Joint NCC & IIS Bull., Comp. Science, 12 (1999), 19–23

On support tools for visual processing of hierarchical graph models*

V.N. Kasyanov, I.A. Lisitsyn

The HIGRES system is considered as a support tool for visual processing of hierarchical graph models. HIGRES is implemented in C++, works under MS Windows 95/NT and is available on WWW at http://pco.iis.nsk.su/higres.

1. Introduction

Graph is the most common abstract structure encountered in computer science. Any system that consists of discrete states or sites (called nodes or vertices) and connections (called edges, arcs or hyperedges) between them can be modeled by a graph. Connections may also carry additional information as labels or weights related to the interpretation of the graph.

Graph models can be used in practice only along with support tools that provide visualizing, editing and processing of graphs. For this reason many graph visualization systems, graph editors and libraries of graph algorithms have been recently developed. Examples of these tools include VCG [10], daVinci [3], Graphlet [5], GLT & GET [8] and LEDA [9]. A comprehensive bibliography on graph drawing [1] cites more than 300 papers written before 1993, and the problem is now the subject of the annual conference with more than 100 participants.

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, more powerful graph formalisms have been introduced, e.g. higraphs [4], compound digraphs [11], clustered graphs [2]. One of the recent graph formalisms is the *hierarchical graphs* [6]. A hierarchical graph consists of a classical graph and its recursive partitioning into subgraphs. Hierarchical graphs can be used in many areas where strong structuring of information is needed.

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 hierarchical graphs. Another weak point is that usual graph 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.

In this paper we present the HIGRES system which is a visualization tool and an editor for attributed hierarchical graphs and a platform for execution and animation of graph algorithms. HIGRES is implemented in C++ and works under MS Windows 95/NT.

The rest of this paper is organized as follows. Section 2 outlines the hierarchical graph formalism. A hierarchical graph supported by the HIGRES system is considered in Section 3. Sections 4 and 5 describe the visualization possibilities and the user interface of the system. Section 6 is a conclusion.

2. Hierarchical graph models

This section contains a brief description of the hierarchical graph formalism; the reader is referred to [6] for further details.

*Supported by the Russian Foundation for Basic Research under Grant 98–01–00748.

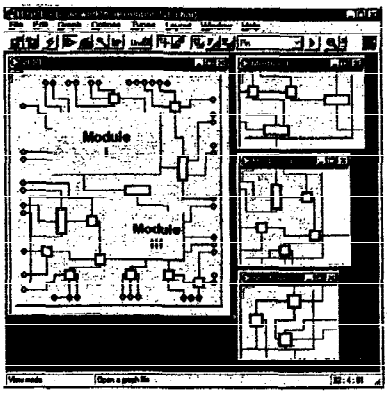

Figure 1. A VLSI model represented by a hierarchical graph. The model consists of three modules folded into fragments. These fragments are displayed in separate windows with different scales. One fragment (Module II) is open. Its content is also displayed inside the VLSI window. Two other fragments are closed and shown in this window as covered rectangles.

Let G be a graph of some type, e.g., G can be an undirected graph, a digraph or a hypergraph. The vertices and edges of G are called the elements of the graph. A graph C of the same type is called a *fragment* (or a *subgraph*) of G, denoted by $C \subseteq G$, if C includes only the elements of G.

F is called *a hierarchy of nested fragments* of the graph G, if F is a set of fragments of G such that $G \in F$ and for any C_1 and C_2 from F one of the following properties holds: $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, $C_1 \cap C_2 = \emptyset$. The fragment G is the *main* fragment of the hierarchy F. A fragment $C \in F$ is a *simple* fragment, if F includes no subfragments of C. For any two $C_1, C_2 \in F$ the fragment C_1 is called an *immediate* subfragment of C_2 (or C_2 *immediately* includes C_1) if $C_1 \subseteq C_2$ and there is no fragment $C_3 \in F \setminus \{C_1, C_2\}$ such that $C_1 \subseteq C_3 \subseteq C_2$.

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. The nodes of T are all fragments of the hierarchy F. The graph G is called the *underlying graph* of H. The tree T is called the *inclusion tree* of H.

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

A hierarchical graph model is defined as a set of labelled hierarchical graphs together with an equivalence relation between them.

Let V be a set of objects called *labels*. For example, V can include some subset of numbers, strings, terms and graphs. Let W be a set of types of graph elements and let a label set $V(w) = V_{i_1} \times V_{i_2} \times \ldots \times V_{i_s}$, where $V_{i_j} \subseteq V$ for any j, 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. For example, the equivalence relation 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.

3. Graph models in HIGRES

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. Each fragment is an induced subgraph and is associated with a set of vertices, so that if the vertex sets of two fragments intersect, then one of these fragments is a subfragment of another one.

