Математички Билтен
Vol. 39(LXV) No. 1
2015 (25-32)
Скопје, Македонија

ISSN 0351-336X
UDC: 519.171:004.925.021

INFORMATION VISUALIZATION ON THE BASE OF HIERARCHICAL GRAPHS

Victor N. Kasyanov[1], Elena V. Kasyanova[2]

Abstract. Graphs are the most common abstract structure encountered in computer science and are widely used for abstract information representation. In the paper, we consider a practical and general graph formalism called hierarchical graphs. It is suited for visual processing and can be used in many areas where the strong structuring of information is needed. We present also the Higres and Visual Graph systems that are aimed at supporting of information visualization on the base hierarchical graph modes.

1. INTRODUCTION

Visualization is a process of transformation of large and complex abstract forms of information into visual form, strengthening user’s cognitive abilities and allowing them to take the most optimal decisions. Graphs are the most common abstract structure encountered in computer science and are widely used for structural information representation [3], [8], [13], [14], [19]. Many graph visualization systems, graph editors and libraries of graph algorithms have been developed in recent years. Examples of these tools include VCG [18], daVinci [5], Graphlet [9], GLT&GET [17].

In some application areas the organization of information is too complex to be modeled by a classical graph. To represent a hierarchical kind of diagramming objects, some more powerful graph formalisms have been introduced, e.g. higraphs [6] and compound digraphs [20]. The higraphs are an extension of hypergraphs and can represent complex relations, using multilevel "blobs" that can enclose or intersect each other. The compound digraphs are an extension of directed graphs and allow both inclusion relations and adjacency relations between vertices, but they are less general then the higraph formalism. One of the recent non-classical graph formalisms is the clustered graphs [4]. A clustered graph consists of an undirected graph and its recursive partitioning into subgraphs. It is a relatively general graph formalism that can handle many applications with hierarchical information, and is amenable to graph drawing.

Hence, there is a need for tools capable of visualization of such structures. Although some general-purpose graph visualization systems provide recursive folding of subgraphs, this feature is used only to hide a part of information and cannot help us to visualize hierarchically structural information. Another weak point is that usual graph

2010 *Mathematics Subject Classification.* Primary: 68R10
Key words and phrases. Information visualization, hierarchical graphs

editors do not have a support for attributed graphs. Though the GML file format, used by Graphlet, can store an arbitrary number of labels associated with graph elements, it is impossible to edit and visualize these labels in the Graphlet graph editor. The standard situation for graph editors is to have one text label for each vertex and, optionally, for each edge.

The size of the graph model to view is a key issue in graph visualization [8]. Large graphs pose several difficult problems. If the number of graph elements is large it can compromise performance or even reach the limits of the viewing platform. Even if it is possible to layout and display all the graph elements, the issue of viewability or usability arises, because it will become impossible to discern between nodes and edges of graph model. It is well known that comprehension and detailed analysis of data in graph structures is easiest when the size of the displayed graph is small. Since none of the static layouts can overcome the problems caused by large graphs, hierarchical presentation, interaction and navigation are essential complements in information visualization.

In the paper, we consider a practical and general graph formalism called hierarchical graphs [10]. It is suited for visual processing and can be used in many areas where the strong structuring of information is needed [11], [15], [16]. We present also the Higres and Visual Graph systems that are aimed at supporting of information visualization on the base hierarchical graph modes. The Higres system (http://pco.iis.nsk.su/higres) is a visualization tool and an editor for attributed hierarchical graphs and a platform for execution and animation of graph algorithms. The Visual Graph system (http://www.visualgraph.sourceforge.net) was developed to visualize and explore large hierarchical graphs that present the internal structured information typically found in compilers.

2. HIERARCHICAL GRAPHS AND GRAPH MODELS

Let G be a graph of some type, e.g. G can be an undirected graph, a digraph or a hypergraph (see, e.g. [2]). A graph C is called a *fragment* of G, denoted by $C\subseteq G$, if C includes only elements (vertices and edges) of G. A set of fragments F is called *a hierarchy of nested fragments* of the graph G, if $G\in F$ and $C_1\subseteq C_2$, $C_2\subseteq C_1$ or $C_1 \cap C_2=\varnothing$ for any $C_1, C_2 \in F$.

A *hierarchical graph* $H = (G,T)$ consists of a graph G and a rooted tree T that represents an immediate inclusion relation between fragments of a hierarchy F of nested fragments of G. G is called the *underlying graph* of H. T is called the *inclusion tree* of H.

A hierarchical graph H is called a *connected* one, if each fragment from F is a connected graph, and a *simple* one, if all fragments from F are induced subgraphs of G.

