Making Interactive Prediction Count In Algorithm Visualization — Yes, It Will Be On The Test!

Thomas L. Naps
Computer Science Department
University of Wisconsin - Oshkosh
naps@uwosh.edu

Stanley Makalew
Computer Science Department
University of Wisconsin - Oshkosh
makals@uwosh.edu

Abstract

In algorithm visualization, interactive prediction refers to the technique whereby viewers are presented with questions at key stages of an algorithm's execution. These questions ask the viewers to predict what will happen in future stages of the execution of the algorithm they are watching. In this paper we present an overview of work that has already been done in interactive prediction and discuss the effectiveness of that work. In the context of this background and perceived shortcomings of previous systems, we describe our current project in which interactive prediction is combined with distributed database technology to incorporate course management and evaluation facilities into an algorithm visualization system. We also discuss security issues involved in the development of such a system. These are particularly relevant when the instructor uses such facilities as a significant component of students' grades. Finally, we present somewhat inconclusive results obtained from a small-scale field test of the system.

Background

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, four in '99 ITiCSE proceedings, eight in '00 SIGCSE proceedings, nine in '00 ITiCSE proceddings, and three in '01 SIGCSE proceedings). However, despite the abundance of algorithm visualization tools now available, their promise as a pedagogical tool is largely unfulfilled. According to Baecker [1], "little of the work (in algorithm visualization) has been adapted by the mainstream of computer science education and practice." Stasko and Lawrence [11] maintain "algorithm animations used as passive videos of an algorithm's operations will have minimal impact on learning." The apparent correlation between students' passively watching AV and its ineffectiveness has spurred recent work in incorporating interactive prediction into visualizations. Essentially this means that the student's viewing of the visualization is interrupted by some form of questioning in which she must predict what the visualization will next show about the algorithm.

A study by Byrne, Catrambone, and Stasko [2] found that forcing students to do prediction during the animation of a depth-first search resulted in significantly better results on a post-test. The methodology used for this study was to have students in a closed lab setting watch an animation presented in the XTango system and have them orally predict what would happen at pre-defined breakpoints in the animation.
In related work, Korhonen and Malmi [5] have used their TRAKLA and TRED systems to have students manually trace the execution of an algorithm. Although not strictly AV in the sense we have defined above, the TRAKLA/TRED system offers students individualized problems in tracing algorithms and then provides them with a graphical data structure editor to record their responses to questions about the data structures. An example given in the Korhonen/Malmi paper is: "Insert the keys X H R U F G I J C M A K (data individualized for each student) in this order into an initially empty AVL tree. Give the tree (using the graphical data structure editor) both after 6 rotations and after all keys have been inserted." The response of the student using the editor is automatically assessed on-line, giving immediate feedback. Korhonen and Malmi report excellent results for their system, with over half of their students scoring 90 percent or above on the simulation exercises.

At the 2000 SIGCSE Symposium, we reported anecdotal success with the incorporation of stop-and-think questions into our JHAVÉ AV system [8]. Such questions are automatically inserted into our visualization at interesting events — precisely where we want to establish the importance of the next step in the 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.

The seeming promise of these efforts was dampened by results presented at the 2000 SIGCSE Symposium by Jarc, Feldman, and Heller [4]. Their Interactive Data Structure Visualizations (IDSV) software presents students with algorithm animations in one of two modes — Show Me or I'll Try. In the former mode students passively watch the animation, trying to learn the behavior of the algorithm. When the student switches to I'll Try mode, she is engaged with interactive predictive questions. Jarc, Feldman, and Heller carried out extensive statistical tests with their students using IDSV on eleven different algorithms. The results were disheartening — in summary form, they are:

