Pergamon

PII: S0360-1315(97)00015-8

Computers Educ. Vol. 28, No. 4, pp. 223–228, 1997
© 1997 Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0360-1315/97 $17.00+0.00

INTEGRATING ALGORITHM ANIMATION INTO A LEARNING ENVIRONMENT

CHARLES KANN, ROBERT W. LINDEMAN and RACHELLE HELLER

Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC 20052, U.S.A.

(Received 1 January 1997; accepted 1 May 1997)

Abstract—Algorithm animation would seem to be a useful tool for teaching algorithms. However, previous empirical studies of using algorithm animation have produced mixed results. This paper presents an empirical study in which the subjects programmed the algorithm which they had seen animated. The results of the experiment indicate that combining the animation with the implementation of the algorithm was an effective way to teach the animation, and also produced transfer effects for general recursion problems. © 1997 Elsevier Science Ltd

INTRODUCTION

Graphics have always been used to explain algorithms in computer science. Most textbooks which teach algorithms use abstract visualizations to explain how the algorithms work. For example, a linked list is often represented as a series of boxes, called nodes, which contain fields for data. At least one of the data fields in a node contains a reference to some other node(s), which creates a link between the nodes [1]. This abstract graphical representation of a linked list is just one example of how graphics are used in textbooks to teach algorithms.

However, the graphical representations of algorithms used in text books have some limitations. First only a small number of graphic images can be included in a text book, so it is necessary to omit certain key steps, and leave it up to the reader to fill in the missing details. Second, printed text represents an historical document, one which must be complete and unchanging. Therefore students cannot run their own simulations of the algorithm, to see how special cases of the data affect steps in the process of the algorithm to solve a problem. To overcome these limitations, instructors generally present the algorithm, using the same graphics as in the text book, explaining the steps which are involved, and solving special problems in the algorithm.

Many of the problems with the graphics used in textbooks can be solved using modern computers, which have the power to make the generation of simple "flip card" animations trivial. Flip card animations are animations where one image overlays another, giving the illusion of motion. This simple type of animation is sufficient for representing most algorithms, and has been used by a number of algorithm animation systems, such as Balsa-II [2], and Polka [3]. These systems have four advantages over the pictures which were used in textbooks. First, since the number of pictures, and thus the number of steps in the algorithm represented, is not as limited as in textbooks, more detail about the algorithm can be shown. Second, since these systems are general purpose algorithm animation systems, instructors and algorithm implementors can develop their own algorithm animations. Third, since these algorithms are run in real-time, it is possible for students to run their own example problems, allowing them to experiment with different data, and thus increasing their understanding of the algorithm. Finally, these algorithm animation systems allow algorithm designers to run simulations and study the behavior of the algorithms to gain a better understanding of the performance of the algorithm.

While the tools to produce algorithm animations have advanced, an understanding of how to effectively use algorithm animation in education is still not well understood. Even though the value of using algorithm animation seems obvious, the results of using algorithm animation in empirical experiments with subjects learning algorithms have been mixed. This research looks at one way to use algorithm animation which the authors have found to be effective, using the animations as part of the overall learning experience where students implement the algorithms.

The rest of the paper presents an empirical study in using algorithm animation in teaching recursion. The paper first covers existing empirical research in using algorithm animation in education. Next the empirical study of using algorithm animation is presented, followed by reporting and discussion of the results of this experiment. Finally some conclusions are given.

BACKGROUND

It seems obvious that an effective way to teach algorithms is to demonstrate by animation how the algorithms work. In practice animation does not live up to its potential. It has been shown that students like animations, and seem to enjoy projects which involve animation [4,5]. However the results of using animations in empirical studies where the animation has been used to help the students learn the algorithm have been decidedly mixed. In this section four studies are presented. The first two studies show animation to have a positive impact on the ability of students to learn algorithms, the second two studies show that animation had no effect on the ability of students to learn algorithms. From these studies, three factors which influence the effectiveness of an animation are proposed, and an empirical study to test influencing one of those factors is described.

