Effective Features of Algorithm Visualizations

Purvi Saraiya, Clifford A. Shaffer, D. Scott McCrickard and Chris North
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
{psaraiya|shaffer|mccricks|north}@cs.vt.edu

ABSTRACT

Many algorithm visualizations have been created, but little is known about which features are most important to their success. We believe that pedagogically useful visualizations exhibit certain features that hold across a wide range of visualization styles and content. We began our efforts to identify these features with a review that attempted to identify an initial set of candidates. We then ran two experiments that attempted to identify the effectiveness for a subset of features from the list. We identified a small number of features for algorithm visualizations that seem to have a significant impact on their pedagogical effectiveness, and found that several others appear to have little impact. The single most important feature studied is the ability to directly control the pace of the visualization. An algorithm visualization having a minimum of distracting features, and which focuses on the logical steps of an algorithm, appears to be best for procedural understanding of the algorithm. Providing a good example for the visualization to operate on proved significantly more effective than letting students construct their own data sets. Finally, a pseudocode display, a series of questions to guide exploration of the algorithm, or the ability to back up within the visualization did not show a significant effect on learning.

Categories and Subject Descriptors

F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems – *sorting and searching*; H.5.1 [Information Interfaces and Presentation (e.g., HCI)]: Multimedia Information Systems – *animations; evaluation/methodology*

General Terms

Algorithms, experimentation

Keywords

Algorithm visualization, heapsort, courseware

1. INTRODUCTION

Algorithms and data structures are essential topics in the undergraduate computer science curriculum. In our department, the sophomore-level data structures course is typically considered to be one of the toughest by students. Many teachers are trying to find methods to make this material easier to understand. One of the most popular methods currently being investigated is algorithm visualization and animation (an algorithm visualization is hereafter referred to as an AV). It is true that students can learn algorithms without using an AV. But motivated by the age-old adage "a picture speaks more than thousand words",[1] many researchers and educators assume that students would learn an algorithm faster and more thoroughly using an AV [3, 4]. Yet, while thousands of AVs can be found on the Internet, only a very few of these are of any use for learning a new algorithm.

The effectiveness of an AV is determined by measuring its pedagogical value [3, 4]. By pedagogical value of an AV we generally mean how much students learned about that algorithm by using the AV. In this way, the pedagogical value for an AV might be compared to some other method for learning about the algorithm such as a textbook or a lecture. Others have found that by actively involving students [4] as they watch an AV and making students mentally analyze what they are doing, the pedagogical value of the AV increases. Only about half of experimental studies conducted on AVs demonstrate a significant effect for the AV, and lack of significant improvement is often attributable to the AV not actively engaging the student. According to Hundhausen [4], participants can be actively involved with the AV by making them

- construct their own input data sets [6].
- do what-if analyses of an algorithm behavior [6].
- make predictions about the next algorithm steps [7].
- program the target algorithm [5].
- answer questions about the algorithm being visualized [3].
- construct their own AV [4].

Since our goal is to increase the effectiveness of instruction for algorithms and data structures, we feel that it is important to understand the underlying principles of what makes

[1]It is worth noting that this adage might take AV designers down the wrong path. As discussed in [4], the way in which AV technology is used is fundamentally more important than the presentation form of the AV itself.

for good AVs. We seek to identify a collection of key features that are generally exhibited by successful AVs. To this end, we first identified a candidate list of features, and then conducted experiments aimed at determining which features actually have a significant effect.

2. THE LIST OF FEATURES

To prepare an initial list of potential key features, we used an "expert study" approach. Three of us (Shaffer, North, McCrickard) are experienced instructors of data structures and algorithms courses, and we are also experienced both in designing and implementing visualizations and in evaluating usability [1, 9, 10]. We began our efforts by assembling a collection of AVs from the Internet on the heapsort algorithm. Heapsort was selected because a variety of AVs are readily available, it is an algorithm commonly taught in data structures courses, and it is of moderate difficulty to understand. In particular, there is an important distinction between physical and logical representations for the heap data structure that must be recognized in order to understand the heapsort algorithm.

