Bull. Nov. Comp. Center, Comp. Science, 37 (2014), 163–180

Visual Graph: an interactive system for the visualization of hierarchical attributed graphs*

T. A. Zolotuhin

Abstract. The paper presents information about the development of the Visual Graph system, its application area, as well as the main problems encountered during the system design and their solutions.

Keywords: attributed hierarchical graphs, graph visualization, graph navigation, search algorithm of maximum common subgraph of two graphs.

1. Introduction

Visualization of information plays an important role in human life. People are believed to get about 90% of information visually. It took thousands of years to go from the easiest ways of visualization in the form of rock paintings to maps, charts, and diagrams. Today, visualization is an integral part of processing complex information about the structure of objects.

Many data structures, which are of practical interest for mathematics and computer science, can be represented as graphs. The main class of such graphs is the hierarchical attributed graphs [1, 2].

As a rule, the advantages of using graphs in research can only be perceived if a good system for their visualization and processing is available. Therefore, there has recently been a lot of interest in the world concerning the methods and means of graph visualization, as evidenced by the growing number of publications containing the description of new algorithms and methods for graph visualization, as well as their implementation in systems [3, 5].

This paper describes one of such systems called the Visual Graph, provided under the BSD license and being developed in the frames of a project on the visualization and search of information in hierarchical attributed graphs (hereinafter, the "graph model").

2. Application areal

During the development of compilers, several structures have been created that are represented as graphs: syntax trees, control flow graphs, and call graphs. Each of these graphs has its practical application. Thus, syntax

*Partially supported by the Russian Foundation for Basic Research under Grant N 12-07-0091.

trees are used for the internal representation of a program in compilers or interpreters, control flow graphs are good for optimizations in compilers and tools of static code analysis, and call graphs are used for program debugging. All these graphs can be regarded as hierarchical attributed graphs.

The representation of these graphs in different compilers may be different, and it may even differ in different versions of the same compiler.

To solve the problem, developers have chosen a method where either a compiler or a helper program translates a graph from the internal representation to a file in a format supported by the Visual Graph. After that, the Visual Graph system can read this graph model from a file of the supported format and visualize it.

3. Existing problems and their solutions

When designing the Visual Graph system, developers faced many problems. This section will deal with the main ones and with their solutions in the Visual Graph system.

The first problem is storing graphs outside the Visual Graph system, i.e. their representation in a file system, in a text format. There are a lot of languages for the representation of simple graphs in the text form, but few of them support the description of attributed hierarchical graphs. One of these languages is GraphML [8].

GraphML is a language for describing the XML-based graphs. This format allows describing directed, undirected, mixed, hyper, and hierarchical graphs and specific attributes for applications. Accordingly, GraphML fully supports hierarchical attributed graphs.

Another problem is graph storage inside the Visual Graph system. The size of an input graph can be hundreds of megabytes. Therefore, storing this amount of information in RAM can quickly exhaust it. A possible solution to this problem is caching data to a hard drive using a relational database.

The embedded SQLite database was chosen as a relational database [9]. Unlike most popular relational databases, SQLite does not require installing a server and the client-server architecture is reduced to working with files.

Another problem is the extensibility of the Visual Graph system. This problem occurs because some parts of the system, such as navigation, visualization and analysis of graphs, are submitted by a set of tools that can be expanded both by the Visual Graph system developers and third-party developers.

To construct an expandable system, it was decided to use the product Apache Felix [14], which, in turn, is an implementation of the specification OSGi [13]. This solution is de facto for this type of tasks and allows developers to extend the system by writing new plug-ins.

Yet another problem discussed is navigating through graphs. As is

known, there is no universal set of navigation tools that would satisfy any needs of any user to solve any of his/her problems. Therefore, it was decided to focus on the application area described above and provide it with the following tools:

- Desktop is an instrument that contains a set of tabs that can be opened by the user for visualizing the selected regions of a graph model as a graph on the plane. To improve the resulting image, the user can change the shape of vertices and edges, layout and its parameters, displayable attributes, the scale of the visible area, and many other parameters.
- Minimap is an instrument that allows the user to see the whole graph placed in a current tab of the desktop, and also move and zoom the visible area.

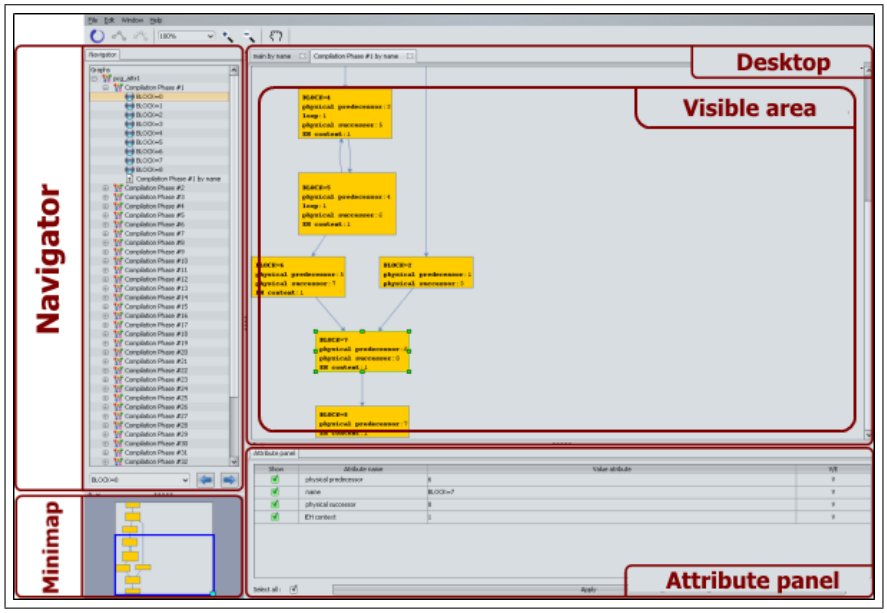

Figure 1. Screenshot of user interface of Visual Graph system

- Navigator is an instrument that visualizes graph models in the tree form. For a quick search of the tree, search line is implemented, which allows the user to easily find the elements he/she is interested in, using regular expression. After that, the user can select the interesting elements and open them in a new tab using the desktop.