The first study which showed positive results from the use of animation was by Shih and Alessi [6]. In this study, the authors proposed that animation could be used to help students develop a mental representation (or mental model) of computer memory. They designed a system which used a warehouse model of memory to show how a computer uses memory when running a program. The authors were able to show that this type of animation produced significant effect on the ability of students to evaluate a program, and that there were transfer effects which allowed the students who were taught evaluative tasks (reading and understanding programs) were better able to complete generative tasks (creating a program). The second study which showed a positive result is Ramani and Rao [7]. In this study a graphical animation of the solution tree for solving an integer programming problem was built interactively. The results of a test of knowledge of integer programming showed the animation had a positive influence on the scores for the subjects who used the program over those who did not.

The third study, which did not show a positive result from the use of animation, was by Anderssen and Myburg [8]. In this study, operating systems concepts such the dining philosophers problem were taught using computer animation. In this study no significant difference was found in the scores on a test between the subjects who used the graphical animation and those who did not. The fourth study by Stasko *et al.* [9] applied a graphical animation to the teaching of a pairing heap. While this study showed that students who used the animation did achieve better scores on a test of concepts about the pairing heap, this difference was neither large nor significant.

In the study by Stasko *et al.*, the authors conjectured that no positive impact was found in the use of animation because, to be effective, algorithm animations must be used as part of a larger instructional setting, and that students might benefit from constructing the algorithm, which is "active" learning, as opposed to watching the algorithm, which is "passive" learning. However, from these four studies, we feel that there are at least three issues which must be taken into account when developing an animation. These are:

1. The type of the animation. In the study by Shih and Alessi, the authors designed the animation to provide a mental model to the subjects of a small, specific area of programming. Likewise in the study by Ramani and Rao, the purpose was to simply show the steps involved in solving an integer programming problem. It is possible that animations which represent a specific goal, e.g. teaching a specific concept like computer memory, might be more useful than animations which show an abstract representation of an algorithm. For example, what should be animated in a recursive problem such as a program drawing a fractal curve? There are at least two choices: to show the student the curve (animating the results of a recursive program) [10]; or to represent the state of the program stack (to animate the recursive nature of the program) [5]. What types of animation can be generated, and what is to be taught by the animation, should influence the animation which is created.
2. How effectively the algorithm can be animated. Animations, even simple ones, are not that easy to create. To be effective, algorithms must first be complicated enough to benefit from being animated, and yet not too complicated so that the details of the algorithm are lost in the animation. For example, consider the two tree animations in [1], a Heapsort and an AVL tree. The details of implementing a rotation of a tree needed in the AVL program are much harder to represent than anything which is represented in a Heapsort. This does not mean that it is not useful to create simple or complex animations, however care must be taken in developing effective animations.

3. How the animation is integrated into a learning environment. This issue is an extension of the "passive" vs "active" argument above. We do not believe that the real value in algorithm animation comes from simply viewing the animation. In order for it to be effective, the animation should be part of an overall learning environment. This environment could be simply having a lecturer show how the animation represents the algorithm, or the students could be asked to do something with the algorithm, such as program the algorithm or generate the animation. Regardless of the technique chosen, we feel that to make an algorithm animation more effective, it should not be used in isolation but as part of the learning experience.

This paper presents an empirical experiment which looks at how to incorporate an algorithm animation into an overall learning experience. This experiment looks at two questions. The first question is whether students who view an algorithm animation and then program that algorithm have a better understanding of the algorithm than students who simply program it. The second question is whether or not problem specific animation routines can be used by students to generate animations.

RESEARCH QUESTIONS

In this experiment, two issues about how animation can be used to teach algorithms will be addressed. The first issue examined is whether actively using the information from the animation makes a difference in the ability of the subjects to understand the algorithm. All the subjects are given a written explanation of the algorithm, and are asked to then implement the algorithm. Half of the subjects are also allowed to view an animation of the algorithm, the other half do not see the animation. After implementing the algorithm all the subjects are given a test of their understanding of the algorithm. The null hypothesis is that the use of animation does not significantly effect the subjects ability to correctly implement the algorithm, or their scores on a test measuring their knowledge of the algorithm and recursion in general.

