

Designing Educationally Effective Algorithm Visualizations

Steven Hansen

Department of Modeling
and Simulations
Air Command and Staff
College
225 Chennault Circle
Maxwell AFB
AL 36112, USA

N. Hari Narayanan*

Intelligent & Interactive
Systems Laboratory
Department of Computer
Science & Software
Engineering
Auburn University
AL 36849, USA

Mary Hegarty

Department of Psychology
University of California
Santa Barbara
CA 93106, USA

To appear in *Journal of Visual Languages and Computing*, Academic Press, 2002.

*Direct all correspondence to this author. Postal address: CSSE Dept., 107 Dunstan Hall, Auburn University, Auburn, AL 36849, USA.
Tel: +1 334-844-6312 Fax: +1 334-844-6329 Email: narayan@eng.auburn.edu

Abstract

Despite the intuitively compelling adage “a picture is worth a thousand words,” attempts over the past decade to use animations to explain algorithms to students have produced disappointing results. In most cases interesting algorithm animations were designed, but no formal, systematic evaluations were conducted. When such evaluations were performed the results were mixed, with compelling evidence for the instructional superiority of algorithm animations failing to emerge. It is in this context that we embarked on a research program to develop educationally effective algorithm visualizations. This program was based on the premise that animations needed to be embedded in a knowledge and context providing hypermedia environment in order to effectively harness their power to enhance learning. This paper describes the architecture of the resulting Hypermedia Algorithm Visualization system HalVis. Four empirical studies with HalVis are described, which demonstrated that the extent of learning exhibited by students who used HalVis was significantly greater than that of students who used means of traditional instruction or a typical algorithm animation.

Keywords: algorithms; animation; empirical evaluation; learning; visualization.

1. Introduction

Computer science students generally find the analysis and design of algorithms to be a difficult subject. One reason for this may be that an algorithm describes a process that is both abstract and dynamic, while the methods typically used to teach algorithms are not. For more than a decade, researchers and educators have pursued the intuitively compelling notion that using computer animations to illustrate the dynamic behavior of an algorithm will prove effective in helping students overcome the difficult task of learning algorithms. Two implicit assumptions underlie this pursuit: graphical representations are better than textual explanations, and dynamic graphics is better than static graphics. But either of these is far from obvious [1]. For instance, Petre persuasively argues that “reading graphics” is an acquired skill that experts are better at [2], a notion at odds with the often unstated assumption that novice students will grasp an algorithm better if it is graphically presented.

Such concerns aside, most algorithm animation research papers present interesting algorithm animations or novel systems for creating and viewing algorithm animations. For instance, AACE [3] is an early system with several built-in algorithm animations constructed by experts. In contrast, Aladdin [4], AlgorithmExplorer [5] and Eliot [6] are systems that teachers and students can use to build their own algorithm animations. Collaborative Active Textbooks [7] and Internet Software Visualization Laboratory [8] are examples of systems that deliver animations over a network such as the web to remote students. Declarative approaches to describing and constructing algorithm animations are presented in [9,10]. However, even though such research is explicitly or implicitly aimed at educational applications such as learning about algorithms or writing and debugging programs, system-oriented papers are usually silent about whether the animations help students master the subject better than traditional means of learning and instruction. From an educational perspective therefore, this leaves open the question of whether the time and effort needed to build algorithm animations using such systems produce comparable learning benefits. This question is important because the investment needed is non-trivial, as noted in an article on experiences in software visualization based teaching at a major university [11].

Several researchers have indeed investigated the educational efficacy of algorithm animations. While pictures and animations are almost always enthusiastically received by the students [12,13], these studies have not produced

convincing proof that animations actually *improve* learning [13,14,15]. Hundhausen, Douglas and Stasko [16] provide a comprehensive survey of the literature on algorithm animation and its evaluation. In a recent paper on visual language design, Blackwell reports not finding evidence for the hypothesis that pictorial representations provide significant assistance in forming mental images of abstract situations or processes. In some cases, pictorial representations may in fact inhibit the formation of mental images of abstract situations [17]. Beyond the domains of algorithms and visual programming, researchers have begun to question the purported benefits of animations in general to aid comprehension when compared with informationally equivalent paper-based presentations such as a series of diagrams illustrating the same phenomenon [18,19].

Given this background, the present paper focuses on the design and use of animations that *do* help students better understand algorithms. We believe that previous attempts at using animation to teach algorithm behavior were unsatisfactory not because of a flaw with animation as a tool for learning, but because of the approach used to convey the animations. Our research is based on the hypothesis that animations are indeed powerful vehicles for effectively conveying the dynamic behavior of algorithms, but that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. By *enhanced learning*, we mean a better understanding of the description and behavior of an algorithm resulting in an increased ability to accurately answer questions about the pseudocode (a step-wise description of an algorithm using a mixture of natural language and mathematical notation) and operations of the algorithm. Building upon research in cognitive science and human-computer interaction on multimedia design [20,21], we have developed an architecture for multimedia presentations of algorithms, called Hypermedia Algorithm Visualization (HalVis). The term *visualization* suggests a richer process than merely watching an animation, and the term *hypermedia* reflects the use of multiple media, semantic links and other cognitive devices to help the student form accurate mental models of algorithms.

In the next section we present key features and the architecture of HalVis. Section 3 contains detailed descriptions of four empirical studies involving over a hundred students. Results from these experiments indicate that learning by interacting with hypermedia visualization of algorithms is significantly more effective than learning by reading a textbook, listening to a lecture, or interacting with an algorithm animation typical of extant research on this topic. A discussion of the implications of these experiments appears in Section 4.

2. Hypermedia Algorithm Visualization

2.1. Key Features

This algorithm visualization architecture embodies several aspects of the *learner-centered design* approach [22]. We consider an algorithm visualization to be more than an algorithm animation. A *visualization* provides an interactive environment that elicits active student participation using a carefully orchestrated presentation of information in various media (such as animations, text, static diagrams, aural narratives, etc.) with appropriate temporal, spatial and hyperlink connections. Our position is that visualization involves more than the visual. In other words, attracting the visual attention of a learner through engaging and informative animations is only one part of the visualization process. Additionally, features of an interactive environment such as those described below designed to attract the cognitive attention and engage the mind of the learner are also critical elements of an effective visualization. These key features of the HalVis Architecture are described below.

(1) *Embedding animations within a hypermedia environment that also employs textual descriptions, static diagrams and interactive examples to provide contextual information.* The focus is not on the animation, but on providing relevant and sufficient information in cognitively appropriate media [23] to support the achievement of specific learning objectives. The learning environment within which animations are embedded provides a context for learning, a major characteristic of learner-centered design.

(2) *Embedding an animated analogy in the very first view of the algorithm that a student sees.* The motivation behind this feature is two-fold. First, it has already been observed that students naturally tend to employ analogies in describing how algorithms operate [24,25]. Second, analogies serve as scaffolding for comprehension of the procedural details of an algorithm from subsequent visualizations. Scaffolding is the notion of providing support to a learner to accomplish a learning task that he or she could not have successfully accomplished otherwise, a support that eventually the learner will no longer need [26]. Scaffolding is another key aspect of learner-centered design.

(3) *Providing three distinct kinds of animations to illustrate qualitatively different views of algorithm behavior.*

Initially, an algorithm's essential behavior is illustrated using animated analogies based on familiar situations. This interactive presentation serves to ground the abstract components of the algorithm in a familiar real-world example.

The animation of the analogy and accompanying textual explanations provide a bridge between events in the analogy and abstract steps of the algorithm. The second kind of animation is a micro-level animation of specific algorithmic operations on a small data set (about 10 elements) in tandem with pseudocode highlighting and textual explanations. The third kind of animation is a macro-level animation that illustrates the algorithm's aggregate behavior and performance characteristics on a relatively large data set (about 50 elements). Learners can interact with all three kinds of animations in various ways. Providing multiple and multimodal representations for learners to peruse and manipulate is the third critical property of learner centered-design.

(4) Presenting algorithm animations in discrete segments accompanied by explanations of the specific actions being accomplished. By segmenting animations into meaningful “bite-sized” pieces and providing logical pauses between segments, the student is able to better digest the abstract, dynamic information being presented. Allowing the student to adjust the level of segmenting tailors the flow of information to meet individual needs. Furthermore, students can provide their own data for the animation or select between different kinds of data sets. All these functionalities together form a flexible interface for the learner – this is the fourth important characteristic of learner-centered design. This is in contrast with most current algorithm animation systems which present the detailed dynamics of the algorithm as a one-shot, stand-alone show that is entertaining to watch but tends to obscure the very details a student needs to learn.

(5) Encouraging student participation by allowing rich interactions with the animations and using probes or questions that stimulate thinking and foster self-explanations. Students can (and are prompted to) explore algorithm behavior more thoroughly by running the micro-level animation multiple times with data of their own choosing. Students can also select from different data sets for the macro-level animation. HalVis periodically poses questions to the student. There are two kinds of questions. One is called a “tickler”, which is a question that pops up in random order but always in an appropriate context. Tickler questions are open-ended, focus student attention on specific issues, challenge their understanding, and promote self-explanations [27] to improve comprehension. Their answers are not entered into the computer nor is feedback provided. Multiple choice questions are placed at “articulation points” between modules of the visualization. These require students to enter answers and immediate feedback is provided by the system. Incorporating interactive devices to engage and motivate the learner is the fifth characteristic of learner-centered design.

2.2. System Architecture

Figure 1 shows the architecture of HalVis. It consists of four main components and two supporting components. The first component explains the core operations of an algorithm and illustrates those using a familiar analogy. The second component is designed to aid the student in understanding the pseudocode by presenting it side-by-side with a textual explanation. Technical terms in this explanation are hyperlinked to definitions and additional illustrations of fundamental algorithmic principles in one of the supporting components. The third component presents four representations simultaneously. One is a detailed, or micro-level, animation of the algorithm on a small data set. The second is the pseudocode of the algorithm in which steps are highlighted synchronously with the animation. The third is a verbal explanation of the events occurring in the animation. The fourth is a panel of variables that illustrates how their values change as the algorithm executes. These are qualitatively different representations of the same process, intended to help with the construction of a dynamic mental model [20,21]. Occasionally, “tickler” questions are generated and presented to the learner by this component while the animation is paused. The fourth component is intended to reinforce the learner’s mental model by presenting a macro-level animation of the algorithm’s operation on a much larger data set, without these other representations. It also allows the student to make predictions about parameters of the algorithm’s behavior and compare predicted values against actual values. Another supporting component provides different kinds of questions. The six components are grouped into five modules, as explained below. This architecture was implemented using Asymetrix Toolbook.

