Visualization and Interaction in Learning Computer Science Concepts

Susan H. Rodger
Duke University
Durham, NC USA
rodger@cs.duke.edu

ABSTRACT

Our focus is on addressing the use of visualization and interaction software to aid in the learning of computer science concepts. We are interested in specifically if visualization and interaction aid in the understanding of algorithms in the area of theoretical foundations of computer science. As an example, we describe our previous work in the area of theoretical foundations of computer science, including software we have developed and a study we ran. We then ask several research questions that apply to that area.

1. INTRODUCTION

Visualization of an abstract concept provides an alternative representation of the concept the learner can associate with and may remember easier than the textual representation of the concept. Interaction with a concrete representation of an abstract concept allows one to experiment with the concept. There are many tools developed for aiding in the learning of computer science concepts that visualize the concepts and allow the user to experiment with the concept. These include algorithm animation tools for learning algorithms and data structures in the first two years. Computer science students have difficulty understanding what their computer program is doing. In CS 1, they may not understand the order elements are being processed in an array for an algorithm such as quicksort. In CS 2, they may think the nodes in their linked list are all hooked together when they are not. In an algorithms course, they may not understand the order the nodes in a graph are processed in a shortest path algorithm. An animation of the actions applied to the main data structure in their program may be helpful to the student's understanding by providing an alternative visual and interactive view.

A large focus on representing the understanding of algorithms is the area of algorithm visualization. Many software visualization packages were developed early on such as Balsa[2], Tango[12], Polka [13], Zeus[3], and AACE[4] that provide animations of common algorithms and data structures. Scripting packages such as JAWAA [1] were created to allow students to easily generate scripting commands that can be placed on a web page for animation. Newer algorithm visualization systems such as JHAVÉ[6] include additional engagement by integrating in questions that appear during the animation. Some studies such as [14, 15, 5] have been done to determine the effectiveness of using such tools to learn the concepts, but many more studies are needed.

There have been levels of engagement defined for algorithm animations. In [7] they define six levels of engagement as 1) no viewing, 2) viewing, 3) responding, 4) changing, 5) constructing, and 6) presenting, with 6 the highest level of engagement. Viewing an animation written by others is a passive activity in which the student can spend part of the time trying to figure out the representation of shapes and the use of colors, and still not fully understand the animation. Responding to questions about an animation challenges the student to think about what is happening in the animation. An animation that is interactive by allowing the student to change the input, and step forward and backward through the animation allows for better understanding. The student must think about what makes a good data set in order to test the animation for most cases. If the student has to make changes in the code, they must understand the code well and what each parts of the code represent. For even more understanding, the student should create the animation, but this must be fast and easy to do, or the student will spend too much time creating the animation instead of learning the concept. The highest level is presenting since a student must really understand the algorithm in detail in order to explain it to others.

2. PREVIOUS WORK

We have focused on developing animations of algorithms for a specific area, theoretical foundations of computing, developing the software JFLAP [10, 9] to experiment with formal languages and automata theory. JFLAP allows one to simulate theoretical machines and grammars, and to experiment with construction type proofs such as converting a nondeterministic finite automaton (NFA) to a deterministic finite automaton (DFA). The development of JFLAP spans twenty years and covers the experimentation of a full semester of topics in formal languages.

We ran a two-year study [11] in 2005-2007 to try to determine the effectiveness of JFLAP in enhancing the learning process and what additional value JFLAP added to the formal languages course. Our study included twelve universities. Results showed that students improved their scores from the pretest to the posttest, but the difference between the control group and the JFLAP group was not significantly significant. Our study did show that the students thought JFLAP was easy to use, they thought it made learning the concepts easier, it made the course more enjoyable, and they felt more engaged in the course.

3. INTERACTION AND VISUALIZATION QUESTIONS
We now propose several research questions on how visualization and interaction impact the learning of computer science concepts. Our examples will come from the computer science area of theoretical foundations and automata, but our questions could be applied to algorithms in other areas of computer science.

1. Does visualization and interaction with abstract computer science concepts aid in learning those concepts? What type of visualization or interaction aids in more learning than other types? How do we modify the software to gain more understanding?