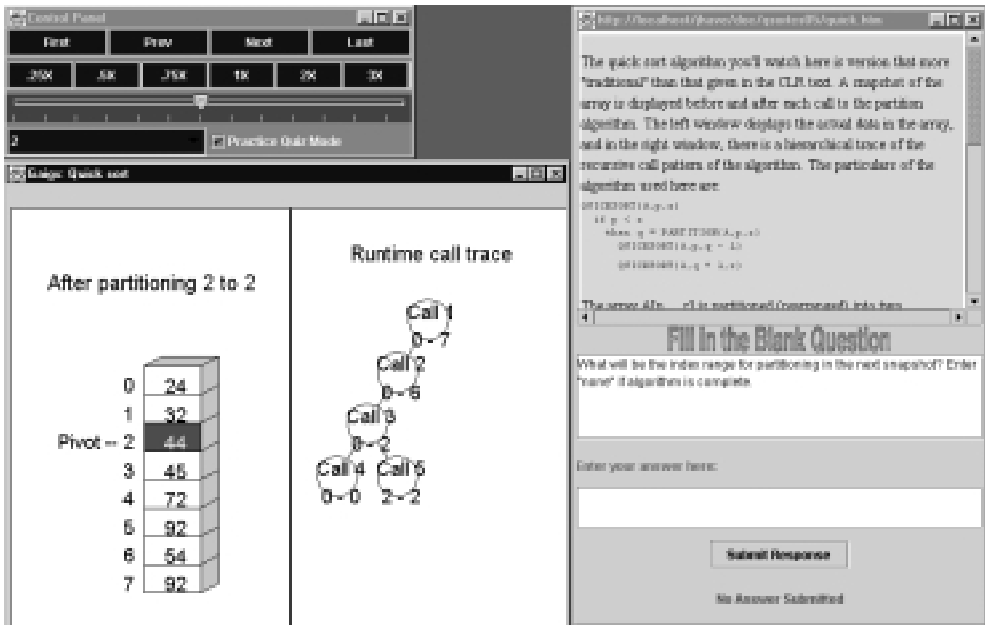
- The control group that used the interactive prediction feature of IDSV performed marginally, but not significantly, worse than the group that did not use it.
- Further breakdown showed (again with no statistical significance) that weaker students were hurt by using interactive prediction while better students were helped.

Jarc, Feldman, and Heller hypothesize that the reasons for these results are that the weaker students are "entertained" by the animations but tend to view interactive prediction as a guessing game. Their level of engagement is more that of someone playing a video game (Jarc, Feldman, and Heller's analogy) where quick responses are of the essence. Hence they fail to think through their answers to the questions that are posed, but instead guess at answers, either hoping to see a "correct answer pattern" that might emerge from the feedback they are given or, even worse, not caring anymore about the visualization because they are simply lost.

Rationale for Features Added to the JHAVÉ AV System

The preceding section has set the stage for the features we have added to JHAVÉ since first reporting on it in [8]. 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. In JHAVÉ's original configuration, there were two such engines -- one for the Samba animation scripting language designed by Stasko [10] and one for the GAIGS data structure visualization language developed by Naps [7]. Since developing that original configuration, we have worked in conjunction with Guido Roessling to add an engine for Roessling's Animal animation scripting language[9]. 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 the aforementioned 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. In JHAVÉ's client-server architecture, a remote client application begins a session by instantiating a connection to the JHAVÉ server application. This results in the client's receiving a listing of available algorithms. When the user selects an algorithm from that list, the client application 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. In addition to the rendering information, the script file syntax also incorporates the stop-and-think questions that are used to make the viewer interactively predict what future states of the visualization will show. Note that, because both the rendering information for the animation and the stop-and-think questions are dynamically generated each time a student requests a particular algorithm, the visualization server becomes, in effect, an infinite repository of exercises of the form "Trace algorithm X on data set Y". Figure 1 illustrates how a user interacts with a JHAVÉ visualization of quick sort that has been interrupted to have the student respond to a stop-and-think question.

