DIDAGRAPH : Software for Teaching Graph Theory Algorithms

V. Dagdilelis
Dept. of Applied Informatics, University of Macedonia
156 Egnatia str., P.O.Box 1591
54006 Thessaloniki, Greece
++3031-891890
dagdil@macedonia.uom.gr

M. Satratzemi
Dept. of Applied Informatics, University of Macedonia
156 Egnatia str., P.O.Box 1591
54006 Thessaloniki, Greece
++3031-891897
maya@macedonia.uom.gr

1. ABSTRACT

Graph theory and in particular its algorithmic aspect is known as being a difficult topic in Computer Science. In this paper we propose the software DIDAGRAPH, which we are in the process of developing, as a support for teaching graph algorithms. The environment of DIDAGRAPH offers the possibility of visualisation and experimentation so as to overcome didactic problems, i.e. the intermediate stages of an algorithm, their implementation in a programming language etc. In DIDAGRAPH we are developing two different frameworks to explore an algorithm: one to explore in detail predetermined algorithms and a second to develop arbitrary algorithms expressed with command language in a visual environment.

1.1 Keywords

Principles of design educational software, computer support for teaching graph theory, graph visualisation, educational software.

2. INTRODUCTION

Graph Theory is a compulsory course of our undergraduate curriculum. The course is usually organised with conventional lectures where we present notions of graph theory and their algorithms. Graph theory, and in particular its algorithmic aspect, is known as being a difficult topic in Computer Science, since students have problems in understanding its algorithms. These problems are due perhaps to the fact that:

- taught algorithms are intrinsically difficult to understand
- we use blackboard or transparencies (which are static) for the description of systems that are dynamic

Of course, graphs have an advantage, since most of their notions and problems can be expressed with the use of drawings. In other words graphs can be based on the visual representation of their concepts and relations.

Our hypothesis is that the students' difficulties with graphs have some special characteristics that can be categorised as follows:

- Difficulties related to the recognition of the details of an algorithm. Students can perceive how an algorithm works in general, i.e. in terms of a generic description in a kind of «natural language», or even to follow a schematic representation of its application on a graph, but they have problems in understanding its details. For example, students cannot understand why Dijkstra's algorithm works only for graphs with positive weights nor the reasons for the existence of additional conditions of its validation.
- Difficulties in fully understanding the meaning of intermediate stages and/or in interpreting the intermediate results. For example, in the labelling algorithms, the role of intermediate labels seems not to be clear.
- The fact that common programming languages are not suitably designed for graph algorithm implementation, also constitutes a problem for the students. Data structures that we use, seem not to have a direct relation to the algorithm and the details that concern their implementation are disproportionate to the simplicity and the shortness of the expressed algorithm in its «natural» pseudo-language. For example, the determination of the successors of a vertex in an oriented graph, is a trivial task when done by hand, but a complex one when expressed in a programming language.

The above points have a general character, i.e. they are not directly connected with the inherent complexity of an algorithm (viz. the Hungarian algorithm)

The possibility of graphical representation of notions and problems combined with the contemporary interface technology, led to the invention of visual systems, used for didactic purposes, to help students overcome the difficulties described above, for research or both. Systems of this type are Cabri-Graph [6], LEDA [7], LINK [2]. In this paper we describe the planing of a similar system, DIDAGRAPH, that has a clear didactic orientation. This software is currently under development and the entire project is financially supported by the Greek Ministry of Education.

