EVEGA: An Educational Visualization Environment for Graph Algorithms

Sami Khuri
Dept of Math and Computer Science
San José State University
San José, CA 95192, USA
011–408–924–5081
khuri@cs.sjsu.edu

Klaus Holzapfel
Institut für Informatik
Technische Universität München
D–80290 München, Germany
49–89–289–28494
Klaus.Holzapfel@in.tum.de

ABSTRACT
This paper describes the package EVEGA (**E**ducational **V**isualization **E**nvironment for **G**raph **A**lgorithms) and possible ways of incorporating it into the teaching of algorithms. The tool is freely available, platform– and network–independent, and highly interactive. The tool is designed for three different groups of users: students, instructors, and developers. Interaction with EVEGA can be achieved through the exploration of existing default visualizations, through the direct manipulation of graphical objects, or through the implementation and visualization of new algorithms using existing classes.

1. INTRODUCTION
Some of the well–known problems covered in an undergraduate course on the design and analysis of algorithms are those dealing with graphs such as minimum spanning trees, shortest–paths, maximum–flows, and both weighted and nonweighted matching problems. These problems are motivated by real–world examples and students learn about different strategies for solving them. While the workings of the various algorithms can be presented through static visualization on the white board or overhead slides, there is a trend in CS education to make the learning process more independent, individualized, interactive and intuitive. Interactive algorithm visualizations can give the students more autonomy in their learning process and independent learning gives them a feeling of accomplishment and also frees some of the instructor's time. Students can explore, discover and reach their own conclusions with only minimal guidance. Programs may be individualized for each student, providing an opportunity for self–paced learning. Students are able to learn interactively by receiving feedback provided either by the instructor, by their peers, or by a program. Finally, intuitive learning: the ability to find solutions via nontraditional or unexpected paths can be done through visualizing a problem, and by graphing and modeling it.

There is a vast amount of research on different visualization tools conducted over the years. While WWW has numerous repositories of algorithm visualizations (see for example [6]), the authors decided to design and implement yet another visualization tool. The decision had been made because the authors' search for a tool suitable for teaching graph algorithm was unsuccessful. There are many interesting and impressive visualization packages, some of which have functionality similar to that of EVEGA, but they were not suitable for the intended use. Some of the existing tools are platform–dependent, such as AVS [7], and LINK [1], or network–dependent, or they are not freely available, such as CATBox [2].

EVEGA was designed for use in classroom demonstrations, homework assignments, and analysis of algorithms. It can be used by students, instructors, and developers or programmers. The demarcation between these groups of users is sometimes loose. A student for example, might play the role of the developer when implementing a new algorithm for a semester project.

The main scope of the tool is visualization of graph algorithms, which allowed us to concentrate on interactivity, correctness and attractiveness of the tool. Although the trend in algorithm visualization research is to design general–purpose systems [6], we believe that specialized tools are easier to use, maintain, upgrade, and document. EVEGA is easy to interact with and is platform– and network–independent. It is implemented as a stand–alone Java application to accommodate students who have different computer systems at home or no access to the Internet.

The rest of the paper is organized as follows. Section 2 presents the functionality of EVEGA, and Section 3 shows how it can be used as a lecture support tool. Section 4 describes EVEGA's framework for developing and visualizing algorithms. The paper then gives an informal evaluation of the package and concludes with some future directions in Section 5.

2. THE EVEGA PACKAGE
In designing EVEGA, we followed guidelines derived from our experience in writing visualization tools and using them in the classroom. EVEGA has consistent interface and visual representations of algorithms and uses colors, multiple views, and textual narratives. The most important issue in designing algorithm visualization tools is that of interactivity, since it distinguishes algorithm visualization systems from simple drawings of an algorithm. Most of the algorithm visualizations designed so far permit only limited forms of interaction, such as

stopping/starting the visualization or changing its speed. EVEGA has a very high degree of interactivity providing traditional as well as advanced features, such as the direct manipulation of graphical objects. The "Editor" window, displayed in Figure 1, is the interface between the end–users and the tool. The main features of the "Editor" are the control mechanisms, the graph drawing tool, the graph generators, the error handling mechanism and the on–line help files.

One of the strong features of EVEGA is its interface for graph drawing and editing. It is easy to directly manipulate and create graphs. Graphs can be constructed by selecting one of the options from the "Tools" menu or by using graph drawing icons displayed on the left hand–side of the "Editor" window. The user can, for example, select with the mouse one of the vertices of the graph and move it to a new position, swap edges, create edges and vertices, remove one of the graph elements, edit graph properties, snap the graph to grid, and align it. Figure 1 displays the process of constructing a new graph (which will be used in the next sections), where the user is about to enter the value of the capacity (the weight) of the edge going from vertex 2 to vertex 6.

