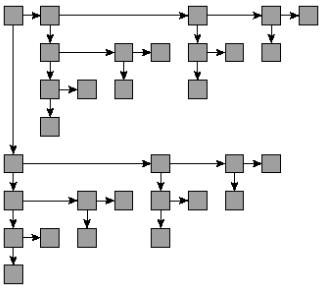

Figure 4: HV-Tree layout[4]

Radial layouts are another kind of graph layout type. They present different variations. The most common is the radial view [8] shown in figure 5, where from a centred root, branches spread out successively in several circles until they reach the leaves. These circles indicate the different depth levels. The balloon view [9] (figure 6), although it is similar to the previous one, depicts one important difference: every new subtree becomes the centre of a new radial graph.

Tree maps (figure 7) are a kind of layout where each node is presented as a nested box inside another

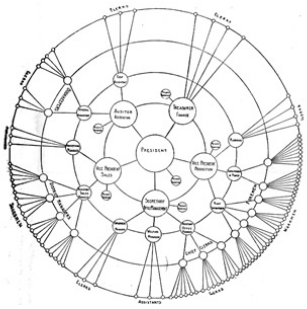

Figure 5: Radial view layout[5]

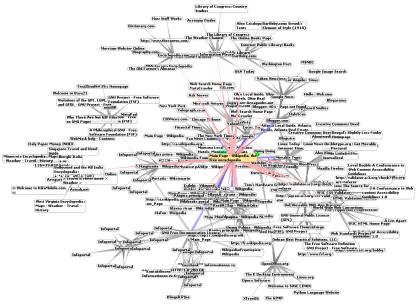

Figure 6: Balloon layout[6]

one depending on its level. The area of the outlined rectangle is significant.

Hierarchical layouts (figure 8) are directed graphs where nodes are distributed along different layers forming horizontal levels. One of the first techniques was proposed by Sugiyama, Tagawa and Toda [10]. Like other cases, subsequent refinements have been done.

In orthogonal layouts (figure 9) every edge is aligned to the axis. These kind of graphs try to achieve the maximum planarity, minimizing the number of crossings and bends [11].

Force-directed layouts (figure 10) can be viewed as a system of nodes with forces interacting between them. The algorithm looks for such a configuration of forces in order to balance the whole system [7].

[3] http://upload.wikimedia.org/wikipedia/commons/a/af/H_tree.svg

[4] Extracted from the GOBLIN graph library (http://sourceforge.net/projects/goblin2/files/goblin2/goblin.2.8b30)

[5] http://upload.wikimedia.org/wikipedia/commons/d/d7/Radial_tree_-_Graphic_Statistics_in_Management.jpg

[6] http://upload.wikimedia.org/wikipedia/commons/b/b9/WorldWideWebAroundWikipedia.png

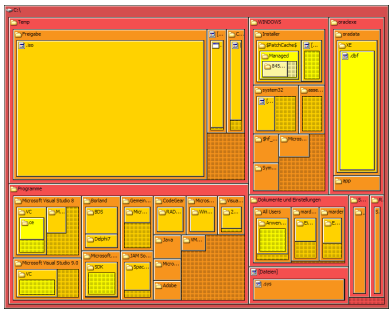

Figure 7: Tree map layout[7]

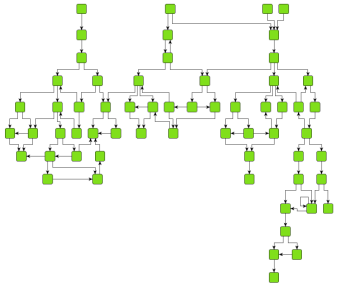

Figure 8: Hierarchical layout[8]

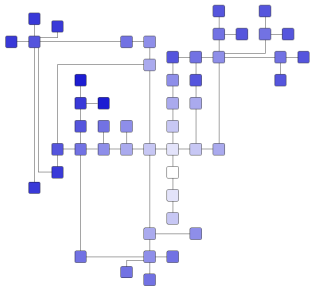

Figure 9: Orthogonal layout[9]

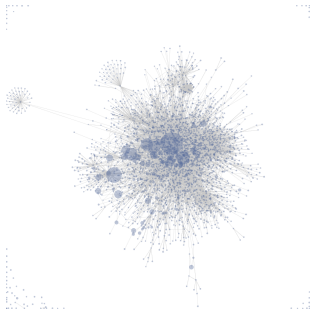

Figure 10: Force-directed layout[10]

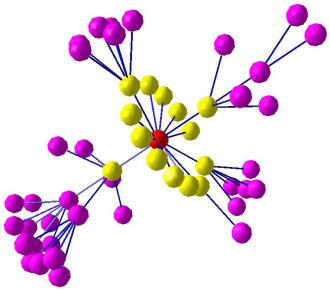

Figure 11: 3D radial layout[11]

In 3D layouts an extra dimension is added. Therefore, there is more space to represent large graphs, increasing scalability. On the other hand, new depicting problems appears as some objects could occlude others. Changing properties like transparency or interactive methods like modifying perspective are some of the techniques to work out this problem. The simplest layout is the 3D version of the radial layout (figure 11). Another is the cone tree [12] (figure 12), but in this case some kind of interaction is necessary to show the hidden elements.