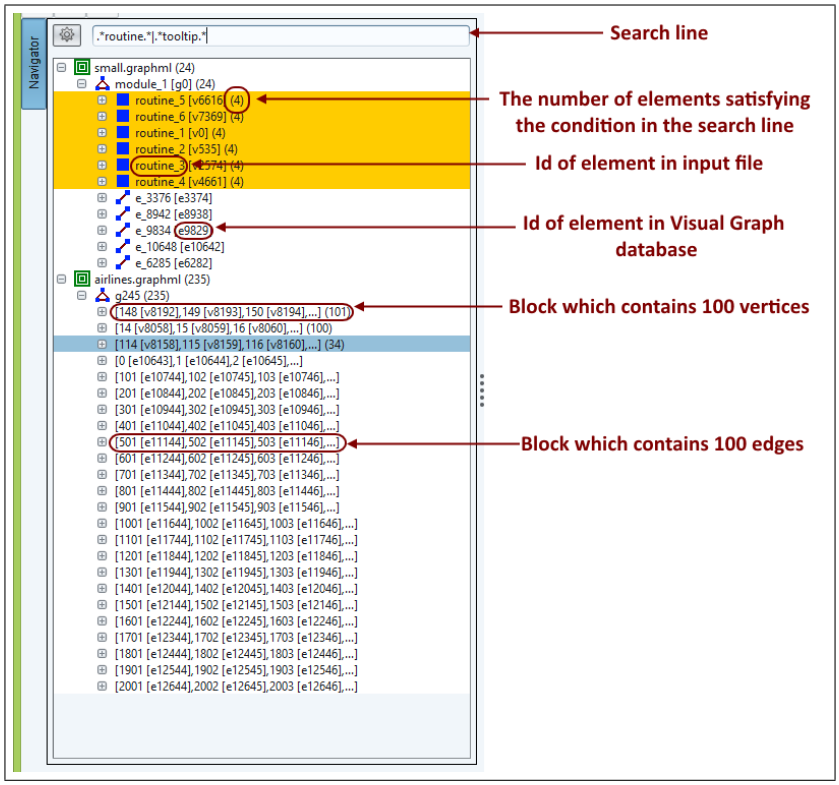

Figure 2. Screenshot of the filter instrument

Also, a Visual Graph system navigator can work with big graphs. It can download the necessary elements dynamically, when the user uncollapses an element containing inner elements. If the number of inner elements is too large), the elements can be combined in blocks. Each block contains no more than 100 elements.

- Attribute panel is an instrument that allows the user to control rendering attributes for selected vertices and edges in the current tab in the desktop. To do this, the user needs to select in the current tab the vertices and edges he/she is interested in, and then mark the attributes he/she wants to visualize in the attribute panel (see Figure 1).
- Filter is an instrument that searches vertices and edges in the current tab using the conditions given. The conditions are specified by the user and use the names and values of attributes. Figure 1 shows a mechanism for specifying the conditions. User-defined conditions can be combined into expressions by means of logical structures, as well as by brackets.

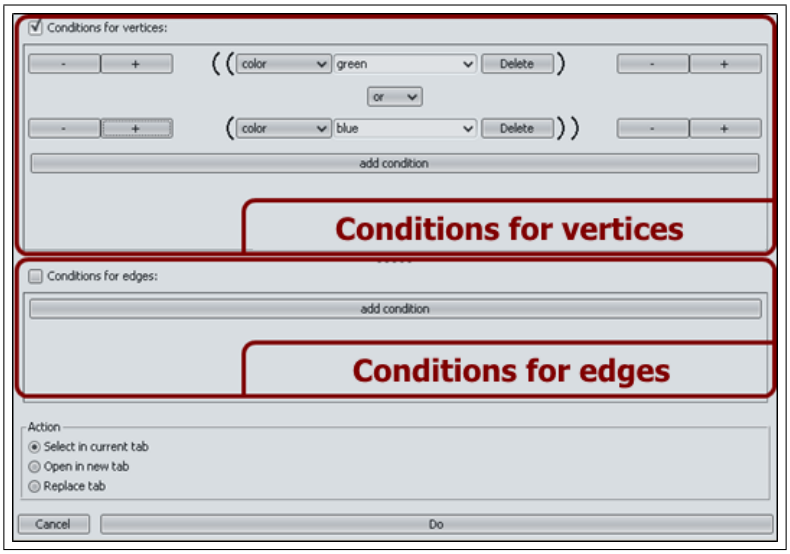

Figure 3. Screenshot of the filter instrument

As a result, the user will receive a set of vertices and edges that will satisfy the given conditions (see Figure 4).

