JHAVÉ -- An Environment to Actively Engage Students in Web-based Algorithm Visualizations

Thomas L. Naps, James R. Eagan, Laura L. Norton
Lawrence University
{thomas.naps, james.r.eagan, laura.l.norton}@lawrence.edu

Abstract
In this paper, we describe JHAVÉ (Java-hosted Algorithm Visualization Environment), a client-server environment for delivering algorithm visualizations over the Web. The first section of the paper briefly summarizes prior research by a variety of investigators into the pedagogical effectiveness of algorithm visualization (AV). The design goals of JHAVÉ are then placed in the context of this research. After a discussion of some technical details of the JHAVÉ architecture, we present two examples of algorithms depicted in JHAVÉ. The results of students' exploring these algorithms with JHAVÉ are analyzed. We close with a discussion of the general conclusions reached from our current work and future directions in which it may lead.

1 Background and Rationale

Algorithm visualization (AV) depicts the execution of an algorithm as a discrete or continuous sequence of graphical images, the viewing of which is controlled by the user. Increasingly many algorithm visualization tools have been developed and presented at recent SIGCSE and ITiCSE conferences (eight papers in '98 SIGCSE proceedings, nine in '98 ITiCSE proceedings, ten in '99 SIGCSE proceedings). These tools can be broken down into two groups. Many are designed to make it relatively easy for students to construct their own visualizations by annotating their programs with calls to functions that, as much as possible, hide the details of producing the graphics connected with the visualization. Others merely have the students watch predefined visualizations designed by the instructor. There is little doubt that a student will learn an algorithm in greater depth if she codes her own visualization of it. This result has been borne out in work by Rodger [7], Stasko [12], Hundhausen [3], and Naps [6]. However, although having students code their own visualizations can be very valuable, it can also be extremely time-consuming. Hence, students are limited to doing this for only a small number of algorithms in a particular course. The work we describe in this paper focuses on an environment for delivering predefined visualizations to students over the Web. JHAVÉ (Java-hosted Algorithm Visualization Environment) has been designed with specific and realistic instructional goals and takes into account pedagogical issues that have been identified by a number of researchers. To understand the rationale behind JHAVÉ requires summarizing the findings of these researchers.

One of the first systematic studies on the effectiveness of AV systems in teaching algorithms and data structures was conducted by Stasko, Badre, and Lewis [11] and reported results that seemed discouraging:

- Subjects trying to learn an algorithm using both textual materials and algorithm animations outperformed text-only subjects, but the difference was not large and not statistically significant. The very passive mode of engagement on the part of the viewer was cited as a key reason for this disappointing result.
- To be effective, algorithm visualizations must be accompanied by comprehensive teacher-provided explanations. The visualization of an algorithm is a mapping from a sequence of digital operations within the computer to an abstract rendering of them on the graphics screen. Unless the student is provided with explanations to make her aware of this mapping, the desired effect of the visualization may be totally lost.
- Without a facility to rewind the visualization to a previous state in the algorithm's execution, students frequently become confused.

Other investigators reported that somewhat better results were achieved by forcing the students to be more "active" in watching the visualization. For example, in 1993, Lawrence [4] found that students who constructed their own input data for the algorithm being viewed scored significantly higher on a post-test than students who watched the visualization passively. In 1996, Byrne, Catrambone, and Stasko [1] conducted an experiment in which viewers were forced to make predictions about what they would see during the visualization. These viewers scored significantly better on a post-test than others who merely watched identical visualizations without making such predictions. The success of Wolfe's TERA system [8] also reinforces the hypothesis that requiring students to make predictions about what they will see can positively affect their learning. TERA allows students to explore the effects of graphics rendering algorithms in two modes -- explore mode and quiz mode. In quiz mode, students are presented with a scene and must determine the rendering algorithm used to produce that scene. A recent study by Wolfe, Grissom, and Naps [14] found that three groups of students who used TERA did extraordinarily well on exam questions about the rendering algorithms they had studied. Findings such as these all point to the same conclusion -- actively engaging students by using hooks that force their interaction with the visualization is essential to making AV an effective instructional tool.

We must also be careful in defining the instructional goals of an AV system. The study of algorithms in CS2 or an upper-level algorithms course involves five different types of understanding: (1) understanding the algorithm as a “recipe”, (2) understanding why the recipe solves the problem, (3) formally proving the algorithm is correct, (4) performing an efficiency analysis of the algorithm, and (5) coding the algorithm in a programming language.

