Software environment for rapid prototyping of graph and network algorithms

K. BANACH, R. KASPRZYK
kamil@banach.net.pl, rkasprzyk@wat.edu.pl

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw

The article presents an innovative software environment for rapid prototyping of graph and network algorithms. The environment consists of two main components: an editor of graphs & networks and an editor of algorithms' code. The presented environment enables interactive visualization of algorithms implemented therein, which in turn allows quick verification of algorithms results as well as its correctness. The aim of the environment construction was to provide a solution for rapid prototyping of novel algorithms. The developed tool can also be successfully used for educational purposes.

Keywords: graph and network theory, algorithms prototyping, algorithms visualization.

1. Introduction

Last years have seen a huge interest in network systems. The number of interdisciplinary researches undertaken in this field is affected by the strategic importance of network systems, particularly from the perspective of crisis management. In fact, networks, which are understood as sets of vertices and branches for representing different kinds of relationships between vertices, are ubiquitous. There are many examples of systems modelled with the use of networks, including the Internet, WWW, transport networks, transmission networks, electrical grids, and finally social networks.

Nowadays, graph and network theory is used to model, explore and optimise network systems [2], [3], [4], [5]. There are lots of graph and network algorithms, which help researchers or decision makers to resolve theoretical or practical problems e.g. shortest path in graphs and networks, minimum spanning trees or graph colouring. It's worth to mention that in the 90s, researches started focus on creating of a wide range of algorithms which model the evolution of genuine graphs and networks.

The challenge facing us is to provide software solution for rapid prototyping of novel graph and network algorithms. The presented environment enables interactive visualization of algorithms implemented therein, which in turn allows quick verification of the algorithms results as well as its correctness. The developed tool can also be successfully used for educational purposes.

2. Definition and notation

Graph is an abstract representation of the structure of any system. Formally, graph can be defined as follows [1], [6]:

$$G = \langle V, B, I \rangle \tag{1}$$

where:

V – a set of graph vertices;
B – a set of graph branches;
I – an incident relation ($I \subset V \times B \times V$), which meets two conditions:

1) $\forall\, b \in B\ \exists\, x, y \in V \quad \langle x, b, y \rangle \in I$

2) $$\forall\, b \in B\ \exists\, x, y, v, z \in V \quad \langle x, b, y \rangle \in I \wedge \langle v, b, z \rangle \in I \Rightarrow (x = v \wedge y = z) \vee (x = z \wedge y = v)$$

Based on the incident relation I, three types of branches can be indicated:

$\tilde{B}$ – a set of edges, which meets a condition $\langle x, b, y \rangle \in I \wedge \langle y, b, x \rangle \in I \wedge x \neq y$

$\vec{B}$ – a set of arcs, which meets a condition $\langle x, b, y \rangle \in I \wedge \langle y, b, x \rangle \notin I \wedge x \neq y$

$\dot{B}$ – a set of loops, which meets a condition $\langle x, b, x \rangle \in I$

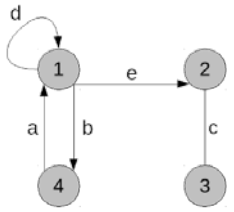

Fig. 1. Graph example: vertices – 1, 2, 3, 4; edge – c; arcs – a, b, e; loop – d

In the Figure 1 there is an example graph with four vertices, one edge, three arcs and one loop. Numbers and letters on vertices and branches are so called labels. It is worth to mention that labels are introduced only for identification of those elements (vertices and branches). It means that labels are not the description of the systems modeled using graphs. To describe elements of graph a concept called network is introduced.

Let's now define the network as follows:

$$N = \left\langle G, \{f_i(v)\}_{\substack{i\in\{1,\ldots,NF\}\\ v\in V}}, \{h_j(b)\}_{\substack{j\in\{1,\ldots,NH\}\\ b\in B}} \right\rangle \quad (2)$$

where:

G – is the graph defined by (1);

$f_i : V \to ValV_i$ – the *i-th* function on the graph vertices, $i = 1,..,NF$, $(NF$ – number of vertex functions), $ValV_i$ – is a set of f_i values;

$h_j : B \to ValB_j$ – the *j-th* function on the graph branches, $j = 1,..,NH$, $(NH$ – number of branch functions), $ValB_j$ – is a set of h_j values.