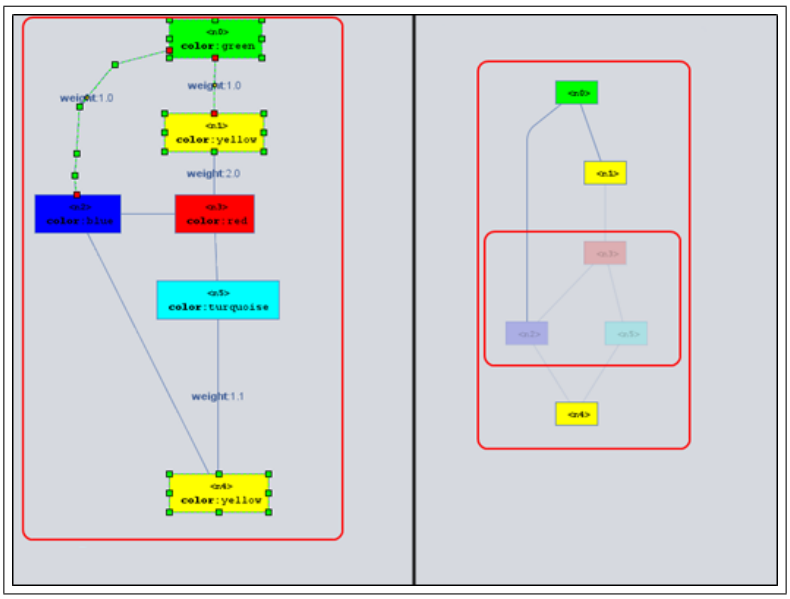

Figure 4. Screenshot of the filters result

In the future, the developers plan to extend the existing functionality and give users the possibility to make their requests to the database using an API, for example SPARQL.

- Search panel is a tool that, similarly to the filter, uses the fact that the user works with attributed graphs. In contrast to the filter, this tool allows you to set conditions only for vertices and performs search not only for the graph in the current tab, but also for all its internal graphs and their internal graphs and so on, in the hierarchical order. The result of this search is a tree (see Figure 5), in which the red crosses mark the vertices that do not satisfy the conditions specified.

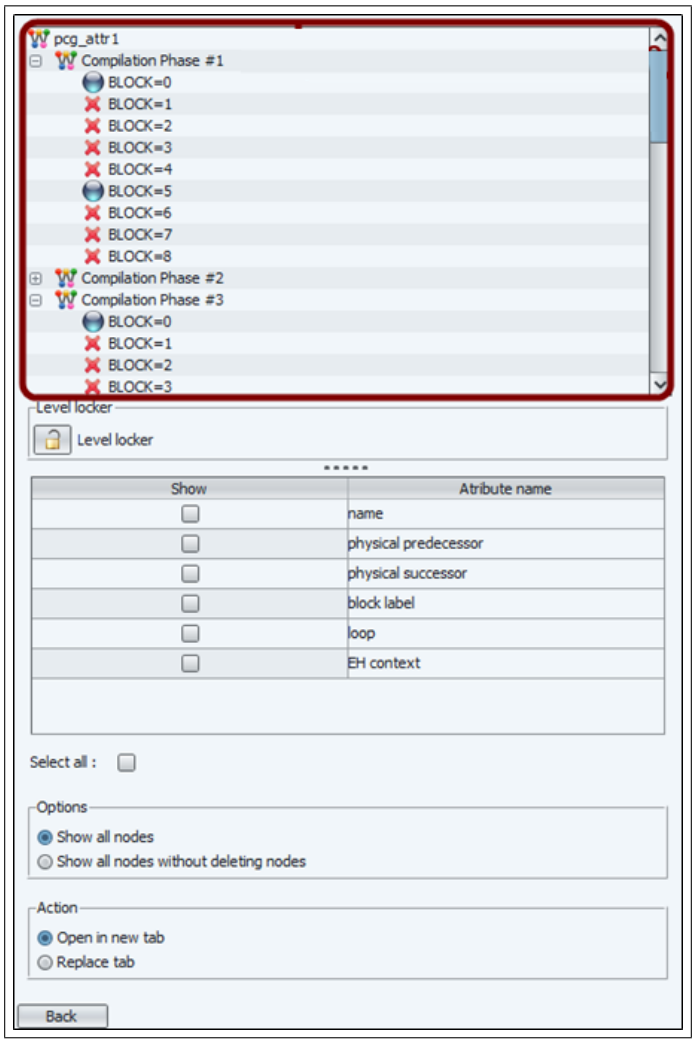

Figure 5. Screenshot of the search panel result

- Notebook is an instrument that allows you to upload files with additional information and link them to the graph models. After that, the user can move from a vertex of a graph model to additional textual

information. For example, it can be used for joining the graph model with the source code.

The tools of structural analysis can also be considered navigation tools, i.e. various graph algorithms help the user to get various kind of information.

The Visual Graph system contains the following tools for structural analysis:

- lighting of the shortest path,
- lighting of cycles,
- lighting of neighborhood vertices,
- lighting of all paths from the source vertex to the target vertex, and
- finding the maximum common subgraph of two graphs [4].

The latter tool is worth considering in more detail. Let us deal with the problem it solves and an algorithm for its solution.

The problem of finding the maximum common subgraph of two graphs occurs when the user wants to find the difference in the behavior of different versions of the same compiler (for example, the first and second versions). This difference can be seen owing to the graphs arising in the compiler at the compile time. In particular, thanks to them the user can find out what optimizations have been made or not made by the compiler of the second version as compared with the compiler of the first version.

For small graphs (about 5 vertices and 3-5 edges), the user can try to solve this problem without resorting to additional instruments. However, if a graph contains more than 10 vertices and 7-10 edges, this problem will not be trivial.

Now let us consider the algorithm for solving the above problem.

The search for the maximum common subgraph of two graphs belongs to the class NP and is solved using brute-force and different heuristics for reducing the number of options. The proposed algorithm follows the same path and works as follows:

1. two attributed (or not attributed) graphs are input, as well as additional information about the weight of a particular attribute for the vertices and edges;
2. a bipartite graph is constructed with vertices from the first and second graphs (the first and second partite sets, respectively). Furthermore, from each vertex of the first partite set edge is constructed to the second partite set with the weight equal to the percentage of coincidence of these two vertices (between zero and one, respectively), where 0 means a radical mismatch of vertices and 1 means their perfect complete match;

3. the Hungarian algorithm for solving the assignment problem is applied to the resulting bigraph to obtain the maximal matching of the maximum weight [6, 7]. Obviously, such a matching may not be in the singular, so all of these options have to be sorted out;
4. the output is the maximum common subgraph of two input graphs, as well the vertices and edges not published in it of the first and second graph, respectively.