In [2], deMarneffe argues that using technology, including predefined algorithm visualizations, cannot make a substantive difference in (3), (4), and (5). Certainly the effect of AV in achieving these types of understanding will be limited. Consequently, in designing JHAVÉ, we focused on developing a system that would help students achieve the first and second types of understanding. After using JHAVÉ to view examples of an algorithm's execution, we want students to be able to manually trace the execution of the algorithm with a small data set and feel that they understand why the algorithm works. If in fact the visualizations delivered by JHAVÉ succeed in reaching these goals, it will allow the instructor to spend her contact time with students discussing the weightier issues involved in the latter types of understanding cited above.

With this as background, the design goals for JHAVÉ emerged as follows:

- Present students with at least two types of visualizations -- smoothly running animations and sequences of discrete snapshots. Both should have a rewind capability.
- Supplement the visualization of the algorithm with context-sensitive textual material in a Web browser window. "Context-sensitive" means that the textual material should change to fit the current state of the visualization.
- Provide the student with *input generators* -- pop-up windows that allow the student to provide input to the algorithm so that she can engage in "what-if" kinds of exploration. Just as one learns concepts in mathematics by repeatedly doing exercises, so one comes to an understanding of how an algorithm works by repeatedly tracing through how the algorithm manipulates different data sets. In this sense, the visualizations delivered by JHAVÉ should be "predefined" only in the sense that, after the student has triggered the input of data to the algorithm, the graphic rendering of the algorithm's execution is dictated by the instructor's design of the visualization.
- Force the student into active participation by interrupting the visualization with "stop-and-think" questions. Such questions make the student predict what she will see in the next step of the visualized algorithm. Without such questions, once a student becomes confused, continuing to watch a visualization is akin to watching a movie in which one has lost interest. Stop-and-think questions change this dramatically. When a confused student answers a question incorrectly, continuing with the visualization not only provides the student with the correct answer but serves to reset the student's perception of the algorithm back on the track intended by the instructor.

2 The JHAVÉ Architecture

JHAVÉ in itself is not an algorithm visualization system. Rather it is a client-server architecture, implemented in Java, into which more specific AV engines may be plugged. As it is currently configured, there are two such engines -- one for the Samba animation scripting language designed by Stasko [10,13] and one for the GAIGS data structure visualization language developed by Naps [5]. Once such an engine "plugs into" JHAVÉ, designers of a visualization using that engine can easily incorporate the pedagogical tools that come with the JHAVÉ environment -- context-sensitive documentation in a browser window, input generators, and stop-and-think questions. The main criterion for an AV engine to be plug-compatible with JHAVÉ is that it must produce its visualization using a script file. With the script file methodology, a program implementing the algorithm to be visualized executes and, instead of directly rendering a visualization of the algorithm, it writes visualization commands to the script file. When the algorithm terminates, this script file is parsed and rendered by the AV engine.

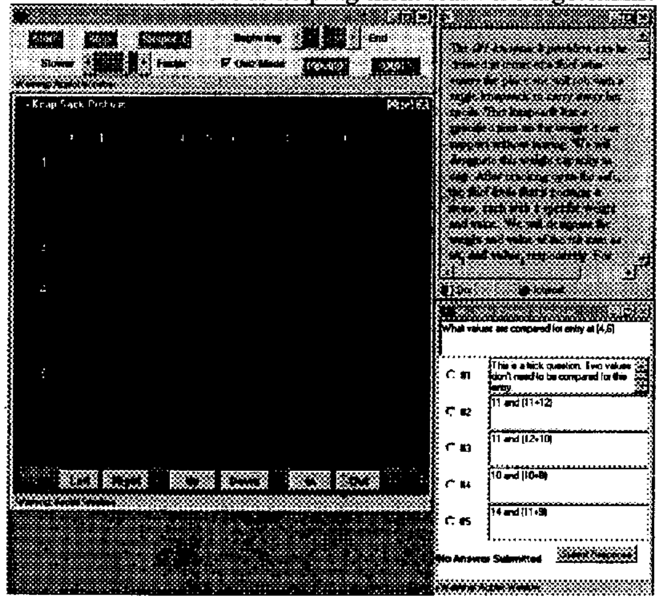
In JHAVÉ's client-server model, the server application manages the available algorithms and generates the visualization script files that the client can display. In a standard session, a Web surfer first launches an instance of the AVClient applet, which displays a listing of available algorithms. When the user selects an algorithm from that list, the client applet sends a request to the server, which will run the program that generates the script file for that algorithm and send it back to the client. The client then renders it with the appropriate engine. If the algorithm requires input from the user, the server sends an input generator object to the client. This is just a frame with appropriate input areas for the user. Once the user fills out these areas, the client returns the user's input to the server as a data set to use when running the algorithm.