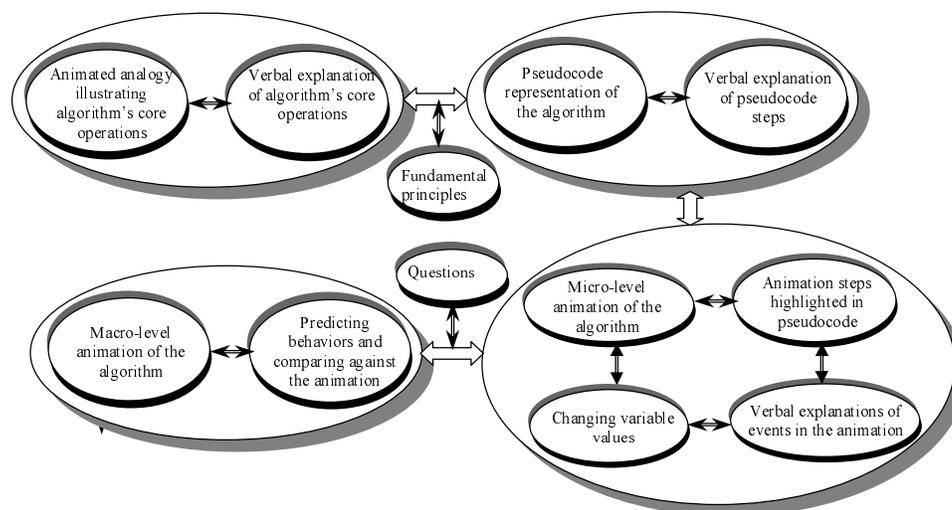


Fig. 1. Architecture of HalVis

(1) *Fundamentals*: This module contains information about basic building blocks of algorithms common to virtually all algorithms. Examples of concepts explained in this module include comparing & swapping data, looping operation and recursion. This module is not directly accessible to the student. It can be accessed only through hyperlinks from other modules, so that the basic information is presented *on demand* (in response to a learner request in the form of clicking on a hyperlink) and *in context* (of algorithm-specific information within which the hyperlink is embedded).

(2) *Conceptual View*: This module introduces a specific algorithm in very general terms using an analogy. For instance, the analogy for the MergeSort algorithm is animated playing cards that divide and merge to create a sorted sequence (Figure 2). This module uses animations, text, and interactivity to provide the student with an understanding of the essential aspects of the algorithm.

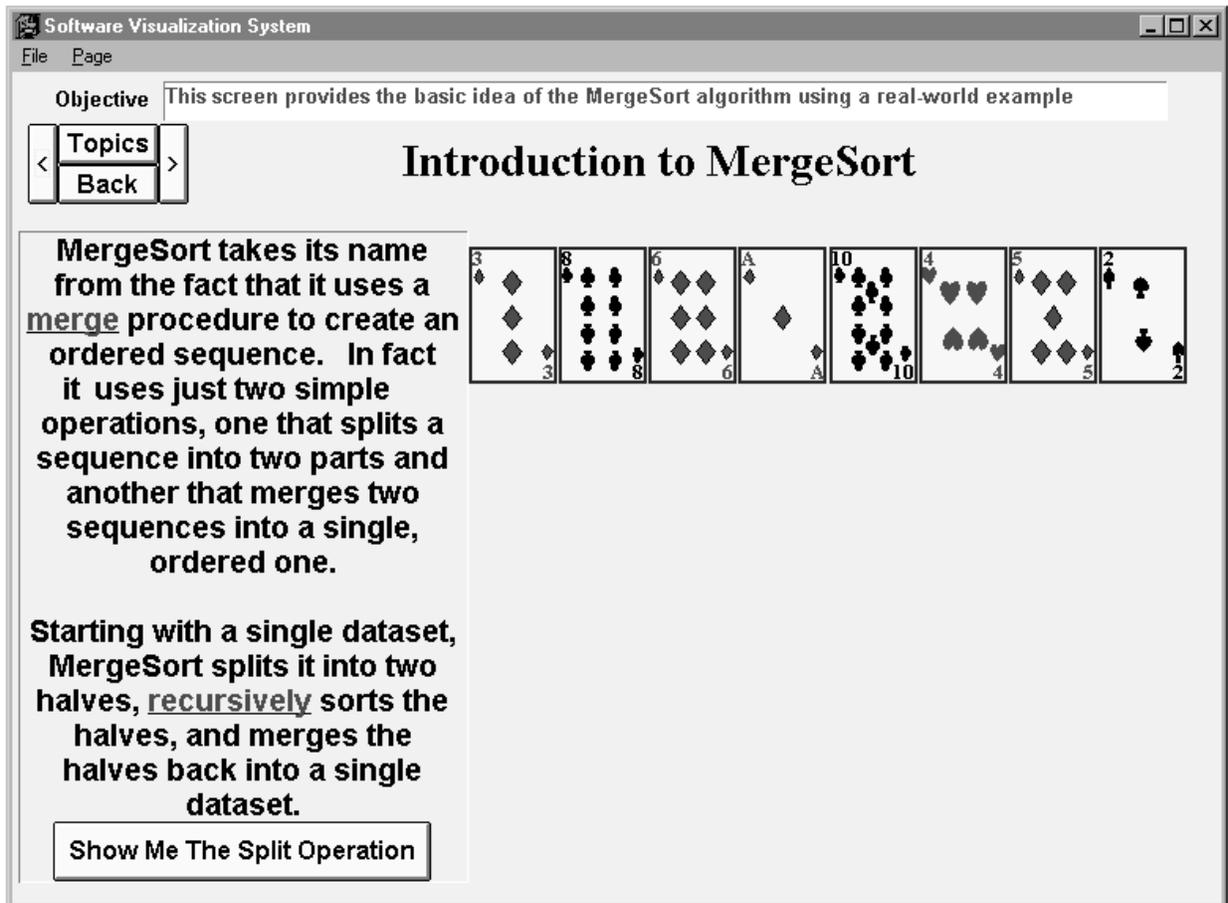


Fig. 2. Conceptual View of the Merge Sort Algorithm

(3) *Detailed View*: This module describes the algorithm at a very detailed level using two presentations. One consists of a textual explanation of the algorithm alongside a pseudocode representation of it. Embedded in the text are hyperlinks to related information in the Fundamentals module. The second presentation (Figure 3) contains four windows that depict various aspects of the algorithm's behavior. The Execution Animation window shows how steps of the algorithm modify data using smooth animation. The animation is segmented at multiple levels of granularity corresponding to meaningful units of the algorithm's behavior, with the level of segmenting selectable by the learner. At the lowest level, the animation displays the execution of an individual statement, pausing for the learner's signal to proceed. The next level corresponds to a logical operation, like completion of a single execution of a loop. At the highest level, the animation proceeds to completion without pausing. The Execution Status Message window provides comments and textual feedback to the student about key events and actions during execution. The Pseudocode window shows the steps involved in the algorithm, which are highlighted synchronously with the animation. Finally, the Execution Variables window displays a scoreboard-like panorama of the variables involved in the algorithm and their changing values. Before launching the animation, students can change the data input to the algorithm as well as the speed and granularity of animation and feedback. Execution of each step of the algorithm affects the display in the four windows simultaneously. Figure 3 shows seven data elements to be sorted using the MergeSort algorithm. When the user presses the ShowMe button, the four windows spring to life, moving the seven data items as needed and pausing between animation segments until the execution is finished. HalVis intentionally limits the number of data items in the Execution Animation window to focus attention on the micro-level behavior of the algorithm.