Now let us discuss how this is implemented in the Visual Graph system. As it was said before, the input is two attributed (or not attributed) graphs, and the user wants to find the maximum common subgraph (see Figure 6).

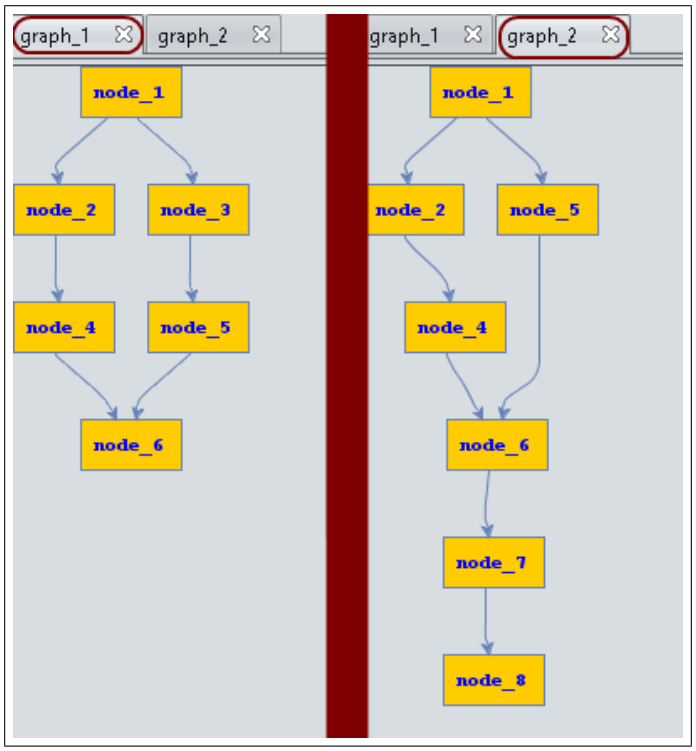

Figure 6. Fig. 6: Example of two input graphs

In the Visual Graph system, the user has two options:

1. selecting two graphs in the navigator and calling the function for comparing the two graphs in the opened popup menu;
2. selecting two vertices with inner graphs in the current tab on the desktop and calling the function of comparing the two graphs in the popup menu.

After that, the desktop will contain a new tab with the result of comparing the two graphs (see Figure 7).

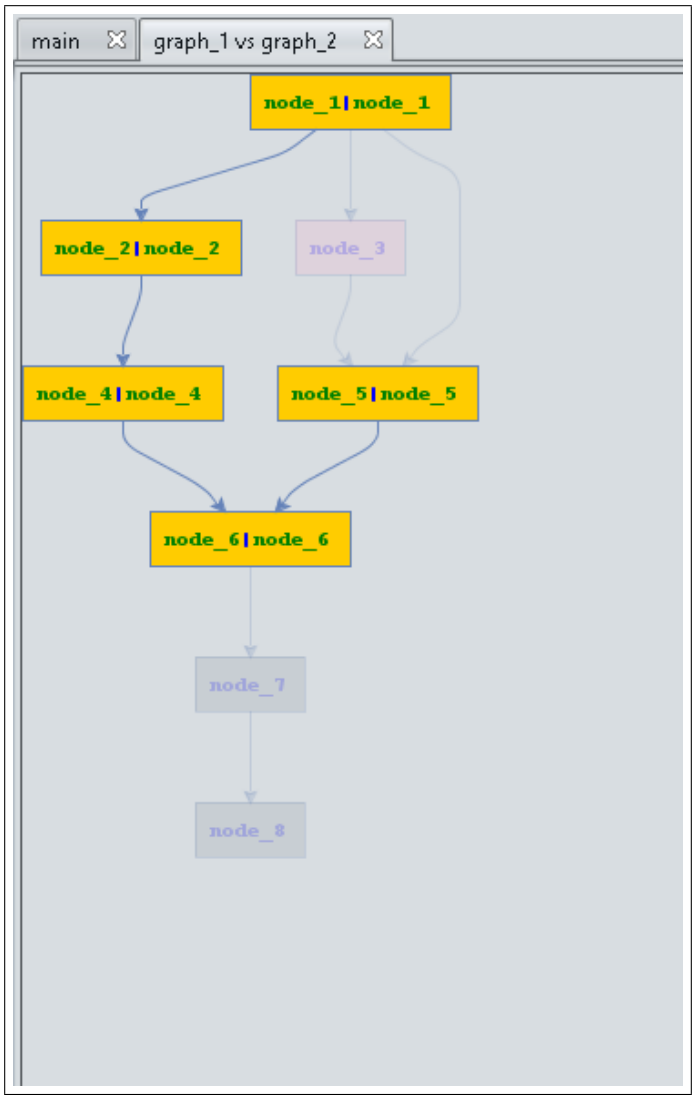

Figure 7. The tab with the result of comparing the two graphs

In Figure 7, we can see the maximum common subgraph of the two source graphs; the transparent colors mark the outside elements: red is for the outside elements of the first graph, and blue is for the second.

The user can change the weights of any attributes, thereby affecting the matching of the vertices and edges (see Figure 8).