Finally, hyperbolic layouts [13] [14] (figure 13) are a graph layout which is closely related to its interaction. The geometry used here is radically different. First, the layout is realized in the hyperbolic space and then is visualized in the Euclidean space.

[7] http://upload.wikimedia.org/wikipedia/commons/8/8b/Tree_Map.png

[8] Figure extracted from yFiles Online Demos (http://www.yworks.com/en/products_yfiles_practicalinfo_demos.html)

[9] Figure extracted from yFiles Online Demos (http://www.yworks.com/en/products_yfiles_practicalinfo_demos.html)

[11] Extracted from the GEOMI documentation

[10] http://upload.wikimedia.org/wikipedia/commons/9/90/Visualization_of_wiki_structure_using_prefuse_visualization_package.png

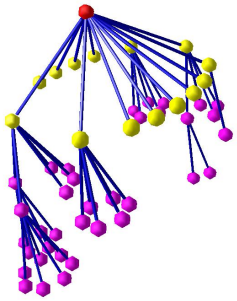

Figure 12: Cone tree layout[12]

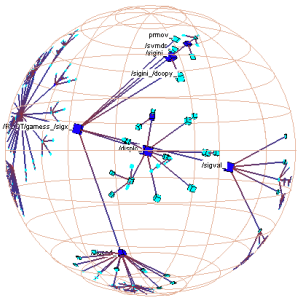

Figure 13: Hyperbolic view of a tree in 3D[13]

3 State of the art

3.1 Preliminaries

The DOM (Document Object Model) [15, 16, 17] is a multi-platform and language-independent interface which allows to access HTML and XHTML documents by its representation as nested objects. These objects are organized in a tree where the document is the root, and all the elements of the document can be accessed through objects' methods.

With the fifth version of HTML language, two new elements came up: the *canvas* and the *svg* element. Both can be embedded in HTML pages in order to enhance websites.

The canvas [18] element allows to create and manipulate 2D and 3D (WebGL [19]) images at the pixel level. However, it is not possible to browse within this elements with DOM objects.

SVG [20, 21] stands for Scalable Vector Graphics and it is used to define vector-based graphics. All SVG elements are embedded as DOM elements and therefore, can be browsed through the HTML document.

With the increase of the web usage in many areas, there have also appeared libraries for visualizing and interacting with graph drawings. But the biggest increase in libraries and web applications happened after the development of version 5 of HTML, with these new elements described before. Most of these libraries rely on the SVG element due to its advantages in contrast with the canvas element such a lossless zoom in and out, and the capacity of being embedded into the document object model.

The state of the art of these libraries and applications will be described, including a review of some relevant desktop applications with the most remarkable features. Finally, a summary of these features will be depicted in several tables. Only tools that are able to perform layouts have been take into account.

3.2 Desktop libraries and applications

This kind of software has evolved from single graph libraries, where just a few types of simple diagrams were supported, to complex libraries allowing the creation of automated graph drawing layouts. The computational model used within the libraries has also evolved from small pieces of software running in specific platforms to frameworks or environments able to run in current desktop clients or even on the cloud. A review of the most popular desktop applications is next presented.

Graphviz [22] [23] is a collection of tools for the visualization and manipulation of abstract graphs. Different types of graph layouts through a C library, command line utilities, GUI tools and web browser tools are implemented. It is composed by two main libraries: Libgraph and Dinagraph. Libgraph focuses on static layouts while Dinagraph deals with incremental and interactive layouts. A variant of the Sugiyama algorithm to make it more aesthetic is used for layered layouts. Two models are used for sym-

[12] Extracted from the GEOMI documentation

[13] Figure generated through h3.exe from `http://graphics.stanford.edu/~munzner/h3/h3.exe`

metrical layouts: the Kamda-Kawai spring layout algorithm and a Fruchterman-Reingold extension. An Eades-based algorithm implementation is used for radial layouts. There are two layout tools [24]: Dot (successor of DAG) for directed graphs drawings and Neato for non-directed ones. The main goal is to make good diagrams with reasonable size and able to scale properly. Graphviz is used in software engineering, bioinformatics applications, web applications, among others. Most of the implementation is written in C, but there are also implementations written in C++ and Java. The software is released under the Eclipse Public License (EPL).

Tom Sawyer Software [25] is a commercial product for producing layouts and visualization of graph drawings used by many companies, software vendors and educational institutions. The product is divided into two tools: Tom Sawyer Layout (formerly Graph Layout Toolkit) and Tom Sawyer Visualization (formerly Graph Editor Toolkit). Layouts supported in Tom Sawyer Layout are circular, hierarchical, orthogonal and symmetric. The product is available for several desktop platforms (Windows, Linux, Apple, Solaris) with C++, Java, ActiveX and .NET editions.

