

On visualization in computer-aided teaching of mathematics

Victor N. Kasyanov, Ivan A. Lisitsyn
Institute of Informatics Systems,
Novosibirsk, 630090, Russia

1 Introduction

The visualization of information, especially information of complex and intricate nature, is a key component of support tools for computer-aided teaching of mathematics. The information that interests us here is nonqualitative, but rather, of a structural, set-theoretical, and relational nature.

Graphs are commonly used to model information of this kind and current systems for computer-aided teaching include graph drawing and visual processing functions.

In this paper, we consider the Higes system aimed at visual processing of hierarchical graph models. A hierarchical graph ¹ consists of a graph and its recursive partition into subgraphs called fragments. Higes can be used as a visualization tool and an editor for attributed hierarchical graphs and a platform for execution and animation of graph algorithms. The system is implemented in C++ and works under Windows 95/NT.

The Higes system is available on WWW at <http://pco.iis.nsk.su/higes>.

2 Hierarchical graphs and graph models

Let G be an undirected graph, a directed graph or a hypergraph. A graph C is called a *fragment* of G if $C \subseteq G$. A set of fragments F is a *fragment*

¹Kasyanov V.N. Hierarchical graphs and visual processing, In: Intern. Congress of Mathematicians (ICM98), Berlin, 1998.

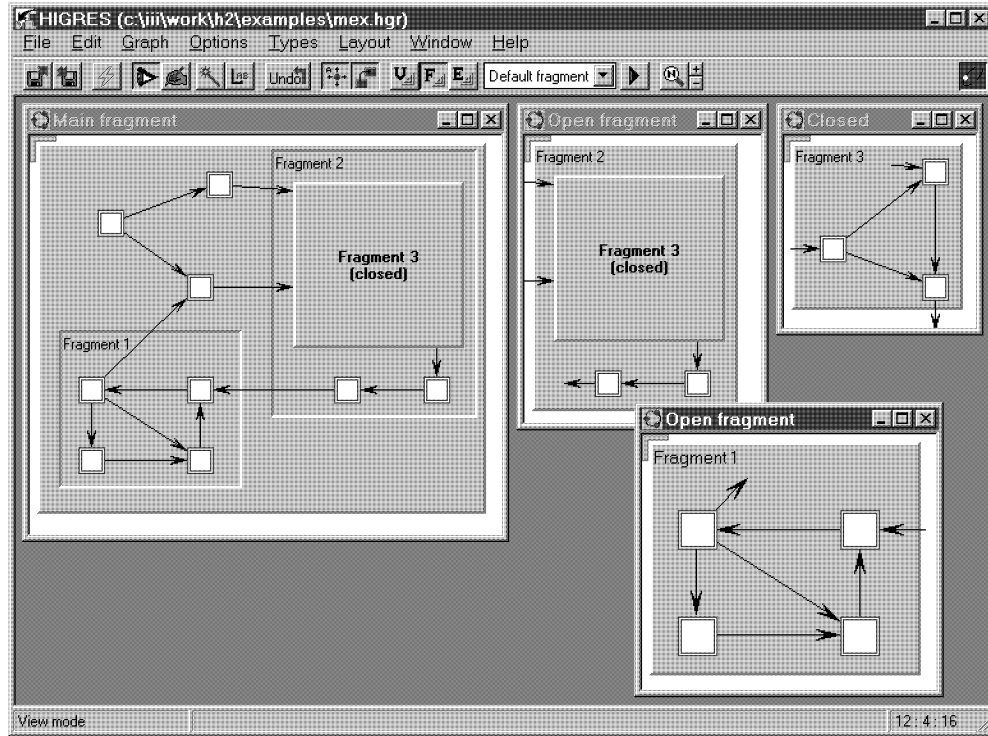


Figure 1: Higres working with a hierarchical graph.

hierarchy if $G \in F$ and $C_1 \cap C_2 = \emptyset$, $C_1 \subset C_2$ or $C_2 \subset C_1$ for any $C_1, C_2 \in F$. The graph $G \in F$ is called the *main fragment* of F .

A *hierarchical graph* H is a pair (G, T) where T is a rooted tree which represents inclusion relation between elements of a fragment hierarchy F . It is said that H consists of the *underlying graph* G and the *inclusion tree* T .

In the Higres system any representation of H in the plane consists of vertices, fragments and edges, which we will call objects. Vertices and edges form the underlying graph G . A fragment from F is represented by rectangle (See Fig.1). All vertices that belong to 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 labels text inside it.

A separate window can be opened to observe each fragment. In this window only content of this fragment is shown, though it is possible to see this content inside windows of parent fragments, if the fragment is open.

The semantics of H is represented in Higes 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 to types represent the semantics of the graph. New object types and labels can be created by the user.

3 Visualization

Most part of visual attributes of an object is defined by its type. This means that semantically relative objects have similar visual representation.

Higes uses flexible technique to visualize object labels. The user specifies a text template for each object type. This template is used to create labels text of objects of the given type by inserting labels values of an object.

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

4 User interface

Comfortable and intuitive user interface was one of our main objectives in developing Higes.