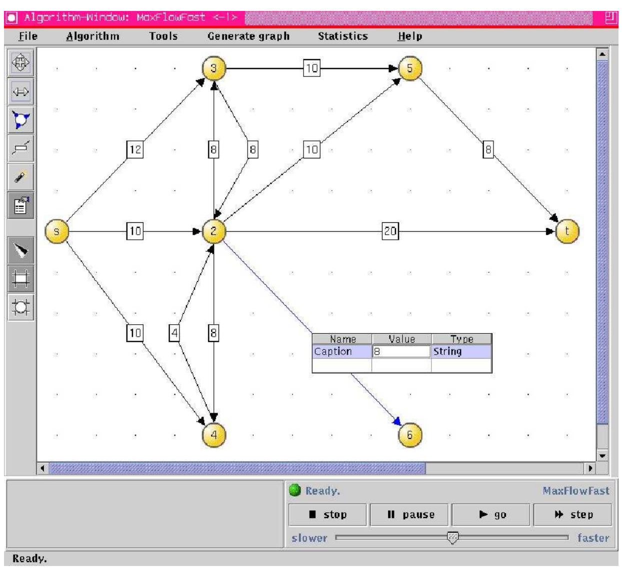

Figure 1. Using the "Editor" to create a flow network.

Another way of creating graphs is by using the built–in graph generators. Four different graph generators [3] can be selected from the "Generate graph" menu. This option is mainly used for testing the performance of algorithms on large input graphs.

Automating the graph drawing has the following benefits to EVEGA's users. It can reduce students' anxiety during graph drawing and allow them to concentrate on what is really important – the algorithm in question. Instructors' time is also spent more productively. They can quickly create or change the graph for a new in–class demonstration or for answering "what–if" questions. The graphs can be created in advance and saved for later exploration. The developers of new algorithms do not need to "re–invent" the wheel and just need to call methods of existing classes.

In designing EVEGA, special attention has been paid to the handling of erroneous user actions. In interactive systems, users might press the wrong button, input invalid data, or manipulate the wrong graphic object. Efforts have been made to prevent the possibility of errors by having default values wherever possible. The tool makes sure that number fields only accept numbers; and it provides lists of choices or file selection dialogs rather than asking users to type names. Error messages that clearly state the mistake appear in pop–up windows.
EVEGA comes with on–line help. The help files are easy to access, easy to exit from and provide specific and accurate information. Students can learn how to use the package or read about details of the algorithms.

3. USING EVEGA IN THE CLASSROOM: AN EXAMPLE

This section describes how EVEGA can be used to explain graph algorithms in class. For demonstration purposes, we choose the maximum–flow algorithm. The explanation starts with a real–life problem.

Example: The **A**ndalucia **A**ceituna **C**ompany (AAC) buys black and green olives from the farmers in Southern Spain and puts them in jars in Málaga. The jars are placed in one–ton crates, and are sent to Barcelona to be shipped to the rest of Europe. AAC leases space on trucks from the **B**arcelona **C**amión **C**ompany (BCC). The trucks of BCC travel over specified routes between major cities in Spain and have only limited capacity (the maximum number of crates that can be shipped over a certain route) for AAC's crates since BCC also does business with other manufacturers. The goal of AAC is to ship the largest number of crates per day that can reach Barcelona that same day. Thus, the problem consists in finding an algorithm that will output both, the largest possible number of crates that could be shipped from the source to the destination, and the route for such a shipment.

We then give a formal definition of the maximum–flow problem. A flow network is a directed graph G = (V, E) with two special vertices: the source s and the sink t. Each edge (u, v) has a nonnegative capacity $c(u, v) \geq 0$ and f(u, v) is the flow from vertex u to vertex v. The value of a flow is the total net flow out of the source and is denoted by $|f|$. The maximum–flow problem consists in finding a flow of maximum value from the source s to the sink t for a given flow network G [4].

We then use EVEGA (see Figure 1) to construct an example of a flow for the AAC problem. Edge (u, v) is labeled with f(u, v)/c(u, v) (see Figure 2). For example, AAC can move a maximum number of 8 crates of olives from Salamanca to Madrid, but the current flow is 5.

Upon computing the flow of the network given in Figure 2, we get: $|f| = 25$. To motivate the algorithms for solving the AAC problem, we ask students if 25 is the maximum possible flow from Málaga to Barcelona.