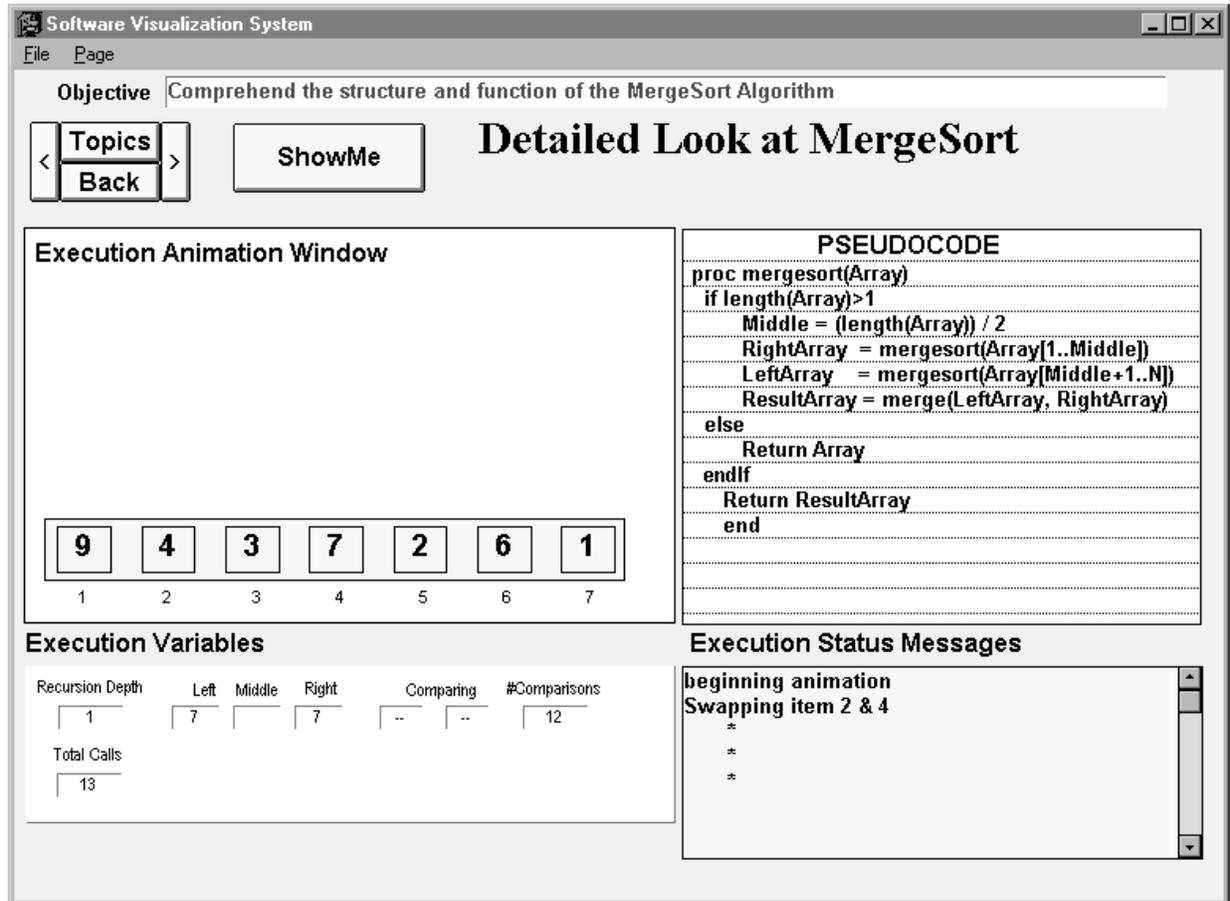


Fig. 3. Detailed View of the MergeSort Algorithm

(4) *Populated View*: This module provides an animated view of the algorithm on large data sets to make its macro-level behavior explicit. In this view (Figure 4), many of the details presented in the Detailed View are removed to allow the student to concentrate on the algorithm’s aggregate performance. Animations embedded in this view are similar to algorithm animations found in earlier systems. A novel feature is a facility for the student to make predictions about different parameters (such as the number of comparisons, swaps, or recursive calls) of algorithm performance and then compare those against the actual performance when the animation is running. When the learner presses the ShowMe button, the system prompts the learner to choose a data set (random, already sorted in the ascending order or already sorted in the descending order) and to make predictions. Then the animation begins. Color coding is used to convey information such as already processed data and data elements currently being processed.

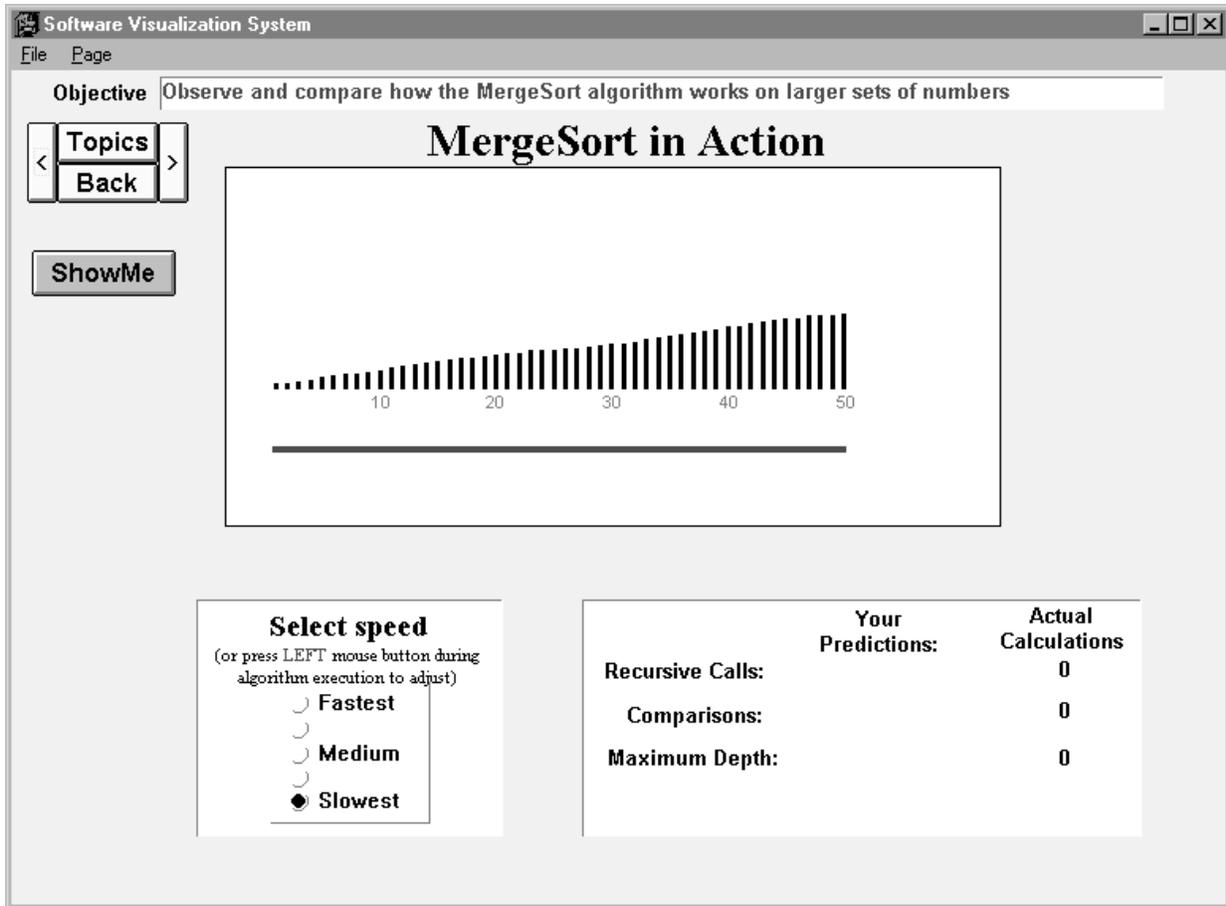
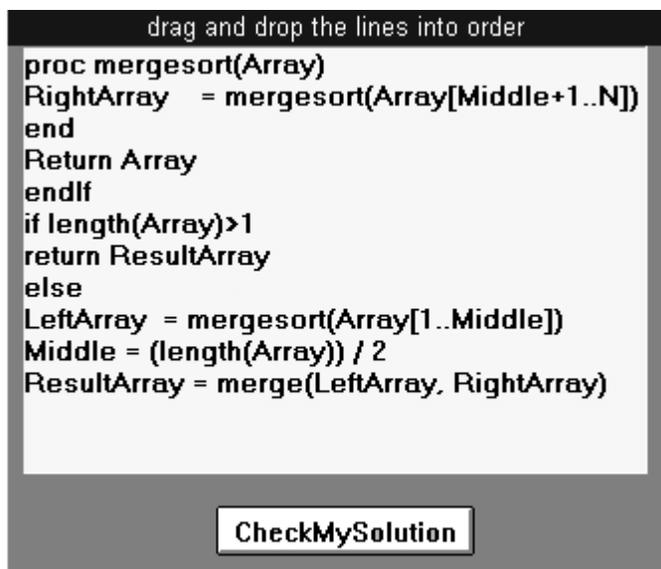


Fig. 4. Populated View of the MergeSort Algorithm

(5) *Questions*: This module presents the student with several questions. A combination of multiple choice, true-false, and algorithm step reordering questions are provided. Students get immediate feedback on their answers. Multiple choice and true-false questions appear with radio buttons or check boxes for indicating answers. The pseudocode-reordering question shows steps of an algorithm in a jumbled order (Figure 5). The student can then drag and drop the steps into a different order and have his/her solution checked for correctness.



```

drag and drop the lines into order
proc mergesort(Array)
RightArray = mergesort(Array[Middle+1..N])
end
Return Array
endIf
if length(Array)>1
return ResultArray
else
LeftArray = mergesort(Array[1..Middle])
Middle = (length(Array)) / 2
ResultArray = merge(LeftArray, RightArray)

```

CheckMySolution

Fig. 5. An Example from the Questions Module

3. Experiments on Learning from Hypermedia Algorithm Visualization

3.1. Summary

We designed and conducted a series of experiments, summarized in Table 1, to evaluate the effectiveness of hypermedia algorithm visualization, specifically to test our conjecture that such visualizations are more beneficial than traditional teaching methods and algorithm animations. In these experiments we measured the learning of students by their performance in pre-tests and post-tests. The first two experiments compared learning from HalVis to learning from textbooks alone. The third experiment compared learning from HalVis to learning from lectures. The fourth experiment compared learning from HalVis to learning from the combination of an algorithm animation typical of extant research and a textbook description. Even though these experiments compared stand-alone learning with HalVis, we do not intend HalVis as a stand-alone system. Rather, we believe that it needs to be incorporated into the curriculum so that working with HalVis supplements other instructional activities such as reading the textbook, attending lectures and doing labs. However, if in these controlled experiments HalVis proved to be significantly more effective than other means of learning, then it would be likely that HalVis incorporated into a curriculum would serve to enhance its overall effectiveness.

Table 1. Experiment Summary

	Level	Comparison Groups		Algorithm(s)
Experiment I	sophomore	HalVis	Textbook	MergeSort (MS)
Experiment II	junior	HalVis	Textbook	MergeSort QuickSort (QS)
Experiment III	sophomore	HalVis	Lectures	SelectionSort MergeSort
Experiment IV	junior	HalVis	Textbook and Animation	Shortest Path (SP)

3.2. General Procedure

In this section we describe procedures common to all four experiments. Deviations from these are explained when discussing individual experiments. In all experiments, volunteer subjects received extra course credit for their participation. They completed a demographic survey providing data on course performance, GPA, and ACT/SAT scores. This information was used to rank the participants, and create a matched pair of experimental and control groups.

The experimental group generally worked with HalVis. Members of this group met in a public computer laboratory. Prior to the start of any experiment, the group was given a 5-minute introduction to HalVis, which oriented subjects to the various screens they would encounter and provided them with basic navigational tips. The students were then assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm in question. The computers were Pentium-class systems with 15-inch color monitors. Members of this group were not given any supplemental materials to study.

In all cases, we ensured that an experiment took place prior to the algorithms used being covered in the course from which volunteers were drawn. No time limit was imposed for either group. After each subject indicated he/she was done with the study phase, he/she was given a post-test. Upon completion of this test, the subject was thanked and allowed to leave.

A pre-test/post-test combination was used to measure individual learning performance. These tests contained questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to (1) recognize and reorder pseudocode descriptions of algorithms, (2) mentally simulate algorithmic

operations, and (3) predict resulting data structure changes. The pre-test measured prior knowledge about an algorithm, and the post-test measured change in knowledge due to experimental treatments. Examples of each kind of question are provided below for illustrative purposes.

Probing conceptual knowledge: “What is the name of the element used in QuickSort to partition the input array?”

Probing procedural knowledge: “Consider the pseudocode of the MergeSort algorithm (shown as part of the question). Does it sort numbers in the input array in the ascending order or the descending order? Make necessary changes to the pseudocode to make it produce ascending order sort (unless it already does so).”