3. DESIGN OF EDUCATIONAL SOFTWARE

With the belief that educational software should combine the progress of technology with progress of didactic knowledge, we present some design principles of modern educational software:

- *Tool Logic* : The educational software is developed in such a way so as to be an effective tool for the achievement of a particular didactic aim [9].
- *Concentration on a specific goal*: Software should allow the user to concentrate on the subject being studied and avoid a series of time-consuming, tiring and unavoidable operations when paper and pencil are used.
- Errors sometimes express a *misconception*, knowledge that is wrong, insufficient or in general inappropriate. Educational software should not impose but allow free expression of the user's conceptions even though they might be wrong. When a user expresses a misconception the software should lead him to a logical contradiction or to explain why the system does not respond.
- *Educational software should develop the logic of a microworld*: an environment where the user can define objects, the relations between them, their functions and their properties. In these microworlds, objects have a dynamic existence, i.e. the possibility of changing them (their geometric shape, equation, image, sound, text, graph) when the parameters on which they depend also change. In addition, the microworld should be able to adapt to some degree to the user's needs.
- *A microworld should incorporate the possibility of transforming functions, notions and relations to objects.* We think that this possibility constitutes the most important and revolutionary difference between the educational software of the past generation and the present, modern one. The user's commands are not written statements which specify the next stage of the system but almost «natural» operations on the depicted objects. An example is the possibility to «take» a hyperbole by using the mouse and move it around the screen, «bend» it as if it was made of wire and simultaneously observe the changes on the analytical or algebraic expression of the curve (software that our department is developing).
- *A microworld can embody the possibility of the object's direct manipulation* [8]. The user can directly manipulate the objects and relations of microworld rather than indirectly, with the use of statements etc.
- *In a microworld the different contexts of the expression of related notions should be equivalent*: for instance, a function can be expressed in an analytical manner - with an algebraic expression - with a matrix or with a graphical representation. The equivalence of interaction between all these different contexts, should be a possibility of microworlds.

The above list consists of a large number of characteristics that are desired in contemporary educational software. It is obvious that educational software can rarely embody all the above described possibilities and in particular to the same degree. It can be said that all these possibilities can be embodied to different degrees, according to the object treated by an algorithm ([1], [3], [5]). Many of these possibilities constitute significant algorithmic and programming challenges in any circumstances, and if they are to be incorporated, the following problems may be encountered:

- *design problems* (how a relation can be transformed to an object or an icon ?)
- *problems of an algorithmic nature* (such as, if in a geometric software a bisector is constructed in two different ways, how can the system decide that the two bisectors are one ?)
- *programming problems* (such as, which is the best way to implement an algorithm ?)

During the development of DIDAGRPH, we have faced a series of problems, but our goal is to embody as many of the above mentioned possibilities.

4. DESCRIPTION OF THE SYSTEM

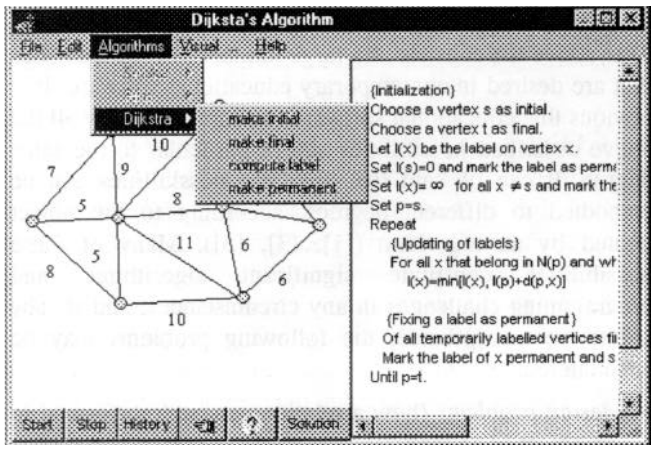
DIDAGRAPH is an environment for the visualisation and exploration of graph algorithms, so as to overcome didactic problems that have previously been described. Its basic design has not yet been finalised, since we wish to test it in a didactic situation and its fundamental structure will be changed if necessary. This is an essential part of its development, since the orientation of its construction comes from the didactic problems and not from the software itself. In DIDAGRAPH we are developing two different frameworks of an algorithm's exploration:

- one to explore in detail predetermined algorithms; and

- one to develop arbitrary algorithms expressed with a command language in a visual environment.