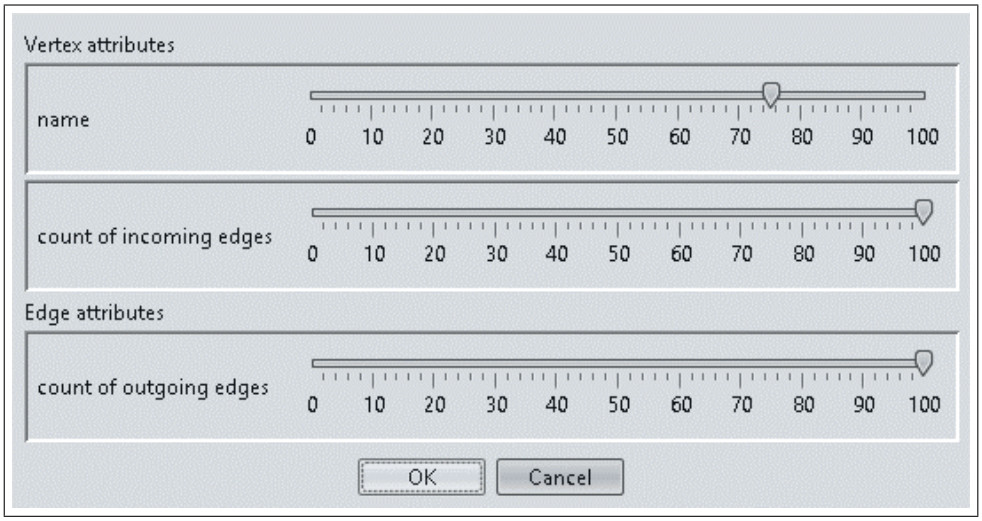

Figure 8. Menu for changing the weights of attributes

In the menu in Figure 9, the user can select a strong or weak matching:

1. In the case of a strong matching, the values of two attributes are considered equal only if they coincide completely (1 or 0);
2. In the case of a weak matching, the following function is used for numerical attributes:

$$1, if a = b \tag{1}$$
$$0, if a >= 0, b < 0 \tag{2}$$
$$0, if b >= 0, a < 0 \tag{3}$$
$$1 - \frac{|a-b|}{|max(a,b)|}, otherwise \tag{4}$$

To match the string attributes, the following function is used:

$$\frac{2i}{length1 + length2}, \tag{5}$$

where i is the position in which the strings are different or the length of the string if the strings are equal; length1 and length2 are the length of the first and the second string, respectively.

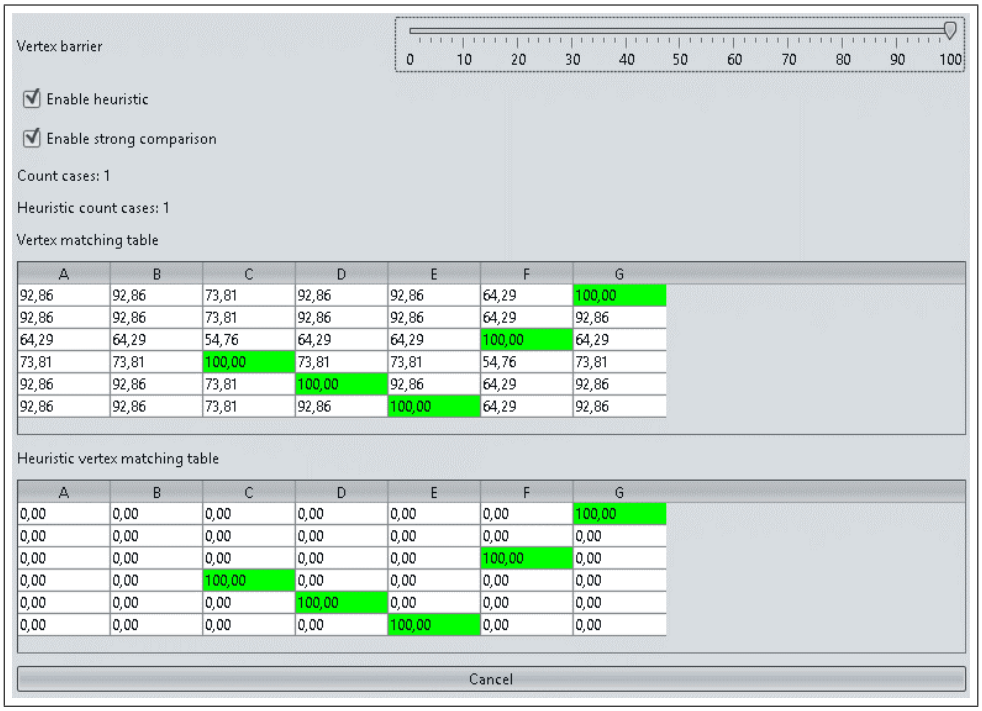

Figure 9. Menu for selecting type of comparing and barrier

In the menu, the user can estimate the maximum number of iterations in the algorithm. In addition, the user can stop the process of searching for the best solution at any time and obtain a solution that will not necessarily be the best.

The use of this tool has shown that the algorithm proposed copes well with the graphs whose elements contain quite diverse values of the same attributes and fails in other cases. The number of vertices in the test graphs ranged from 10 to 1000.

The last problem to be considered in this section is the problem of graph mapping, namely, the layout of the graph in the plane.

The layout problem layout does not have an optimal solution and cannot have it because the sets of the criteria for assessing the quality of layouts are different for different graphs. For some graphs, a certain criterion will be of greater importance than the others in the readability of the graph. However, a user can pick up the kinds of layouts, their parameters and approximate algorithms for the specific classes of graphs that are typically visualized. As mentioned earlier, the main graphs the compiler processes are syntax trees, control flow graphs and call graphs. All these graphs have a small number of edges and a hierarchical structure. As a rule, a hierarchical layout is good for these types of graphs. Depending on the number of graph vertices

and edges, we can apply heuristic algorithms, or algorithms to minimize the width or height of the image pickup, and to decrease the intersection between the edges.