System uses two basic modes: view and edit. In the view mode it is only possible 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

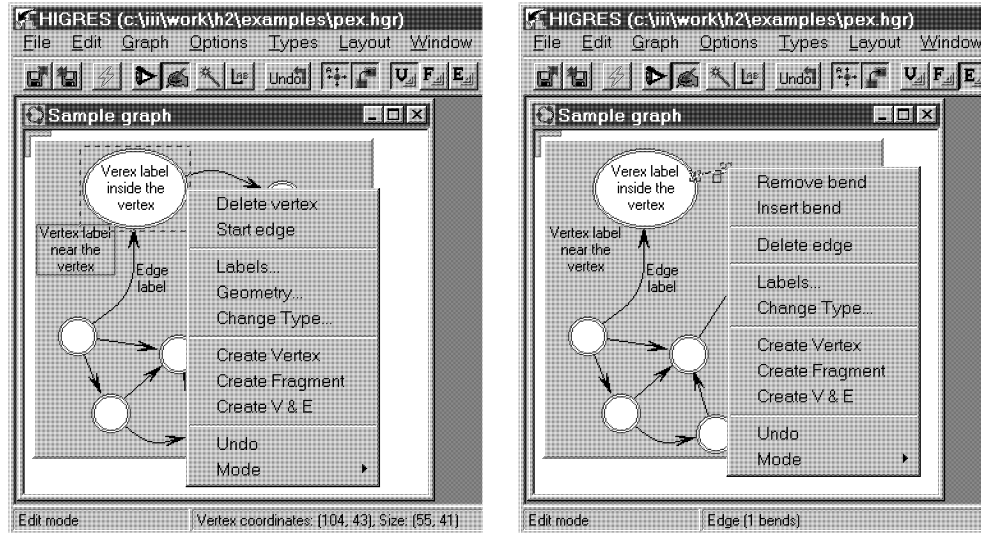


Figure 2: Popup menus in the edit mode: a). menu for a selected vertex; b). menu for a selected edge.

objects by selecting commands in this menu (See Fig.2).

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 user interface; online help available for each menu, dialog box and editor mode.

5 Algorithms 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 run-time animation of algorithms. It also caches samples for repeated and backward animation.

A set of parameters can be defined inside a module. These parameters can be changed by the user at any execution step. The module can input text and numbers from the user. It can also send any textual information to

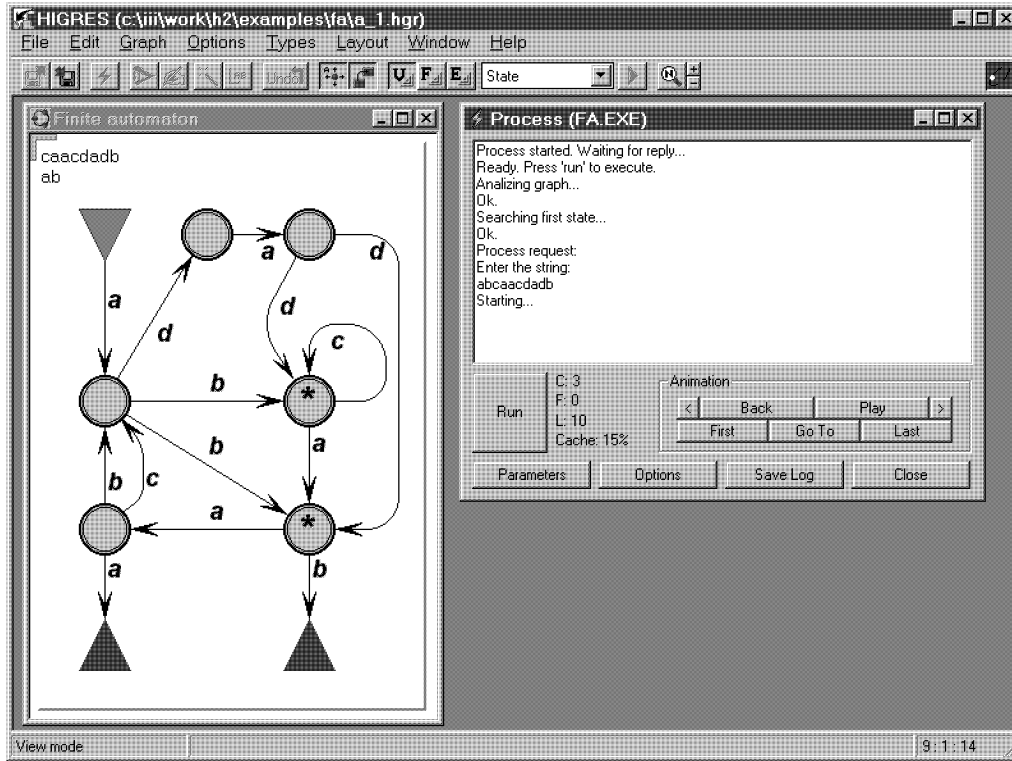


Figure 3: Visual simulation of a given finite-state automaton.

the protocol, which is shown in the process window.

A wide range of semantic and graph drawing algorithms can be implemented as external modules. Screen shot in Fig. 3 shows some example of an external module which emulates the work of a given finite-state automaton. Asterisks show active states. Two lines in the top left corner are the tail of the string that was input from the user and the part of this string that is already processed.

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 use this API, to know the details of the internal representation of graphs. Hence, the creation of new modules is a rather simple work.