Within the listing of available algorithms, algorithms can be grouped by type to indicate whether they can interchange data sets. Thus, when a user requests a visualization of, for example, Prim's minimum spanning tree algorithm, she can subsequently view how Kruskal's algorithm would work on the same data set. Alternatively, the server could provide visualizations of the same algorithm for different engines, allowing the user to observe the algorithm's execution from the perspective of GAIGS' discrete snapshots or Samba's smooth animations.

One key advantage of JHAVÉ's design lies in its portability. Written in Java, the server application itself can be relocated from one operating system to another without making any changes. This is in stark contrast to older environments such as WebGAIGS [5] that made extensive use of CGI scripts to launch visualizations. Programs that implement algorithms producing visualization scripts, however, can be written in *any* language. They are merely executed at the direction of the Java server application. This means that, under the JHAVÉ environment, an instructor who wishes to provide her own visualizations for students is not tied to writing them in Java. Hence there is a great deal of variation in the ways one can produce visualizations that ultimately are viewed in JHAVÉ. For instance, prior to the advent of the Web, we had written many Pascal and C++ programs that produced GAIGS and Samba visualization scripts. Whereas the visualizations produced by these programs once had to be viewed locally using operating system specific tools, they now can be conveniently distributed over the Web. All we had to do to our old programs is modify them so that, inside the scripts they produced, annotations were inserted as to when context-sensitive documentation windows and stop-and-think questions should pop up during the course of the visualization. This methodology also allows students in a course to write Web-viewable visualizations even though they are not programming in Java. Merely load the students' script-producing programs on the departmental Web server, and JHAVÉ can make the visualizations produced by these programs available for everyone to explore.

3 Examples and User Testing

To initially test the system, we prepared visualizations and accompanying materials for two algorithms studied in our upper level algorithms course -- the 0/1 knapsack problem and the Knuth-Morris-Pratt string search algorithm. For a discussion of these algorithms, the reader is referred to [9]. The visualization for the knapsack problem was built from a C++ program that created a Samba script while that for the KMP algorithm came from a Pascal program that produced a GAIGS script. During the past summer, these visualizations were presented to a small group of five students who had completed our CS2 course but who had not yet been exposed to the algorithms. For the knapsack problem, we used JHAVÉ's stop-and-think questions to interrupt the animation with multiple-choice questions that asked what two previously computed values in the optimal value matrix had to be compared to fill in the currently empty matrix position (see Figure 1). The visualization for the KMP algorithm interspersed GAIGS snapshots with fill-in-the-blank questions that asked students first for the values to be stored in the pattern string's alignment array and later about the indices at which realignment would occur in the second phase of the algorithm (see Figure 2). After studying the algorithms for approximately two hours, the students were asked to manually trace each algorithm on a small data set and then to respond to a set of subjective questions regarding the effectiveness of the JHAVÉ environment in helping them learn the algorithms.


What values are compared for entry at (4,6)

#1 This is a trick question. Two values don't need to be compared for this entry.

#2 11 and (11+12)

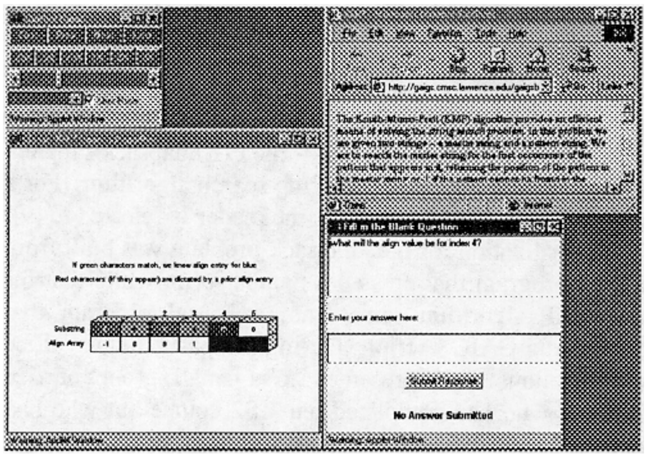
#3 11 and (12+10)

#4 10 and (10+8)