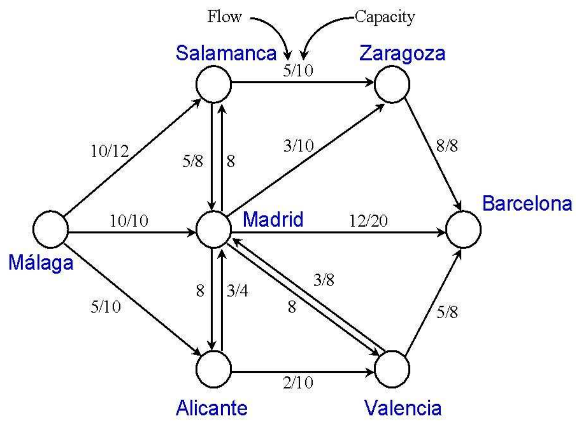

Figure 2. An example of a flow for the AAC problem.

3.1 Preflow−push algorithms

The algorithm can be understood by thinking of the flow network as being a system of interconnected pipes. The vertices are pipe junctions and each vertex u has an outflow pipe leading to an arbitrarily large reservoir, e(u) to accommodate excess flow and is on a platform whose height, h(u) increases as the algorithm progresses. Flow is pushed from a high vertex to a lower one. The order in which active nodes (nodes that have an excess flow) are considered can be random, FIFO, or highest label first (HLF). The FIFO preflow−push procedure maintains a set list as a queue and consists of two parts:

Initialization: The sink is of height 0, vertices that share an edge with t will be of height one, vertices two hops from t have 2 as initial height, etc. The initial height of the source is set to |V|. The source's excess flow is initialized to the sum of the capacities of the edges leaving s. The excess flow and the height of each vertex appear right underneath each vertex u. In Figure 3, 32/7 under node s means that the excess flow of s is 32 and the vertex is of height 7. Initially, each edge leaving s is filled to capacity; other edges carry no flow at all.

Push or Lift procedures: The algorithm selects the vertex u from the front of the list, performs pushes from this node, and adds newly active vertices to the rear of the queue. The algorithm terminates when the queue of active nodes is empty.

In the example of Figure 3, vertices 2, 3, and 4 are active nodes. In EVEGA, active nodes are colored in blue to differentiate them from inactive (yellow) nodes. The queue of active nodes (List = {2,4,3}) is displayed in the right hand−side of Figure 3. The algorithm next removes vertex 2 from the queue and examines it. A push of 10 crates from vertex 2 to t is then performed. Since node 2 gets rid of its excess overflow, it is not an active vertex for the time being. The algorithm then considers vertex 4 (the current node in List) and continues until the queue of active nodes is empty.

While demonstrating the workings of the algorithm, the instructor can step through the example, pause, ask students to predict what will happen next, and check the visualization to see if the predictions were correct.

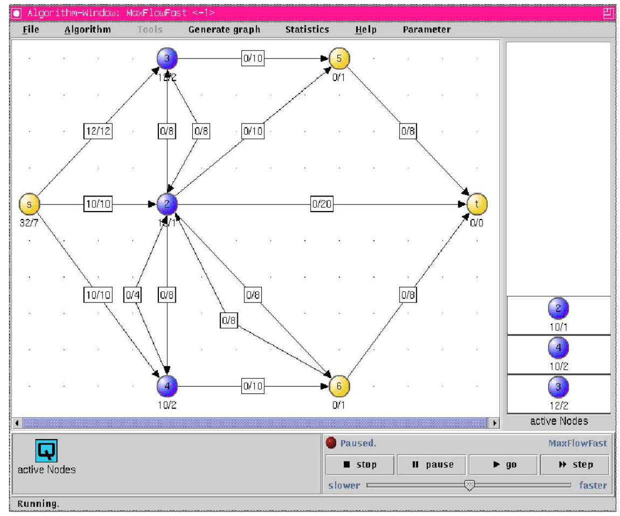

Figure 3. Edges leaving s are filled to capacity.

In general, the design and study of an algorithm is followed by its analysis. Most existing educational tools are designed and used for explaining the steps of an algorithm, but only a few allow the users to perform algorithm analysis. We have implemented an analysis tool for the maximum−flow algorithms that can be invoked in the "Editor" window. Analysis tools similar to ours could easily be constructed to analyze other algorithms. To perform the analysis of an algorithm, the user needs the results of an algorithm's run. The following statistics are collected and displayed at the end of each algorithm's run: running time (total, push and adjustment phases), number of push−operations (saturating and nonsaturating pushes), and the number of relabel operations. Of course, the running time of an algorithm is dependent on the running time of the visualization, which is undesirable for the analysis of algorithms. So the user can disable the visualization feature and collect the above statistics for one or more algorithms. Finally, users can compare the performance of their algorithm on various graphs or different implementations of the same algorithm and display the results graphically (see Figure 4).