Figure 1



The quick sort algorithm you'll watch here is version that more "traditional" than that given in the CLR text. A snapshot of the array is displayed before and after each call to the partition algorithm. The left window displays the actual data in the array, and in the right window, there is a hierarchical trace of the recursive call pattern of the algorithm. The particulars of the algorithm used here are:

```
QUICKSORT(A,p,r)
  if p < r
    then q = PARTITION(A,p,r)
      QUICKSORT(A,p,q - 1)
      QUICKSORT(A,q + 1,r)
```

The array A[p... r] is partitioned (rearranged) into two

What will be the index range for partitioning in the next snapshot? Enter "none" if algorithm is complete.

Features that we have added to JHAVÉ directly address the findings and weaknesses reported by Jarc, Feldman, and Heller. First, the problem of students becoming so lost in watching a visualization that they randomly starting guessing at answers to predictive questions should not be unexpected. For this reason, we need to give students the capability to rewind the animation to the point where they became lost. If AV systems are to be considered analogous to electronic textbooks, then we must remember how one reads highly technical material such as mathematics. We read a passage until we become lost or confused. We then backtrack (though not necessarily to the beginning of the passage) and re-read, trying to recapture the thread of understanding that we lost. The ability to backtrack and re-read is absolutely essential to meaningful reading of technical material. We maintain that similar backtracking abilities are essential to effectively using AV to achieve any non-trivial level of understanding. This was borne out in a study done by Stasko and Lawrence [11] in which students were quizzed on a pairing heap algorithm after studying it with the help of the XTango AV system. According to Stasko and Lawrence, "the most often cited negative comment (on the part of the participants) was the inability to rewind the animation. The participants said that, after an operation occurred, they often wanted to look at the heap as it appeared before the operation." Unfortunately such rewind facilities are not present in many current AV systems. However, all of the scripting language engines in JHAVÉ support a rewind capability. Moreover, Animal, the scripting language we have recently added, supports what Gloor

[3] calls a s*tructural view of the algorithm.* This allows the student to jump directly to selected key points of the animation by clicking on an element in this structural view. The feature goes beyond a rewind capability in that it requires the capability to jump to a specific animation state in either direction.

However, even if we provide a rewind capability, we have no assurance that students will see a need for using the AV system. Unfortunately, many students are driven largely by the goal of attaining a particular grade in a course. Ultimately, the question such students always seem to want answered is: "Will it be on the test?" (Perhaps we shouldn't be too critical here — giving students some idea of what we expect them to learn in a course can go a long way to helping us improve our teaching.) Our response? Incorporate on-line testing into the JHAVÉ AV environment. Hence in the new version of JHAVÉ students are presented with stop-and-think questions in two modes — practice mode and "for real" mode. In practice mode, stop-and-think questions are presented just as they were in the original version of JHAVÉ. Students have the option of answering them or not. When they are answering the questions in practice mode, they receive feedback, but their responses are not recorded. This feedback provided to the student is generated completely on the client, without any further communication to the server.

However, when the student chooses "for real" mode, her ability to move forward in the animation (and hence see future states of the algorithm) is disabled until the question is answered. Once the student responds to the question, the response is transmitted back to the server for evaluation and to be recorded in a server-side database. The result of this evaluation is then transmitted back to the client where the student is informed of her results on the completed question and allowed to move forward to future states of the visualization. Once recorded in the database, the results of students who have taken for-real quizzes may be viewed by the instructor, who may use them for grading purposes or merely to get some feedback on how well students are understanding a particular topic. Clearly, caution must be exercised in weighing these recorded results too heavily for grading purposes. Because students are taking these for-real quizzes at remote locations, there can be no real control over how the student took the quiz. For example, a given student may have had her best (and most talented) friend sitting at her side, coaching her through what the answers to the questions should be for the animation that she is viewing. Moreover, there are technical aspects to the client-server relationship in the JHAVÉ architecture that may allow a clever student to compromise the security of the fashion in which questions are answered and then evaluated.

