The Use of Computers for Teaching Artificial Intelligence at Rensselaer

Ellen L. Walker
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
walkere@cs.rpi.edu

Abstract

This paper describes the use of computers in the course Introduction to Artificial Intelligence at Rensselaer. Polytechnic Institute. In addition to programming projects completed by groups of students outside of class, in-class workstations are used for teaching the LISP language and for demonstrating relevant software. Finally, a class repository and newsgroup are used for communication between members of the class.

Introduction

An important component of the course Introduction to Artificial Intelligence at Rensselaer Polytechnic Institute is hands-on experience with problem-solving on the computer, both inside and outside class. This paper discusses two aspects of computing in the course: outside programming projects, and the use of in-class computing for instruction and demonstration.

Introduction to Artificial Intelligence is a 400-level course taken as an elective by computer science majors (around 65% over the last two years), computer engineering majors (around 15%), and students from other areas including engineering, mathematics, other sciences, and business. Approximately 75% of the students are undergraduates; the rest are graduate students, mostly but not entirely Masters candidates. For many (most?) students, this is the only course in artificial intelligence that they will ever take. Thus, it is more important to give them a broad overview and "literacy" in artificial intelligence in this course, than it is to give them a strong formal background upon which future courses can be based.

The topics covered in the course can be loosely grouped into four segments: LISP programming, problem solving as search, knowledge representation and its use in problem solving, and additional techniques and applications. Each of the four segments has at least one programming project associated with it. (See Figure 1).

The textbook used is *Artificial Intelligence* (Rich & Knight 1991). Textbook material is supplemented with articles from semi-technical sources such as *AI Magazine*, *IEEE Expert*, and *IEEE Computer* as appropriate, but we do not look at the primary research literature in this course. For LISP, students purchase either of two recommended textbooks: *Common LISP* (Hennessey 1989), or *Common LISP the Language* (Steele 1990). They are told that the first is a fairly good (but unavoidably incomplete) textbook, and that the second is a complete (but sometimes difficult-to-follow) manual. About half of the students that buy a LISP book choose each.

At Rensselaer, we are fortunate to have an advanced computing environment consisting of approximately 500 networked workstations, several of which are in classrooms with projection facilities. Most introductory courses in science and engineering make use of these workstations, so that most students are familiar with the systems by the time they take Introduction to Artificial Intelligence. All work-

<u>LISP (2 weeks)</u>
Programming assignment 1: introduction to LISP

<u>Problem solving and search</u>
Problem description
Solution recognition & ranking
Heuristics
Adversary search, game playing
Programming assignment 2: heuristic search

<u>Knowledge representation & problem solving</u>
Forms of representation
Logic, including brief introduction to PROLOG
Rules (forward & backward chaining, RETE)
Programming assignment 3: backward chaining
Expert systems in practice
Semantic nets & frames
Reasoning under uncertainty
Programming assignment 4: frames

<u>Additional techniques & applications</u>
Planning
Symbolic learning
Connectionist systems
Programming assignment 5: neural network
Vision
Natural language understanding

Figure 1: Introduction to AI Topics

stations have a common file system, which includes department directories as well as user directories. Thus, all course-related software, sample programs, and usage examples is collected into a single course directory, accessible to all students.

Each semester, I attempt to design a set of programming projects that span the topics of the course, have a design component, are relatively fun for the students, and provide a reasonable mix between asking the students to write custom code for AI algorithms and to use AI tools that they were given. In teaching Artificial Intelligence, I have found that the use of existing code is important to get students to complete a reasonable project in a reasonable amount of time. From talking with students in my class, it is clear that they have chosen this particular class as opposed to courses in artificial intelligence offered by other departments, including philosophy and electrical engineering, specifically because they know that this course is the one where they will get the most programming experience. Some students want the programming because they enjoy it, while others want projects to add to their resumes. In any case, the material covered by the programming projects is invariably the material that the students learn the best, as measured by exams.