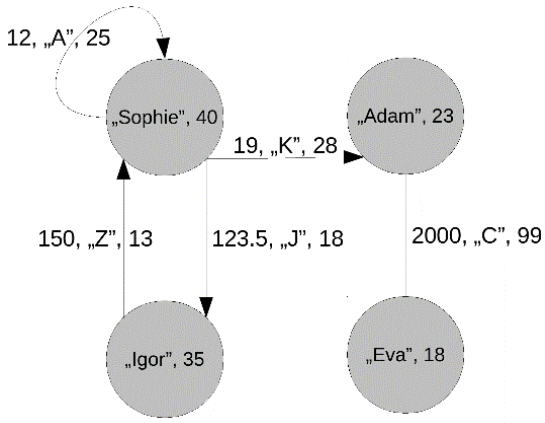

Fig. 2. Network example: two functions described on vertices and three functions described on branches

In the Figure 2 we can see a network example based on the graph from the Figure 1. Results of functions on vertices and branches are commonly called weights. In example above we can see text and numerical weights.

Networks are commonly referred to as weighted graphs by many authors. There are lots of other concepts, which are used in graph theory. A directed graph, called also digraph, is a graph which contains only arcs and loops (set of edges is empty). An undirected graph is a graph which contains only edges and loops (set of arcs is empty). A multigraph is a graph which can have multiple branches of the same type between the same pair of vertices. Opposite to multigraph is an unigraph which not allows to have multiple branches of the same type between two vertices.

Also in literature we can notice different names for vertices and branches i.e. vertices are called nodes, edges are called lines and arcs are called directed edges or directed lines.

3. The concept of the environment

During many researches we used different software solutions. Every one of them demanded of us learning about their architecture, used technologies, libraries and dependencies. There were many restrictions when we tried to adapt them to our needs. Any current existing environments doesn't allow us to simply write and test algorithms.

For the reasons given above we wanted to create a simple, but extensible, environment, where users can just type in and test graph and network algorithms. In addition, that environment should be used by end users without downloading and installing additional software. That requirements are met by the web application which can be run in any web browsers.

Created environment allows user to quickly type algorithm in *JavaScript* language and test it. We believe that choose of *JavaScript* (to be exact – some subset of it), as a language used to describe algorithms, is most appropriate choice, because of its simplicity. Also its syntax is similar to many other widely used languages like *C#* or *Java*.

4. The architecture and GUI of the software environment

The solution – the software environment for rapid prototyping of graph and network algorithms – that was created, contains two connected software components.

The first component is visual editor which allows to create and modify a graph or a network. It allows to set various properties of created vertices and branches like colour or weights (both number and text).

The second component, called algorithm's code editor allows to type an algorithm and run it in an isolated environment – *sandbox*. Inside of this part there is an interpreter which parses code into the *Abstract Syntax Tree (AST)*. Code is interpreted and run by walking over that tree. Because it's not intuitive for people to run algorithm in this way, our modification of interpreter allows user to run an algorithm line

by line. Thanks to that modification user can easily track execution of any graph and network algorithms.

The main screen of the software environment is presented in the Figure 3.

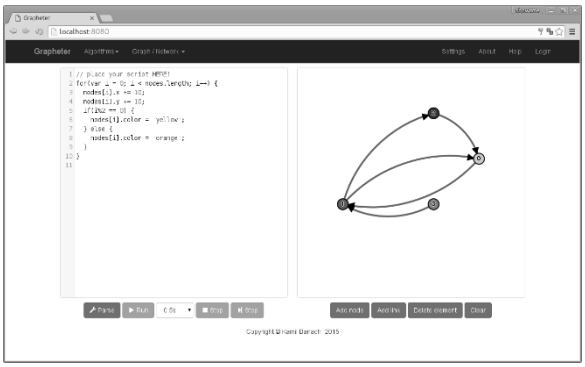

Fig. 3. The main screen of the environment

In the Figure 4. is presented a window for editing node's properties.

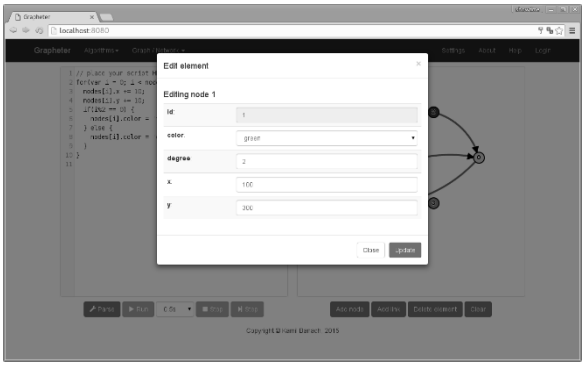

Fig. 4. Edit node screen

Both components i.e. the visual editor and the code editor are connected by shared *JavaScript* objects. These objects allow to use a graph or a network created in the visual editor during the execution of algorithms implemented in the code editor. What is more, the outcomes of the algorithm's execution is presented in real--time on a graph or a network drawn within the visual editor.