It should be noted that any clustered graph H can be considered as a simple hierarchical graph $H=(G, T)$, such that G is an undirected graph and the leaves of T are exactly the trivial subgraphs of G.

A *drawing* (or *layout*) D of a hierarchical graph $H = (G,T)$ is a representation of H in the plane such that the following properties hold. Each vertex of G is represented either by a point or by a simple closed region. The region is defined by its *boundary* — a simple closed curve in the plane. Each edge of G is represented by a simple curve between the drawings of its endpoints. Each fragment of H is drawn as a simple closed region which includes all vertices, edges and subfragments of the fragment.

D is a *structural* drawing of H if for any fragment C all vertices and fragments that are not included in C are located outside the region R of C and for any edge $u=\{p, q\}$ of G intersection of representation of u with the boundary of R is nonempty only if $u \notin C$ and consists of no many than $|\{p, q\} \cap C|$ points.

A hierarchical graph is called a *planar* one if it has such a structural drawing that there are no crossing between distinct edges and the boundaries of distinct fragments.

The following properties hold.

Theorem 1. *There are nonplanar hierarchical graphs $H=(G,T)$ with planar underlying graphs G.*

Theorem 2. *There are nonplanar hierarchical graphs $H=(G,T)$ having nonstructural planar drawing.*

Theorem 3. *A simple connected hierarchical graph $H=(G,T)$ is a planar graph if and only if there is such a planar drawing D of G that for any vertex p of T all vertices and edges of G-$G(p)$ are in the outer face of the drawing of $G(p)$.*

Let V be a set of objects called simple *labels* (e.g. V can include some numbers, strings, terms and graphs). Let W be a set of *label types* of graph elements and let a label set $V(w)= V_1 \times V_2 \times \ldots \times V_s$, where $s \geq 1$ and for any i, $1 \leq i \leq s$, $V_i \subseteq V$, be associated with each $w \in W$. *A labelled hierarchical graph* is a triple (H,M,L), where H is a hierarchical graph, M is *a type function* which assigns to each element (vertex, edge and fragment) h of H its type $M(h) \in W$, and L is a *label function,* which assigns to each element h of H its label $L(h) \in V(M(h))$.

The semantics of a hierarchical graph model is provided by an equivalence relation which can be specified in different ways, e.g. it can be defined via *invariants* (i.e. properties being inherent in equivalent labelled graphs) or by means of so-called *equivalent* transformations that preserve the invariants.

Many problems in program optimization have been solved by applying a technique called *interval analysis* to the control flow graph of the program [7], [12]. A control flow graph which is susceptible to this type of analysis is called *reducible*.

Let F be a minimal set which includes G and is closed under the following property: if $C \in F$ and p is such an entry vertex of C that subgraph $\{p\}$ does not belong to F then F contains all maximum strongly connected subgraphs of graph which is obtained from C by removing of all edges which are ingoing in p. Let $H_F=(G,T)$ be such a simple hierarchical graph that *T* represents an immediate inclusion relation between fragments of the hierarchy *F*.

The following properties hold.

Theorem 4. *A control flow graph G is reducible if and only if for the simple hierarchical graph* $H_F=(G,T)$ *the set of all fragments corresponding vertices* $p \in T$ *is a hierarchy of nested single-entry strongly connected regions.*

Theorem 5. *A control flow graph G is reducible if and only if that for any* $p \in T$ *of the simple hierarchical graph* $H_F=(G,T)$ *the fragment which is obtained from fragment corresponding to p by reducing all its inner fragments from F into their entry vertices is an interval.*

3. THE HIGRES SYSTEM

A hierarchical graph supported by the Higres consists of vertices, fragments and edges which we call objects. Vertices and edges form an underlying graph. This graph can be directed or undirected. Multiple edges and loops are also allowed.

The semantics of a hierarchical graph is represented in Higres by means of object types and external modules. Each object in the graph belongs to an object type with a defined set of labels. Each label has its data type, name and several other parameters. A set of values is associated with each object according to the set of labels defined for the object type to which this object belongs. These values, along with partitioning of objects to types, represent the semantics of the graph. New object types and labels can be created by the user.

In the Higres system each fragment is represented by a rectangle. All vertices of this fragment and all subfragments are located inside this rectangle. Fragments, as well as vertices, never overlap each other. Each fragment can be closed or open. When a fragment is open, its content is visible; when it is closed, it is drawn as an empty rectangle with only label text inside it. A separate window can be opened to observe each fragment. Only content of this fragment is shown in this window, though it is possible to see this content inside windows of parent fragments if the fragment is open.