The examples which follow describe the didactic possibilities of DIDAGRAPH. As an example of the first framework we use Dijkstra's algorithm and for the second Kruskal's algorithm.

Following, is an illustration of a typical intermediate stage of Dijkstra's algorithm. The graphics environment is almost self explanatory. The presentation window (Figure 1) consists of four major parts: the menu area, the navigation panel, one area with graph, and the other where the algorithm is displayed in a pseudo- language. The graph can be produced in a number of different ways, such as determining the number of vertices and edges or with direct manipulation or from a file.



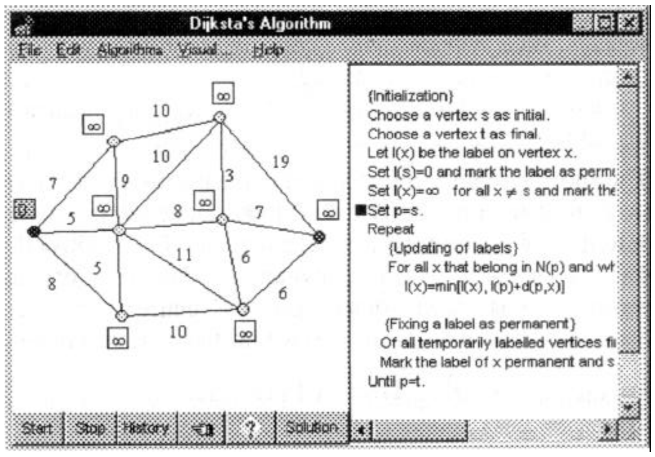
```
{Initialization}
Choose a vertex s as initial.
Choose a vertex t as final.
Let l(x) be the label on vertex x.
Set l(s)=0 and mark the label as perm
Set l(x)= ∞  for all x ≠ s and mark the
Set p=s.
Repeat
    {Updating of labels}
    For all x that belong in N(p) and wh
        l(x)=min[l(x), l(p)+d(p,x)]

    {Fixing a label as permanent}
   Of all temporarily labelled vertices fi
   Mark the label of x permanent and s
Until p=t.
```

Figure 1

A Student can choose an algorithm from the menu option «Algorithms» and then he has two possibilities: either to see the algorithm's execution by using the button «Solution», or to apply the selected algorithm by himself, describing the next step of the algorithm in manipulating directly the elements of the graph (i.e. vertices, edges etc.). The didactic function of the option «Solution» is obvious. The system provides the solution for the proposed graph and also permits the student to see a large number of executions and/or on complex graphs, in order to fully understand its operation. When a student chooses to describe the next stage of the algorithm himself and comes to an impasse, the option «Solution» can help him.

For each algorithm we have defined operations which the student can choose and then apply to every element of the graph, i.e. in Dijkstra's algorithm the student can choose between the four operations shown in figure 1.

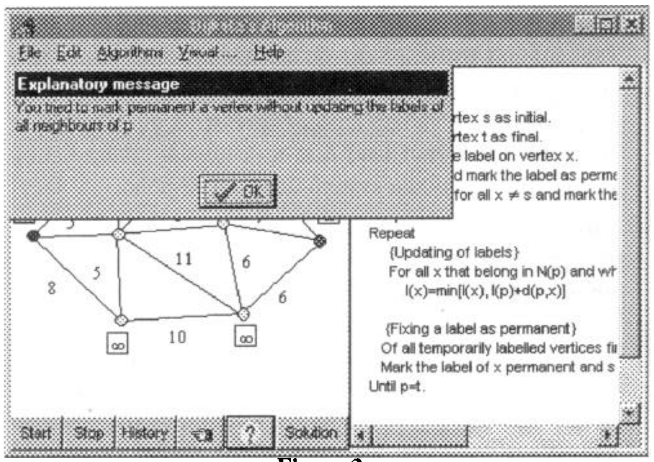
Figure 2 depicts the graph after the completion of the initial step. The student chose the initial and final vertex, set the initial values of labels for all vertices and marked the label of the initial vertex permanent. He described this step by first choosing the vertex of the graph and then applying one of the appropriate commands from the submenu of the selected algorithm.