Recognizing and reordering algorithm descriptions: “Consider the pseudocode description of an algorithm shown above. Which sorting algorithm does it implement?” See Figure 5 for an example of the pseudocode-reordering question.

Mentally simulating algorithmic operations: “If the array of numbers {52, 91, 7, 41, 34, 10, 55} is passed to the MergeSort algorithm as input, what is the array passed as input to the recursive call on line 5 (the pseudocode is given as part of the question) when it is executed for the second time?”

Predicting changes to data structures: “If the array {52, 91, 7, 41, 34, 10, 55} is passed as input parameter to the MergeSort algorithm, produce a diagram showing the sequence of recursive calls. Show both the arrays passed as input parameters to each recursive call, and the resulting arrays when the calls are completed.”

3.3. Experiment I: Do Novice Students Learn More from HalVis than from a Typical Textbook

Chapter?

3.3.1 Subjects

This experiment involved students relatively new to the topic of algorithms. Twenty-eight second-year undergraduates enrolled in an introductory data structures and algorithms course at Auburn University volunteered to participate. We chose to use the MergeSort algorithm for this experiment because novices find its recursive structure somewhat challenging to comprehend. Subjects were assigned to a matched pair of groups: one group (called the "Text" group) would learn about the MergeSort algorithm using textbook descriptions, and another group (called the "Algorithm Visualization" (AV) group) would learn the MergeSort algorithm using the HalVis algorithm visualization tool. Twelve students in the Text group and sixteen students in the AV group completed the experiment.

3.3.2 Materials

Text Group: The Text group received a photocopied 6-page section from their course textbook [28]. This section contained a description of the MergeSort algorithm, its analysis, various diagrams, and program code. AV Group: The AV group learned about the MergeSort algorithm using the HalVis system with no supplementary materials provided.

3.3.3 Procedure

We timed the experiment to follow class lectures covering basic program design and fundamental data structures, but precede lectures that covered sorting algorithms. Participants were first asked to complete a pre-test about the MergeSort algorithm. The pre-test results helped us verify that the two groups were evenly balanced, and provided a baseline against which to compare subsequent changes. The pre-test scores indicated that the subjects did not know this algorithm and that the groups were matched (Average = 27% for the Text group and 28% for the AV group). This was followed by the experimental treatment and post-test. No student in the AV group took more than 60 minutes for the entire experiment. No student in the text group took more than 45 minutes for the entire experiment.

3.3.4 Results

The overall results are shown in Figure 6. The pre-test results indicate that both groups were equally unfamiliar with the MergeSort algorithm. The post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 74% compared to the Text group's 43%, and the results are significant for both the overall performance ($F(1,27)=10.9, p<0.003$) and for pre- to post-test improvement ($F(1,27)=6.7, p<0.015$). The statistical summary is shown in Table 2 (SD: Standard Deviation).

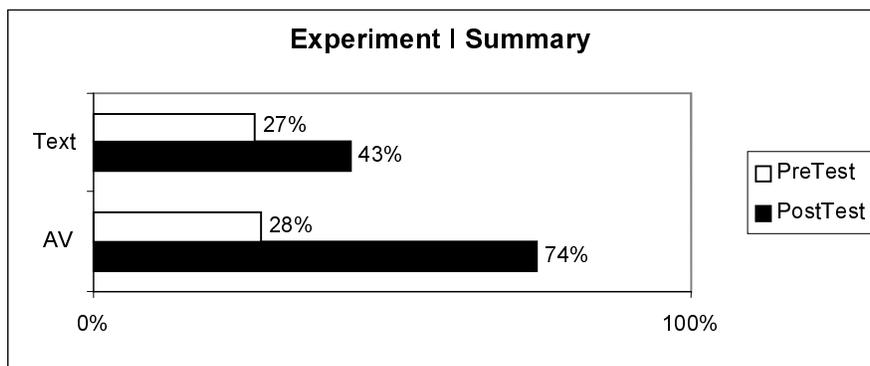


Fig. 6. Experiment I Overall Summary

Table 2. Experiment I Statistical Summary

Statistical Summary			
	Pre-Test	Post-Test	Improvement
Text Group	27% (SD = 28.61)	43% (SD = 31.08)	16%
AV Group	28% (SD = 28.09)	74% (SD = 19.77)	46%
F(1,27)	0.01	10.9	6.7
Significance level	p<0.93	p<0.003	p<0.015

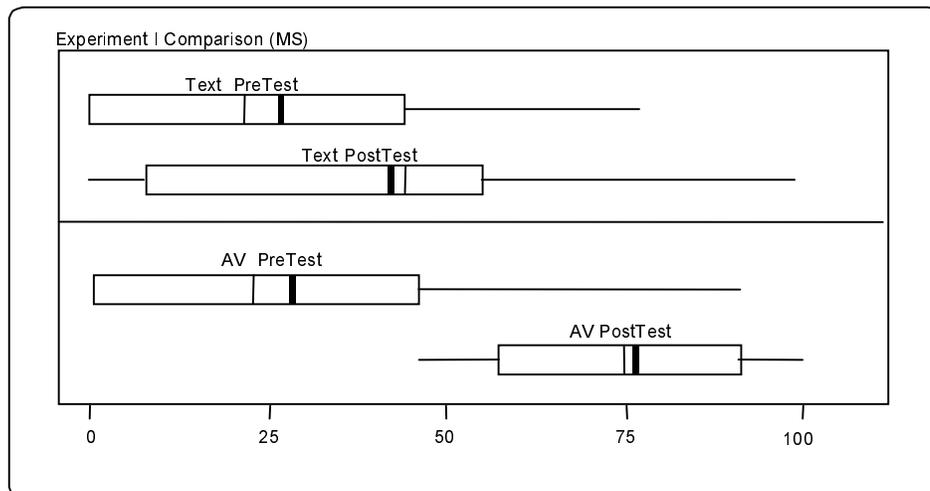


Fig. 7. Experiment I Box Plots

Another way of visualizing these results appears in Figure 7. In these box plots, the boxes indicate the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in a box indicates the mean, and the thin vertical line represents the median value for each group.

3.3.5 Discussion

These results suggest that novice students better learned conceptual and procedural aspects of the MergeSort algorithm after working with HalVis than after studying a typical textbook. However, there are some factors that

must be mentioned to keep these results in perspective. First, differences in student motivation between the groups could have influenced the results. The level of enthusiasm observed in the HalVis group was much higher than that of the Text group. The novelty of the visualization and the interactive features of HalVis seemed to engage the students' interest. In contrast, there was nothing new or uniquely motivating for the Text group. The motivational argument in favor of algorithm animations has been made by other researchers as well [12].

Second, familiarity with the learning materials could have had an influence. The Text group did not have to acquaint themselves with a new user interface, software system, or learning by interacting with visualizations. They were already familiar with reading and learning from a textbook. The students in the AV group had to contend with a new interface and a different way of learning. None of them had used an interactive visualization system before. If this factor indeed played a role, the AV group still exhibited a higher level of comprehension despite any additional cognitive effort involved in learning the interaction and navigation facilities of HalVis.

Third, one could argue that a different textbook could have led to different results. We chose a textbook that has been in use at Auburn University for more than five years. During this time several faculty have taught the first algorithms course that undergraduate computer science students take (about a hundred students go through this course each year) using this textbook. The general opinion among faculty who have taught the course is that it is as good as most other textbooks. We also know that this textbook is used at several other universities. These facts support our belief that the textbook extract used in the experiment is a representative one.

Last, only novice students participated in this experiment. It is possible that more advanced students may be more capable of learning from a textbook explanation of an algorithm. The next experiment investigated this.

3.4. Experiment II: Do Advanced Students Learn More from HalVis than from a Typical Textbook Chapter?

This experiment was similar to Experiment I in that the goal was to compare the effectiveness of learning using HalVis to learning from text. Our aim was to test whether the results of Experiment I could be replicated with more sophisticated algorithms and more advanced students. In this experiment we asked third-year undergraduates to learn two algorithms: MergeSort and QuickSort.

3.4.1 Subjects

This experiment involved 22 undergraduate computer science students enrolled in a third year algorithm design and analysis course at Auburn University. Participants were assigned to a matched pair of groups as before: a “Text” group and an “Algorithm Visualization” (AV) group. Eleven students in the Text group and eleven students in the AV group completed the experiment.

3.4.2 Materials

Text Group: The Text group received a ten page photocopied extract from the course textbook [29] that discussed the MergeSort and QuickSort algorithms. AV Group: The AV group learned about MergeSort and QuickSort algorithms using the HalVis system with no supplementary materials provided.

3.4.3 Procedure

A procedure similar to the previous experiment was used. Subjects in the AV group completed the experiment in less than 120 minutes, and subjects in the Text group finished in less than 90 minutes.

3.4.4 Results

The overall results are shown in Figure 8. Pre-test results indicate that both groups were equally unfamiliar with both algorithms. Post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 63% compared to the Text group’s 44%, and the results are significant for both the overall results ($F(1,21)=4.96$, $p<0.038$) and for improvement ($F(1,21)=9.29$, $p<0.006$). The statistical summary is given in Table 3, showing both aggregate and algorithm-specific results for each group.