JViews [26] is a commercial component suite in Java made by the ILOG company (absorbed by IBM, in January 2009). The framework is easily integrated into third party applications. One of the most important components is the graph drawing layout module. It has a great variety of layouts: mesh, spring, force-directed, tree, radial, hierarchical, bus, circular, grid, orthogonal and nested.

yFiles [27] is another graph drawing visualization software that performs automatic layouts. It is designed as a Java library with a commercial license. One of the priorities is to keep a flexible and extensible architecture. The library consists of three components that cooperate mutually: the yFiles Basic Component containing the classes, data structures and algorithms, a viewer/editor component that contains an intuitive and easy to use graph drawing editor (yEd), and the yFiles Component Layout containing a multitude of layouts. The software uses several layout types: circular-radial layout; orthogonal layout, following the Tamassia approach; force-directed layout, layered layout following the Sugiyama approach with different alternatives (this technique is also used by GraphViz); and tree layout, a variant of the Reingold-Tilford approach. The graph drawing supported formats are ygf (own), GraphML and GML, and also drawing can also be exported to SVG and PDF. There is also a version for the web (described further).

Cytoscape [28] [29] is a general purpose environment for graph visualization which was originally designed for working on biomolecular networks. Cytoscape components provide the basic functionality for integrating arbitrary data, representing it visually, selecting and filtering it, and even link it with other software through an interface implemented as plug-ins. The data in the graph drawing is integrated through attributes and annotations representing a hierarchical classification. Regarding the layout, it supports a variety of automatic layouts (spring, hierarchical, circular...). The software is written in Java and the license is LGPL although some algorithms use the commercial yFiles graphics library. The software also facilitates the use of plug-ins, in order to implement new algorithms or carry out new graph analysis among other things. There is no restriction when adding plug-ins, they may have different licenses, even commercial. It has a powerful API, which allows to create additional features that they can be distributed through its own store, the Cytoscape App Store.

Prefuse [30] is a framework for creating dynamic visualizations through structured or unstructured data offered as an extensible toolkit. It allows abstractions to filter data before being displayed and then performs several actions on the display (transformations, layouts, colors...). It also implements a great variety of layouts: tree-maps, cone trees, perspective walls, starfield displays, hyperbolic trees, DOI (Degree Of Interest) trees, space trees, force-directed, radial, among others. Prefuse is written in Java using Java2D graphics library and it is distributed under open source license.

JGraphX [31] is a library written in Java for graph drawing visualization. It has been create primarily with the Java Swing components. This library has an open source license, the 3-clause BSD license. However, the web library version, which is called mx-

Graph and described further, is released under a commercial license.

Goblin [32] is an application written in C++ and Tcl/Tk for displaying and manipulating graph drawings. It has a graphical interface, an API and Tcl/Tk programming interface. Supported layouts are: tree, orthogonal, hierarchical and force-directed, besides different graph optimizations for the most known techniques. This software is distributed under GNU GPL.

H3Viewer [33] is a graph library that supports interactive navigation in graph drawings up to a few hundred of thousand edges through an adaptive algorithm that ensures a specific frame rate. The layout and navigation occur in a hyperbolic space which easily allows changing the point of interest. H3Viewer motivation's is that other libraries do not scale properly in a large number of nodes and edges, especially in automatic layouts. This library achieves an interactive navigation in large graph drawings through the use of a spanning tree, which fits properly with trees and directed acyclic graphs. The layout technique depends, not on the total number of nodes and edges, but on the visible ones. The algorithm implementation is a loop where it starts drawing from the centre of the sphere heading around until time runs out, in order to ensure a specific frame rate. The library is available for SGI, Linux and Windows released under the SGI licence (free for non commercial use).

Geomi [34, 35] is a software for the analysis and visualization of graph drawings, specialized in 3D layouts. It is composed of three main components: network analysis, layout engine and a component for interactions. Over this layer, a multitude of plug-ins can be integrated. Three-dimensional supported layouts are: hierarchical, radial, cone-tree and clustered. The software is written in Java and released under the LGPL license.

3.3 Web libraries

Protovis [36] is a library for graph visualization on the web. It uses JavaScript and the SVG element to display those graphs, which means that it does not need any special plug-in (just a modern browser). The library is open source and is released under the MIT license. In 2011, its development stopped in favour of the D^3 library.

D^3 (Data-Driven Documents) [37, 38] is the spiritual successor of Protovis. It has been written in JavaScript, where data manipulation is done through selections (a widely used technique used in other JavaScript libraries that consists on select elements of a web page in order to modify, append or remove these items like applying operations on a data set). Data is specified as arrays of arbitrary data. The library supports event handlers, which it makes possible to create animated transitions, interpolating the current value to a specified value gradually over the time. It also has different modules, some of which provide chart and histograms shapes, layouts for graph drawings (extensible via plug-ins)and interaction techniques (behaviour). Applications made with the library range from real-world applications to teaching examples for new users. Like Protovis library, D^3 is also open source and is released under the MIT license.

Sigma.js [39] is a lightweight library for displaying graphs. It also uses HTML canvas and has been specially designed to view static graphs interactively, whether data is imported or generated on the fly. However, the development is in a very preliminary stage. The library is open source and distributed under the MIT license, and it has been written in JavaScript.

Raphaël [40] is another reduced-size JavaScript library that simplifies vector graphics creation. Each graphical object is an object which can be linked to JavaScript event handlers or modified from code. The library is released under the MIT license.

yFiles for the web is the web version of yFiles [27]. It is available in several technologies: HTML5, Silverlight, Flex and AJAX. There are also client and server versions. Allowed layouts are hierarchical, organic, tree, radial, and orthogonal. It has a commercial license.