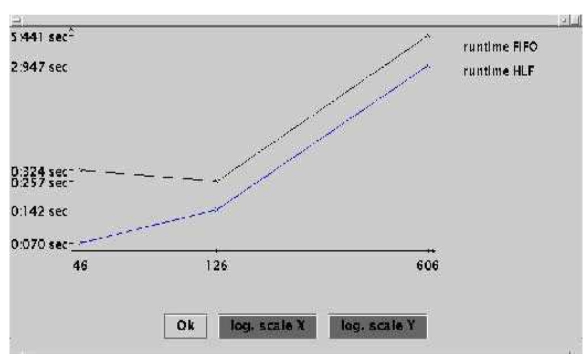

Figure 4. Comparison of FIFO and HLF.

4. USING EVEGA TO IMPLEMENT NEW ALGORITHMS

Another useful and interesting way of using EVEGA is to have students write their own implementations of algorithms. This can be done in programming homework assignments or in a semester−long project. As mentioned in Section 1, EVEGA is implemented in the Java programming language and it includes packages of classes that can be used to implement and visualize new algorithms. To implement a new algorithm, the developer has to extend an existing class and implement the run() method. As mentioned in Section 2, users of EVEGA have access to the documentation for developers generated using the javadoc utility. See [5] for a more detailed description of the implementation.

The advantage of these visual assignments and projects is the students' ability to see the results of their programs in graphical form. The graders can also notice errors in the algorithm's implementation more easily. The programs have to be written in Java, and the users can either use existing classes or implement their own. Of course, projects of this sort are not suitable for beginning students.

5. CONCLUSION

This paper introduced the Educational Visualization Environment for Graph Algorithms. EVEGA can be used by students, instructors, and developers. Instructors can use it to explain the steps of a graph algorithm in the classroom. Students can use EVEGA to explore algorithms on their own, and developers can create new visualizations of graph algorithms by using existing classes and methods. EVEGA offers a set of powerful features to create and edit graphs, to display visualizations and to perform comparisons of algorithms.

Our students like EVEGA and its features, but it is difficult to measure its impact on the learning process quantitatively. We performed an informal evaluation to test its strengths and weaknesses. Students participating in the evaluation had already studied maximum−flow algorithms in an algorithms and data structures course and they rated their knowledge of these algorithms as medium. They had never used visualization tools before. As part of the evaluation, the students were asked to create their own graphs, run the algorithms under different parametric settings and comment on all aspects of the package, such as the GUI, the graph drawing tool, and the colors. Overall, students' reaction was very favorable. They continued experimenting with the tool even after returning the evaluation forms. They found EVEGA extremely easy to use and easy to understand. Students recommended a few small changes, some of which we implemented.

Although EVEGA is very easy to use and fast to learn, it is important to realize that it cannot be relied upon as a stand−alone educational tool. As with any other visualization tool, it must be carefully incorporated into, and supported by other teaching techniques. The tool should be first demonstrated in class by an instructor or by a teaching assistant in an open lab to show the students its power. It does not replace the lecture, but rather supplements it. Students should be encouraged to use the visualization interactively to meet a set of objectives, and their process should be monitored or scaffolded when learning by exploration to keep them on track.

There are several directions for future work. First, additional algorithms will be implemented and included as default settings. Second, we are in the process of testing a client−server version of EVEGA, as well as incorporating it into an electronic textbook on graph algorithms. The EVEGA package is available for download from http://www14.in.tum.de/EVEGA/index.html

6. ACKNOWLEDGEMENTS

The authors would like to thank Thomas Erlebach for his guidance and comments during the MS writing project.

REFERENCES

[1] Berry, J., "Improving Discrete Mathematics and Algorithms Curricula with LINK", Proceedings of ITiCSE'97, ACM Press, pp. 14−20, 1997.

[2] "CATBox − the Combinatorial Algorithm Toolbox", Springer Verlag, 2000.

[3] Cherkassky, B. and Goldberg, A., "On implementing push−relabel method for the maximum flow problem", *Technical Report*, CS Department, Stanford University, September, 1994.

[4] Cormen, T., Leiserson, C., and Rivest R., "Introduction to Algorithms", MIT Press, Cambridge, 1990.

[5] Holzapfel, K., "WWW−Visualisierung und Analyse von Push−Relabel−Flussalgorithmen" (in German), *Masters Thesis*, Institut für Informatik, Technische Universität München, 1999.

[6] Khuri, S., "Designing Effective Algorithm Visualizations", invited lecture at the Program Visualization Workshop, Porvoo, Finland, July 2000, available at http://www.mathcs.sjsu.edu/faculty/khuri/animations.html.

[7] Shannon, G., MacCuish, J., and Johnson, E., "Animating Maximum Flow Algorithms: A Case Study", *Proceedings of the DIMACS Algorithms Implementation Challenge: Network Flows and Matching*, 1991.