MAVIS: A Multi-Level Algorithm Visualization System within a Collaborative Distance Learning Environment

Igal Koifman Ilan Shimshoni
ilans@ie.technion.ac.il
Dept. of Indusrial Engineering and Management
Technion – Israel Inst. of Technology

Ayellet Tal
ayellet@ee.technion.ac.il
Dept. of Electrical Engineering
Technion – Israel Inst. of Technology

Abstract

We present in this paper a new model for an algorithm visualization system. Our model views the visualization system as an integral part of a broader distance learning environment. As such, it supports the heterogeneity of the Internet the visualization is expected to run on and the diversity of the expected users. It does so by defining a few manners for handling multi-level visualizations. First, a visualization can run in various abstraction levels of the algorithm, depending on the familiarity of the students with the studied materials. Second, a visualization can use various levels of graphics, depending on the capabilities of the client machines. Third, the messages sent between the machines can be of various levels, depending on the communication loads. Another important aspect of a distance learning environment, which is supported by our model, is to facilitate collaboration and data sharing between the students and the instructor and between the students themselves. This paper also presents a system, MAVIS, that realizes the model, and demonstrates its use on case studies.

Keywords: Algorithm visualization, algorithm animation, distance learning, collaboration.

1 Introduction

Algorithm visualization refers to the use of graphics and motion to explain algorithmic ideas and data structures [3, 4, 5, 6, 7, 11, 15, 16]. It has a great potential to assist in the design of algorithms, in the debug process, and for teaching algorithms to students and colleagues.

Distance learning refers to a learning process in which the instructor resides in a different geographical location than the students, thus communication is needed in order to convey the materials [1, 2, 10, 14]. The advent of the Internet along with the development of multimedia capabilities made the two a convenient platform for developing comfortable and affordable distance learning tools.

Within a distance learning environment, a virtual lecture is performed once the user is surfing to a specific web site. Just like any real classroom, a virtual classroom consists of an instructor and a group of students. The users should be able to interact with each other, collaborate and share information. The only difference from a regular classroom is that each individual can be at a different location when the class takes place.

Most algorithm visualization systems are not designed to be used within a general distance learning environment. In this paper we present a novel conceptual model for algorithm visualization within a virtual classroom framework. We also present our system, MAVIS (Multi-level Algorithm VIsualization System), that realizes this model.

Running a virtual lecture and a visualization over the Internet means that no assumption can be made regarding the processing capabilities of the clients. In addition, communication loads might vary greatly on various links. As a result, delays might be caused and synchronization becomes a difficult task. Thus, a major emphasis of our system is to support the expected heterogeneity in a dynamic manner.

To do it, rather than accompanying an algorithm with one specific visualization, our model supports *multi-level* visualizations. That is to say, rather than producing a single visualization for a given algorithmic event, the visualization system should be capable of producing a whole spectrum of visualizations. There are three cases where this capability is desirable.

First, using different visualizations can be beneficial for teaching purposes. The instructor (or even the student) is interested in explaining the algorithm in varying details depending on the familiarity with this algorithm. For instance, upon the first encounter with an algorithm, only the high-level events of the

algorithm are visualized, while once the student gets more familiar with the algorithm, more detailed events can be visualized.

Second, the diversity of the Internet means that every machine has different capabilities for supporting visualizations. Simplified visualizations are displayed on low-end machines while elaborate visualizations are displayed on high-end machines. Without this latter capability, weaker machines could not participate in a virtual classroom and keep synchronized with the instructor.

Third, the communication loads on the various links might vary greatly. A system should be able to monitor the links and send data suitable for the specific communication load. For instance, machines having overloaded links can run sub-functions of the algorithm locally and save on messages regarding algorithmic events happening in these functions.

Another major concern in every distance learning environment is how to facilitate collaboration between the participants. A couple of tools that assist information sharing are supported by our model. With a *guided Internet tour* tool, the instructor can change the course of the class by surfing to various web sites, and make the students follow the surf. With a *chat* mechanism the instructor and the students can exchange messages in real time.