Flare [41] is an ActionScript (Flash) library created by the UC Berkeley Visualization Lab. From basic graphs to complex interactive graph drawings can be created with this tool. It supports data management, animation and several interaction techniques. Some of the available layouts are tree, force-directed,

indent, radial, circle, dendrograms, bubbles, circle pack, icicle and sunburst. The library is distributed under the BSD license.

Cytoscape.js [42] is the successor of Cytoscape Web. Both libraries are the web version of Cytoscape. In the case of Cytoscape Web had two versions, one for JavaScript and another for Flex. However, the new library keeps only the JavaScript version. Although Cytoscape is an entire desktop platform (application, libraries, and plugins), Cytoscape.js is just a library. According to its website: *Though Cytoscape.js shares its name with Cytoscape, Cytoscape.js is not exactly the same as Cytoscape desktop. Cytoscape.js is a JavaScript library for programmers. It is not an app for end-users, and developers need to write code around Cytoscape.js to build graphcentric webapps*[14]. This library is distributed under GNU LGPL version 3.

According to its website *mxGraph* [43] *is the only fully client-side JavaScript Graph Visualization and Layout Solution*. Available layouts are tree, organic and circle. The library can be used in conjunction with Java, .NET or PHP. The mxGraph library is licensed under a standard commercial license.

Table 2 depicts a review of the most relevant features. It has been taken into account the minimal size for web libraries, since most of them have been reduced or compressed. This is an critical feature because it increases the performance when loading a web page.

3.4 Web applications

Tom Sawyer [25] is one of the oldest graph visualization tools. It has two separated products for graph drawing visualization: Tom Sawyer Visualization and Tom Sawyer Layout. Through the Java platform and working on several servers it allows the execution of their software on the web as JSP pages. Moreover, it is also possible to run the software through the .NET platform, creating ASP.NET pages. Tom Sawyer software is distributed under a commercial license.

[14]Cytoscape.js & Cytoscape section. `http://cytoscape.github.io/cytoscape.js`

Graphity [44] is a free graph editor made with the yFiles for Flex libraries. Allowed layouts are hierarchical, organic, tree, radial, and orthogonal.

Draw.io [45] is a web application realized with the commercial mxGraph library developed by the company JGraph. It has a Google Apps look and feel and it can work directly with some cloud services (Google Drive and Dropbox). Available layouts are tree, organic and circle. Although the mxGraph library is licensed under a standard commercial license, Draw.io can be used freely.

DAGitty [46, 47] is a tool for drawing and analysing casual directed acyclic graphs (DAG). It is a specific purpose application and does not have many features. There is only one layout available, and it is not well defined. DAGitty is available under the GNU GPL license.

On the other hand, there is also some applications that allow graph and diagram creation like Gliffy [48], LucidChart [49] or Creately [50]; but they are not able to perform any layout.

In table 3 a brief description of the main features is listed, likewise as in previous table, except the size, which is not available.

4 Application

In the previous section, the state of the art of libraries and tools for graph drawing visualization and manipulation have been described. The development of the HTML version 5 has caused the emergence of many web libraries, including some of them especially designed for graph drawing visualization and manipulation. Among these last ones, there are also an increase of the open source libraries.

However, in the case of web applications, this is a very virgin field. Although some commercial applications are found, which they usually have a free access to use them, there is a complete lack of general purpose open source applications. *webGraphEd* attempts to fill this gap. Another important point is the absence of use of open file formats, especially for graph drawings. Most of this applications have their own file format, which in most of the cases, it is unknown to the end user. This is a considerably con-

Library	Version	Size	Year	Language/Environment	License
Cytoscape.js	2.0.4	216 Kb	2013	Flex, JavaScript	Open Source. LGPL
D3.js	3.3.11	146 Kb	2013	JavaScript	Open Source. BSD
Flare	2009.01.24	1.2 Mb	2009	ActionScript	Open Source. BSD
mxGraph	2.8.1.0	N/A	2014	JavaScript	Commercial
Protovis	3.3.1	1.3 Mb	2011	JavaScript	Open Source. BSD
Raphaël	2.1.2	91 Kb	2012	JavaScript	Open Source. MIT
sigma.js	1.0.2	109 Kb	2014	JavaScript	Open Source. MIT
yFiles	1.1/2.3/1.8/2.1	N/A	2014	HTML5, Silverlight, Flex, AJAX	Commercial

Table 2: Web libraries summary

Application	Version	Year	Language/Environment	License
Tom Sawyer	9.2	2011	ASP.Net, JSP	Commercial
Graphity	2.10	2012	Flex	Commercial
Draw.io	N/A	2014	JavaScript	Commercial
DAGitty	2.1	2014	JavaScript	Open Source. GPL

Table 3: Web applications summary

straint to the interoperability between applications.

The aim of the application is to be able to create graphs drawings, lay out and manipulate them, with a friendly interface. All of these in a open source lightweight application (currently, less than 650 KB) with the capacity to be embedded into a web page. webGraphEd is designed to be intuitive and easy to use for end user, containing basic editing and visualizing functionalities. In addition to these basic features, some others are implemented as well like layout detection and several graph formats support.