Dijksta's Algorithm
File Edit Algorithms Visual Help
Start Stop History ? Solution

```
{Initialization}
Choose a vertex s as initial.
Choose a vertex t as final.
Let l(x) be the label on vertex x.
Set l(s)=0 and mark the label as perm
Set l(x)=∞  for all x ≠ s and mark the
■Set p=s.
Repeat
    {Updating of labels}
    For all x that belong in N(p) and wh
        l(x)=min[l(x), l(p)+d(p,x)]

    {Fixing a label as permanent}
   Of all temporarily labelled vertices fi
   Mark the label of x permanent and s
Until p=t.
```

Figure 2

If his next command is not appropriate, the system does not respond. However, whether it is appropriate or not, by pressing the button «Question mark» an explanatory message appears. In figure 3, we can see an explanatory message, when student chose to mark the label of a vertex x permanent, while he didn't compute the labels for all neighbours of p (p is the last vertex marked permanent). In the case of a correct answer the system will provide an explanation as to why the step was the right one, i.e. a kind of descriptive, informal proof which will validate for the student the correctness of the step.


File Edit Algorithms Visual Help
Explanatory message
OK
Start Stop History ? Solution

You tried to mark permanent a vertex without updating the labels of all neighbours of p

```
rtex s as initial.
rtex t as final.
e label on vertex x.
d mark the label as perm
for all x ≠ s and mark the
Repeat
    {Updating of labels}
    For all x that belong in N(p) and wh
        l(x)=min[l(x), l(p)+d(p,x)]

    {Fixing a label as permanent}
   Of all temporarily labelled vertices fi
   Mark the label of x permanent and s
Until p=t.
```

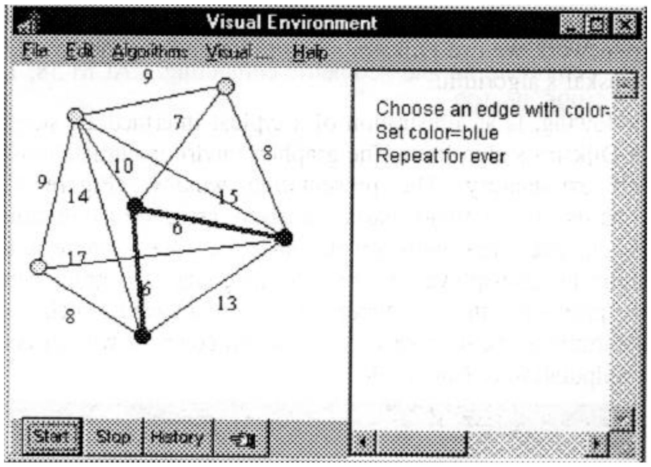
Figure 3

The button «Back stage» causes the appearance of the previous stage. Every command of the user is recorded and so the user can see his operations as a sequence of snapshots by using button «History». When the user completes a command, the system moves an index in front of the pseudo- code.

The application of predefined algorithms, has the advantage of permitting the incorporation of the didactic knowledge about particular points or the instructor's experience into the software and to organise the appropriate reaction. For example, when a student experiments with Dijkstra's algorithm, it is possible to choose a graph with negative weights. What happens in this case? In accordance with our principles this graph is not rejected. At the end of the process, however, the system demonstrates that Dijkstra's algorithm does not produce a correct solution.

The authors believe that this environment provides genuine help in order to recognise the intermediate stages of an algorithm. The software gives us the possibility to illustrate these stages and it also provides a description of the algorithm in a graph oriented pseudo-language. The user can indicate the intermediate stages without being obliged to effectively implement the algorithm. After the completion of the algorithm the instructor (or the system) can propose questions related to some intermediate stages of the algorithm by using the «History» option. At this point, we wish to mention that the possibility of a more detailed experiment with an algorithm will be incorporated in a future version of DIDAGRAPH, i.e. the student will be able to compute the label of a vertex and not only the system himself.