We built a system, MAVIS, which realizes the model. MAVIS is an object oriented system, implemented in Java, and designed such that the algorithms, its objects and its graphical support can be easily extended by unexperienced users. MAVIS is structured as an Internet client-server architecture.

The rest of the paper is organized as follows. In Section 2 we present our conceptual model. In Section 3 we describe our system. In Section 4 we illustrate the process of producing an algorithm visualization with our system. We conclude in Section 5.

2 Conceptual Model

In this section we describe our conceptual model for an algorithm visualization system within a general interactive and collaborative distance learning environment. We consider three types of users: the instructor, the students and groups of students.

After choosing an algorithm to teach, the instructor can prepare various input examples in order to demonstrate the actions of the algorithm. The instructor can choose to display the algorithm in various levels of detail, for instance, by teaching it top-down and visualize each level when appropriate. As will be explained later, this does not require more than one implementation of the algorithm and of the visualization.

It was shown in [18] that algorithm visualization assists learning when the students have active tasks. Moreover, a comprehensive understanding of the algorithm is facilitated when the students themselves implements the visualization [12, 13, 17]. Thus, a major concern of our model is to let the students be active (and thus the system – interactive).

The student can use the system in three manners. First, the student can participate in an online virtual classroom. In this case, the student remains active by being able to ask the instructor questions and get feedback, and by being able to collaborate with other students in the virtual classroom. Second, the student can run the visualization independently in a standalone machine. This way, the student can explore the algorithm by changing the inputs and checking how the visualization of the algorithm changes accordingly. Finally, through a special comfortable interface, the student can implement an algorithm and create a visualization illustrating it.

In our model, students can form study groups whose members can collaborate, share information, and work together on their assignments. Student collaboration is important since it has a potential to expose the students to different ways of thinking, to mutual assistance in solving problems, and to greater joy in studying.

Our model enables the definition and management of several virtual classrooms that run concurrently. Managing the classrooms is done by forming data channels, as illustrated in Figure 1. Data channels for a specific classroom can handle various types of messages. A student who registers to a specific classroom, automatically gets all the information passing in the relevant channels. Defining the broadcasting mechanism in this manner has the benefit of flexibility. New types of messages can be easily added. Moreover, external systems can be connected by defining special channels for them, while the system automatically handles the messages.

Our algorithm visualization model is based on the classical model of *interesting events* [3], indicating algorithm events that are of interest when the program runs. An interesting event can be either *simple* or *compound*. A simple event represents a change made to a single object. A compound event is composed of several simple events, which are animated concurrently.

The main deviation of our model from previous models is that we aim at supporting the diversity of students and the heterogeneity of the Internet, by providing *multi-level visualizations*. That is to say, an

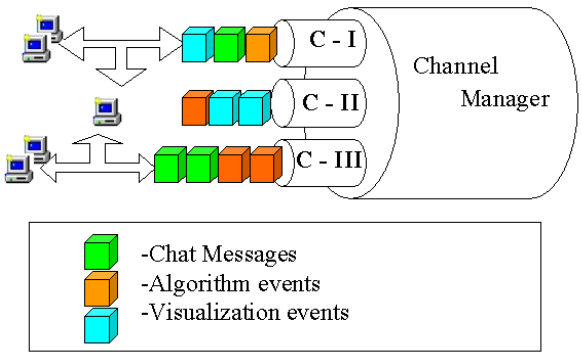

Figure 1: Data channel manager

algorithm can be accompanied by a visualization selected from a set of possible visualizations. Less detailed visualizations present only the higher level logical operations of the algorithm, while more elaborate visualizations present additional interesting events, and each such event is visualized in more detail. Another aspect of diversity in the types of visualizations that a system can generate relates to the computer graphics being used. A visualization of a specific interesting event can represent each algorithmic object by various types of graphical objects (e.g., a circle, a square or a sphere), can use various motions, and can vary the number of frames utilized. There are three factors that influence the choice of a specific visualization over all other possible visualizations.

