An Interactive Lecture Approach to Teaching Computer Science

Susan H. Rodger[1]
Computer Science Department
Duke University
Durham, NC, 27708-0129
email: rodger@cs.duke.edu

Abstract

Students get more out of an interactive lecture than a passive lecture because they are given time to think. This time allows them to determine if they understand a concept, and if not to ask questions. This understanding is crucial when concepts build on one another. We describe our positive experiences in teaching sophomore-level computer science courses in an interactive lecture format with a computer in the classroom.

1 Introduction

In an interactive lecture, students interact with the instructor and with other students. When a new concept is introduced, students are immediately given a problem to solve that forces them to think about the concept. Moreover, if students are placed in small groups, they can compare solutions with each other, which helps to point out their misunderstandings, and to build confidence when they are correct. This collaboration results in increased participation in class discussions. Although less material is covered during class, students obtain a deeper understanding of this material and can expand on this understanding outside of class.

In a traditional classroom, an instructor stands at the front lecturing while students furiously copy verbatim notes from a blackboard or overhead projector. Students concentrate so much on taking notes, that only a small percentage of the lecture is retained. In addition, whenever a concept is introduced that students do not understand, they rarely understand any of the material that follows. This is further hampered by students' shyness in asking questions, especially when the instructor is further into the lecture.

In other disciplines such as humanities, interactive lectures are common. For example, in a foreign language class, students and the instructor engage in conversation most of the class period, either discussing exercises or acting out real-life situations.

This paper describes our positive experience converting sophomore-level computer science courses (data structures, algorithms, and automata theory) from traditional lectures to interactive lectures and integrating interactive and visual computer tools into these lectures. Our lectures consist of a series of mini-lectures, problem solving, and discussions. Problems are solved in a number of ways during class: individually, in small groups, and on the computer with the instructor entering input suggestions from students.

In Section 2 we give background and in Section 3 we discuss the format of interactive lectures, with and without a computer. In Section 4 we provide details of three interactive lectures taught in the spring of 1994 and in Section 5 we mention tools for algorithms, data structures, and automata theory. In Section 6 we evaluate our approach, and in Section 7 we give concluding remarks.

2 Background

For the past five years at Rensselaer we have been teaching Fundamental Structures of Computer Science (FSCS) I and II, a sequence of sophomore level courses covering data structures, algorithms, and automata theory. FSCS I and II are the third and fourth courses for computer science majors. The first two years we taught these courses in the traditional lecture format, the next two years in the interactive lecture format, and this past year (93-94) in the interactive lecture format

[1] Supported in part by a Fellowship from the Lilly Foundation, a Rensselaer CIUE Development Grant for Educational Innovation, and the National Science Foundation's Division of Undergraduate Education through grant DUE-9354791. This work was done while this author was at Rensselaer Polytechnic Institute.

with a computer in the classroom. For the latter, an Xterminal connected to the campus file system had its output projected to a screen during class. The interactive and visual tools used include Xtango [13] for algorithm animation, FLAP [9] for automata theory, and LRparse [4] for parsing. The usual class size over five years was 30, but sometimes was as high as 60. Each class met two days a week for 80 minutes per class. We used two textbooks, one for automata theory [8] and one for algorithms [11]. In addition, we have supplemented many of the lectures with our own notes.

3 Format of an Interactive Lecture

We discuss our format of an interactive lecture, without a computer in the classroom, and with a computer in the classroom. If there is a computer, a mix of the two formats can be used.

3.1 Lecture Without a Computer

In classrooms lacking a computer, an interactive lecture consists of activities that use the available resources: paper and pencil, overhead projector and neighbors. We utilize these resources by fill-in-the-blank classnotes and group problem solving, both intertwined with discussion within groups of students and with the whole class.

We hand out notes that are mostly complete, but contain missing pieces and exercises that students fill in during class in a variety of ways. The instructor uses slides with abbreviated notes, also with missing pieces that are filled in as lecture proceeds, usually with input from the class. For short problems, students are given a couple of minutes to think and then volunteers suggest solutions. With the majority of notes on the handout, students can spend time listening to the instructor and thinking about the material.

Another form of interaction, group problem solving [12], simulates a real world environment. Students are partitioned into assigned groups of size three or four, and group members sit together during class. It is critical that students not pick their own groups, because they sit with their friends and are tempted to talk instead of paying attention. New groups are assigned every four to five weeks, so students will have the opportunity to meet and work with other students in the class. During a typical class, groups solve one to three problems ranging in length from five to twenty minutes. Volunteers present solutions by either describing in words, or drawing a picture on the overhead projector. Groups with alternative solutions are often eager to present their solution. The class finds it stimulating to see that a problem may have several solutions, and also to further discuss which, if any, solution is better. Sometimes, groups grade other group solutions. If two different problems are assigned, one to each half of the class, each group will grade a problem they did not solve, exposing them to additional problems during class.