Technical Security Issues in Generation and Evaluation of For-real Quizzes in JHAVÉ

As any other network application, JHAVE also has security issues that need to be considered. At this stage of development, we have not implemented solutions to most of them. However, we do realize their potential damage, and have started on techniques to manage them. These security issues include, but are not limited to, the textual presence of the answers in the animation script delivered from the server, the lack of error checking for the input to the server-side SQL database, the ability to decompile the Java class files that comprise the client application, and the ability to read packets that run through the network.

When a student takes a quiz, the server invokes a program that creates a script containing information as to how the visualization should be rendered and questions to the quiz. However, the script syntax requires that every question has to have an answer; therefore, at the end of the script the answer to each question is appended. Thus, if the student manages to read the script, she will be able to ascertain the answers to the quiz. Although the script itself will reside in the computer memory after being transferred from the server, there are several possible ways that a student can read it. The first approach, although it is relatively hard to do, would be for the student to read the computer memory using a hexadecimal editor. A more plausible approach that the student can employ is to read the script when it is being sent from the server. With the existence of software known as a "packet sniffer", it is possible that she can read the data that are sent from the server. There are potentially other ways that a student can surreptitiously read the script. As a result, we decided that the best way to safeguard the answers is to incrementally encrypt the script. At present, an encryption method has been implemented as part of the system; however, it has not been fully utilized. Before the script is sent to the client, it is encrypted to prevent anyone from reading the answers. Furthermore, each question will be encrypted with a different key. Each key will be sent to the client to decrypt a question *after* the student answers the previous question. Hence the student won't be able to read the question ahead nor obtain the answer from the script.

Another security issue that we examined is the possibility of a student strategically placing special escape characters in information that the client parses into a SQL statement transmitted from the client to the server. When the user enters her information, such as, username, this information is sent to the server as part of a SQL statement that creates a query or updates the database. However, since it used directly as part of the SQL statement, a knowledgeable user can purposefully place special escape characters, such as an apostrophe, to modify the conditional part of the SQL statement. This implies that a user might be able to bypass the identification checking, or even change other users' passwords. The apparent remedy for this is simply more error checking. We have to make sure that the user doesn't enter any special escape character that could potentially be used to alter the SQL statement constructed by the server. For tighter security, this error checking should be implemented on the server side, though this may cause more network traffic. One of the reasons that it is done on the server is because, as we will describe next, it is possible to create a "fake client."

A fake client is an application that connects to JHAVÉ server posing as a JHAVÉ client. In order to write such a program, a student needs to know the inner workings of the JHAVÉ client works, such as what port number it uses, and the structure of the messages that are sent to the server. Unfortunately, since the client application is written in Java, the student can use a JAVA decompiler to get the source code to the JHAVÉ client classes. If the student obtains the source code, it will not be hard for her to create a fake client. A fake client can be made for different purposes. For example, it can parse the script for taking a quiz, and send the answers, which are appended to the script, back to the server as if the student is really taking a quiz. Furthermore, built into the JHAVÉ client are certain instructor functions. These instructor functions enable the instructor to see how a student did in a quiz. They display the quiz and its visualization, along with the answers that the student submitted; therefore, by using a fake client a student could actually see other students' quiz results and answers. To guard against this type of intrusion, we will eventually have to place most error checking on the server instead of the client, because the fake client could otherwise simulate all error checking. We will also have to separate the instructor functionality from existing in the same application that the student uses, and create a new client for the instructor. Thus, it can be saved in a directory to which students do not have access physical access.

As a final note on these security issues, we remind the reader that they become an issue only if the results of the for-real quizzes weigh significantly in the determination of a student's grade. Our own usage has been to weigh them only a very small percentage. Doing this encourages students to take advantage of the system as one additional resource that they can use to prepare for particular types of questions on in-class exams, which count much more toward their final grade. However, we also realize that AV systems integrated with on-line testing will have to address the more stringent security issues we have raised here if they are to find widespread use in certain types of distance learning applications.