The chosen license has been the GNU GPL version 3. A copyleft license is a must and version 3 is the most suitable because it is compatible with BSD license (3-clause), Apache License 2.0 and MIT license, in order to use properly the required libraries.

4.1 Features

The most basic features for graph drawing visualization and manipulation have been implemented in webGraphEd. A brief description of all these features is shown below.

- Detection of graph drawing type [7] (connected and disconnected graphs, cyclic and acyclic graphs, trees and binary trees).
- An extensible variety of layouts. Currently, horizontal tree, vertical tree, radial tree and force-directed layouts are available.
- Ability to draw a graph with a fixed layout algorithm, depending on its type. For instance, for a cyclic graph drawing or an acyclic non-tree, only force-directed layout is available since orthogonal and hierarchical layouts are not implemented yet, but for a binary tree, all kind of tree layouts are also available.
- The application recognizes the main graph formats and it is capable of importing and exporting them considering the most relevant features of these formats. The supported graph formats are GML (Graph Modelling Language) [51] and GraphML [52], although the totality of the features of those notations are not implemented.
- The basic actions are:
  - Pan and zoom.
  - Drag and drop nodes as well as links.
  - Select and unselect nodes and links.
  - Add and remove nodes and links.
  - Change basic properties of nodes (shape and color) and links (thickness and color).
  - Center layout when the nodes are moved freely, except for force-directed layout, which is always centred due to its own gravity.

- The advanced actions are:
  - Change layout algorithm on the fly.
  - Collapse and uncollapse nodes.
  - Fit layout to the available screen, in order to give more space and readability to the graph drawing.

The application can also be embedded into a web page. The graph being drawn can be referred in two different ways: embedding into the URL or indicating the URL where the file is located.

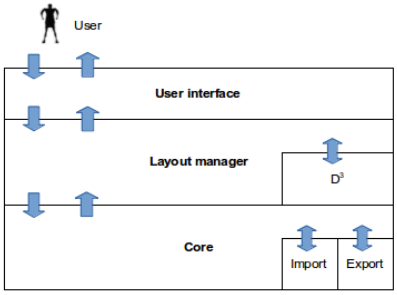

Figure 14: webGraphEd's architecture

5 Architecture

The developed application have been designed in 3 layers (figure 14): core, layout manager and user interface. The goal is to abstract the full functionality encapsulating them into different layers, allowing a different approach for a specific layer if it is necessary.

The *core layer* implements the kernel of the application, containing the most basic functions, internal graph drawing and data structures, the import and export modules, and several utilities related to maths calculations, DOM and JSON.

Above it is the *layout manager*, which links to the graphics libraries (in this case D^3) through a well defined interface, in order not to rely exclusively on a single library. It chooses the most suitable layout for every graph drawing.

Finally, the user *interface layer* is the top layer. It is in charge of the aspect of visualization, including the GUI, manipulation and interaction tools, properties and behaviour.

6 Implementation

webGraphEd is fully implemented in JavaScript [53, 54], which means that the application is executed on the client. Among the different client-side languages, like Java applets or ActionScript, JavaScript is the most widely used, most compliant with browsers and the code is interpreted, so the code can be browsed. This does not mean that JavaScript applications are open source, but they are close.

Although in JavaScript there are no classes (but objects), an OOP [55] approach has been used. Thus, different functionalities from distinct layers have been encapsulated properly. A Test Driven Development [56] methodology have also been used, and a large list of unitary tests have been written.

Several libraries are used, which all of them are open source with their respective licenses and they are further described. The documentation of the whole application has been done with JsDoc [57], which provides a comprehensible HTML help for developers. With the aim to reduce the code size, and therefore increase the application's performance, compression tools have also been used. JavaScript code is reduced with the Closure Compiler [58], which optimizes and verifies the code as well. The cascade style sheets (CSS) have been compressed with the YUI Compressor [59].

A local copy of the libraries is used in order to prevent side effects, such a server downfall, location changes, or even behavioural changes (new versions, deprecated functionalities...)

6.1 External libraries

Several external libraries have been used in order to implement the different layers. These libraries are next described and every one is employed in a specific layer.

At the *core layer* two libraries have been used. The jQuery library [60] is a general purpose library. It allows to keep your application independent from

Library	Version	License
D^3	3.4.6	BSD
Closure	N/A	Apache 2.0
QUnit	1.12.0	MIT
JQuery	2.1.0	MIT
JQuery Impromptu	5.2.3	MIT

Table 4: Library licences

browsers among other features, such as a powerful API or an easier manipulation of HTML elements. The library used for unit testing has been the QUnit library [61], from the same authors of jQuery. Library's choice has been made by its simplicity. The low-level development of the application has been closely linked to the unit tests. QUnit is distributed under the same jQuery license, the MIT license.

The application's main library is found at the *layout manager*, the graphical library D^3.js [37, 38], which is aforementioned in section 3.

A widely used library in the *interface layer* is the Closure Library [62], developed and maintained by Google. This library is part of the Closure Tools, containing, in addition to the library, a JavaScript optimizer and other related utilities. All graphics such as menus and dialogs are performed with this library. It is distributed under the Apache 2.0 license. An additional library used at this level is the jQuery Impromptu [63]. Due to some limitations of the Closure library, jQuery Impromptu has been used for a richer dialogs, which requires some outputs to be different from the standard ones.