The semantics of a hierarchical graph is represented in HIGRES by means of object types. Each object in the graph belongs to an object type. A set of labels is defined for each object type. 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 into types, represent the semantics of the graph. New object types and labels can be created by the user.

4. Visualization

In HIGRES 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 only with label text inside it. A separate window can be opened to observe each fragment. Only the content of this fragment is shown in this window, though it is possible to see this content inside the 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. HIGRES 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;
- color selection for all graph components;
- the possibility to scale graph image to an arbitrary size;
- edge text movable along the edge line;
- external vertex text movable around the vertex;
- font selection for the label text;
- two graphical output formats;
- a number of options to control the graph visualization.

Now HIGRES uses three graph drawing algorithms for automatic graph allocation. The first algorithm is a force method which is very close to the original algorithm from [7]. The second is our improvement of the first one. The third method allocates rooted trees on layers.

5. The user interface

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.

All edit operations are gathered in a single edit mode. To our opinion, it is a more useful approach (especially for unexperienced 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 label editing.

Other interface features include the following:

- almost unlimited number of undo levels;

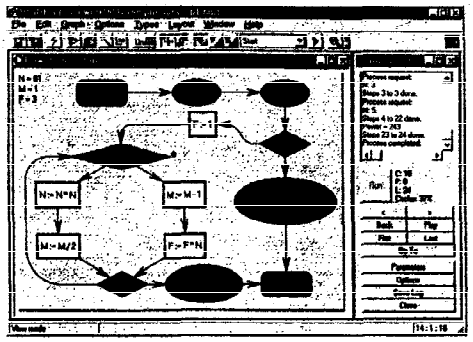

Figure 2. Program scheme interpreted by external module. Asterisk shows which operator has control. Current values of variables are listed in the top left corner of the fragment rectangle. This screenshot was made after the completion of the process and rewinding several samples back.

- optimized screen update;
- automatic elimination of object overlapping;
- automatic vertex size adjusting;
- a grid with several parameters;
- a number of options that configure the user interface;
- online help available for each menu, dialog box and editor mode.

6. Algorithm animation

To run an algorithm, the user should select an external module in the dialog box. The system starts this module and opens the process window which 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 the user to input strings and numbers. It can also send any textual information to the protocol which is shown in the process window.

A wide range of semantical and graph drawing algorithms can be implemented as external modules. For example, we have modules which simulate finite automata, Petri nets and imperative program schemes (see Fig. 2).

The animation feature can be used for algorithm debugging, educational purposes and exploration of iteration processes such as force methods in graph drawing.

We provide a special C++ API that can be used to create external modules. This API includes functions for graph modification and functions that provide interaction with the 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 becomes a rather simple work.

7. Conclusion

We presented the HIGRES system which is a visualization tool and an editor for attributed hierarchical graphs and a platform for execution and animation of graph algorithms. The release of the system is available and has applications to education, research, and practice. However, it can be extended and improved in some aspects which are the subjects of our future work. Our nearest plans concern the integration of additional graph drawing algorithms, including methods specially designed for hier-

archical graphs. We also have ideas on further improvements of the system interface and visualization features.

The system is available on WWW at http://pco.iis.nsk.su/higres.

References

[1] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Comput. Geom. Theory Appl., **4**, 1994, 235–282.

[2] Q.W. Feng, R.F. Cohen, P. Eades, *Planarity for clustered graphs*, Lect. Notes Comput. Sci., **979**, 1995, 213–226.

[3] M. Fröhlich, M. Werner, *Demonstration of the interactive graph visualization system daVinci*, Lect. Notes Comput. Sci., **959**, 1995, 266–269.

[4] D. Harel, *On visual formalism*, Commun. ACM, **31**, No 5, 1988, 514–530.

[5] M. Himsolt, *The Graphlet system (system demonstration)*, Lect. Notes Comput. Sci., **1190**, 1997, 233–240.

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

[7] P.Eades, *A Heuristic For Graph Drawing*, Congressus Numerantium, **42**, 1984, 149–160.

[8] B. Madden, P. Madden, S. Powers, M. Himsolt, *Portable graph layout and editing*, Lect. Notes Comput. Sci., **1027**, 1996, 385–395.

[9] K. Mehlhorn, S. Näher, *LEDA: a platform for combinatorial and geometric computing*, Communs. ACM, **38**, No 1, 1995, 96–102.

[10] G. Sander, *Graph layout through the VCG tool*, Lect. Notes Comput. Sci., **959**, 1995, 194–205.

[11] K. Sugiyama, K. Misue, *Visualization of structured digraphs*, IEEE Trans. on Systems, Man and Cybernatics, **21**, No 4, 1991, 876–892.