When we teach automata and proofs about them, students learn this topic in two ways. For one way, they learn the theoretical definitions of structures, such as representing a finite state automaton mathematically as a 5-tuple using set notation. This method is textual and written down on paper by students. As an alternative way, they learn the visual representation shown as a graph. This method is also interactive, as they can build and experiment with the automaton. Once they build an automaton, they can run simulations on data and explore the processing of the data in steps.

A harder task is to take the automata and understand the proofs that use them. For example, consider the proof that for any NFA there is an equivalent DFA. This proof is a construction type proof that takes a general DFA and converts it into an equivalent NFA and then shows that a string accepted in one must also be accepted in the other. This proof can be taught in two ways. The first way is learning the proof mathematically, using the mathematical notation for a general DFA and converting that specific notation into the mathematical notation for an equivalent NFA. The second way is to see by example that the construction method actually works. Here a user would construct an actual DFA and then using JFLAP, go through the algorithm to construct an equivalent NFA based on this DFA. We understand this construction of a specific example is not a proof, but we believe that by doing both methods, this latter method helps in the understanding of the mathematical proof. That is, when a student has a visual image of the DFA in their mind, they can associate that visual picture with the mathematical representation needed in the formal proof.

There are also many issues as to whether or not the visualization and interaction are the best for learning, and what modifications should be made to the software to improve learning. Would observation by video or logging keystrokes be useful in determining how students are using JFLAP? Are they using the software in the best way? From past versions of JFLAP we have learned over the years to make sure the learner is engaged, but does not have to type in so much that they become disengaged. There is the correct balance between entering in text to make the user think and having some of it entered in automatically. The automatic part should start when it is determined the user has completed enough work to understand the concept.

2. Does visualization and interaction with abstract computer science concepts result in a more satisfied learning experience?

Our two-year JFLAP study mentioned earlier was based on a short survey to students. The positive response indicates a more thorough study would be of value to see what part of JFLAP or its use results in a more satisfied learning experience. Are they feeling more satisfied because any type of interaction is engagement, and any engagement is more satisfying? Does that feeling of satisfaction apply while constructing an automaton? Does it also apply to the proof construction algorithms, which is the more in depth material? Would interviews with students or groups of students aid in understanding what part of JFLAP the students are satisfied with? Would an in-depth study of JFLAP help in guiding the creation of other algorithm animation software? Would interviews help in finding other types of interaction that would be useful and satisfying?

3. Would combining automatic questions with the visualization and interaction improve the learning?

As mentioned earlier, the JHAVÉ system has quiz questions that appear during an animation that must be answered before moving on. JFLAP does not have that built in. If automatic questions were added to JFLAP, especially during simulations of input or during a proof construction, would they aid in the learning of the concept?

4. Does visualization and interaction with abstract computer science concepts result in a better retention rate?

The theoretical foundations course in the eyes of students is closer to a mathematics course than a computer science course. Most of their computer science courses involve programming, likely in a high level language. Traditionally, the theoretical foundations course is taught like a mathematics course with pencil and paper exercises, making it difficult to get immediate feedback and resulting in tedious errors. By creating software that visualizes concepts and allows one to get immediate feedback on problems, does that increase the retention rate for computer science students? Some students may be in computer science but are not so keen on math, and steer away from those computer science courses that have more in-depth math. Does the visualizations and interaction help retain those type of students?

5. Does visualization and interaction with abstract computer science concepts result in more diversity in the field of computer science?

There are small numbers of underrepresented groups and women in computer science courses. Being a minority, they may be more hesitant to ask questions in class and questions in general. Does the visual and interact software that provides a way to get automatic feedback on many questions help to retain those type of students?

4. IMPACT OF RESEARCH

Although our focus is on visualization, interaction and learning in the area of algorithms in theoretical foundations
of computer science, learning gains achieved by studying this area could be applied to studying the algorithms in other areas of computer science, especially the more mathematical areas.

The NCWIT Scorecard [8] mentions that the U.S. Bureau of Labor Statistics predicts a total of 1.4 million computing-related jobs by 2018 and that one-third of them will be unfilled because not enough students are choosing computer science as a major. Since the foundations are usually part of a computer science major, either as a course or a part of a course, improving the learning in this course would impact almost all computer science majors. If gains are shown in retention rates for this course, especially for women and underrepresented groups, that would lead to an increase in the number of students completing the major, and an increase in women and underrepresented groups majoring in computer science.

5. REFERENCES