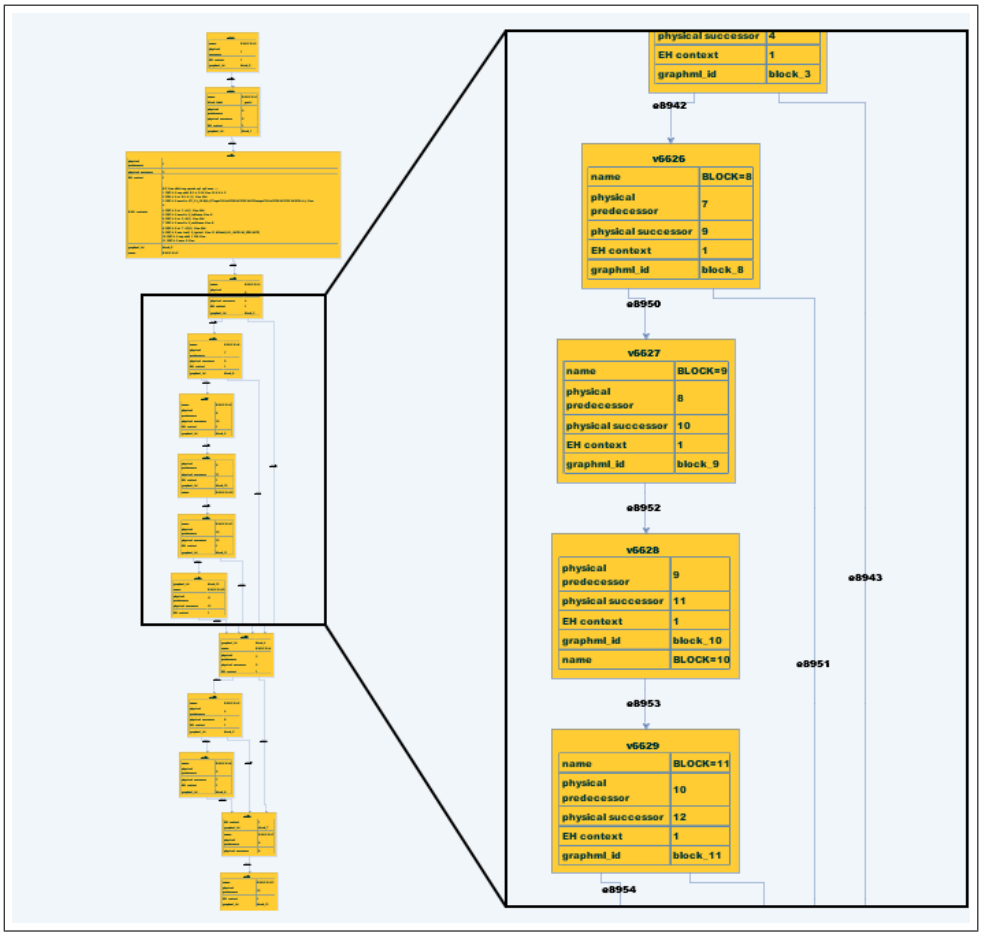

Figure 10. Screenshot of using the hierarchical layout for control flow graph

4. Features of the Visual Graph system and its analogs

Currently, the market offers a fairly wide range of systems working with graphs. The most popular are aiSee [10], yEd [11] and Cytoscape [12].

The domain area of these systems partially overlaps with the domain area of Visual Graph but on the whole it is wider. For instance, many of them are designed to edit the existing graphs and create new, while the Visual Graph system is oriented exclusively to viewing the existing graphs without the editing option.

The main objectives of such systems are displaying the graphs and nav-

igating through them. Therefore, let us consider the characteristics of each of the systems in the context of these problems.

aiSee is a fee-based system, which automatically draws the layout of the graph described in the language of GDL. Then a user can interactively explore (without editing) a given graph, print it and save it in various formats.

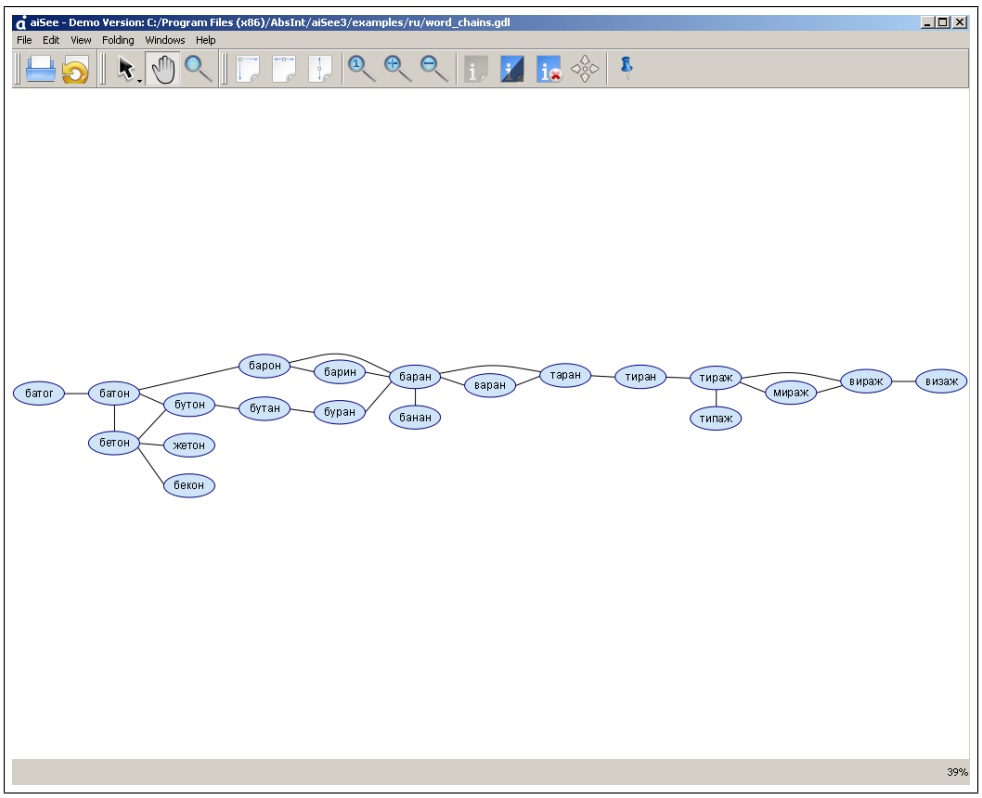

Figure 11. Screenshot of aiSee system

AiSee was originally developed to visualize data structures handled by compilers. Today, it is used by tens of thousands of people around the world in various fields, including:

- business management (block diagrams enterprises, visualization of business processes),
- genealogy (family trees), and
- software development (block diagrams, control flow graphs, and graphs of function calls).