#5 14 and (11+9)

No Answer Submitted

Figure 1 -- *Knapsack problem with Samba controls, context-sensitive documentation, AV rendering window, and multiple-choice question.*


File Edit View Favorites Tools Help
Address http://gaigs.cmsc.lawrence.edu/gaigsb
Fill in the Blank Question
Substring
Align Array
Submit Response
Warning: Applet Window

The Knuth-Morris-Pratt (KMP) algorithm provides an efficient means of solving the string search problem. In this problem we are given two strings -- a master string and a pattern string. We are to search the master string for the first occurrence of the pattern that appears in it, returning the position of the pattern in the master string or -1 if the pattern cannot be found in the

If green characters match, we know align entry for blue

Red characters (if they appear) are dictated by a prior align entry

What will the align value be for index 4?

Enter your answer here:

No Answer Submitted

Figure 2 -- KMP algorithm with GAIGS controls, context-sensitive documentation, AV rendering window, and fill-in-the-blank question.

The ability of the students to trace the algorithms after participating in the visualizations was guardedly encouraging. For the knapsack problem, all students arrived at an understanding of the algorithm to the extent that they could fill in the matrix of values computed by the algorithm for a small instance of the problem. Two of the students made a minor arithmetic error at one stage of the trace, but an examination of their responses made it clear that their error was purely one of calculation and not of understanding. On the post-test questions for the KMP algorithm, only two of the five students were correctly able to trace the entire algorithm, that is, both the phase that determines the values for the pattern string's alignment array and also the phase in which the alignment array is consequently used to perform the actual search. The three students who made errors did so during their trace of the algorithm's first phase. Their traces of the realignments during the second phase of the algorithm were then consistent with their earlier errors.

Student comments on the strengths and weaknesses of the system were very helpful. There was a split as to whether the discrete snapshots of GAIGS were more effective than the smooth animations of Samba. One student who preferred Samba remarked that the animation "allowed me to see where things go, what things go together to make the next step." Another who preferred the GAIGS visualization noted, "Animations are hard to follow. It takes some detailed thinking to get to the next step, and the animation doesn't wait for you to think." The students were unanimous in emphasizing the importance of the context-sensitive documentation windows, offering comments such as "The visual effects do help, but without the accompanying description of them, I would have no idea what's happening." Several students criticized that the particular documentation we provided with these two algorithms was not enough. For example, one said, "there should be a better description of how to interpret a particular picture." Finally, the stop-and-think questions were universally well received. According to one of the participants, "Questions are by far the feature most conducive to understanding the algorithm. They force you to figure out what's going on. However, I found that I was able to see the answers to the questions before I completely understood why those were the right answers."

4 Future Directions and Conclusions

In many respects, JHAVÉ just represents a starting point. There are a variety of directions to proceed from here:

1. We're anxious to have other computer science instructors try JHAVÉ. This can be done at several levels. If you merely want to try the algorithms we already have incorporated into JHAVÉ, point your browser at http://gaigs.cmsc.lawrence.edu. If you have a program that you used in the past to produce GAIGS or Samba visualization scripts, we would be happy to review that program and add it to the collection of visualizations that can be launched with JHAVÉ from our server. If you've written an AV system that could become an AV engine for JHAVÉ or if you would like to run JHAVÉ directly on your Web server, please contact us.
2. Much research needs to be done on how to effectively generate and use stop-and-think questions during the course of a visualization. These questions have to be generated in automated fashion by the same program that is producing the script files and are therefore highly dependent on the data the program is manipulating. There are significant computer-assisted-instruction and artificial intelligence issues involved in the effective production of such questions.
3. Although JHAVÉ is able to synchronize a visualization with context-sensitive documentation, the documentation we presently use is still static in that it knows nothing about the specific data in that image. For instance, the documentation might inform the viewer "the red node will be moved up a level in the tree," but it doesn't know enough to say "the red node *containing 4* will be moved up a level in the tree." This is because the documentation has been written statically, that is, before the execution of the algorithm that it tries to explain. It would be even more valuable if the documentation could be dynamically produced as the algorithm executes. It would then have much more awareness of the data being manipulated and could offer a much more specific explanation to the student.