5. Technologies and libraries used to create environment

The environment, as stated before, was created as the web application. That implies splitting the software into tiers, for our purpose two tiers.

First tier, in literature often called backed, in our case is responsible for communication with a database. It's built with usage of the *Spring Framework* [8]. One part of that framework, *Spring Data*, was used to communicate with non-relational database *MongoDB* [9]. In the database there are stored data about saved graphs, networks and algorithms.

Second tier, called frontend, is a graphical user interface, containing two components mentioned in section 4. It was built in modern web technologies with *Open Source* libraries:

- *JS-Interpreter* [10] as algorithms interpreter;
- *CodeMirror* [11] as text editor with syntax coloring;
- *D3.js* [12] as graph and network visualization library;
- *jQuery* [13] to communicate with backend part.

Both tiers are bundled into one package, which can be deployed to any Java application server e.g. *Wildfly* or *Jetty*.

6. Algorithms build in interpreter

There are some algorithms embedded into the software environment for rapid prototyping. The algorithms are divided into four groups:

- basic graph algorithms;
- basic network algorithms;
- graph and network creation algorithms;
- diffusion in networks.

The first group, basic graph algorithms, contains algorithms that operate only on graph. There are three built-in algorithms: *Breadth First Search*, *Deep First Search* and *Largest First Coloring* algorithm.

The second group, basic network algorithms, contains two algorithms that operate on networks. Those algorithms are *Prim's Algorithm* that finds *Minimum Spanning Tree* on networks and *Dijkstra's Algorithm* that finds *Shortest Paths* in networks.

The third group, graph and network creation algorithms, at present contains only one algorithm – *Barabási-Albert* model which generate so called *Scale Free* networks. That model is based on two main assumptions: constant growth and preferential attachment.

The fourth group is empty for now but it will include algorithms for simulation of different kind of phenomenon in networks [5].

The list of algorithms can be easily expanded. The software environment for rapid prototyping of graph and network algorithms contains all tools necessary to create any graph or network algorithm. Users of the software are able to manipulate weights or colors of vertices and branches. Also there is a possibility to add

new vertices and branches. That means ones can shape graphs and networks directly within visual editor or indirectly using the code editor.

In the Figure 5. we can see code and results of running the *Largest First Coloring* algorithm.

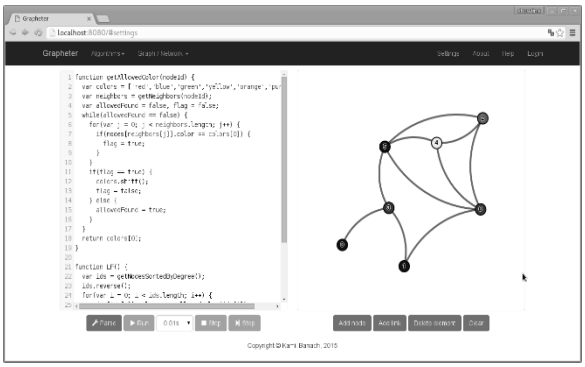

Fig. 5. Screenshot with result of running the *Largest First Coloring* algorithm

7. Simple case study

As a simple case study we will slightly modify basic *Barabási-Albert* model by adding starting vertex attractiveness to a probability of connection to already existing vertices within a graph [7]. The probability for vertex i will be calculated as follows:

$$P(i) = \frac{k_i + a}{\sum_{j=1}^{n}(k_j + a)} \tag{3}$$

where:

a – starting vertex attractiveness;

k_i – degree of vertex *i*;

n – number of vertices in a graph.

In Figure 6 we can see source code of the elementary *Barabási–Albert* model labelled *function BA*. It has three parameters:

$m0$ – size of initial full graph;

N – number of vertices to add;

M – number of arcs to add with a vertex.

Most interesting for us is line 21, where the probability of connection to randomly selected vertex is calculated.

```
13 function BA(m0, N, M) {
14     createFullGraph(m0);
15     stop();
16     for(var i = m0; i < N; i++) {
17         var nodeId = addNode();
18         var edgesAdded = 0;
19         while (edgesAdded < M) {
20             var randomId = getRandomInt(nodeId - 1);
21             var propOfNode = nodes[randomId].degree / getSumOfDegree();
22             var random = Math.random();
23             if(random < propOfNode) {
24                 if(!isLinkExist(nodeId, randomId)) {
25                     addLink(nodeId, randomId);
26                     edgesAdded++;
27                 }
28             }
29         }
30     }
31 }
```

Fig. 6. Screenshot with code of BA generation algorithm

Let's start modification of the elementary *Barabási–Albert* model by adding additional parameter a (our starting attractiveness) to the signature of *function BA*. Next we have to modify line 21, where the probability of selecting randomly chosen vertex is calculated. For the part above the fraction bar it is obvious – simply adding the parameter a is enough. For the part under the fraction bar it seems to be more complicated – we need to add the parameter to every vertex degree.