The three member "expert panel" together viewed five heapsort AVs. We compared and contrasted the AVs, took observations about their different characteristics, and attempted to identify what features might increase algorithm understanding and promote active learning in students. After completing the review session, we re-analyzed our observations and comments to identify a specific list of key features that we hypothesize might be characteristic of good AVs. The features thus identified are as follows.

Ease of use: Interface (usability) flaws can easily ruin an AV. The typical user will use a given AV only once. He/she will not be willing to put much effort into learning to use the AV. If the interface is difficult to use, the student will need to make a conscious effort to learn it. This that can prove distracting to a user who is at the same time attempting to learn an algorithm. Likewise, any operational bugs in the program will make it impossible to use, or at least distracting. Any pedagogically useful AV must eliminate any usability and reliability concerns.

Appropriate feedback: AVs make assumptions about the background knowledge of their users. AV designers should make explicit what level of background is expected of the user, and support that level as necessary. Feedback messages and other information should be appropriate to the expected level of algorithm knowledge.

State changes: All AVs display a series of state changes caused by the algorithm acting on some data structure. Designers need to define clearly each logical step in the algorithm, and the visualization that is associated with each state or state change. Sometimes it is good to provide textual explanation for some state changes, with this text perhaps displayed as a series of messages within a particular area of the visualization.

Window management: If an AV uses multiple windows, it can be difficult to relate actions in one to corresponding actions and updates in another (e.g., text in one section can be difficult to relate to actions elsewhere within the AV). Implementors should make sure that when the user starts the AV, all windows are made visible to the user. AVs might allow the user to resize or reposition the windows in a way that he/she is comfortable with.

Multiple views: Many AVs show both a physical view and a logical view for a data structure. As an example, a heap can be viewed as an array (its typical physical implementation) or a tree (its logical representation). The AV can show clearly the distinctions and connections between these two views by showing both simultaneously.

User control: Different users have different rates for understanding and grasping information. Even the same user may need to work at different speeds when progressing through the different stages of understanding. Sometimes users might not "get it" the first time they are working with a test case. Thus, users should be able to control the speed of presentation, and they should be able to re-run a test case or perhaps be able to "back up" through a visualization.

Canned examples and data input: On the one hand, users are likely to benefit from good test cases that show them the best and worst performance of an algorithm. On the other hand, users should also be able to enter their own test cases that allow them to explore their questions about how the algorithm behaves.

Pseudocode: Pseudocode for the algorithm can help the user relate state changes directly to the algorithm's process. A pseudocode view can indicate which line of the algorithm is being executed at each step. Pseudocode should be easy to understand. Pseudocode will be clearer if presented using a high-level language rather than a particular programming language.

We note that the features identified by this process are generally oriented toward how an AV is used (and abilities provided to the user) rather than how the AV looks. This was not a conscious decision made *a priori* by the reviewers, but rather seemed a natural result to us. This result matches the conclusions from [4].

3. EXPERIMENT 1

The list of features in Section 2 represents our initial hypotheses for what constitutes effective AVs. Our next goal after compiling the feature list was to determine which if any have a significant impact on the ability of an AV to affect student learning. The feature list actually contains qualitatively different sorts of features. Most AVs are failures because they are buggy or their interface is so poor as to make them unusable. Yet however important this factor is, as a "feature" we must take fundamental stability and usability for granted in that it does not make sense to test whether a program that is unusable or has bugs is inferior to an otherwise equivalent one that is usable and bug-free. Likewise, the features of relevant feedback, clearly defined state changes, and window management are all useful directives for AV designers to keep in mind when creating an AV However, we have not thought of a way to test this hypothesis, in that these do not appear to represent capabilities that an AV might or might not have.

In contrast, the other proposed features represent capabilities that might or might not be provided to a user. An AV might permit user control of the pacing or not, it might allow users to enter examples or not, it might provide pre-defined examples or not, it might provide a pseudocode view or not, and it might present multiple views of the data structure or not. The work of Hundhausen et al. [4] and Hansen et al. [3] indicates that AVs are more effective when the user is interactive in the sense of being forced to answer questions about the algorithm, especially to predict future behavior. This capability can be provided by the AV in some manner.