A brief summary of the used libraries in the application is shown in table 4.

7 Results

In this section, qualitative and quantitative data have been analysed. Several layouts, distribution and spatial organization, layout readability and aesthetical parameters like colour and shape have been analysed as a qualitative data. Quantitative data has been focused basically on compatibility and response times in order to assess the application performance.

7.1 Quantitative data

First of all, compatibility among browsers have been verified. The most common actions have been checked for the most popular browsers. Although there are some cross-platform browsers, tests have been done in a Windows 7 environment, since the most browsers have an implementation for this operating system. The latest versions of these browsers have been used:

- Google Chrome 35.0
- Mozilla Firefox 29.0
- Internet Explorer 11
- Safari 5.1.7
- Opera 22.0

In table 5, the compatibility for the most common actions is shown for several browsers. Safari does not support local file handling, although the latest version is from 2012. Moreover, Internet Explorer lets drag parts of the graph outside the SVG element. Even the force-directed layout is partially drawn out of the bounds when the simulation is still going on.

Another qualitative data analysed has been the time, especially load times and execution times for a 30 nodes graph drawing. Table 6 depicts these times. Data has been retrieved from a Dell Inspiron 15 (Intel Pentium T4500 2.30 GHz) running a Windows 7 operating system.

Application load time is a critical point. It relies on two main parameters, the server where the web application is allocated (response time, bandwidth, ...)and the optimization of the application. *webGraphEd* has been optimized and compacted with the Closure Compiler, and thus, the load time performance has been improved.

Nevertheless, some dramatic differences are observed such as the excessive time for loading a file in Internet Explorer. Regarding the layout execution, the Safari browser is slightly faster than Chrome and Opera (these two do not have many differences, as they are based on the same web browser engine, Blink). On average, the less favourable times are for Internet Explorer.

Scalability of the web application have also been tested, through a comparative of the execution time

Action	Chrome	Firefox	Explorer	Safari	Opera
Graph drawing					
Pan & zoom	yes	yes	yes	yes	yes
Add node	yes	yes	yes	yes	yes
Drag & drop	yes	yes	yes	yes	yes
Hide node	yes	yes	yes	yes	yes
Link node	yes	yes	yes	yes	yes
Node's properties	yes	yes	yes	yes	yes
Others					
Open file	yes	yes	yes	no	yes
Save file	yes	yes	yes	no	yes
Dialog	yes	yes	yes	yes	yes

Table 5: Browser compatibility

Action	Chrome	Firefox	Explorer	Safari	Opera
Load times					
Application load time	3.3	4.0	3.9	4.3	3.9
File load time	0.012	0.015	0.072	N/A	0.029
Execution times					
Horizontal tree	0.03-0.04	0.05-0.06	0.05-0.08	0.02-0.03	0.03-0.05
Vertical tree	0.03-0.04	0.05-0.06	0.05-0.08	0.02-0.03	0.03-0.05
Radial tree	0.03-0.04	0.07-0.08	0.05-0.08	0.02-0.03	0.03-0.05
Force-directed	0.01-0.03	0.02-0.03	0.02-0.04	0.01-0.02	0.01-0.03

Table 6: Load and execution times

of different layouts depending on the number of nodes. Table 7 shows these results of this comparative, where the values are in seconds. The force-directed layout has better execution times because it is executed in two steps: first, it locates all the nodes in a not well defined position; and then the simulation of the forces begins, which takes longer times. In the case of 5000 nodes, the simulation never stops, because it enters in an infinite iteration. However, the time of the simulation depends on several factors that can be customized, such as gravity, distance between nodes, force and charge [64].

It is also observed that the progression of layout execution time is linear, proportionally to the number of nodes. Figure 15 outlines the temporal relationship with regard to the number of nodes for each layout.

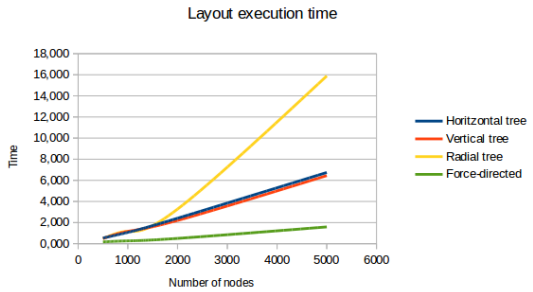

Figure 15: Layout execution time

7.2 Qualitative data

Currently, the application has 4 different implemented layouts, though 3 of them are trees. Nonetheless, a great variety of graph typologies can be illustrated. Figure 16 depicts a vertical layout for 30 nodes while figure 17 shows a force-directed layout with 50 nodes.

Another implemented features that can be customized are colour and shape for nodes, and colour and thickness for links. Figure 18 shows a radial layout with several customized nodes and links.

Readability has been improved with functions like center and fit, which lays out the graph according the screen boundaries.