First, as every algorithm can be described in several levels, it can also be visualized according to these levels. The instructor can dynamically choose the most suitable level for the classroom at a certain time. For instance, the explanation starts with a high level description of the algorithm, and then discusses each step in more detail. Moreover, every student can also pick the level of visualization that fits his understanding. Upon the first encounter with an algorithm, the student can choose to visualize only the major events of the algorithm, while once the student gets more familiar with the algorithm, more details are sought, and thus additional interesting events are displayed.

For instance, given a scene consisting of points and edges, the *visibility graph* consists of line segments between points which can see each other (i.e., do not intersect any edge). The most naive algorithm for constructing a visibility graph checks for every pair of vertices whether the segment connecting them intersects any given edge of the scene. Thus, in the detailed level visualization, each such intersection test is visualized (i.e., visualizing $O(n^3)$ tests). A less detailed visualization consists of displaying every candidate segment in turn (i.e., visualizing $O(n^2)$ events), without showing the intersection tests. An even less detailed visualization consists of displaying all the visibility segments emanating from a given vertex (i.e., $O(n)$ events). Finally, in the least detailed visualization only the resulting visibility graph is displayed (i.e., $O(1)$ events).

Second, in a diverse environment as the Internet, every end-point machine has different capabilities in general and different graphical capabilities in particular. Each user can set the quality of the graphical display according to the capabilities of the machine. Low end-machines receive messages which include only basic changes to the display objects. High-end machines receive messages containing more elaborate changes.

Third, a common problem in a heterogeneous environment that characterizes distance learning is that of communication loads. Consider an end-point machine which does not receive its messages in time due to communication overloads, and thus makes it impossible for the student to synchronize with the classroom. This problem can be dealt with in two ways. The first solution is not to send a detailed visualization as described above. This solution is not satisfactory because it can reduce the understanding of the student. A better solution is to send the end-point machine portions of the code of the algorithm to be run locally on that machine and generate the visualization using the machines' hardware and software capabilities. Our system supports this capability at every nesting level of the code. That is, when a certain function of the algorithm is called, either its code is sent to the end-point machine and the visual events are displayed locally, or the same code is executed on the instructor's machine and only the interesting events are sent to the client one by one. Note that the algorithm implementor need not do anything for this option be take place. The code needs to be implemented only once.

A major aspect of a system that handles communication overloads is its ability to automatically detect this situation. Our system monitors the various links and the message queues on these links. If an end-point computer is found to be incapable of synchronizing with the classroom due to long queues, the system changes the type of messages sent to this machine.

Another important aspect of our model is the support for collaboration between the instructor and the students and among the students themselves. This is done by integrating a couple of tools: a *guided surfing tool* and a *chat tool*. The guided surfing tool enables a guided tour of the Internet. When the instructor changes the Internet address the instructor is cur-

rently in, the whole classroom automatically follows. This capability allows the instructor to use external information during class and during algorithm visualization. The chat tool lets the users exchange textual data with each other. This is important both for asking questions and answering them, and for using textual information within the visualization, explaining the algorithm and presenting the code.

3 The System

MAVIS, our system, was implemented in Java. This programming language was chosen for the following reasons. First, Java supports multiple platforms, and thus the code can run on many kinds of endpoint machines. Second, Java applets allow a simple deployment, access and execution through standard browsers. Third, the RMI (Remote Method Invocation) mechanism of Java lets the code call and execute functions which reside in remote objects. Fourth, the language supports code migration to other machines and its execution on the target machine. Fifth, Java supports computer graphics capabilities essential for visualization. Sixth, the object oriented nature of Java facilitates the extension of the model via inheritance, and thus facilitates the creation of new visualizations by instructors and students alike. Seventh, a security mechanism, the *sandbox*, protects the user from destructive operations.