We conducted an initial experiment with the goal of determining which of these capabilities make a significant difference to student understanding of the algorithm. We implemented our own AV for the heapsort algorithm in such a way that various features could be added or removed. The (optional) features were:

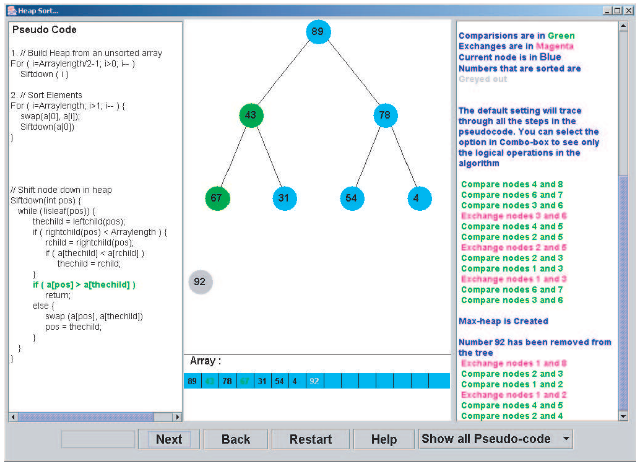
- The ability by users to enter their own data sets.
- An example data set given by the AV.
- A pseudocode display.
- A "back" button enabling users to back up any number of steps in the presentation.
- A "guide" which presented a series of questions meant to guide users through the AV and force them to engage with the content of the AV.

Since the space of possible feature combinations is fairly large, and our population of test subjects is limited, we then decided on five specific variations of feature sets to compare. We hypothesized that an AV having more features from the set would tend to improve pedagogical value, and therefore students using these features would show increased learning about the algorithm as compared to those students using an AV with fewer of these features.

Methods: We recruited as testers undergraduate students who had prior knowledge about simple sorting algorithms and knew basic data structures like stacks, queues, and linked lists. Most of these students had no previous knowledge about the heapsort algorithm or heaps, and had little knowledge about tree data structures in general. Students were asked to work (there was no time limit) with one of five variations of the heapsort AV defined below (see Figure 1). When the students thought that they were sufficiently familiar with the AV, they were asked to take a test on the heapsort algorithm. All variations used the same graphical representation of the algorithm. An important feature common to all these AVs was that students could control the speed of the algorithm's execution via use of a "next" button. Each time the "next" button was pushed, the AV advanced to the next logical step. Before starting the visualization, all students were provided with some background information on the heapsort algorithm.

The full-featured version of the heapsort AV used for the experiment is shown in Figure 1. The features associated with each of the five versions is shown in Table 1. The available features indicated in the table include a "next" button to control speed of the tutorial, a "back" button to back up one logical step in the presentation, an example given by the AV, the ability for the user to enter his or her own example, pseudocode, and a question guide, repsectively.

From the table, we see that **Version 1** had (compared to the others) minimal capabilities. It provided a pre-selected data set, but students were not permitted to enter their own test cases. The "back" button was not enabled, nor was there a pseudocode display. Students were given no guide questions. **Version 2** was slightly more sophisticated than Version 1. Students were permitted to enter their own data set, but were not given any example. They could use the "back" button to revert to previous steps in the algorithm. **Version 3** added a pseudocode display to the features of Version 2. Again, students were not provided with an example data set to start. **Version 4** was identical to Version 2, and added the guide. **Version 5** was identical to Version 3, and added the guide. The difference from Version 4 was the



Pseudo Code

```
1. // Build Heap from an unsorted array
For ( i=Arraylength/2-1; i>0; i-- )
   Siftdown ( i )

2. // Sort Elements
For ( i=Arraylength; i>1; i-- ) {
   swap(a[0], a[i]);
   Siftdown(a[0])
}

// Shift node down in heap
Siftdown(int pos) {
  while (!isleaf(pos)) {
        thechild = leftchild(pos);
        if ( rightchild(pos) < Arraylength ) {
            rchild = rightchild(pos);
            if ( a[thechild] < a[rchild] )
                thechild = rchild;
        }
        if ( a[pos] > a[thechild] )
            return;
        else {
            swap (a[pos], a[thechild])
            pos = thechild;
        }
  }
}
```