3.2 Lecture With a Computer

A computer in the classroom is a powerful tool for solving complicated problems quickly, visually, and in more detail than can be shown drawing by hand on a blackboard or using several predrawn slides on an overhead projector. In particular, we use the computer during class to show animations of algorithms, to show how to use tools effectively, and to answer questions by instantly generating the answers.

The animation of an algorithm allows one to show the complete picture from start to finish of the processing of data. Flexible animations include user-supplied input, speed control, and pausing capabilities. Animations that allow the user to create the input allow for experimentation of the algorithm in different cases. Speed control allows one to examine a large amount of input, moving quickly over stages that are easy to understand in order to reach the more complicated stages, and showing these in slow motion. Pausing at a stage allows the class to discuss what just happened, and to allow students time to think about what will happen in the next stage.

The computer can be used during class to show students how to use tools effectively. Showing several examples using a particular tool will let the students see its capabilities. When students see how easy or useful a tool is, they are more inclined to use it themselves.

An advantage to having a computer in the classroom is being able to answer questions by *showing* the answer. Questions of the type "what will happen if we try ..." can be answered easily by trying it. Examples include modifying, recompiling, and running a program, and running an animation on different input.

Not only can the computer be used effectively during class in the ways discussed above, but saving the data used during class allows students outside of class to recreate animations and problems solved during class.

4 Example Interactive Lectures

We describe three interactive lectures with a computer given in the spring of 1994, and give a brief list of other topics covered in this manner.

4.1 Lecture on Pushdown Automata

We begin this lecture by giving the definition of pushdown automata (PDA), explaining the differences between PDA and finite automata, and describing the representation of transitions. Then the following problem is given.

Problem: Construct a nondeterministic pushdown automaton for L=$\{a^n b^n \mid n > 0\}$ that accepts by final state.

At this point, we use the computer and start the tool FLAP. A drawing window appears, and we show students how to edit a transition diagram by showing how to create states, identify start and final states, and create a transition between two states. Students are given five minutes to think about solving the problem above, and then we ask for volunteers from the class to describe their design for this PDA. With their suggestions, we draw the transition diagram. When the class is satisfied with the drawing, we show them how to enter an input string and run the string through the PDA. In FLAP we can quickly test whether the string was accepted, receiving a yes or no answer. If the string was not accepted but we think it should have been, we can run a trace. A trace processes one symbol in the input at a time, and shows the stack contents. Once the transition diagram for this PDA is correct, we save it in a file so students can recreate the example on their own.

Next we give the following problem to determine if students understand the concept of nondeterminism.

Problem: Construct a nondeterministic pushdown automaton for L=$\{a^n b^m \mid m > 0, m <= n <= 3m\}$ that accepts by final state.

Students are already sitting in assigned groups. For this problem, students are given fifteen minutes to discuss a possible solution with other members of their group. The instructor wanders around the room, sitting in on group discussions. If most students are stumped, the instructor halts the problem session and holds a discussion of nondeterminism. Then students continue working on the problem in their group. Finally, one group volunteers to describe their solution and we draw the solution using FLAP. This time when testing an input string, the slower step-by-step trace is used to show the nondeterminism. Each possible configuration (stack, current state, and remaining input) is shown (up to 12) that can be reached after a specific number of steps from the start state. Students are shown how to control the run when the number of configurations grows exponentially, by selecting certain configurations that are unlikely to end in a final state and either *killing* or *freezing* these configurations. Again, once the transition diagram is correct, we save it in a file. A solution to this problem using FLAP is shown in Figure 1.