MAVIS is structured as a client-server architecture. Figure 2 illustrates the logical data flow in the system. The server is in charge of running the channel manager, which creates new virtual classrooms and opens a data channel for each classroom. The server is also in charge of monitoring the message queues at the clients. Once the instructor initiates the creation of a virtual classroom, the algorithm code is loaded by the *Algorithm executer*. During the run, and upon reaching interesting events, messages are broadcasted via a relevant data channel to all the users who have registered to that channel (illustrated by dotted lines). Sometimes, clients will run functions of the algorithms. In this case, the code is uploaded from the server (illustrated by double lines).

Figures 3–4 illustrate the configurations of the server and of the client, respectively. The system is configured in layers, which are described below.

- The *communication layer* is in charge of sending messages in the IP level, using the RMI mechanism of Java.
- The *routing layer* is in charge of routing the messages and the events between the server and the clients. This layer is also in charge of filtering the

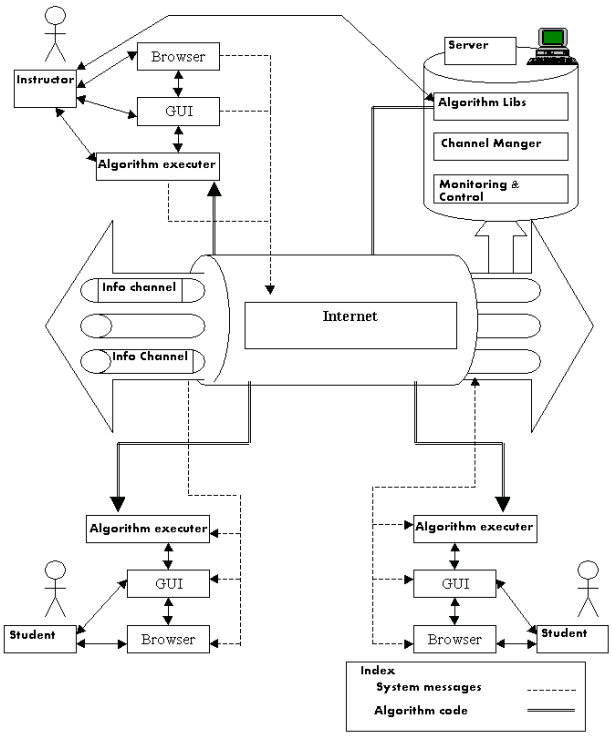

Figure 2: Data flow in MAVIS

messages based on criteria defined for each client (influenced by the capabilities and the communication loads of the client, as described in Section 2).

- The *dissemination & object layer* defines a unified format for all the objects transferred between the server and the clients. When an algorithmic object is to be transferred, it is encapsulated in an appropriate message object which contains data about the sender, the receiver and any other necessary data. This infrastructure can be easily extended to transfer new object types.
- The *event manager* lets the applications define the events happening in the applications. Using this mechanism, it is possible to create events happening in the server or in the clients and handle them in other clients.
- The *application layer* contains the applications implemented or used in the system such as the collaboration tools. This layer can be extended to contain other applications.
- The *algorithm infrastructure layer* is the basic

layer for implementing algorithms and their visualizations with MAVIS. It contains two types of objects: algorithm objects and visualization objects. These objects are modified by the algorithm during its execution. Obviously, these objects can be extended in order to create different visualizations and implement new algorithms.

- The *algorithm layer* contains the objects that implement the algorithm. This layer uses the objects of the algorithm infrastructure layer.
- The *display layer* is in charge of displaying the visualization. This is the layer that supports the multi-level presentation of the visual events. Separating this layer from the other layers is essential since it lets the user change and extend the forms of display, without modifying lower layers, and in particular without modifying the algorithm.
- The *control and monitor manager* monitors the message queues, changes the parameters accordingly and informs the routing layer.
- The *channel manager* is in charge of adding and deleting data channels and registering users to the various channels.