Cytoscape is free system with an open source code used to visualize and analyze networks.

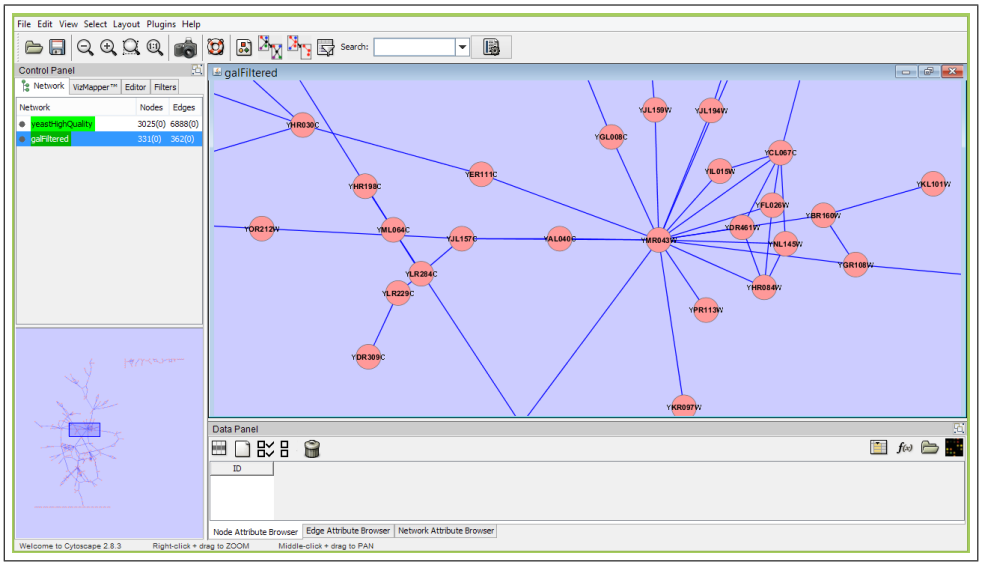

Figure 12. Screenshot of the Cytoscape system

The main domain area of this system is bioinformatics. This system owes its popularity to the fact that it is easily extensible, allowing third-party developers to write various plug-ins. **yEd** is system that can be used to quickly create and edit high-quality diagrams.

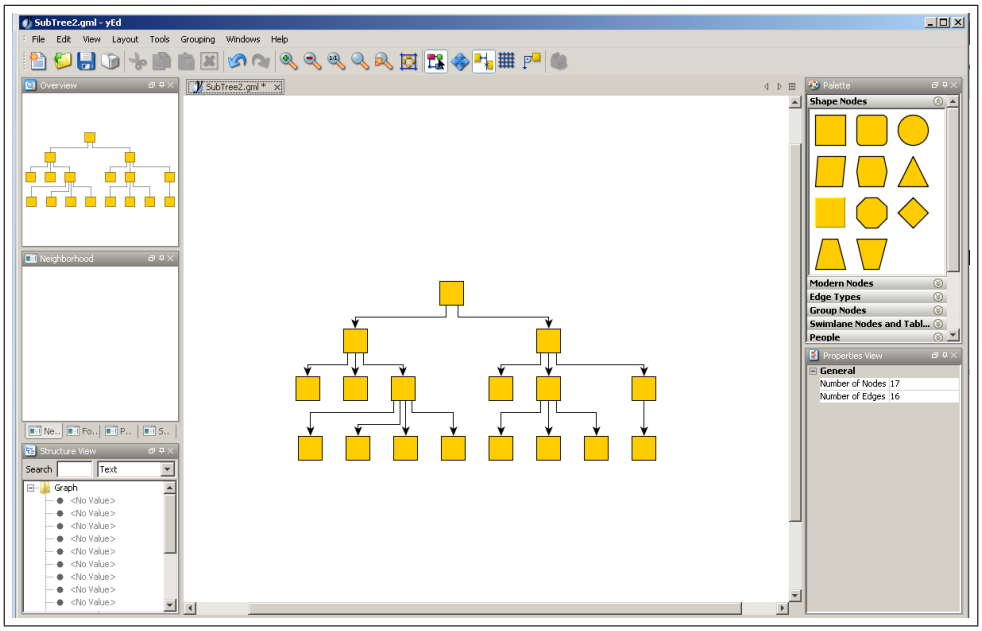

Figure 13. Screenshot of the yEd system

The user can create diagrams manually or import them from external data. A wide range of algorithms can be applied to the resulting graph in order to analyze it and obtain the necessary information.

Each of the systems considered displays graphs and has its own methods to control the displaying of these graphs.

In aiSee, there are several layouts and a lot of settings whose influence on the result is negligible. The quality of the layout of certain types of graphs is very high.

yEd has a large number of layouts and options for them. The user can fine-tune each type of a graph.

Cytoscape has its own layout algorithms, as well as third-party algorithms, such as yFiles. It is also worth noting that the results obtained in yEd are much better than similar results in Cytoscape with settings by default.

The Visual Graph system provides several layouts, the main of which is the hierarchical layout, which has been maximally adapted to work with graphs used in compilers. If desired, the layout may be modified and / or adjusted for other types of tasks.