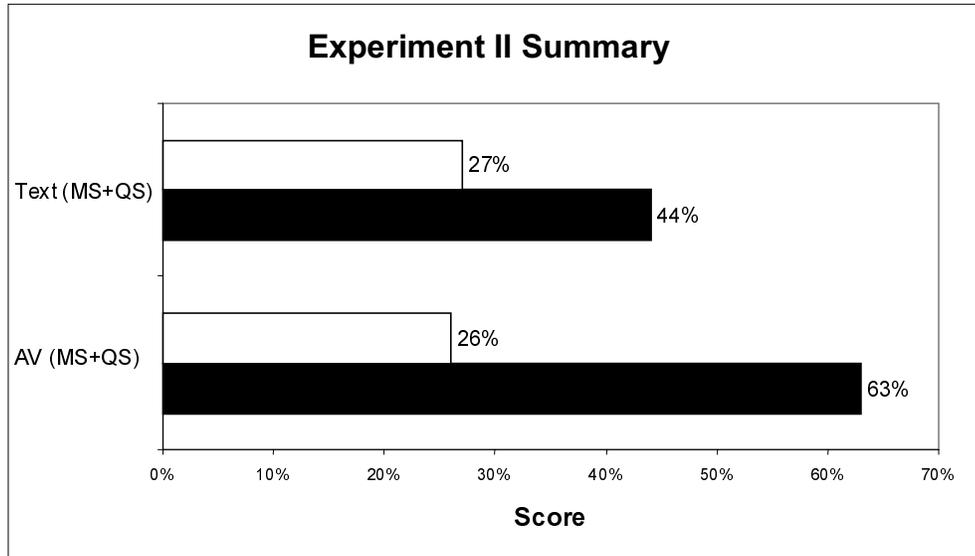


Fig. 8. Experiment II Overall Summary

Table 3. Experiment II Statistical Summary

Statistical Summary			
	Pre-test	Post-test	Improvement
Text (MS)	44%	53%	9%
AV (MS)	48%	71%	23%
Text (QS)	10%	35%	25%
AV (QS)	4%	55%	51%
Text (MS+QS)	27%	44%	17%
AV (MS+QS)	26%	63%	37%
F(1,21)	0.02	4.96	9.29
Significance level	p<0.89	p<0.038	p<0.006

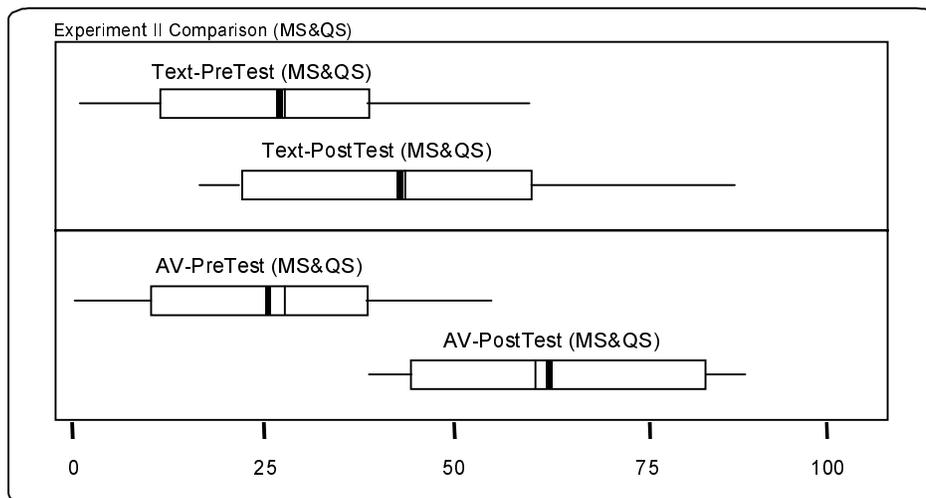


Fig. 9. Experiment II Box Plots

The results are summarized as box plots in Figure 9. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group.

3.4.5 Discussion

These results parallel those of Experiment I in suggesting that students perform better in answering conceptual and procedural questions about the MergeSort and QuickSort algorithms after using HalVis to learn than after studying a typical textbook. As Table 3 indicates, AV groups improved their performance by nearly three times (from 26% to 63% correct). It is interesting to note that in both experiments, students spent more time with the visualization than with the textbook. In the second experiment, they spent almost twice as much time. As they had complete control over how long they wanted to study, this indicates a higher level of motivation, translating into longer study times and better learning. As shown in Table 4, the more advanced status of the students led to higher prior knowledge (as indicated by the pre-test scores) of the MergeSort algorithm, compared to the pre-test levels observed in the novice students of Experiment I. Another noteworthy point is that the AV groups in both experiments reached the same level of performance on MergeSort in the post-test after interacting with HalVis though they started off with different levels of prior knowledge. These two experiments together suggest that interactive visualization is a more effective method for individual self-paced learning than learning from a textbook regardless of the level of students.

Table 4. Comparison of Results of Experiments I and II

	Experiment I		Experiment II	
	Pre-test	Post-test	Pre-test	Post-test
Text Group	27%	43%	44%	53%
AV Group	28%	74%	48%	71%

3.5. Experiment III: Do Novice Students Learn More from HalVis than from a Lecture?

This experiment was designed to compare HalVis with classroom lectures, and also to investigate how HalVis and lectures *together* contribute to learning. Our hypothesis was that HalVis would facilitate learning better than a lecture because the system provided animated illustrations of algorithms that a lecturer cannot, and it allowed

students to pace their learning whereas the pace of a lecture was beyond the control of individual students. In addition, we predicted that HalVis used in conjunction with lectures would assist learning even more because of the redundant exposure to the subject matter through two different instructional methods. The third issue investigated in this experiment was whether the order (HalVis before lecture, or vice versa) would make a difference in performance when students learned by both methods. Participants in this experiment were novice computer science students, and the algorithms used were SelectionSort and MergeSort.

3.5.1 Subjects

This experiment involved 20 second-year undergraduates enrolled in an introductory data structures and programming course at Auburn University. Students were assigned to a matched pair of groups, a Lecture-Visualization (LV) group and a Visualization-Lecture (VL) group. The LV group received a class lecture discussing the algorithms, then interacted with visualizations of the two algorithms in a computer laboratory. The VL group interacted first with HalVis, and then attended the class lecture covering the algorithms. There were nine students in the LV group and eleven students in the VL group.

3.5.2 Materials

Lecture: All participants attended the same two lecture sessions conducted by an Auburn University computer science faculty member. These sessions consisted of 100 minutes of verbal instruction accompanied by pictures drawn on the blackboard, overhead transparencies, and a lecture summary handout. The lecturer also responded to several questions from students in the class during the sessions.

Visualization: Both groups interacted with the same visualizations of the SelectionSort and MergeSort algorithms, with no supplementary materials provided.

Test Questions: Three tests were used. A pre-test was given prior to the experiment, a mid-test was given after the first treatment, and a post-test was given after the second treatment.

3.5.3 Procedure

The phases of this experiment were carefully synchronized with the class syllabus. Students were given the pre-test a week before the scheduled lecture about sorting algorithms. The day before the lecture, the VL group met in a

computer laboratory on campus, interacted with HalVis to learn the two algorithms, and completed the mid-test. Both groups then attended the same lecture. We chose to use a live classroom lecture instead of presenting a videotaped lecture to students in order to allow student interaction with the instructor and to simulate a realistic learning environment. Having both groups attend the same lecture eliminated variations between separate lectures covering the same topic. The day after the lecture, the LV group met in a computer laboratory on campus and first completed the same mid-test taken by the VL group. Then the group was asked to interact with HalVis to learn the two algorithms, followed by the post-test. On this same day, the VL group met in a classroom and completed the same post-test.

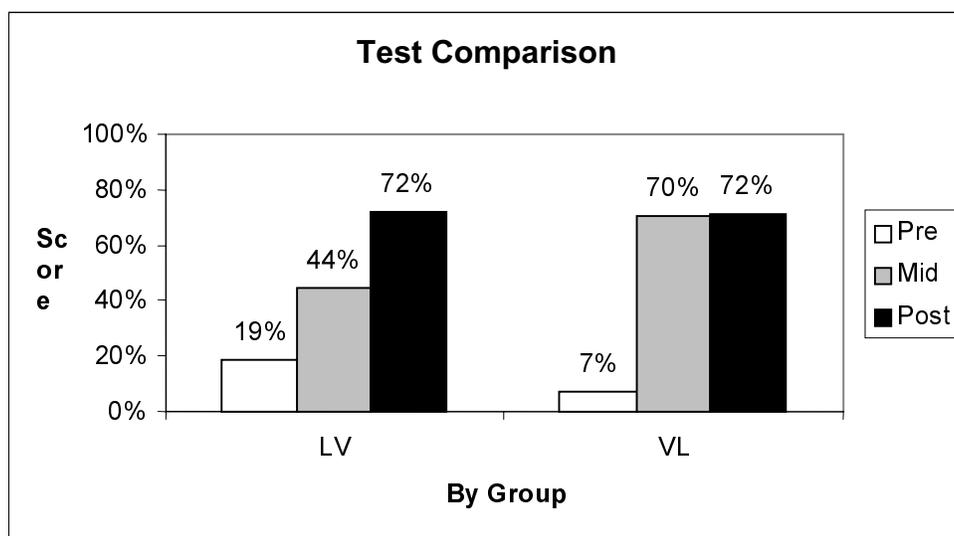


Fig. 10. Comparison of Group Pre-Test, Mid-Test and Post-Test Scores

We designed the experiment to minimize outside interactions that might affect the results. We asked students to refrain from discussing any aspect of the experiment during its course. We did not explicitly prevent students from reading the course textbook or trying to learn more about the algorithms from other sources. However, only one of the algorithms was discussed in the course textbook. A question in the mid- and post-tests asked students whether they had attempted to learn about the algorithms on their own during the experiment. None of the students indicated that they had done so in the mid-test, and four indicated that they read the textbook description of one algorithm in the post-test.

3.5.4 Results

Examining the results depicted in Figure 10, we see that both groups were relatively unfamiliar with the algorithms

based on their pre-test averages (7% for the VL group and 19% for the LV group). The mid-test results show that the VL group learned more than the LV group. Following the session with HalVis, the VL group average score was 70% compared to 44% for the LV group after the lecture. Despite having less prior knowledge about the algorithms, the VL group (after interacting with HalVis) significantly outperformed the LV group (that received the lecture) in the mid-test. However, HalVis helped the LV group catch up with the VL group by the time of the post-test, whereas additional knowledge gained by the VL group from the lecture session was relatively small. The VL group experienced a 63% improvement after the visualization, and the following lecture provided an improvement of 2%. The LV group experienced a 25% improvement in average score after receiving the lecture, then improved another 28% following the AV interaction. Both groups eventually reached similar levels (72%) of performance. The LV group showed steady improvements following the lecture and then the visualization. The VL group showed a significant increase resulting from visualization alone, to which the lecture did not add considerably.

Statistical data are provided in Tables 5, 6 and 7. Table 5 shows the between-group statistical results. Post-test results show that the order in which lectures and visualizations are presented does not appear to make a difference, as both groups scored the same (72% for the VL group and 72% for the LV group; $F(1,19)=0.001$, $p<0.97$). While these post-test results are not significantly in favor of either group, Table 5 shows that the mid-test scores are in favor of the group that interacted with the visualization first (VL group) compared to the group receiving the lecture, ($F(1,19)=5.3$, $p<0.033$).