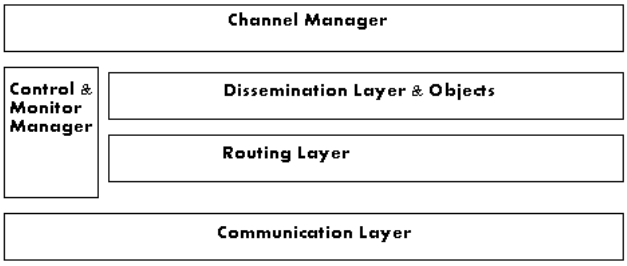

Figure 3: Server configuration

Adding new algorithms and new display mechanisms for specialized graphic hardware is very easy with MAVIS. This is due to the class inheritance feature of Java and due to the layered architecture of MAVIS. When a new object class is added to the system, it should inherit from the appropriate base class. Then all the layers which deal with objects of that type can manipulate the new class automatically. Easiness of use has been demonstrated when unexperienced users were able to quickly add new algorithms to the system as will be discussed in the next section.

4 Case Study

To check the utility of the model, we asked two teams of undergraduate students to implement a few

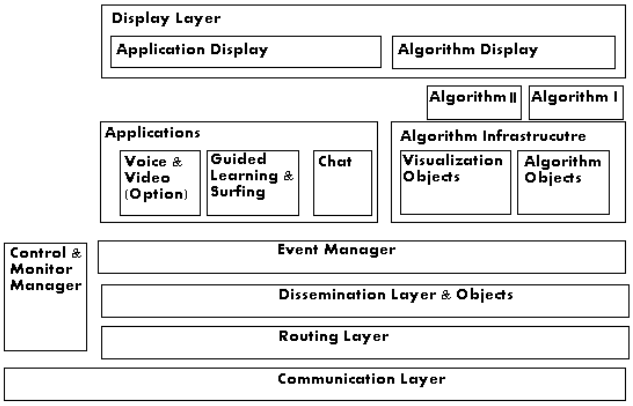

Figure 4: Client configuration

algorithms with MAVIS and create their visualizations. The algorithms were chosen so as to represent different families of algorithms. In particular, one team implemented merge-sort and Dijkstra's algorithm for finding the single source shortest paths in a weighted directed graph [8]. The second team implemented an algorithm for constructing a visibility graph and a quad-tree based algorithm for path planning, both used in robotics [9].

Generally, the development of a new visualization is done in three steps. First, the algorithmic objects are identified and the algorithm implemented in Java. Second, the visualization objects are identified and implemented. These are the graphical display objects. Third, the interesting events of the algorithms are identified and implemented and each event is assigned a priority level.

In our tests, each algorithm was executed and examined in the following configurations: on a standalone machine, on a client-server configuration over the Internet, and over the Internet using various load parameters.

For lack of space we will concentrate hereafter on a single algorithm, Dijkstra's algorithm [8]. Given a weighted directed graph $G(V, E)$ where all the weights are non-negative and a source node $s \in V$, the algorithm maintains a set S of vertices whose final shortest paths from s have already been determined, and repeatedly selects a vertex $u \in V - S$ with the minimum shortest path estimate. The vertex u is inserted into S and all the arcs leaving u are relaxed. The vertices contained in $V - S$ are maintained in a priority queue.

The algorithmic objects are the graph, consisting of nodes and arcs, and the priority queue consisting of queue elements. In the visualization, the nodes are presented as spheres, the arcs as straight lines, and

the queue elements as spheres as well.

The visualization consists of the following interesting events. The first interesting event, *initialization*, consists of the display of the original graph and the formation of the priority queue. In the next interesting event, *tree growing*, the first node in the priority queue is chosen and inserted into S. This event is animated by removing the node from the queue and moving a brush on the arc which connects it to the tree, painting it in red. In the next interesting event, *cost update*, all the outgoing arcs of this new node are examined, and costs are updated if necessary. This event is animated by moving a brush on each examined arc in turn, painting it in cyan. The next event, *queue update*, updates the queue according to the new costs. In the visualization, when the position of a node in the priority queue is changed, it is shown by an arrow pointing from the node's current location to its new location, followed by the move. The final interesting event, *results*, displays the resulting shortest paths tree in red.