Most part of visual attributes of an object is defined by its type. This means that semantically relative objects have similar visual representation. The Higres system uses a flexible technique to visualize object labels. The user specifies a text template for each object type. This template is used to create the label text of objects of the given type by inserting labels' values of an object.

Other visualization features include the following: various shapes and styles for vertices; polyline and smooth curved edges; various styles for edge lines and arrows; the possibility to scale graph image to an arbitrary size; edge text movable along the edge line; colour selection for all graph components; external vertex text movable around the vertex; font selection for labels text; two graphical output formats; a number of options to control the graph visualization.

The comfortable and intuitive user interface was one of our main objectives in developing Higres. The system's main window contains a toolbar that provides a quick access to frequently used menu commands and object type selection for creation of new objects. The status bar displays menu and toolbar hints and other useful information on current edit operation.

The system uses two basic modes: view and edit. In the view mode it is possible only to open/close fragments and fragment windows, but the scrolling operations are extended with mouse scrolling. In the edit mode the left mouse button is used to select objects and the right mouse button displays the popup menu, in which the user can choose the operation he/she wants to perform. It is also possible to create new objects by selecting commands in this menu. The left mouse button can be also used to move vertices, fragments, labels texts and edge bends, and resize vertices and fragments. All edit operations are gathered in a single edit mode. To our opinion, it is more useful approach (especially for inexperienced users) than division into several modes. However, for adherents of the last case we provide two additional modes. Their usage is optional but in some cases they may be useful: the "creation" mode for object creation and "labels" mode for labels editing.

Other interface features include the following: almost unlimited number of undo levels; optimized screen update; automatic elimination of objects overlapping; automatic vertex size adjusting; grid with several parameters; a number of options that configure the user interface; online help available for each menu, dialog box and editor mode.

To run an algorithm in the Higres system, the user should select an external module in the dialog box. The system starts this module and opens the process window that is used to control the algorithm execution.

Higres provides the run-time animation of algorithms. It also caches samples for the repeated and backward animation. A set of parameters is defined inside a module. These parameters can be changed by the user at any execution step. The module can ask user to input strings and numbers. It can also send any textual information to the protocol that is shown in the process window.

A wide range of semantic and graph drawing algorithms can be implemented as external modules. As examples now we have modules that simulate finite automata, Petry nets and imperative program schemes. The animation feature can be used for algorithm debugging, educational purposes and exploration of iteration processes such as force methods in graph drawing.

A special C++ API that can be used to create external modules is provided. This API includes functions for graph modification and functions that provide interaction with the Higres system. It is unnecessary for programmer, who uses this API, to know the details of the internal representation of graphs and system/module communication interface. Hence, the creation of new modules in the Higres system is a rather simple work.

4. THE VISUAL GRAPH SYSTEM

Visual Graph is a tool that automatically calculates a customizable multi-aspect layout of hierarchical graph models specified in GraphML laguage [1]. This layout is then displayed, and can be interactively explored, extended and analyzed.

Visual Graph was developed to visualize and explore large graphs that present the internal structured information typically found in compilers. Visual Graph reads a textual and human-readable GraphML-specification and visualizes the hierarchical graph models specified. Its design has been optimized to handle large graphs automatically generated by compilers and other applications.

Visual Graph provides tools for analyzing graph structures. *Structural analysis* means solving advanced questions that relate to a graph structure, for instance, determining a shortest path between two nodes.

Simple possibilities to extend the functionality of Visual Graph (for example, to add a new layout, search, analysis or navigating algorithm, a new tool for processing information associated with elements of graph models and so on) are provided.

GraphML (Graph Markup Language) is a comprehensive and easy-to-use file format for graphs [1]. It consists of a language core (known as the Structual Layer) to describe structural properties of one or more graphs and a flexible extension mechanism, e.g. to add application-specific data. Its main features include support of directed, undirected, mixed multigraphs, hypergraphs, hierarchical graphs, multiple graphs in a single file, application-specific data, and references to external data.

Two extensions, adding support of meta-information for light-weight parsers (Parse Extension) and typed attribute data (Attributes Extension) are currently part of the GraphML specification.

Unlike many other file formats for graphs, GraphML does not use a custom syntax. Instead, it is defined as an XML (Extensible Markup Language) sublanguage and hence ideally suited as an exchange format for all kinds of services generating or processing graphs.

Visual Graph was designed to explore large graphs that consist of many hundreds of thousands of elements. However, the layout of large graphs may require considerable time. Thus, there are two main ways to speed up the layout algorithm: multi-aspect layout of graph and control of layout algorithms.

The first way in visualizing a large graph is aimed at avoiding computing the layout of parts of the graph that are currently not of interest. Interactive exploring of graph is based on step by step construction of so-called *multi-aspect layout* of graph being a set of drawings of some subgraphs of the graph.