In addition to these avenues for continued research, there are lessons learned from our experiences with JHAVÉ that apply to the design and use of all AV systems. In particular, AV can be effective in helping students achieve the first and second types of algorithm understanding that were cited in Section 1. What does this imply about the role of AV in a course like CS2 or an upper-level algorithms course? One particularly appropriate use of AV is to help students familiarize themselves with an algorithm before the lecture in which that algorithm will be covered. If we, as instructors, can expect that students will walk into class with a solid understanding of the steps involved in an algorithm, then we can spend much more class time on tougher topics like correctness proofs and efficiency analyses. Of course, used in this way, AV is analogous to a reading assignment, so the threat of a "manual trace" quiz at the beginning of class to make sure the assignment is done may be appropriate. Feedback from our test group indicated that JHAVÉ's stop-and-think questions lessen the intimidation factor of such a quiz. In effect, students know exactly what to expect because JHAVÉ has already provided them with an unlimited supply of practice quizzes during their preparation.

Our results reinforce those in of Stasko, Badre, and Lewis in [11] that accompanying textual explanations are absolutely essential to the effective instructional use of AV. As much as computer science instructors may complain about students' abilities to read an algorithm, having them read about it in the context of watching an accompanying visualization may be the ideal motivating factor. Even though we provided them with context-sensitive documentation windows, the members of our test group emphatically asked for more textual material to clarify the visualizations they were watching. In other words, perhaps our focus should change from AV being supplemented by textual materials to textual materials being augmented and motivated by AV. Though subtle, this reversal in perspective could be significant because it substantiates that our work as designers of visualizations does not stop at merely producing high quality graphics. To be truly effective instructional aides at the undergraduate level, AV systems must guide their viewer through a carefully orchestrated exploration of an algorithm. Designers of AV systems should consider techniques to provide such guidance to be of equal importance to the graphical rendering done by the system. Indeed, the incorporation of such techniques may well prove to be the difference between students' largely ignoring the system and their viewing it as an essential resource.

5 Acknowledgement

We are greatly indebted to John Stasko for providing us with the Java source code for his Samba animator [13]. With that as a starting point, we were able to add a limited rewind facility and fit it into the framework of an AV engine for JHAVÉ with relative ease.

6 References

[1] Byrne, M.D., Catrambone, R. and Stasko, J.T., "Do Algorithm Animations Aid Learning?", Tech. Rep. No. GIT-GVU-96-18, Georgia Tech Graphics, Visualization, and Usability Center, 1996

[2] deMarneffe, P.A., "The Problem of Examination Questions in Algorithmics," in Proceedings of the 3rd ITiCSE Conference (Dublin, Ireland, August, 1998).

[3] Hundhausen, C.D., "Toward Effective Algorithm Visualization Artifacts," Doctoral Thesis, University of Oregon, June 1999

[4] Lawrence, A.W., "Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding," Doctoral Thesis, Georgia Tech University, 1993

[5] Naps, T.L. and Bressler, E., "A multi-windowed environment for simultaneous visualization of related algorithms on the World Wide Web," in Proceedings of the SIGCSE Session, ACM Meetings (Atlanta, Georgia, February, 1998).

[6] Naps, T.L. and Chan, E., "Using Visualization to Teach Parallel Algorithms" in Proceedings of the SIGCSE Technical Symposium on Computer Science Education, New Orleans, Louisiana, February 1998.

[7] Rodger, S., "Integrating Animations into Courses" in Proceedings of the Conference on Integrating Technology into Computer Science Education, (Barcelona, Spain, June, 1996).

[8] Sears, A. and Wolfe, R., "Visual Analysis: Adding Breadth to a Computer Graphics Course" in Proceedings of the SIGCSE Technical Symposium, Nashville, TN, March, 1995.

[9] Sedgwick, R., Algorithms in C++, Addison Wesley, Reading, MA, 1990

[10] Stasko, J., "Tango: A Framework and System for Algorithm Animation" in IEEE Computer, September, 1990.

[11] Stasko, J., Badre, A. and Lewis, C., "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis" in Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems, (Amsterdam, Netherlands, April, 1993).

[12] Stasko, J., "Using Student-Built Algorithm Animations as Learning Aids" in Proceedings of the SIGCSE Session, ACM Meetings (San Jose, CA., February, 1997).

[13] Stasko, J., JSamba, described online at http://www.cc.gatech.edu/gvu/softviz/SoftViz.html.

[14] Wolfe, R., Grissom S., Naps, T. and Sears A., "A Tested Tool for Teaching Graphics" in Journal of Computing in Small Colleges (Proceedings of the Third Annual CCSC Midwestern Conference), Greencastle, IN, vol. 12, no. 2, November 1996.