Table 5. Experiment III Statistical Summary: Between Groups (Overall Performance)

	Pre-test	Mid-test	Post-test
LV	19%	44%	72%
VL	7%	70%	72%
$F(1,19)$	1.78	5.3	0.001
Significance	$p<0.198$	$p<0.033$	$p<0.97$

Table 6 shows that the score improvement from the pre-test to the mid-test favored the group receiving the visualization, which for this first phase of the experiment was the VL group ($F(1,19)=26.89$, $p<0.0001$).

Table 6. Experiment III Statistical Summary: Between Groups (Improvement)

	Pre-to-Mid- Test Improvement	Pre-to-Post- Test Improvement
LV	25%	53%
VL	63%	65%
F(1,19)	26.89	1.16
Significance	p<0.00001	p<0.29

These results are summarized as box plots in Figure 11. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group.

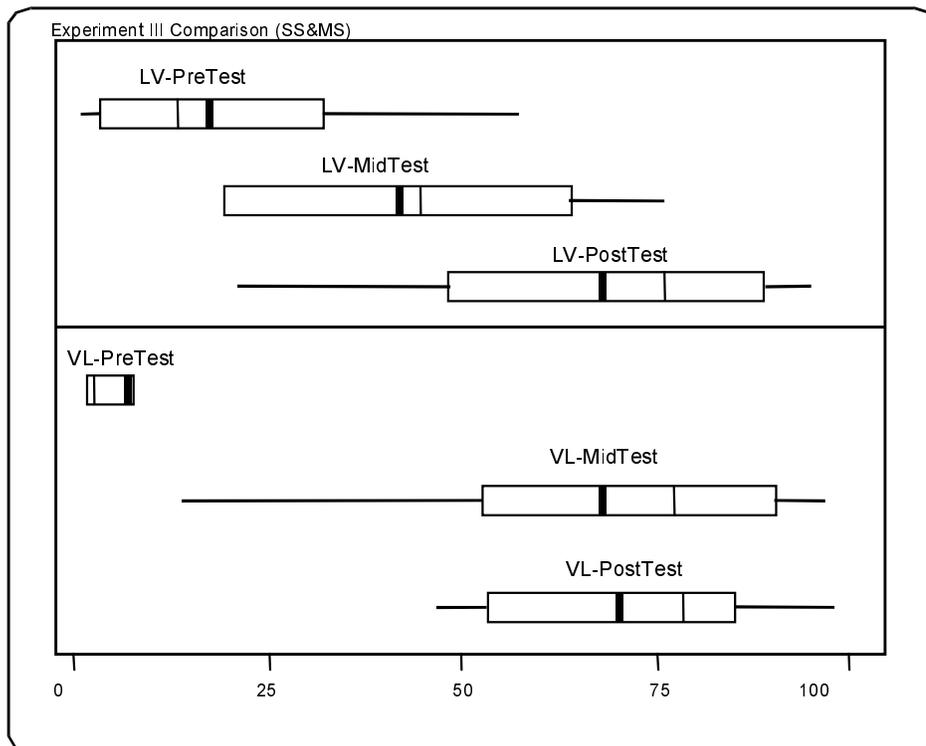


Fig. 11. Experiment III Box Plots

3.5.5 Discussion

This experiment shed light on the three aspects of learning that we set out to investigate. First, the results support the hypothesis that learning by visualization is more effective than learning by lecture alone. Second, these results suggest that the *combination* of learning by visualization and lecture leads to improvements over learning by visualization or lecture alone. Both groups improved after both phases were completed. The improvement due to providing the visualization to the LV group was large and statistically significant. However, the improvement due to providing the lecture to the VL group was only 2% and not statistically significant. The third conclusion is that if students learn from both lectures and algorithm visualizations, the order of the instructional methods is not important. After completing both phases, the two groups performed at about the same level.

There are some issues to consider regarding these results. The most obvious one is the quality of the lectures, for which no objective measure exists. The lecturer had several years of experience teaching introductory computer science courses and had a record of consistently receiving high course evaluations. He received the “best teacher” award from the college of engineering twice, and had taught the introductory data structures and programming course, from which the subjects were drawn, many times. We designed the experiment so that all participants would attend the same lecture sessions, to avoid the possibility that different sessions could have covered the same material in different ways. We also chose to have a live lecture instead of a videotaped one, to allow teacher-student interaction more typical of a classroom environment. Another factor we did not control for was that some students might have tried to learn more about the algorithms during, but outside of the experiment. To reduce this possibility we intentionally used the SelectionSort algorithm, which was not discussed in the course textbook. The MergeSort algorithm was discussed in the textbook. We asked the subjects a question about this in the mid- and post-tests. Only four indicated in the post-test that they had read the text. We did not detect any significant differences in student performance between the algorithm discussed in the textbook and the algorithm not covered in the textbook.

3.6. Experiment IV: Do Advanced Students Learn More from HalVis than from a Typical Algorithm Animation?

The goal of this experiment was to compare the effectiveness of learning from hypermedia algorithm visualization in the style of HalVis to learning from a typical algorithm animation. The algorithm used was Shortest Path, to find

shortest paths from one node to all other nodes in a graph. It is conceptually more difficult, and is different in style, from the algorithms used in the previous experiments. Correspondingly, we chose more advanced students as subjects.

3.6.1 Subjects

This experiment involved 40 undergraduate computer science students enrolled in a third year algorithm design and analysis course at Auburn University. Like previous experiments, participants were assigned to two matched groups: a “Tango” group and a “HalVis” group.

3.6.2 Materials

Algorithm Animation: One of the most mature and widely available algorithm animation platforms is the software suite developed by Stasko [25] – Tango, Polka and Samba – publicly available from Georgia Institute of Technology. This software distribution executes on Windows 95 systems and includes a library of algorithm animations created by researchers and students. We carefully examined eight animations of the Shortest Path algorithm available in this distribution, and selected one that appeared to be easiest to understand and which had the most features in common with the HalVis system. Figure 12 shows a screenshot of this animation. The Tango group worked with this animation. Algorithm Visualization: The HalVis group worked with a visualization of the Shortest Path algorithm in HalVis.

Similarities and differences between the two: Like HalVis, the Tango animation provides multiple representations: a pictorial representation of the graph in which the shortest path is being computed and a textual representation of the pseudocode of the Shortest Path algorithm. The animation, similar to HalVis, shows various steps of the algorithm on the graph through color coding, and the line of the pseudocode that is being executed in the animation is highlighted. Controls for stepping through the animation (which will pause after each step has been executed) and speeding it up or slowing it down are also available in both systems.

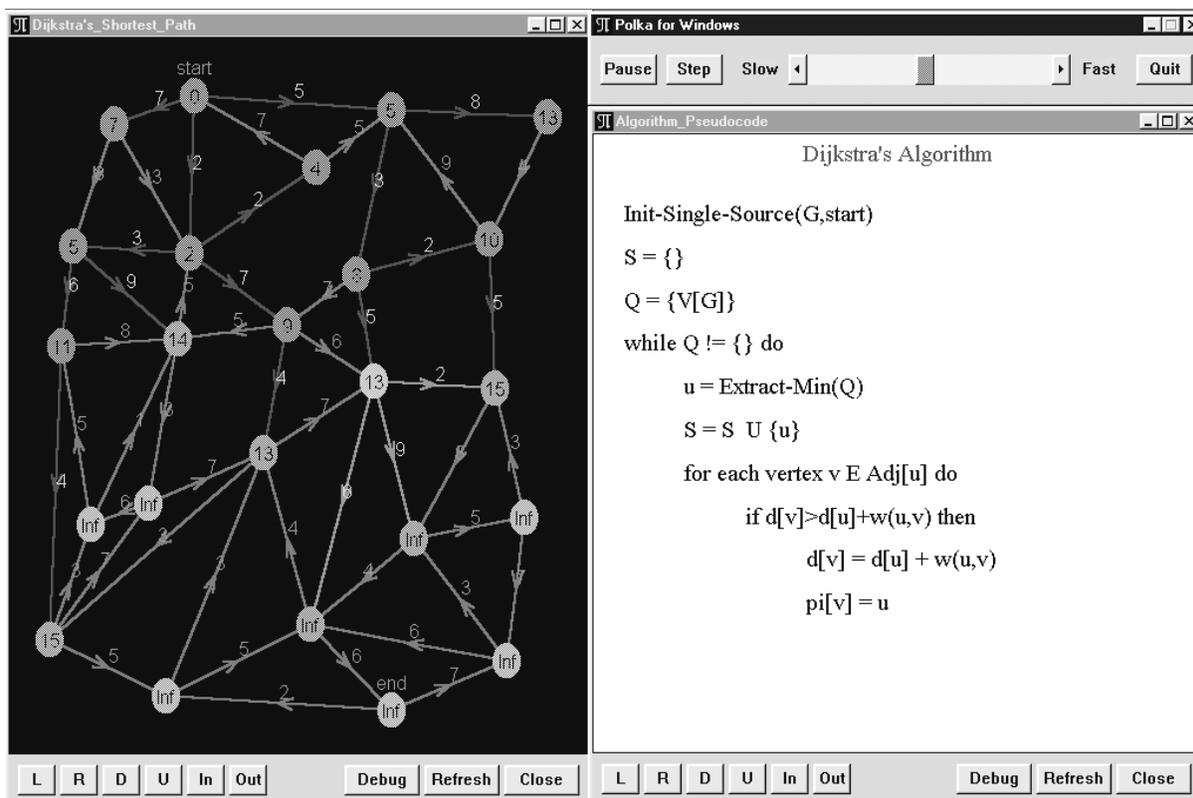


Fig. 12. Algorithm Animation

There are also several differences between the animation and the visualization. One is that the animation, typical of algorithm animations produced in prior research, provides just one module that shows the animation with additional representations such as pseudocode. The module in HalVis that presents a similar animation, the Populated View which contains a macro-level (large-scale) animation, is preceded by a bridging analogy that is animated (the Conceptual View) and a micro-level animation module called the Detailed View. Thus, HalVis provides three kinds of animations. Second, HalVis presents textual explanations, with hyperlinks to additional explanations of the basics of algorithmics in a Fundamentals module at several points. Thus, the animations do not stand-alone; instead, they are contextually embedded in a knowledge-providing hypermedia environment. The third difference is that HalVis poses thought provoking “tickler” questions to students during the course of the animations, and provides “articulation points” consisting of additional questions between modules, in order to give students a way to self-assess their knowledge. Fourth, the algorithm animation in HalVis’s Detailed View is accompanied not only by synchronously highlighted pseudocode, but also by brief explanatory messages about the operations being executed and a view of the changing values of important variables utilized by the algorithm. Fifth, the student has more control over HalVis’s animation in the Detailed View. Besides similar speed controls and a stepping-through

facility, the student can also segment the animation at different levels such as pausing after each loop execution. Finally, in the Populated View, the student is given an opportunity to make predictions about the performance of the algorithm. If he or she chooses to do so, the animation will display the predicted and actual values, thus providing informative feedback.