As mentioned in the introduction, the implementation of an algorithm constitutes problems especially for students that are novices and it is also a time-consuming process for researchers. The test of correctness of new algorithms and the verification of conjectures on graphs are complex tasks due to the implementation details. A graph editor, like the one presented above, is a help but not sufficient, as it does not allow the expression of new algorithms by the user. We can approach this in two ways: the first one is the use of a graph-oriented language, as the version of Scheme, that is used in LINK[1], the second one is to create an environment in which the user could express graph algorithms by using a graph-oriented command language (menu driven), which is the one that we develop. Each step of the algorithm is expressed by choosing the appropriate command from a menu. The set of edges, vertices etc. are distinguished by different morphological characteristics (colour, shape, size etc.). This language at present covers a subset of graph algorithms which is constantly expanding. Obviously, the user can directly express algorithms already known or invented by him and obtain a visual representation of the results. Our system can also memorise the solution procedure and apply it later to a given graph. Thus the user can verify his algorithm with an unlimited variety of graphs and in this way has valuable assistance in the exploration of the algorithm. Figure 4 gives an example of how the student can express the algorithm for the determination of a minimum spanning tree of the graph shown in figure 4 (Kruskal's algorithm) by using the visual environment described above.



Choose an edge with color:
Set color=blue
Repeat for ever

Figure 4

5. CONCLUSIONS

The principles of the design of DIDAGRAPH have been dictated by a didactic rationale, i.e., its design was conducted by well known didactic needs. DIDAGRAPH's initial design will be either accepted or ejected depending on the results of how effective it actually is in teaching. The second part of DIDAGRAPH is still in its primary stages (especially the visual programming and program by example parts) and poses some interesting problems for the elaboration of a specific interface, well suited to deal with a wide range of cases. For example we are designing a graph oriented calculator to express algebraic relations that are necessary to describe some algorithms. Other similar problems are under consideration.

6. REFERENCES

[1] Baulac Y., Bellemain F., Laborde J.M.: (1988), Cabri-géomètre, un logiciel d'aide à l'apprentissage de la géomètrie, Cedic-Nathan,, Paris.

[2] Berry J., Dean N., Fasel P., Goldberg M., Johnson E., MacCuish J., Shannon G and Skiena S., LINK: A combinatorics and Graph Theory workbench for applications and research. Tech. Rep. 95-15, DIMACS, Piscatway, NJ, 1995.

[3] Brousseau G. [1986] Fondements et méthodes de la didactique des mathématiques, Recherches en didactique des mathématiques, vol. 7.2, s. 33-115.

[4] Gallesio E., The Stk reference manual. Tech. Rep. RT 95-31a, I3S CNRS, Université de Nice - Sophia Antipolis, France, 1995.

[5] Jackiw N.: (1989), Geometer's sketchpad, collège de Swarthmore. USA.

[1] LINK's interface is written in STk, Erik Gallesio Scheme interface to John Ousterhout's Tk graphics package [4], [10].

[6] Laborde C., Capponi B.:(1994), Cabri-géomètre constituant d'un milieu pour l'apprentissage de la notion de figure géométrique, *Didactique et intelligence artificielle*, 14 (1), p. 165-210

[7] Melhorn K., Nahger S., LEDA: A platform for combinatorial and geometric computing. CACM 38, 1, 1995, 96-102.

[8] Nanard J. (1990), La manipulation directe en interface homme-machine, Thèse, Université des sciences et techniques du Languedoc, Montpellier.

[9] NATO ASI Series F-78 (1991), Integrating Advanced technology into Technology Education, M. Hacherm A. Gordon, M. de Vries (Eds), Springer-Verlag.

[10] Ousterhout J., Tcl and the Tk Toolkit, Addison-Wesley, 1994.