Recall that each event can get a different priority level. In our visualization, the highest priority (always presented) is assigned to the *initialization* interesting event and to the *results* event. In other words, if a client wishes to see only the highest level of abstraction, this client will visualize the given graph and then the shortest path tree painted in red on the graph. A medium priority is assigned to the *tree growing* event that displays the next selected node that joins the constructed tree and to the *queue update* event which updates the priority queue. Clients having this level will therefore view initialization, the additions of nodes and red arcs to the shortest path tree, each such addition is followed by queue updates, and the display of the final tree. A low level visualization also displays the *cost update* event where every possible arc is examined and painted in cyan. Only clients having this level will view the full algorithm visualization.

Some snapshots from the visualization are given in Figures 5–7. The pseudo code of the algorithm is shown alongside the visualization, where the current line is emphasized. Figure 5 shows the graph and the priority queue after initialization. This snapshot is visualized by all the clients. Figure 6 shows the graph after the first node was extracted from the queue, and its out-going arcs are examined (in cyan). This snapshot is visualized only by clients where the lowest level of abstraction is assigned. It also shows the updates to the queue (by the arrow). Figure 7 displays shortest paths tree in red.

5 Conclusions

We have presented in this paper a conceptual model for an algorithm visualization system embedded within a general distance learning framework. The main feature of this model is the support for multi-level visualizations necessary for addressing the heterogeneity of the Internet, the diversity of the students, and the iterative nature of the learning process. Another important feature of the system is the support for collaboration and information sharing embedded in the model.

We have also presented a system, MAVIS, that realizes the model. MAVIS has a client-server architecture and supports clients running various operating systems. It monitors the loads of the network and adjusts the visualization parameters at each client according to the loads and to the capabilities of each machine. Implementing new algorithms and new algorithm visualizations with MAVIS is easy since its object oriented nature results with a flexible system that can be easily extended by students.

There are a few directions for future research. First, we would like to embed a more advanced statistical monitoring algorithm for handling overloads. Second, we intend to add a recording mechanism that will let users record classes and re-visualize them at the students' convenience. Third, we would like to add video and voice capabilities. Finally, with the advent of portable devices, we would like to extend MAVIS to handle these architectures.

References

[1] H. Abdel-Wahab, K. Maly, E. Stoica. Multimedia integration into a distance learning environment. *Proc. of Third International Conference on Multimedia Modelling*, pp.69-85, Toulouse, France, November 1996.

[2] B. Aktan, C.A. Bohus, L.A. Crowl and M.H. Shor. Distance learning applied to control engineering laboratories. *IEEE Transactions on Education*, Vol. 39, No. 3, pp. 320–326, August 1996.

[3] M.H. Brown. *Algorithm animation.* MIT Press, 1988.

[4] M.H. Brown. Zeus: A system for algorithm animation and multi-view editing. In *1991 IEEE Workshop on Visual Languages*, pages 10–17, October 1991.

[5] M.H. Brown and M.A. Najork. Collaborative active textbooks: A web-based algorithm animation system for an electronic classroom. *IEEE Symp. on Visual Languages*, pages 266–275, 1996.