Handout: The Tango group also received a supplement, a 5-page section describing the Shortest Path algorithm photocopied from their textbook [29]. This was done to mimic conditions under which algorithm animations were previously experimentally evaluated [e.g. 14,15], where the animation groups generally had access to supplementary materials. Members of the HalVis group were not given any supplementary materials.

3.6.3 Procedure

The experiment preceded course lectures that covered the subject of graph algorithms. Twenty students each in the Tango group and the HalVis group completed the experiment. Pre-test results helped verify that the two groups were evenly balanced in terms of prior knowledge. In the following week, both groups met in the same public computer laboratory on campus, but at different times. Both groups received a brief, navigation-only orientation to the visualization software they were to use prior to being assigned individually to computers. No student in the HalVis group took more than 90 minutes for the entire experiment. No student in the Tango group took more than 60 minutes for the entire experiment.

3.6.4 Results

Examining the results shown in Table 8 and Figure 13, we see that both groups were relatively unfamiliar with the algorithm based on the pre-test averages. In the post-test the HalVis group scored on average 89% and the Tango group scored on average 71% ($F(1,37)=12.75, p<0.001$). The pre- to post-test improvement observed in the HalVis group was also significantly greater than that shown by the Tango group ($F(1,37)=4.79, p<0.035$). These results are also summarized as box plots in Figure 14. The distribution of scores is interesting. The HalVis pre-test score distribution is tight and normal-looking, but there are two outliers (shown as black dots) that scored very well, indicating prior knowledge of the Shortest Path algorithm. The distribution of the HalVis group's post-test scores indicates a tighter clustering compared to the post-test score distribution of the Tango group, with one outlier at 69% (shown as a black dot). The post-test score distribution of the Tango group also shows (as a black dot) the presence of one outlier who scored extremely poorly.

Table 8. Experiment IV Statistical Summary

Statistical Summary			
	Pre-test	Post-test	Improvement
Tango	23%	71%	48%
HalVis	22%	89%	68%
F(1,37)	0.01	12.75	4.79
p	p<0.91	p<0.001	p<0.035

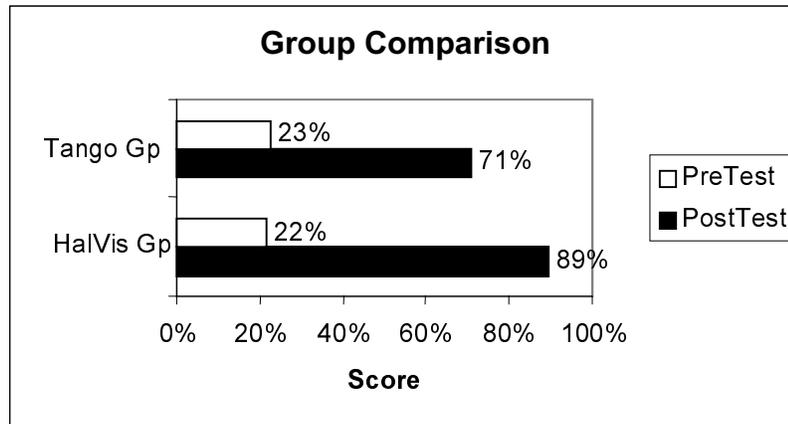


Fig. 13. Experiment IV Overall Summary

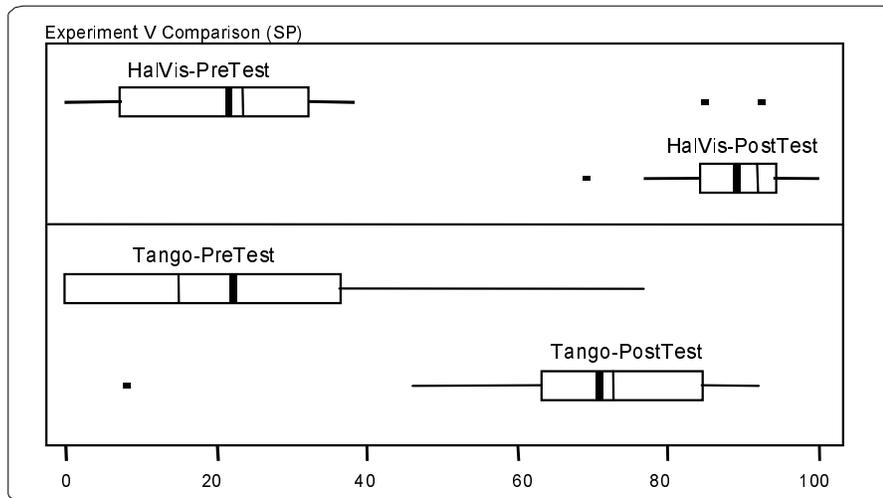


Fig. 14. Experiment IV Box Plots

3.6.5 Discussion

This experiment compared the HalVis algorithm visualization framework with an algorithm animation built using a

popular algorithm animation system. Similar algorithm animations have been the focus of several experimental analyses reported in the literature [e.g. 12,13,14,15]. The algorithm animation was well-paced, showed good use of color to highlight algorithm actions, included a brief textual introduction, and provided the student with the algorithm's pseudocode, whose lines were highlighted synchronously as the animation proceeded. We supplemented this animation with pages describing the algorithm from the course textbook in order to provide the student with as much information about the algorithm as possible in a stand-alone setting, and to replicate as closely as possible the conditions of algorithm animation experiments reported by other researchers. The results indicate that our hypermedia algorithm visualization design is more effective, as a stand-alone self-paced learning environment, than an algorithm animation representative of current approaches.

As with the other experiments, there are several relevant factors to note. The quality of the algorithm animation is one issue. We attempted to address this by selecting the most explanatory and most similar (in features to HalVis) algorithm animation from the eight different animations of the Shortest Path algorithm supplied with the software distribution. It is possible that a different algorithm animation might have led to different results for the Tango group. A similar argument could be made regarding the textual supplement (photocopied pages from the course textbook) that was provided to the Tango group.

4. General Discussion

Our experiments were designed to evaluate a novel approach to the design of educationally effective algorithm visualizations, one in which analogies and animations are embedded in a context and knowledge providing hypermedia environment. Comparisons with learning from a textbook, learning from lectures, and learning by interacting with an algorithm animation representative of current research on the topic demonstrated the benefits of HalVis for self-directed and self-paced learning. Significant learning effects were found for different algorithms and undergraduate students at different levels.

We are aware of only one other system, ALGONET, that takes a similar approach of including algorithm animations in a larger context in which other “cognitive media” such as definitions, examples, pseudocode, questions and case studies are provided [23]. But the authors present data from only an empirical comparison of two versions of ALGONET, rather than comparing ALGONET against other means of instruction or other algorithm visualizations.

There are also significant differences between the architectures of HalVis and ALGONET. At the very beginning, HalVis introduces key aspects of an algorithm with an analogy, and in work reported elsewhere [30] we have demonstrated that these analogies significantly prime learning from subsequent visualizations. The case studies in ALGONET are like analogies, but they are not explicitly designed to be bridges to better comprehension of the more detailed animations to follow. ALGONET animations are accompanied by textual explanations, but not also by synchronously highlighted pseudocode and changing variable values as in the Detailed View of HalVis. ALGONET does not present “tickler” questions to promote self-explanations during animation viewing. There is also no facility for the learner to enter his or her predictions regarding performance parameters of the algorithm and compare those against actual values as the animation progresses.

In addition to examining student performance, we captured a log of user activity and collected student comments. Preliminary analysis of the user logs showed that students executed the micro-level animations (embedded in the Detailed View) an average of three times and viewed the macro-level animations (embedded in the Populated View) an average of four times. We found that over half of the students experimented with entering their own data for at least one execution of the algorithm animation at the detailed level, and every student elected to make performance predictions for at least one animation execution at the populated level. The student surveys overwhelmingly favored HalVis over the other learning methods. Extracts of some of the positive comments include: “Animation made it easier to understand ... more interesting ... much better than reading a book”. Students liked “...being able to rerun the animation until you understand ... step by step...,” as well as the questions and facility to make predictions. One student said, “ I liked being able to see [efficiency] ... I didn't realize that there was that much difference.” There were some negative comments as well. Students provided constructive criticisms like: “It could use more sound,” and “The recurring textual explanations [in the Execution Status Message window of the Detailed View] were annoying.” There were also conflicting comments like one who said, “It was too detailed,” while another that observed, “It was too simplistic.”

In a meta-analysis of empirical studies of algorithm animations, Gurka and Citrin [31] state that “there is a danger ... as more experiments produce inconclusive data, which may result in algorithm animation failing to become an important educational mechanism”. Therefore, the algorithm visualization architecture and positive evaluation results presented in this paper are important steps toward revealing the educational effectiveness of algorithm

animation. While these experiments represent a large-scale systematic effort to uncover the learning benefits of contextually embedded algorithm animations, others have also reported on positive outcomes from algorithm animation. For instance, Crosby and Stelovsky [32] compared the learning of second semester computer science undergraduates from a visualization system and traditional instruction (view graphs and blackboard). The students learned several algorithms in a four-week period. The visualization system displayed either three simultaneous presentations of some algorithms (pseudocode in which steps are highlighted as they are executed, changing variable values and comments, but no graphical animation) or a typical animation (for other algorithms). They found that the visualizations resulted in significantly higher post-test scores (82.69% correct versus 51.41% correct). Kann and colleagues compared post-test scores of students who learned a recursive algorithm for the Knapsack problem from a textual description, and from a very simple animation and the textual description [33]. They found that viewing the animation resulted in significantly higher post-test scores (79% correct versus 64% correct). In a large study involving 300 novice participants (non-computer science majors), Seay and Catrambone [34] found that the transitional motion contained in a smooth animation of the steps of an algorithm is helpful to those who are not exposed to well-designed textual explanations. In the study, students who had access to a textbook extract and viewed an animation performed better on far transfer problems than those who had access to the textbook extract and saw a series of static pictures as well as those who saw only the textbook extract.