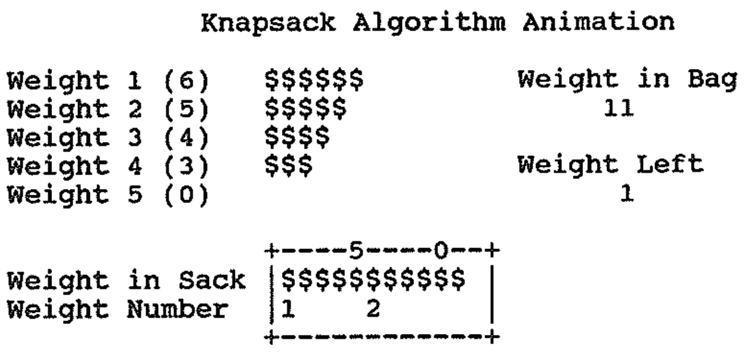
The second issue examined is whether there is educational value in having students use problem specific functions to actually implement the animation. One conjecture by Stakso *et al.* was that subjects would learn the algorithm better if they actually implemented the animation. However the details of implementing an animation were far beyond the ability of the students who were taking part in this experiment. Therefore a library of problem specific functions was developed that allowed the students to implement the animation using calls to functions which can be implemented as part of the algorithm. These functions hide the details of implementing the animation from the subject, and allow them to implement the animation as part of implementing the algorithm. The null hypothesis here is the same as for viewing the animation, that programming the animation will not affect the subjects ability to correctly implement the algorithm, or their scores on a test measuring their knowledge of the algorithm. However the larger purpose is to also gain insight into how students might use problem specific graphic routines when implementing an algorithm.

EXPERIMENTAL METHOD

For this experiment, a recursive problem was chosen. The type of problem was thought to be appropriate because subjects were drawn from a second semester computer science course, and had just learned recursion earlier in the same semester. The specific problem chosen was a knapsack problem [11]. In a knapsack problem, there exists a knapsack which can hold a specific amount of weight, and a number of bars of varying weights which can be placed in the knapsack. The purpose of this algorithm is to see if some combination of the bars can be found which exactly fills the knapsack. The problem involves an exhaustive search of all possible combinations of bars, and is most easily solved recursively.

Subjects for this experiment were 28 student volunteers from a Data Structures (CS2) course taught at the George Washington University. This course is a required course for all undergraduate computer science majors. It is also often a required course for graduate students who do not have a sufficient background in computer science, and can be taken for credit by students in other undergraduate and graduate majors. The background of the subjects was therefore very varied, however all were familiar with the Ada programming language. Subjects were randomly assigned to one of the four treatment groups described below.

The subjects were all given a textural description of the problem, and asked to implement the program



```
                  Knapsack Algorithm Animation

Weight 1 (6)    $$$$$$         Weight in Bag
Weight 2 (5)    $$$$$               11
Weight 3 (4)    $$$$
Weight 4 (3)    $$$            Weight Left
Weight 5 (0)                         1

                +----5----0--+
Weight in Sack  |$$$$$$$$$$$ |
Weight Number   |1    2      |
                +------------+
```

Fig. 1. Example screen from animation.

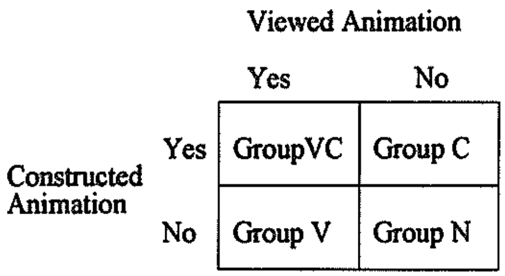
in the Ada programming language. An ASCII animation of the knapsack algorithm, shown in Fig. 1, was available to some of the subjects, as explained in the treatment section below. An ASCII animation was used because at the time of this experiment there was a lack of tools for programming true graphical animations in Ada (with the advent of Ada compilers for the Java VM, this is no longer a problem).