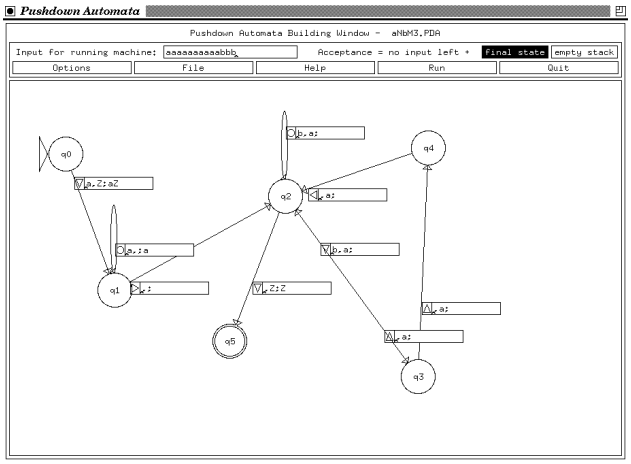

Figure 1: Pushdown Automaton in FLAP

4.2 Lecture on LR Parsing

This lecture begins with a brief definition of LR parsing and the study of an algorithm for converting a context-free grammar to a pushdown automaton that models the LR parsing process. At this point students are already familiar with context-free grammars and pushdown automata. Using the tool FLAP, we construct a picture of the transition diagram for the pushdown automaton constructed from the following grammar: S $\rightarrow$ aSb, S $\rightarrow$ b. Tracing input strings in this grammar, students are shown visually how the resulting PDA is nondeterministic (there are multiple configurations). By using additional information (lookaheads) we show how to decide which transitions to take to reach a final state and accept the input.

We then give a brief lecture explaining the algorithm for LR parsing, and how to use a parse table. We use the tool LRparse in conjunction with brief lectures to explain how to construct an LR(1) parse table for a given LR(1) grammar. First we type in a non-LR(1) grammar and try to continue. LRparse will inform us that the grammar is not LR(1). We modify the grammar to be an LR(1) grammar and then proceed. Second we explain the function FIRST, and then we use LRparse to type in the students' suggestions for the sets. LRparse identifies those sets that are incorrect. Lots of examples are done until the class feels comfortable with this concept. Third, we explain FOLLOW sets, and use LRparse in a similar manner to calculate FOLLOW sets for several grammars. Fourth, we give a brief lecture on item sets and constructing the deterministic finite automaton (DFA) that models the stack, and then use LRparse on the grammar above to construct such a DFA. Finally, we explain how to construct the parse table using the DFA, and use LRparse to construct this table for the grammar above. We then type in input strings and LRparse gives an animation

showing the stack contents and identifying the reduce and shift rules encountered. Additional examples are shown, and in particular, examples of grammars that are not LR(1) are shown. For these, the parse table can still be constructed and it shows the conflicts encountered.

Students are shown how to use LRparse to study the LR parsing process outside of class. In addition, example grammars are saved in files and can easily be input into LRparse to recreate the examples done in class.

4.3 Lecture on Red-Black Trees

This lecture begins by showing the class an Xtango animation on inserting several elements into an initially empty red-black tree, and pointing out the similarities and differences with these operations on binary trees. We then leave the computer for a brief lecture segment to formally define red-black trees, examine its properties and the algorithm for insertion. Next we run Xtango again, inserting elements one at a time. Each number added starts at the root, travels down to the proper insertion point, and then the tree is balanced via recoloring and/or rotations. By inserting numbers in an unbalanced manner, many insertions will require balancing. Before each insertion, the animation is paused and we ask the students to think about what type of balancing will be performed. They make suggestions, we run the animation, and then they can see if they were correct or not. For incorrect answers, we spend additional time discussing the algorithm, focusing on the decision of when to perform rotations and what type of rotations (single or double). Once students have the general idea for insertions, deleting elements is presented in a similar manner. For further study after class, we show the students how to run the animation so they can recreate the demo given during class. A red-black tree in Xtango is shown in Figure 2.

4.4 Topics of Other lectures

We briefly list other topics we have taught in our interactive lecture format with a computer in the classroom.

Xtango has been used to show animations of several sorting algorithms including shellsort, mergesort, quicksort, and heapsort. We have also used it to study the operations on data structures including binary trees, heaps, and binomial heaps.

FLAP has been used to study finite automata and Turing machines in a manner similar to the lecture on pushdown automata. We have also used it to study pushdown automata representing the LL parsing process.

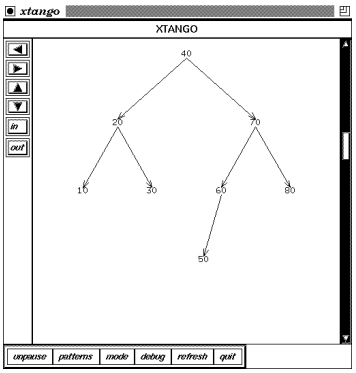

Figure 2: Red-Black Tree in Xtango

LLparse [4] has been used to study the LL(1) parsing process and the construction of LL(1) parse tables.

5 Tools available for Teaching Computer Science

We have mentioned the four tools FLAP, Xtango, LRparse and LLparse, which we have used in our lectures. There are a number of other tools available and many more are becoming available. In 1992, DIMACS held a workshop [5] for computational support for discrete mathematics where many tools were demonstrated for animating and analyzing algorithms. Additional automata theory tools include [1, 7, 14]. Additional tools for algorithm animation include [3, 6, 10, 2].