Comparisions are in Green
Exchanges are in Magenta
Current node is in Blue
Numbers that are sorted are
Greyed out

The default setting will trace through all the steps in the pseudocode. You can select the option in Combo-box to see only the logical operations in the algorithm

Compare nodes 4 and 8
Compare nodes 6 and 7
Compare nodes 3 and 6
Exchange nodes 3 and 6
Compare nodes 4 and 5
Compare nodes 2 and 5
Exchange nodes 2 and 5
Compare nodes 2 and 3
Compare nodes 1 and 3
Exchange nodes 1 and 3
Compare nodes 6 and 7
Compare nodes 3 and 6

Max-heap is Created

Number 92 has been removed from the tree
Exchange nodes 1 and 8
Compare nodes 2 and 3
Compare nodes 1 and 2
Exchange nodes 1 and 2
Compare nodes 4 and 5
Compare nodes 2 and 4

Figure 1: The heapsort visualization, with pseudocode display and back button enabled.

Version	Next	Back	Given Examp	Own Examp	Pseudo	Guide
1.1	X		X			
1.2	X	X		X		
1.3	X	X		X	X	
1.4	X	X		X		X
1.5	X	X		X	X	X
2.1	X			X		
2.2	X	X	X	X		
2.3	X	X	X	X	X	X
2.4				X		

Table 1: Features for each heapsort version, Experiments 1 and 2.

addition of the pseudocode display.

An important goal for our experiment is to find at least one feature set that provides for a significant difference in student performance over some other feature set. While it can be difficult to tease apart what makes an AV effective, unless we can show that at least some AV is more effective than another, there can be no hope of determining the cause of the distinction. This is one reason why we attempted to provide a version that appears to be devoid of (hypothesized) useful features and another that carries all of the (hypothesized) useful features in hopes of generating at least one significantly different pair of AVs.

Results: The following is a summary of the most relevent findings of the experiment. For complete details on this experiment, see [8]. When we say that a result is significant, we mean that the p value is less than 0.05.

66 students participated in the experiment. We omitted the performance of 2 students, as they had no prior instruction about sorting, and thus they were quite different from the target population for this AV. The average GPAs for the groups were similar. We evaluated pedagogical effectiveness based on a post-test containing 19 multiple-choice questions on the heapsort algorithm.

There was no significant difference in the performance of the participants as measured by a single-factor ANOVA on total test scores over all 5 variations of the AV. The overall performance indicated that the users of Version 1 performed somewhat better overall (not significantly) than users of the other versions, while those using Version 2 performed somewhat worse (not significantly).

When analyzing the results for individual questions, some differences emerged. A single-factor ANOVA on test scores for procedural questions only, over all 5 AV conditions, reported a significant effect ($p = .002$). The users of Version 1 significantly outperformed the users of Versions 3–5 on procedural (algorithm mechanics) questions ($p = .00026$, $p = .0424$, $p = .0080$, respectively). Users of Versions 2 and 4 performed significantly better than those using Version 3 on procedural questions ($p = .0251$ and $p = .010$, respectively). Thus, the results suggest that a simple AV that focuses just on the logical steps of an algorithm and shows its effect on the data would provide better understanding of the procedural steps of the algorithm for students who have no previous knowledge of the algorithm.

On a conceptual question that asked the memory requirements of the heapsort algorithm, an ANOVA over all 5 AV conditions reported a significant effect on participant performance ($p = .05$). Users of Version 3 performed significantly better than users of Version 4 ($p = .010$) and somewhat better ($p = 0.087$) than users of Version 2. On a conceptual question that asked participants to identify the array in max-heap format from a given set of arrays, an ANOVA over all 5 AV conditions reported a significant effect on participant performance ($p = .03$). Users of Version 5 significantly outperformed the users of Version 2 ($p = .00281$) and Version 4 ($p = .0459$). The users of Version 1 outperformed the users of Version 2 ($p = .029$).

From these results it seems that a pseudocode display and guide or a data example covering the important cases might provide better conceptual understanding of the algorithm. We were surprised that the participants who had a pseudocode display did not perform better on the pseudocode-related questions as compared to the participants who did not have a pseudocode display. In general, the results failed to demonstrate any real differences between the differing versions. However, the results did provide suggestive information that can be used to guide further experiments.