The experiment was designed as a 2-way ANOVA. All subjects were given a textural description of the knapsack algorithm, and were asked to create a program to solve this problem. The subjects differed in the degrees of animation to which they were exposed, the independent variables of the experiment, which were: (1) whether or not the subjects were allowed to view an animation of how the knapsack algorithm worked; and (2) whether or not the subjects were asked to program the animation of the algorithm. Thus the treatment groups were as follows: subjects had no exposure to any animation (Group N); subjects viewed an animation of the knapsack algorithm (Group V); subjects were asked to construct an animation of the algorithm (Group C); and finally subjects viewed the animation, and were asked to construct their own animation (Group VC). Subjects were randomly assigned to each group, with 7 subjects in each group. Figure 2 shows a graphic representation of the experimental design.

To start the experiment, the subjects did a warm up exercise to make sure that they were ready for the study, and that there were no problems with the programming environment. Once they completed the warm up exercise, they were given the text based description of the knapsack problem, and told that they had 2 hours to complete the assignment. Subjects from the treatment groups which were allowed to view the animation (groups V and VC) were also given instructions on how to view the animation. At the end of 2 hours, the subjects were told to stop working, and given a test to measure their understanding of the algorithm. They were given 20 minutes to complete the test.

The test which was given at the end of the experiment contained 17 questions. Three versions of the test, with different random orderings of the questions, were used to insure that there were no effects from the ordering of the questions. The test contained four different types of questions:

1. declarative questions, such as definitions, which measured the understanding of the subjects about the algorithm;
2. procedural questions, such as example problems, which measured the ability of the subjects to follow the steps in the algorithm;
3. analytical questions, which concerned extending the algorithm to see if the subject could reason



		Viewed Animation	
		Yes	No
Constructed Animation	Yes	GroupVC	Group C
	No	Group V	Group N

Fig. 2. Experimental design.

abstractly about the algorithm;

4. general recursion questions, where the subjects were asked if they could identify the purpose of a recursive function such as a string reversal.

RESULTS AND DISCUSSION

There are two artifacts which were produced by the subjects in this experiment, the program code produced by the subjects, and the results of the post test. These were both used to evaluate the results of this experiment. The first result is the number of students who recognized this problem as being a recursive problem. Of the 28 subjects who ran this experiment, 23 subjects tried to solve the problem using recursion, with the other 5 trying various strategies involving for and while looping. There was at least one subject in each group who did not attempt to use recursion, and so there was no measurable effect from using animation on whether or not the subjects choose to use recursion or not.

Next the subjects' programs were scored based on how many of the recursive conditions were found in their programs. A total of five different recursion or ending conditions are needed to completely solve the problem, and so a program which contains more of these conditions would be considered more correct than one which contained fewer of these conditions. Only one subject had all five conditions specified in their program. The average number of conditions present in programs for subjects who viewed the animation was 2.71; for subjects who did not view the animation it was 2.50. Based on this measure, there was no significant difference in the quality of the solutions generated by the subjects.

One last result from evaluating the subjects' code is that less than half (6 out of 14) of the students who were asked to generate the animation even attempted to do so, and of the 6 who attempted, none came close to generating a complete animation. This would seem to argue against the idea that problem specific functions to generate animations can be used by students. However, other studies [4] do show that students can generate these animations, and indeed seem to enjoy this type of assignment. A better conclusion is that the students saw the animation as something to add after the algorithm was working, and since only one student produced a working algorithm, the students assigned the animation task simply did not get around to doing it.

Next the overall results of the subjects' tests were examined, and are summarized in Fig. 3. This chart shows the subjects' average test scores against the two independent variables, if the subjects viewed the animation, and if the subjects were asked to program the animation. A 2-way ANOVA was run against the test results. This showed that a statistically significant difference (F=7.38, P=0.012) existed in the post test scores of the subjects who viewed the animation and those who did not view the animation. No such correlation was found to exist for subjects who coded the animation, and no cross effects were found. From this result we can conclude that the subjects who viewed the animation and programmed the algorithm had a better understanding of the algorithm (as measured by the score on the post test) than subjects who simply programmed the algorithm. No such effect was found for the subjects who were asked to program the animation, and thus we conclude that programming the animation had no measurable effect on the understanding of the algorithm by the subjects.