6 Evaluation

For two years we taught FSCS in the traditional lecture format, where the instructor lectured and the students wrote down notes. In this passive environment students were likely to fall asleep or if awake, constantly monitoring the remaining time. They were eager to leave and would start gathering materials, rustling papers loudly, at least five minutes before class was over. The majority of students would not volunteer to answer questions in this environment.

For the next two years, we taught this class in the interactive lecture format without a computer in the classroom. In this active environment, students ask more questions because they work problems during class and see right away what they do not understand. Over half the class was eager to volunteer to give the answers to problems that had been solved by their group. They had discussed the problem with other students and were more confident that the *group* solution was

correct. Many times a group problem is given for the last twenty minutes of class. In this case, many students are so involved working on the problem, they continue working even though class is over. Students were very enthusiastic to this type of environment. In fact, one group brought a sign with their group name and hung it up on the wall beside their group.

This past year, we taught this class in the interactive lecture format with a computer in the classroom. Approximately one-third of the lectures used the computer, either by demonstrating a concept via an animation or solving a problem visually and interactively with input from the class. The remaining lectures were taught in the interactive lecture format, and would have used the computer if appropriate software had been available. Overall, the students were very positive about the interactive classroom and the tools. Comments from students taken from the final evaluation forms this past year include the following comments. "The interactive format was very helpful and made the class more interesting." "The tools ... proved to be a really efficient way to check handwritten answers. They helped me to see when I would have been wrong, especially the traces on the PDA's!" "The tools were helpful in fine tuning what we needed to know". In fact, one-quarter of the class showed up for a workshop (on their own time) to demonstrate this method of teaching to other faculty at Rensselaer. Students came to the workshop because they were interested in encouraging more faculty to teach in this manner.

7 Conclusion

Although computer science is a technical field, lectures can be taught in an interactive manner by allowing time during class for students to think about problems, either singly or in small groups. This time to think generates discussion because students are better prepared, more confident, and more willing to volunteer. Less material is presented during class, but students understand this material more thoroughly. In evaluations, students overwhelmingly prefer the interactive lecture format.

A computer in the classroom makes interactive lectures more interesting. In particular, a computer can be used for visualizing and animating concepts, working problems with input from students, answering questions by showing the answer, and enticing students to use tools for their own experimentation.

References

[1] J. Barwise and J. Etchemendy, Turing's World, Kinko's Academic Courseware Exchange, Santa Barbara, CA, 1986.

[2] D. Berque, J. Bogda, B. Fisher, T. Harrison, and N. Rahn, The KLYDE Workbench for Studying Experimental Algorithm Analysis, *Twenty-fifth SIGCSE Technical Symposium on Computer Science Education*, p. 83-87, 1994.

[3] M. Brown, ZEUS: A System for algorithm animation and multi-view editing. *Proceedings of the IEEE 1991 Workshop on Visual Languages*, p. 4-9, Kobe, Japan, Oct. 1991.

[4] S. Blythe, M. James, S. Rodger, LLparse and LRparse: Visual and Interactive Tools for Parsing, *Twenty-fifth SIGCSE Technical Symposium on Computer Science Education*, p. 208-212, 1994.

[5] N. Dean, G. E. Shannon, eds. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Computational Support for Discrete Mathematics*, Vol. 15, American Mathematical Society, 1994.

[6] P. Gloor, *AACE - Algorithm Animation for Computer Science Education*, IEEE Workshop on Visual Languages, p. 25-31, 1992.

[7] D. Hannay, Hypercard Automata Simulation: Finite State, Pushdown and Turing Machines, *SIGCSE Bulletin*, 24, 2, p. 55-58, June 1992.

[8] P. Linz, *An Introduction to Formal Languages and Automata*, D. C. Heath and Company, 1990.

[9] M. LoSacco, and S. H. Rodger, FLAP: A Tool for Drawing and Simulating Automata, *ED-MEDIA 93, World Conference on Educational Multimedia and Hypermedia,* p. 310-317, June 1993.

[10] T. Naps, C. Hundhausen, and B. Swander, *GAIGS User Manual, Ver. 3.0.* Lawrence Computing Center Publications, Appleton, WI, 1993.

[11] R. Sedgewick, *Algorithms in C,* Addison Wesley, 1990.

[12] K. Smith, The Craft of Teaching Cooperative Learning: An Active Learning Strategy, *Frontiers in Education Conference Proceedings,* p.188-192, 1989.

[13] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, Sept. 1990.

[14] K. Sutner, Implementing Finite State Machines, *DIMACS Workshop on Computational Support for Discrete Mathematics*, 1992.