Now we will deal with the navigation features in each of these systems. aiSee system:

1. Desktop is an instrument that visualizes the graph model as a static image, which cannot be modified by moving or changing elements, for example.
2. Searcher is an instrument that allows the user to search elements in the graph model using their names. It supports regular expressions, the choice of categories of elements for the searching and saving of previous searches.

yEd:

1. Desktop is similar to the aiSee desktop except that
   - it is capable of working with several graph models simultaneously in a single instance of the program (a tab for each of the graph is created);
   - it is capable of modifying the elements of the graph, from changing their positions and sizes to setting the attributes that affect their visualization.
2. Navigator is an instrument that displays the graph placed in the current tab.
3. Minimap is an instrument that shows the whole graph placed in the current tab.

Cytoscape:

1. Desktop similar to the yEd desktop. The difference is that Cytoscape can work with hierarchical graphs.
2. Minimap identical to the minimap provided by the yEd system.
3. Attribute panel is a tool that allows the user to manage the visualization of attributes, edit the existing attributes and create the new ones. To this end, the tool builds a table using the attribute names as rows and graph elements as columns. This method has grave disadvantages if the number of attributes and graph elements is large.

The navigation features provided by the Visual Graph system (a detailed description of all the following tools can be found in the section "Existing problems and their solutions"):

1. Desktop similar to the desktop provided by the analog systems. The main difference is that in the analog systems a desktop shows a full graph model in one tab, while the Visual Graph desktop allows the user to select interesting elements (vertices and edges) in the graph model (using the navigator or another instrument) and then to visualize them in different tabs. This strategy, which works well with big graphs, is one of the most important features of the Visual Graph system. In order to understand something complex, we can always break it into many small simple pieces for subsequent analysis;
2. Navigator similar to the yEd navigator. The main difference is that in yEd system the navigator shows a graph that is in the current tab, and the Visual Graph navigator shows all the graphs the user is processing at the moment. In addition, the Visual Graph navigator can dynamically download the necessary elements, when the user uncollapses an element containing inner elements. If there are too many inner elements, they are combined in blocks. Each block contains no more 100 elements. The analogs do not have such a mechanism and so cannot work with the biggest graphs (with about 1-3 billion elements). In the analogs, the navigator wants to show all the elements and this leads to big lags.
3. Minimap identical to the yEd minimap.
4. Attribute panel displays a set of attributes of the elements selected in the current tab and allows the user to manage their visualization.
5. Filter and search panel allows the user to search graph elements using specifying expressions. The analogs have simple searchers incapable of a flexible search of graph elements. They can find a graph element using a substring (not even a regular expression) for a name (or a title

attribute). It is a good (easy and understandable) strategy for small graphs but it does not work for large with a lot of attributes.

6. Although the Notepad does not have many of the features standard for most notepads popular today, such as syntax highlighting, searching with regular expressions, and so on, its functionality suffices to cope with the challenges facing the user of the Visual Graph system, namely:
   - Opening several text files and relating them with graphs, with a subsequent pass from a graph element to a fragment in the text file;
   - Highlighting the search result.
7. Structural analysis instruments, such as searching of cycles, critical paths, neighborhoods of vertices and the maximum common subgraph of two graphs. It is difficult to estimate the pluses and minuses of these instruments, but certainly it is practically impossible to do without them. The analogs do not have most of the tools for structural analysis, such as finding the maximum common subgraph or lighting the critical paths).

5. Conclusion

In this paper, we have discussed the problems encountered during the development of the Visual Graph system and its domain area, and have researched similar systems. As we can see, the Visual Graph system has many advantages and is not inferior to its peers in terms of the visualization of hierarchical attributed graphs used in compilers.

It is also worth noting that using the Visual Graph system is not limited to the visualization of graphs arising in compilers. This system can be used in other related fields which require graph visualization and navigation.

References

[1] Kasyanov V.N. Hierarchical graphs and graph models: problems of visual processing // Problems of Informatics Systems and Programming. – Novosibirsk, 1999. – P. 7–32 (In Russian).

[2] Kasyanov V.N., Evstigneev V.A. Graphs in Programming: Processing, Visualization and Application. – St. Petersburg: BHV-Petersburg, 2003 (In Russian).

[3] Kasyanov V.N., Kasyanova E.V. Visualization of Graphs and Graph Models. – Novosibirsk: Siberian Scientific Publ., 2010 (In Russian).

[4] Zolotuhin T.A. Search algorithm of maximum common subgraph of two graphs in Visual Graph system // 5th Internat. Internet Conf. of young scientists and students "Innovative technology: theory, tools and practice" (InnoTech

2013). – URL: http://www.conference.msa.pstu.ru (accessed: 29.01.2013) (In Russian).

[5] DiBattista G., Eades P., Tamassia R., TollisI G. GraphDrawing: Algorithms for Vizualization of Graphs. – Prentice Hall, 1999.

[6] Kuhn H.W. The Hungarian Method for the assignment problem // Naval Research Logistics Quarterly. – 1955. - P. 83–97.

[7] Munkres J. Algorithms for the assignment and transportation problems // J. of the Society for Industrial and Applied Mathematics. – 1957. – P. 32–38.

[8] GraphML specification. URL: http://graphml.graphdrawing.org/specification.html (accessed: 25.09.2014).

[9] SQLite homepage. URL: http://www.sqlite.org (accessed: 25.09.2014).

[10] aiSee homepage. URL: http://www.aisee.com (accessed: 25.09.2014).

[11] yEd homepage. URL: http://www.yworks.com (accessed: 25.09.2014).

[12] Cytoscape homepage. URL: http://www.cytoscape.org (accessed: 25.09.2014).

[13] OSGi Alliance homepage. URL: http://www.osgi.org/Main/HomePage (accessed: 19.09.2014).

[14] Apache Felix homepage. URL: http://felix.apache.org (accessed: 19.09.2014).