Finally, the average number of questions which the subject answered correctly was correlated against the type of question (i.e. declarative, procedural, analytic, and general recursion), and whether the subject viewed or did not view the animation. These results are given in Fig. 4. From the chart, there are strong

		Viewed Animation		
		Yes	No	Avg
Constructed Animation	Yes	77	61	69
	No	82	68	75
	Avg	79	64	

Fig. 3. Average % correct on post test.

Type of Question	% Correct			
	Viewed	Did not View	F	p
Declarative	91	73	4.19	0.051
Procedural	91	79	3.45	0.074
Analytical	50	50	0.00	1.00
Recursion	64	32	5.35	0.029

Fig. 4. Results for question type vs treatment.

trends that the subjects who viewed the animation did better on the declarative and procedural questions than the subjects who did not view the animation, however the results fail to reach a significance at the 0.05 level. The results from the recursion questions, however, are surprising. There is nothing in the experiment which would lead to the prediction that subjects who had viewed an animation of the knapsack problem would do better at recognizing general recursion problems. However the results here show that there is a significant difference between those who viewed the animation and those who did not in their ability to recognize recursive problems. If these results are indeed valid, then the use of algorithm animation would have transfer effects for recursion problems, and possibly for other types of programs.

CONCLUSIONS

The purpose of this empirical study was to test the hypothesis that one effective way of using animation to teach algorithms is to have the programmers implement the algorithm as part of the overall learning experience. The results of this study argue strongly that this strategy is indeed an effective way to use animation. A second hypothesis is that implementing the animation as part of the programming assignment is also an effective way to use animation. However, the constraints of this experiment did not allow this conjecture to be effectively tested. It did, however, suggest that this type of timed study probably does not lend itself to implementing the animation, and some other experimental method will probably have to be devised to study the effect of using problem specific animations.

The unexpected and exciting result is that there seems to be transfer effects to general problem solving when animation is used as part of the learning experience. In this experiment, that transfer is from solving a specific recursion problem, the knapsack problem, to understanding more general recursion problems. This result needs to be verified, but if valid, has many implications, not only for how animation is used, but for how transfer occurs in learning programming constructs.

REFERENCES

1. Feldman, M. B., *Software Construction and Data Structures with Ada95*. Addison–Wesley, New York, 1996.
2. Brown, M. H., Exploring algorithms using Balsa-II. *Computer*, 1988, **21**, 14–36.
3. Stasko, J. and Kraemer, E., A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing*, 1993, **18**, 258–264.
4. Kann, C., Sibert, J., Feldman, M. and Vembar, N., Experiences using Ada and Java in computer science education. *ASSET Symposium*, 1996, pp. 41–49.
5. Wilcocks, D. and Sanders, I., Animating recursion as an aid to instruction. *Computers & Education*, 1994, **23**, 221–226.
6. Shih, Y. and Alessi, S., Mental models and transfer of learning in computer programming. *Journal of Research on Computing in Education*, 1994, **26**, 155–175.
7. Ramani, K. V. and Rao, T. P., A graphics based computer-aided learning package for integer programming: the branch and bound algorithm. *Computers & Education*, 1994, **23**, 1–10.
8. Anderssen, E. C. and Myburgh, C. P. H., The acquisition of operating systems concepts by computer science students. *Computers & Education*, 1992, **19**, 309–320.
9. Stasko, J., Badre, A. and Lewis, C., Do algorithm animations assist learning? An empirical study and analysis. *INTERCHI Conference'96*, 1996, pp. 61–66.
10. Elenbogen, B. and O'Kennon, M., Teaching recursion using fractals in Prolog. *SIGCSE Bulletin*, 1988, **20**, 263–266.
11. Aho, A., Hopcroft, E. and Ullman, J., *Data Structures and Algorithms*. Addison–Wesley, New York, 1983.