4. EXPERIMENT 2

The first experiment gave generally counter-intuitive results (essentially, that better performance was achieved by students using fewer features). We then hypothesized that two factors not tested for were dominating the results: having control over the pace of the algorithm, and having a good example. Thus, we decided to perform a second experiment to test the hypothesis that these two factors dominate the other features.

Methods: Students were asked to work with one of the four variations of the heapsort AV. When a subject decided that he/she was sufficiently familiar with the algorithm, they took a test on the heapsort algorithm. All four variations used the same graphical representation of the algorithm and the same textual feedback messages as in Experiment 1. Besides the AV, students were provided with some background information on the heapsort algorithm. A log file recorded the time a student spent with the AV and also the number of times that he/she interacted with the various interface controls. The following four versions were used. A summary of each version's features is shown in Table 1.

Version 1 had a limited number of features. Users entered their own data sets, but were given no example data set. This version had a "next" button (thus permitting users to control the pace of the visualization), but no "back" button, no pseudocode panel, and no question guide. **Version 2** was identical to Version 1 with the addition that users were given an example data set, and could use the "back" button to revert to previous steps of the algorithm. **Version 3** extended Version 2 by adding a pseudocode display and a question guide. Thus, this version had the most features. Unlike all other variations, **Version 4** animated the heapsort algorithm at a fixed pace. Students could halt the animation and change the rate of progress (i.e., set the rate at which steps of the algorithm took place), but could not directly control pacing using a "next" button. No example data set was provided, subjects had to enter their own data set. There was no "back" button, no pseudocode, and no question guide. Thus, this version represents the absence of optional features.

Results: The following is a summary of the most relevent findings of the experiment. For complete details on this experiment, see [8].

44 students, 11 for each version, participated in the experiment. The average GPAs for each group were similar. A single-factor ANOVA on total test scores over all 4 variations of the AV showed a significant effect ($p = 0.0002$), warranting further analysis of pairwise comparisons. Users of Versions 1, 2, and 3 significantly outperformed users of Version 4 ($p = .029$, $p = .0001$, $p = .006$, respectively). Versions 1 and 4 were identical except for the control of algorithm execution. Thus, providing users an absolute control on the pace of the AV proved to make a significant difference.

For procedural questions only, a single-factor ANOVA over all 4 variations showed a significant effect ($p = 0.0001$). Users of Version 2 significantly outperformed users of Version 1 on procedural questions ($p = .008$). Although there was no significant difference in performance between users of Versions 2 and 3, users of Version 2 performed marginally better than users of Version 3 both overall ($p = 0.1$), and on procedural questions ($p = .059$). Users of Version 2 performed significantly better than users of Version 4 on procedural questions ($p = .0001$). Thus, providing a good data example and having an absolute control on the algorithm execution proved helpful in procedural understanding of the algorithm.

Version 2 and Version 3 had every feature in common other than the pseudocode display and the study guide. These features appeared to provide only marginal improvement in performance. Analysis of learning time versus performance showed that a larger learning time (at least when that time is spent observing pseudocode) does not mean that participants understood the algorithm any better. Users of Version 3 spent almost double the amount of time with the AV as compared to users of Version 2. This shows that having the right set of features could reduce learning time (versus other feature sets), while using more features may result in increase in the learning time with little pedagogical benefit. This result contradicts [4] which suggests that more time spent with the AV translates into more learning. Our results suggest that some uses of time are more efficient for learning than others.

There was no significant performance difference on the conceptual questions on the test for Experiment 2 between the various groups. Results from Experiment 1 indicated that additional features like pseudocode and guide might prove to be helpful in increasing conceptual understanding of the algorithm, but the data are weak for this hypothesis. Note that [4] suggests that conceptual questions tend to be less useful to distinguish pedagogical advantage than procedural questions.

GPA vs. performance analysis on the tests for both Experiment 1 and Experiment 2 indicated that students having an extremely low GPA often performed relatively well using the AVs. This indicates that if AVs are used for teaching algorithms in class, these students might benefit so as to increase their relative performance overall. However, we did not conduct an experiment explicitly to test this hypothesis. This result might well support the finding of [2] that concrete learners benefit relatively more from AV.