Early Results of Classroom-testing For-real Quizzes in JHAVÉ

To see if having students use for-real quizzes could make a difference, we conducted a small study in the fall semester offering of our file structures course. One of the topics studied in this course is hashing and collision-processing strategies. We created a series of four visualizations to illustrate the difference between linear collision processing, chained collision processing, quadratic collision processing and double hashing. A class of twenty-five students divided themselves into two groups -- those who agreed to take for-real quizzes in using the visualizations and those who did not. There were sixteen students in the first group and nine in the second. All students were informed beforehand that there would be questions on tracing collision-processing strategies on the next in-class exam and that participating in the for-real quizzes would be one additional way of preparing for those questions (in addition to using more traditional materials such as the textbook and class notes). Students were then free to choose the group in which they would participate. Our hypothesis was that the group using for-real quizzes in their preparation would do better on the hashing questions in the in-class exam than those that did not. Essentially the in-class exam played the role of a post-test that would indicate whether or not students learned the concepts better when using AV that included for-real quizzes. The results of this small-scale test were inconclusive. The group using AV with for-real quizzes had an average score of 9.1 (out of 10) on the post-test hashing questions. The other group scored at an average of 8.7 on these same questions. This difference in performance, while indicating that the for-real quizzes may have helped a bit, is not statistically significant.

Anecdotal responses from students regarding the use of JHAVÉ and for-real quizzes to help them prepare for an in-class exam provided encouragement. For instance, on the next in-class exam, which covered, among other topics, dynamic and extendible hashing, several students “complained” that we did not have JHAVÉ visualizations with accompanying quizzes to help them prepare. Clearly, for some students, there is a “comfort level” attained by having taken these online for-real quizzes that make them more confident when they walk into a regular in-class testing situation covering the same material.

It is also apparent that the small size of the two groups used in our test, along with the fact that students could choose which group they wanted to be in, violated basic rules of experimental design. The test was done in this fashion because we strongly felt that the academic integrity of the file structures course being taken by the students was of much greater importance than a perfectly designed statistical test. We wanted no students to feel that they were being coerced into one group or the other for the sake of some hypothesis that we felt compelled to prove, but rather to feel that they decided on their own whether or not to take advantage of one additional study aid. This spring we are planning to organize a much larger multi-campus test of effectiveness in which experimental design factors will be more carefully controlled.

References

1. Baecker, R., "Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science," in Software Visualization: Programming as a Multimedia Experience, The MIT Press, 1998, pp. 369-382.
2. 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
3. Gloor, P. A. User Interface Issues For Algorithm Animation. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 11, pp. 145–152.
4. Jarc, D.J., M.B. Feldman, and R.S. Heller, (March 2000) "Accessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware," in Proceedings of the SIGCSE Session, ACM Meetings, Austin, Texas.
5. Korhonen, A. and L. Malmi, "Algorithm Simulation with Automated Assessment" in Proceedings of the Conference on Innovation and Technology into Computer Science Education, (Helsinki, Finland, July, 2000), pp 160-163.
6. Lawrence, A.W., Badre, A, and Stasko, J., "Empirically Evaluating the use of Animations to Teach Algorithms," in Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO, October 1994, pp. 48-54.
7. Naps, T., (February 1990) "Algorithm Visualization in Computer Science Laboratories," in Proceedings of the SIGCSE Session, ACM Meetings, Washington, DC.
8. Naps, T., Eagan, J, and Norton, L., (March 2000) "JHAVÉ -- An Environment to Actively Engage Students in Web-based Algorithm Visualizations", in Proceedings of the SIGCSE Session, ACM Meetings, Austin, Texas.
9. Rößling, G., Schüler, M., and Freisleben, B. The ANIMAL Algorithm Animation Tool. *5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finland* (July 2000), 37–40.
10. Stasko, J. Samba Algorithm Animation System, 1998. Available online at: http:// www.cc.gatech.edu/gvu/softviz/algoanim/samba.html
11. Stasko, J., and Lawrence, A. Empirically Assessing Algorithm Animations as Learning Aids. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 28, pp. 419–438.