Let's look at the sum in (3). We can split it into two independent sums:

$$P(i) = \frac{k_i + a}{\sum_{j=1}^{n} k_j + \sum_{k=1}^{n} a} \tag{4}$$

As we can see, the second sum can be simplified to $n * a$. That allows us to simply write:

$$P(i) = \frac{k_i + a}{\sum_{j=1}^{n} k_j + n * a} \tag{5}$$

We can now get the part under the fraction bar by simply adding sum of all vertices degrees and the parameter a *n*-times. Number of nodes can be accessed by property *nodes.length*, so we will simply write $nodes.length * a$.

Modified *function BA*, algorithm with starting attractiveness, is presented in Figure 7.

```
function BA(m0, N, M, a) {
    createFullGraph(m0);
    stop();
    for(var i = m0; i < N; i++) {
        var nodeId = addNode();
        var edgesAdded = 0;
        while (edgesAdded < M) {
            var randomId = getRandomInt(nodeId - 1);
            var propOfNode = (nodes[randomId].degree + a) /
                             (getSumOfDegree() + nodes.length*a);
            var random = Math.random();
            if(random < propOfNode) {
                if(!isLinkExist(nodeId, randomId)) {
                    addLink(nodeId, randomId);
                    edgesAdded++;
                }
            }
        }
    }
}
```

Fig. 7. Screenshot with code of BA generation algorithm with starting node attractiveness

8. Summary

Presented software environment is a modern solution for rapid graph and network algorithm prototyping. It enables interactive visualization of algorithm which in return allows to quick verification of the results and algorithms correctness.

Moreover, environment can be also successfully used for educational purposes e.g. on courses include graph and network theory.

9. Bibliography

[1] Korzan B., *Elementy teorii grafów i sieci. Metody i zastosowania*, WNT, 1978.

[2] Tarapata Z., Kasprzyk R., "An application of multicriteria weighted graph similarity method to social networks analyzing", in: *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*, July 20–22, 2009, Athens, Greece, pp. 366–368, IEEE Computer Society, 2009.

[3] Tarapata Z., Kasprzyk R., "Graph-Based Optimization Method for Information Diffusion and Attack Durability in Networks", in: *RSCTC 2010*, LNAI 6086, pp. 698–709, Springer, Heidelberg, 2010.

[4] Bartosiak C., Kasprzyk R., Tarapata Z., "Application of Graphs and Networks Similarity Measures for Analyzing Complex Networks", *Biuletyn Instytutu Systemów Informatycznych*, Vol. 7, 1–7 (2011).

[5] Kasprzyk R., "Diffusion in Networks", *Journal of Telecommunications and Information Technology*, Vol. 2, 99–106 (2012).

[6] Diestel R., *Graph Theory*, Springer-Verlag Berlin Heidelberg, 2010.

[7] Fronczak A., Fronczak P., *Świat sieci złożonych. Od fizyki do Internetu*, Wydawnictwo Naukowe PWN, Warszawa, 2009.

[8] Spring Framework project webpage, https://spring.io/

[9] MongoDB project webpage, https://www.mongodb.com/

[10] JS-Interpreter project source code, https://github.com/NeilFraser/JS-Interpreter

[11] CodeMirror project webpage, https://codemirror.net/

[12] D3.js project webpage, https://d3js.org/

[13] jQuery project webpage, https://jquery.com/

Środowisko programowe do szybkiego prototypowania algorytmów grafowo-sieciowych

K. BANACH, R. KASPRZYK

Artykuł ma na celu przedstawienie autorskiego środowiska programowego do prototypowania algorytmów grafowo-sieciowych. Środowisko składa się z dwóch głównych komponentów wzajemnie od siebie zależnych, tj. edytora grafów i sieci oraz edytora kodu algorytmów. Prezentowane środowisko umożliwia interaktywną wizualizację implementowanych w nim algorytmów, co w konsekwencji pozwala na szybką weryfikację efektów działania algorytmu, w tym jego poprawność. Celem budowy środowiska było dostarczenie rozwiązania pozwalającego na szybkie prototypowanie nowych algorytmów. Opracowane narzędzie może również zostać z powodzeniem wykorzystane do celów dydaktycznych.

Słowa kluczowe: teoria grafów i sieci, prototypowanie algorytmów, wizualizacja algorytmów.