For presentation of multi-aspects layout a set of windows which includes a separate window for visualization of each considered subgraph is used. At each step of the construction a layout algorithm is applied to a subgraph being interested to user at this step. To indicate the interested subgraph the user can select its elements in the active window or in the navigator.

The user can also define some condition in the filter or in the search panel. Then the condition will be used for searching of graph elements which will form the interested subgraph. The search can be performed both locally (in some part of graph, e.g. through a subgraph presented in the active window) or globally (around the entire graph). Multi-aspect drawing of graph models makes every visible part of the graph smaller, thus enabling the layout to be calculated faster and the quality of the layout to be improved.

In order to further reduce layout time, it is possible to control the layout algorithms, e.g. some layout phases can be omitted or the maximum number of iterations of some layout phases can be limited. However, this usually decreases the quality of the layout. The user can improve the layout by hand, e.g. by moving of nodes or changing of their sizes or forms.

Visual Graph offers several tools for navigating through a graph model: minimap, navigator, attribute panel, filter, search panel, notebook.

The authors are thankful to all colleagues taking part in the projects described. The work was partially supported by the Russian Foundation for Basic Research (grants N 12-07-00091, N 15-07-02029).

References

[1] U. Brandes, M.S. Marshall, and S.C. North, *Graph data format workshop report*, Lecture Notes in Computer Science, 1984, (2001), 410–418.

[2] V. A. Evstigneev, V. N. Kasyanov, *Explanatory Dictionary of Graph Theory in Computer Science and Programming*, Nauka Publ., Novosibirsk, 1999. (In Russian).

[3] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, *Graph Drawing: Algorithms for Vizualization of Graphs*, Prentice Hall,1999.

[4] Q. W. Feng, R.F. Cohen, P. Eades, *Planarity for clustered graphs*, Lecture Notes in Computer Science, 979, (1995), 213-226.

[5] M. Fröhlich, M. Werner, *Demonstration of the interactive graph visualization system daVinci*, Lecture Notes in Computer Science, 959, (1995), 266-269.

[6] D. Harel, *On visual formalism*, Comm. ACM, 31, (5), (1988), 514-530.

[7] M. S. Hecht, *Flow Analysis of Computer Programs*, Elsevier, New York, 1977.

[8] I. Herman, G. Melançon, M.S. Marshall, Graph visualization and navigation in information visualization: a survey, *IEEE Trans. on Visualization and Computer Graphics*, 6, (2000), 24-43.

[9] M. Himsolt, *The Graphlet system (system demonstration)*, Lecture Notes in Computer Science, 1190, (1997), 233-240.

[10] V. N. Kasyanov, *Hierarchical graphs and graph models: problems of visual processing*, in: V.N. Kasyanov (ed.), *Problems of Informatics Systems and Programming*, IIS, Novosibirsk, (1999), 7-32. (In Russian)

[11] V. N. Kasyanov, *Hierarchical graph models and information visualization*, in: *Proceedings of the 2012 Third World Congress on Software Engineering (WCSE 2012)*, IEEE Computer Society, 2012, 79-82.

[12] V. N. Kasyanov, V.A. Evstigneev, *Graph Theory for Programmers. Algorithms for Processing Trees*, Kluwer Academic Publ., 2000.

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

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

[15] V. N. Kasyanov, E.V. Kasyanova, *Information visualization based on graph models*. Enterprise Information Systems, 7, (2), (2013), 187-197.

[16] V. N. Kasyanov, I.A. Lisitsyn, *Hierarchical graph models and visual processing*, in: *Proceedings of Conference on Software: Theory and Practice. 16th IFIP World Computer Congress 2000*, PHEI, Beijing, 2000, 179-182.

[17] B. Madden, P. Madden, S. Powers, M. Himsolt, *Portable graph layout and editing*, Lecture Notes in Computer Science, 1027, (1996), 385-395.

[18] G. Sander, *Graph layout through the VCG tool*, Lecture Notes in Computer Science, 959, (1995),194-205.

[19] K. Sugiyama, *Graph drawing and applications. For software and knowledge engineers.* World Scientific, 2002.

[20] K. Sugiyama, K. Misue, *Visualization of structured digraphs*, IEEE Transactions on Systems, Man and Cybernetics, 21, (4), (1999), 876-892.

[1] Institute of Informatics Systems, Novosibirsk State University, Novosibirsk, Russia
E-mail address: kvn@iis.nsk.su
[2] Institute of Informatics Systems, Novosibirsk State University, Novosibirsk, Russia
E-mail address: kev@iis.nsk.su