5. Conclusion

The general conclusion is that interactive hypermedia algorithm visualization, modeled after the HalVis framework (a system in which analogies and animations are embedded within a knowledge and context providing hypermedia environment), can provide significant benefits to learners as an educational tool for self-directed and self-paced learning, either in a stand-alone mode or even more so in combination with other instructional techniques.

There are a number of possible reasons for the observed advantages of algorithm visualizations over traditional algorithm animations. First, in comparison with previous animation systems that only presented animations with some textual feedback, HalVis allows the student to learn incrementally by starting from a real world analogy and then transitioning to the algorithm. Second, the hypermedia structure provides a student access to fundamental building blocks of algorithmic knowledge *in-context* and *on-demand*. Third, we divide dynamic presentations into manageable and meaningful segments, with pauses and self-explanation inducing questions in-between. This makes

it easier for students to stop and reflect, repeat, or access other relevant information through hyperlinks while watching animations. Furthermore, animation segments are presented in synchrony with other representations in other media. This combination of features, we believe, results in the dynamic information being conveyed better in context, and therefore in a more comprehensible fashion. Fourth, rather than providing just one view of an animation as has been the typical approach, HalVis presents three kinds of animations (analogical, micro-level and macro-level), so that the macro behavior is seen after the micro behavior is seen and understood, both following an analogical introduction to the algorithm. Fifth, our framework allows students to actively engage themselves in the visualization by changing data inputs and making performance predictions, contributing to better learning. Sixth, we believe that the different kinds of questions provided by HalVis promote reflection and self-explanation, thereby improving learning.

The following list summarizes conclusions that can be drawn from the reported experiments.

- Advanced as well as novice students perform better in answering conceptual and procedural questions about fundamental algorithms after interacting with hypermedia algorithm visualization than after studying explanations found in typical textbooks.
- Novice students gain more knowledge after interacting with hypermedia algorithm visualization than after hearing a typical classroom lecture. Furthermore, lectures and visualizations supplementing each other provide the best learning scenario, and the order of presentation does not seem to influence extent of learning.
- It appears that interactive hypermedia algorithm visualization is most effective when prior knowledge is limited. However, students with some amount of prior knowledge also derive a significant learning benefit from algorithm visualizations.
- Individual differences in learning from algorithm visualizations exist. These differences may be compensated by the use of multiple modes of instruction. This argues for hypermedia algorithm visualizations supplementing, rather than replacing, traditional instructional methods.
- The framework for algorithm visualization design that HalVis exemplifies is more effective than previous algorithm animation designs.

In conclusion, the research reported in this paper illustrates the benefits of visualizations, conceived as different kinds of animations immersed in contextual knowledge, over simple animations and traditional means of learning.

Thus, it begins to provide a methodology for the design of visualizations to effectively communicate abstract process knowledge to students. Perhaps disheartened by the preponderance of negative evidence regarding the educational effectiveness of algorithm animations, the research community has of late begun to investigate students constructing animations [25,35] and other expository representations [36]. While constructivism in algorithm learning is a pedagogical strategy sorely in need of further research, the message of this paper is that abandoning the design and use of interactive expert-constructed algorithm visualizations as instructional aids in the classroom, or as aids for self-paced learning, is not warranted.

Acknowledgements

This research has been supported by the National Science Foundation under contracts CDA-9616513 and REC-9815016. The experiments reported herein could not have been carried out without the participation of volunteers from the following courses: CSE 220 – Fundamentals of Computer Science and CSE 360 – Fundamental Algorithm Design and Analysis. We thank the anonymous referees for their suggestions on improving this paper.

References

1. M. Scaife & Y. Rogers (1996) External cognition: How do graphical representations work? *International Journal of Human-Computer Studies* **45**, 185-213.
2. M. Petre (1995) Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM* **38**, 33-44.
3. P. A. Gloor (1992) AACE – Algorithm animation for computer science education. In: *Proceedings of the IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Piscataway, NJ, pp. 25-31.
4. E. Helttula, A. Hyrskykari & K-J. Raiha (1990) Principles of Aladdin and other algorithm animation systems. In: *Visual Languages and Applications* (T. Ichikawa, E. Jungert & R. Korfhage, eds.) Plenum Press, New York, pp. 175-187.
5. J. D. McWhirter (1996) AlgorithmExplorer: A student centered algorithm animation system. In: *Proceedings of the IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Piscataway, NJ, pp. 174-181.
6. S.-P. Lahtinen, E. Sutinen & J. Tarhio (1998) Automated animation of algorithms with Eliot. *Journal of Visual Languages and Computing* **9**, 337-349.
7. M. H. Brown & M. A. Najork (1997) Collaborative active textbooks. *Journal of Visual Languages and Computing* **8**, 453-486.
8. J. Domingue & P. Mulholland (1998) An effective web-based software visualization learning environment. *Journal of Visual Languages and Computing* **9**, 485-508.
9. P. Carlson, M. Burnett & J. Cadiz (1996) A seamless integration of algorithm animation into a visual programming language. *Proceedings of AVI'96: International Workshop on Advanced Visual Interfaces*, Gubbio, Italy.
10. C. Demetrescu & I. Finocchi (2001) Smooth animation of algorithms in a declarative framework. *Journal of Visual Languages and Computing* **12**, 253-281.
11. J. Bazik, R. Tamassia, S. P. Reiss & A. van Dam (1998) Software visualization in teaching at Brown university. In: *Software Visualization: Programming as a Multimedia Experience* (J. Stasko, J. Domingue, M. H. Brown & B. A. Price, eds.), MIT Press, Boston, MA, pp. 383-398.
12. C. Kehoe, J. Stasko & A. Taylor (2001) Rethinking the evaluation of algorithm animations as learning aids: An observational study. *International Journal of Human-Computer Studies* **54**, 265-284.

13. J. T. Stasko, A. Badre & C. Lewis (1993) Do algorithm animations assist learning? An empirical study and analysis. In: *Proceedings of the INTERCHI'93 Conference*, Addison-Wesley, Amsterdam, pp. 61-66.
14. A. Lawrence, A. Badre & J. T. Stasko (1994) Empirically evaluating the use of animations to teach algorithms. In: *Proceedings of the IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Piscataway, NJ, pp. 48-54.
15. M. D. Byrne, R. C. Catrambone & J. T. Stasko (1996) Evaluating animations as student aids in learning computer algorithms. *Computers & Education* **33**, 253-278.
16. C. D. Hundhausen, S. A. Douglas & J. T. Stasko (to appear) A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*.
17. A. F. Blackwell (2001). Pictorial representation and metaphor in visual language design. *Journal of Visual Languages and Computing* **12**, 223-252.
18. N. H. Narayanan & M. Hegarty (2000) Communicating dynamic behaviors: Are interactive multimedia presentations better than static mixed-mode presentations? In: *Proceedings of the First International Conference on the Theory and Application of Diagrams*, Springer-Verlag, Berlin, pp. 178-193.
19. J. B. Morrison, B. Tversky & M. Betrancourt (to appear) Animation: Does it facilitate learning? *International Journal of Human-Computer Studies*.
20. N. H. Narayanan & M. Hegarty (1998) On designing comprehensible interactive hypermedia manuals. *International Journal of Human-Computer Studies* **48**, 267-301.
21. N. H. Narayanan & M. Hegarty (to appear) Multimedia design for communication of dynamic information. *International Journal of Human-Computer Studies*.
22. E. Soloway, S. L. Jackson, J. Klein, C. Quintana, J. Reed, J. Spitulnik, S. J. Stratford, S. Studer, S. Jul, J. Eng & N. Scala (1996) Learning theory in practice: Case studies of learner-centered design. In: *Proceedings of the ACM Human Factors in Computing Systems Conference (CHI'96)*, ACM Press, New York, pp. 189-196.
23. M. M. Recker, A. Ram, T. Shikano, G. Li & J. T. Stasko (1995) Cognitive media types for multimedia information access. *Journal of Educational Multimedia and Hypermedia*, **4**, 183-210.
24. S. Douglas, C. Hundhausen & D. McKeown (1995) Toward empirically-based software visualization languages. In: *Proceedings of the IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Piscataway, NJ, pp. 342-349.

25. J. T. Stasko (1997) Using student-built algorithm animations as learning aids. In: *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*, ACM Press, New York, pp. 25-29.
26. C. E. Hmelo & M. Guzdial (1996) Of black and glass boxes: Scaffolding for learning and doing. In: *Proceedings of the Second International Conference on Learning Sciences*, AACE, Charlottesville, VA, pp. 128-134.
27. M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann & R. Glaser (1989) Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science* **13**, 145-182.
28. N. Dale, S. Lilly & J. McCormick (1996) *Ada plus data structures: An object-based approach*. Heath and Company, Lexington, MA.
29. M. A. Weiss (1993) *Data Structures and Algorithm Analysis in Ada*. The Benjamin/Cummings Publishing Company, Redwood City, CA.
30. S. R. Hansen & N. H. Narayanan (2000) On the role of animated analogies in algorithm visualizations. In: *Proceedings of the Fourth International Conference of The Learning Sciences*, Lawrence Erlbaum Associates, Mahwah, NJ, pp. 205-211.
31. J. S. Gurka & W. Citrin (1996) Testing effectiveness of algorithm animation. In: *Proceedings of the IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Piscataway, NJ, pp. 182-189.
32. M. Crosby & J. Stelovsky (1995) From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia* **4**, 147-162.
33. C. Kann, R. W. Lindeman & R. Heller (1997) Integrating algorithm animation into a learning environment. *Computers & Education*, **28**, 223-228.
34. F. Seay & R. Catrambone (2001) Using animations to help students learn computer algorithms: A task analysis approach. In: *Artificial Intelligence in Education* (J. D. Moore et al. eds.), IOS Press, 43-54.
35. C. D. Hundhausen & S. A. Douglas (to appear) Low fidelity algorithm visualization. *Journal of Visual Languages and Computing*.
36. T. Hübscher-Younger & N. H. Narayanan (2001) Features of shared student-created representations. Paper presented at the *Workshop on External Representations in AI-ED: Multiple Forms and Multiple Roles, Tenth International Conference on Artificial Intelligence in Education (AI-ED 2001)*, San Antonio, TX, May. Available as Technical Report CSSE01-13, Computer Science & Software Engineering Dept., Auburn University, Auburn, AL, USA, from <http://www.eng.auburn.edu/department/csse/research/publications/>.