5. CONCLUSIONS

Based on the current AV research, we hypothesized that there are certain characteristics that are common to pedagogically effective AVs. We then proceeded to compile a list of candidate features that we believed might increase the pedagogical effectiveness of an AV. To measure the effect of selected features with the goal of verifying that certain ones do affect the quality of AVs, we conducted two experiments using several variants of a heapsort AV. The following is a list of the key findings from our experiments:

- Allowing a user to step through the AV significantly improves the pedagogical value over allowing control of the rate at which the AV runs the presentation.
- Providing students with an example data set that covers the important cases in an algorithm significantly increases the pedagogical value of an AV.
- Participants having pseudocode display and an activity guide spend substantially more time with the AV as compared to simple AVs that do not have these features. These features do not appear to provide significant pedagogical benefit on procedural understanding.

These results suggest that we were correct in some of our hypotheses, in that certain features do significantly increase the pedagogical effectiveness of an AV, especially control of the AV pace and having a good example. But the experiments also indicated that we were wrong about the effectiveness of other features. A pseudocode display and using a history mechanism to revert to a previous step in the algorithm did not prove to have significant pedagogical effectiveness. Providing users with the ability to enter their own data sets does not provide significant pedagogical value. Finally, providing students with a question guide did not result in significant pedagogical value.

Our finding that students given a good example perform significantly better than students required to construct their own own examples appears to contradict the results of [6] and [3]. Their argument is that constucting test cases is a form of active engagement that should lead to increased learning. However, it does not surprise us that students are generally unable to provide good test cases with which to explore an AV, since these same students are notorious for having poor ability to generate test cases with which to debug their own programs.

While it surprised us that pseudocode displays and being able to back up the AV proved of no significance to learning, it does not seem too hard in retrospect to accept that these are relatively minor details compared to controlled pacing through a good tutorial example. Neither pseudocode display nor the ability to back up really engage the student. Our results do agree with similar research indicating that people prefer to read procedural instructions in pictoral form, even though they prefer to write procedural instructions in textual form [11]. Perhaps program code is effective for implementing algorithms but not for learning algorithms.

In contrast, it greatly surprised us that the question guide seemed to have no significant effect. This finding seems to contradict the conclusions of earlier work, such as that by Hundhausen [4]. It was expected that the guide, whose intent is to engage students intellectually with the material by forcing them to answer questions that require exploration of the algorithm, would increase learning. This issue deserves more study. We note that we did not test the pedagogical effectiveness of requiring students to predict the future behavior of the algorithm, which the work of Hansen et al. [3] suggests is a key feature.

6. REFERENCES

[1] V. Colaso, A. Kamal, P. Saraiya, C. North, S. McCrickard, and C. Shaffer. Learning and retention in data structures: A comparison of visualization, text, and combined methods. In *Proceedings of the World Conference on Educational Multimedia/Hypermedia and Educational Telecommunications (ED-MEDIA 2002)*, June 2002.

[2] M. Crosby and J. Stelovsky. From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia*, 4:147–162, 1995.

[3] S. Hansen, N. Narayanan, and Schrimpsher. Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 1, 2000.

[4] C. Hundhausen, S. Douglas, and J. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 2002.

[5] C. Kann, R. Lindeman, and R. Heller. Integrating algorithm animation into a learning environment. *Computers & Education*, 28:223–228.

[6] A. Lawrence. *Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding.* PhD thesis, Department of Computer Science, Georgia Institute of Technology, 1993.

[7] B. Price, R. Baecker, and I. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4:211–266, 1993.

[8] P. Saraiya. Effective features of algorithm visualizations. Master's thesis, Department of Computer Science, Virginia Tech, July 2002.

[9] C. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis.* Prentice Hall, second edition, 2001.

[10] J. Stasko and D. McCrickard. Real clock time animation support for developing software visualizations. *Australian Computer Journal*, 27(3):118–128, Nov. 1995.

[11] P. Wright, A. Lickorish, A. Hull, and N. Umellen. Graphics in written directions: Appreciated by readers not by writers. *Applied Cognitive Psychology*, 9:41–59, 1995.