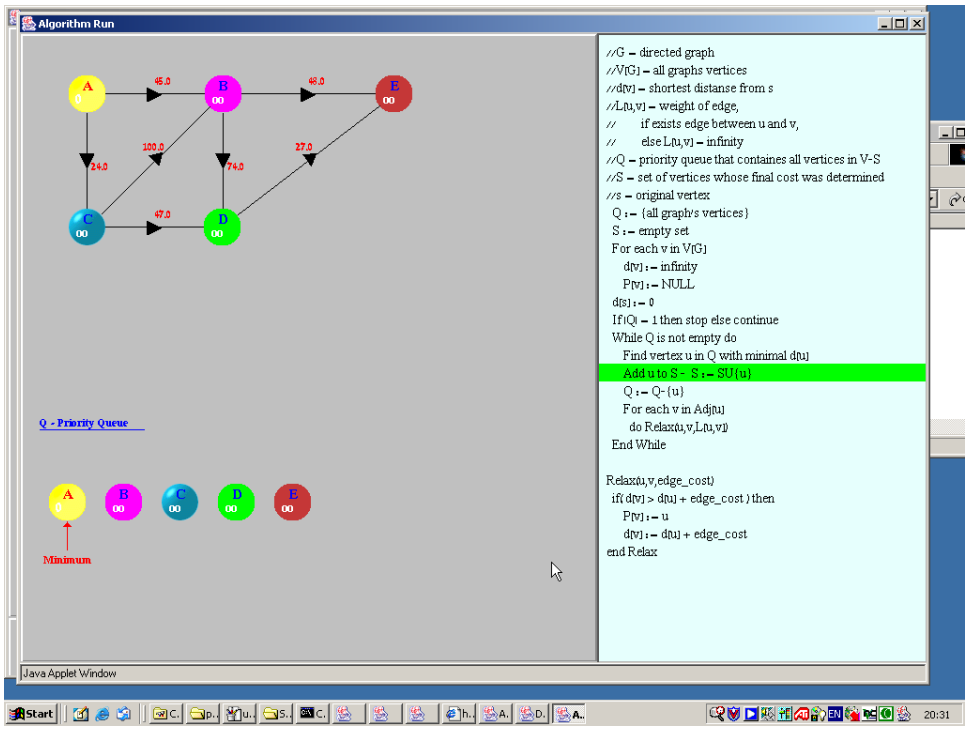

Figure 5: Initialization

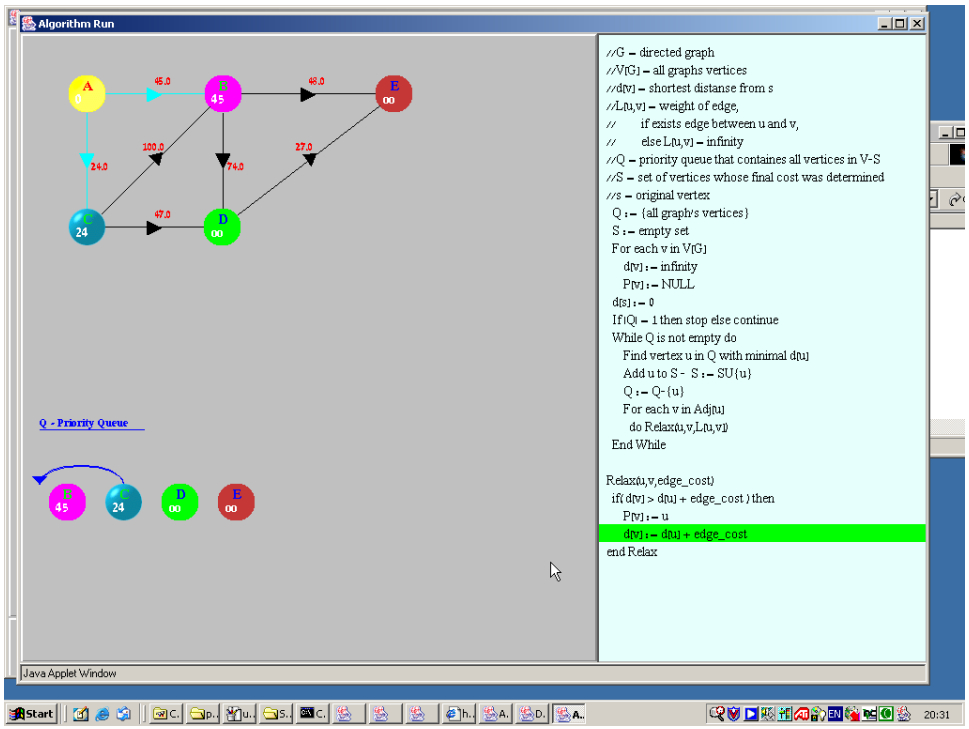

Figure 6: Examining the outgoing arcs and updating the queue

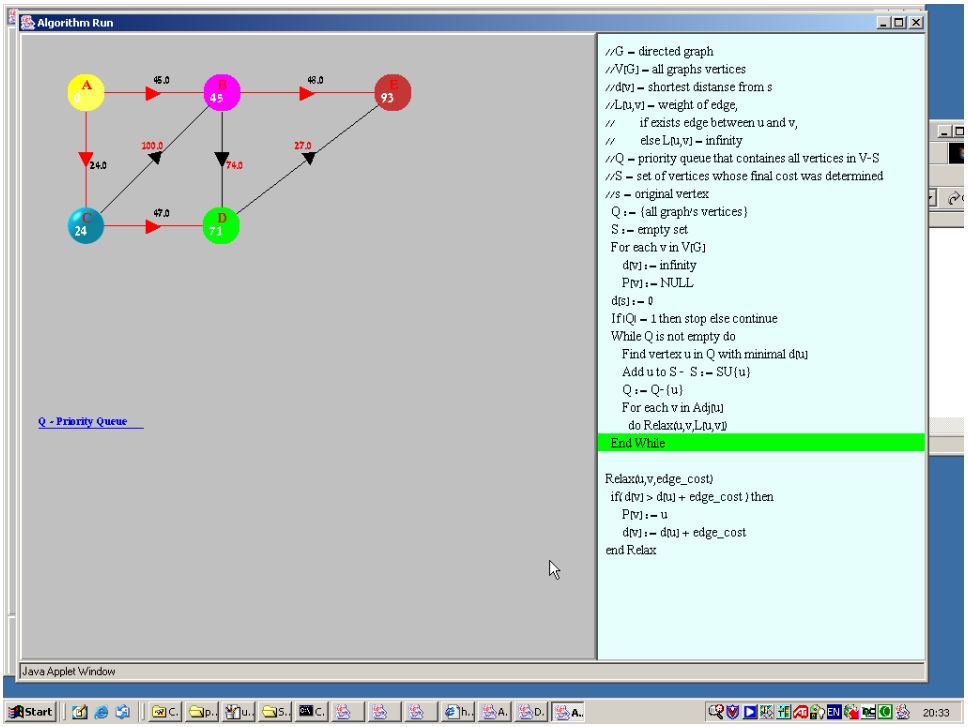

Figure 7: Shortest paths tree

[6] M.H. Brown and R. Raisamo. JCAT: Collaborative active textbooks using Java. *Computer Networks and ISDN Systems*, 29:1577–1586, 1997.

[7] M.H. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2(1):28–39, January 1985.

[8] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to algorithms*, The MIT Press.

[9] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[10] K. Maly, H. Abdel-Wahab, C.M. Overstreet, J.C. Wild, A. Gupta, A. Youssef, E. Stoica, E. Al-Shaer. Interactive distance learning over Intranets. *IEEE Internet Computing*, 1(1):60–71, January–February 1997.

[11] M.A. Najork and M.H. Brown. A library for visualizing combinatorial structures. In *Proc. '94 Visualization*, pages 164–171, October 1994.

[12] V.K. Proulx, R. Rasala and H.J. Fell. Fundamentals of computer science: What are they and how do we teach them. In *SIGCSE Bulletin*, 29:3, 1997, 42–48.

[13] S.H. Rodger. Integrating animations into courses. *ACM SIGCSE/SIGCUE Conference on Integrating Technology in Computer Science Education*, 1996, 72–74.

[14] L. Sherry. Issues in distance learning. *International Journal of Distance Education*, 1(4):337–365, 1996.

[15] M. Shneerson and A. Tal. Visualization of geometric algorithms in an electronic classroom. *Journal of Visual Languages and Computing*, 6(6):615–637, December 2000.

[16] J.T. Stasko. TANGO: A framework and system for algorithm animation. *Computer*, 23(9):27–39, September 1990.

[17] J.T. Stasko. Using student-built algorithm animations as learning aids. *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE '97)*, 1997, pp. 25-29.

[18] J.T. Stasko, A. Badre, and C. Lewis. Do algorithm animations assist learning? An empirical study and analysis. in *Proc. ACM INTERCHI* '93. New York: ACM Press, 1993, pp. 61-66.