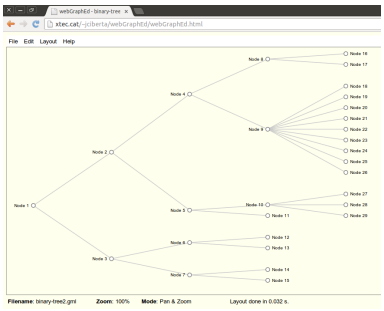

Figure 16: webGraphEd vertical layout

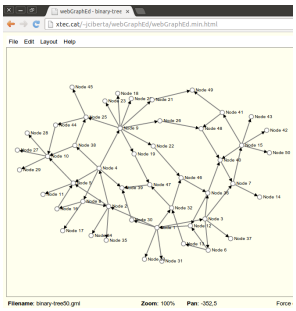

Figure 17: webGraphEd force-directed layout

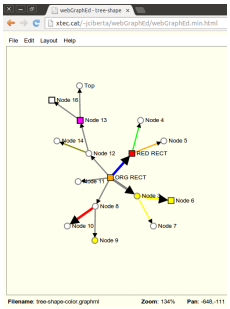

Figure 18: webGraphEd radial layout

8 Conclusions

A new web graph editor has been presented is this paper. One of the most goals has been to create an easy to use application, with a friendly user interface. In this early version, the major relevant features have been highlighted and the available layouts have been described. Several performance test have been car-

Nodes	50	500	1000	2000	5000
File load time	0.02	0.076	0.28	1.067	8.712
Layout execution time					
Horizontal tree	0.053	0.537	1.072	2.413	6.744
Vertical tree	0.047	0.488	1.169	2.204	6.45
Radial tree	0.058	0.473	1.167	3.279	15.887
Force-directed	0.075	0.174	0.268	0.503	1.589
End of simulation	5.047	15.895	29.032	57.37	N/A

Table 7: Scalability of the application

ried out, including scalability, as well.

This is an ongoing project, and new features are foreseen. Available layouts can be increased, even if they are not implemented by the D^3 library, through the own layout engine. Support to more graph formats such as DOT [65], GXL (Graph Exchange Language) [66] and GraphXML [67] would be also convenient. Moreover, new forms of interaction and missing actions, like enlarging nodes, more shapes and customising links are planned.

References

[1] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(1):61–79, 1988.

[2] Ivan Herman, Guy Melançon, and M Scott Marshall. Graph visualization and navigation in information visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24–43, 2000.

[3] John Q. Walker II. Node-positioning algorithm for general trees. *Software - Practice and Experience*, 20(7):685–705, 1990. cited By (since 1996)59.

[4] C. Wetherell and A. Shannon. Tidy drawings of trees. *Software Engineering, IEEE Transactions on*, SE-5(5):514–520, 1979.

[5] Edward M. Reingold and J.S. Tilford. Tidier drawings of trees. *Software Engineering, IEEE Transactions on*, SE-7(2):223–228, 1981.

[6] S. Bhattacharya and W.-T. Tsai. Area efficient binary tree layout. In *VLSI, 1991. Proceedings., First Great Lakes Symposium on*, pages 18–24, 1991.

[7] Ioannis Tollis, Peter Eades, Giuseppe Di Battista, and Loannis Tollis. *Graph drawing: algorithms for the visualization of graphs*, volume 1. Prentice Hall New York, 1998.

[8] Peter Eades. *Drawing free trees*. International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.

[9] Chang-Sung Jeong and A. Pang. Reconfigurable disc trees for visualizing large hierarchical information space. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 19–25, 149, 1998.

[10] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.

[11] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[12] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Information visualization using

3d interactive animation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 461–462, New York, NY, USA, 1991. ACM.

[13] Jonh Lamping and Ramana Rao. The hyperbolic browser: A focus+ context technique for visualizing large hierarchies. *Journal of Visual Languages & Computing*, 7(1):33–55, 1996.

[14] Tamara Munzner and Paul Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *Proceedings of the first symposium on Virtual reality modeling language*, pages 33–38. ACM, 1995.

[15] Lauren Wood, Vidur Apparao, Laurence Cable, Mike Champion, Mark Davis, Joe Kesselman, Tom Pixley, Jonathan Robie, Peter Sharpe, and Chris Wilson. Document object model (dom) level 1 specification. *w3C recommendation*, 1, 1998.

[16] World Wide Web Consortium et al. Document object model (dom) level 2 style specification. 2000.

[17] Gavin Nicol, Lauren Wood, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification. 2001.

[18] World Wide Web Consortium (W3C). HTML Canvas 2D Context. W3C Candidate Recommendation 6 August 2013. `http://www.w3.org/TR/2013/CR-2dcontext-20130806/`.

[19] Chris Marrin. Webgl specification. *Khronos WebGL Working Group*, 2011.

[20] World Wide Web Consortium (W3C). Scalable Vector Graphics 1.1 (Second Edition). W3C Recommendation 16 August 2011. `http://www.w3.org/TR/SVG11/`.

[21] J David Eisenberg. *SVG Essentials: Producing Scalable Vector Graphics with XML*. O'Reilly Media, Inc., 2002.

[22] John Ellson, Emden R Gansner, Eleftherios Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz and dynagraph static and dynamic graph drawing tools. In *Graph drawing software*, Mathematics and Visualization, pages 127–148. Springer, 2004.

[23] Emden R Gansner. Drawing graphs with graphviz. Technical report, Technical report, AT&T Bell Laboratories, Murray, 2009.

[24] Emden R Gansner and Stephen C North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000.

[25] Arne Frick and Brendan Madden. Flexible graph layout and editing for commercial applications. In SueH. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 446–447. Springer Berlin Heidelberg, 1998.

[26] Georg Sander and Adrian Vasiliu. The ilog jviews graph layout module. In *Graph Drawing*, pages 438–439. Springer, 2002.

[27] Roland Wiese, Markus Eiglsperger, and Michael Kaufmann. yfiles visualization and automatic layout of graphs. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 173–191. Springer Berlin Heidelberg, 2004.

[28] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[29] Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2011.

[30] Jeffrey Heer, Stuart K Card, and James A Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM, 2005.

[31] Gaudenz Alder and David Benson. JGraph Ltd. JGraphX library. `http://www.jgraph.com/jgraph.html`.

[32] Christian Fremuth-Paeger. Goblin: A graph object library for network programming problems, 2007. `http://goblin2.sourceforge.net/`.

[33] Tamara Munzner. Drawing large graphs with h3viewer and site manager. In SueH. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 384–393. Springer Berlin Heidelberg, 1998.

[34] Adel Ahmed, Tim Dwyer, Michael Forster, Xiaoyan Fu, Joshua Ho, Seok-Hee Hong, Dirk Koschützki, Colin Murray, Nikola S Nikolov, Ronnie Taib, et al. Geomi: Geometry for maximum insight. In *Graph Drawing*, pages 468–479. Springer, 2006.

[35] Joshua Ho and Seok-Hee Hong. Drawing clustered graphs in three dimensions. In *Graph Drawing*, pages 492–502. Springer, 2006.

[36] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1121–1128, 2009.

[37] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

[38] Mike Dewar. *Getting Started with D^3 - Creating Data-Driven Documents*. O'Reilly, 2012.

[39] Alexis Jacomy and Guillaume Plique. The sigma.js JavaScript library. `http://sigmajs.org/`.

[40] Dmitry Baranovskiy. Raphaël JavaScript library. `http://raphaeljs.com/`.

[41] UC Berkeley Visualization Lab. Flare: Data visualization for the web. `http://flare.prefuse.org/`.

[42] Donnelly Centre. University of Toronto. The Cytoscape.js library. `http://cytoscape.github.io/cytoscape.js/`.

[43] Gaudenz Alder and David Benson. JGraph Ltd. mxGraph library. `http://www.jgraph.com/mxgraph.html`.

[44] yWorks GmbH. Graphity diagram editor. `http://live.yworks.com/graphity/`.

[45] JGraph Ltd. Draw.io. `https://www.draw.io/`.

[46] Johannes Textor, Juliane Hardt, and Sven Knüppel. Dagitty: a graphical tool for analyzing causal diagrams. *Epidemiology*, 22(5):745, 2011.

[47] Johannes Textor and Maciej Liskiewicz. Adjustment criteria in causal diagrams: An algorithmic perspective. *arXiv preprint arXiv:1202.3764*, 2012.

[48] Gliffy: Online diagram software. `http://www.gliffy.com/`.

[49] LucidChart: Flow Chart Maker & Online Diagram Software. `https://www.lucidchart.com/`.

[50] Creately: Online Diagram Software to draw Flowcharts, UML & more. `http://creately.com/`.

[51] Michael Himsolt. GML: A portable graph file format, 1997. Universität Passau. `http://www.fmi.uni-passau.de/graphlet/gml/gml-tr.html`.

[52] Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. *Graph markup language (GraphML)*. Bibliothek der Universität Konstanz, 2010.

[53] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[54] Stoyan Stefanov. *JavaScript patterns*. O'Reilly Media, Inc., 2010.

[55] Peter Coad and Jill Nicola. *Object-Oriented Programming*. Prentice Hall, 1993.

[56] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[57] Nicholas C Zakas. *Maintainable JavaScript*. O'Reilly Media, Inc., 2012.

[58] Google developers. The Closure Compiler. `https://developers.google.com/closure/compiler/`.

[59] YUI Compressor. The Yahoo! JavaScript and CSS Compressor. `http://yui.github.io/yuicompressor/`.

[60] Jonathan Chaffer. *Learning jQuery*. Packt Publishing Ltd, 2013.

[61] Dmitry Sheiko. *Instant Testing with QUnit*. Packt Publishing Ltd, 2013.

[62] Michael Bolin. *Closure - The Definitive Guide: Google Tools to Add Power to Your JavaScript*. O'Reilly, 2010.

[63] Trent Richardson. jQuery Impromptu. `http://trentrichardson.com/Impromptu/`.

[64] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[65] Emden Gansner, Eleftherios Koutsofios, and Stephen North. Drawing graphs with DOT. Technical report, Technical report, AT&T Research. URL `http://www.graphviz.org/Documentation/dotguide.pdf`, 2006.

[66] Andreas Winter, Bernt Kullbach, and Volker Riediger. An overview of the GXL graph exchange language. In *Software Visualization*, pages 324–336. Springer, 2002.

[67] Ivan Herman and M Scott Marshall. GraphXML. an XML-based graph description format. In *Graph Drawing*, pages 52–62. Springer, 2001.