Programming Projects

Although project time is limited, students master the material better when given significant design freedom, beyond what is often thought of as "modifying existing code". Thus, in this course, students write complete modules to be integrated with given modules into a larger program. Students are encouraged to work in groups, and each group reports who contributed what as part of their project write-up. In this way, strong programmers contribute more to the implementation, but even weak programmers contribute equally to the design. Since design is crucial to the understanding of the various topics, this allows weak programmers to get more from the projects than they would working alone, on even an easier project.

The remainder of this section describes the five projects from the Fall 1993 semester. Except for the first, each project emphasized a technique and a problem to solve, and included some code to help the students solve the problem, but not enough to hamper their design freedom. In projects 2 and 3, the students were given modules to link with their own to build more interesting systems than they could otherwise do in the allotted time. For projects 4 and 5, the provided code was in the form of tools for the students to use rather than customized modules. In this way, the students were given experience with the kind of tools that artificial intelligence professionals might be expected to use.

Project 1 — Introduction to LISP

The first project was designed to give each student a basic understanding of the LISP programming style and the use of recursion as a programming paradigm. Unlike the remaining projects, no code was given, and the students were required to work alone. The students were asked to recursively determine the complete set of prerequisites for courses, given the catalog specifications for individual courses. Essentially, the students had to determine the transitive closure of a prerequisite graph.

Although the project was short and simple, it introduced the students to two essential features of LISP: recursive procedures and property lists (used to store the catalog information). Additionally, students were encouraged to use the interactive debugger to understand their programs so that they would be able to use them to debug more complex projects later.

Project 2 — Adversary Search

For the second project, the technique was adversary search. The students were given the code necessary to play a game (Dots), exclusive of the control portion to choose the next move. Using this infrastructure, their job was to write the control portion for an automatic dots-playing program. Included in the given code was a "dummy program" that allowed a human to play against the machine. Part of the grading process was to run a tournament among the student programs for extra credit points.

One of the advantages of this assignment was that it was clear to the students that their part of the project was to add "intelligence" to a dumb game-playing program. The competitive nature of the assignment added to the enthusiasm with which it was received.

I wrote the code for Dots, and similar code for Klondike solitaire. In each case, the code included the necessary data structures to represent the playing field, a function to perform each type of move, a function to check move legality, and a top-level function to play the game, calling the players' control functions alternately in the case of dots, repeatedly in the case of solitaire. I would be happy to make this code available to others who teach AI if they feel it would be helpful.

Project 3 — Backward Chaining

For the third program, the technique was backward chaining. Students were given a unification procedure and a library to handle variables, and were required to write a backward chainer for deductive retrieval. Rules for a sample domain (family relationships) were given, but as part of the assignment, each team had to apply their reasoner to their own domain and set of rules. Thus, the students first had to develop an AI tool (the reasoner), and then use it (by creating their own rules and facts).

The unification procedure and variable library were adapted from code in an FTP repository provided by Kevin Knight to accompany the textbook, *Artificial Intelligence*. In general, I have found this FTP repository quite useful, and would highly recommend it to others using the book.

The neural network backpropagation simulator, used in project 5, also came from this repository.

Project 4 — Frame Representation

The technique addressed in the fourth program was using frames as a representational tool. Students implemented a simplified story-understanding program, using the Framekit frame system. Students were given the Framekit system (and manuals), and were required to design frames to understand stories about going to the movies. Since the representation was the main point of the project, they were allowed to translate the stories into frames by hand, but the answers to the questions had to be extracted automatically from the frames. Given more time, a natural extension to this project would be to apply natural language techniques to automatically translate the stories to frames.

This project was primarily a design project; the students generally wrote very little code, although their frames were extensive. Because of the open nature of the assignment (a frame for "seeing movies" that could encompass several stories, including one where two people rented a video and watched the movie at home), there was the greatest variety of solutions to this project. The students, as users of the given frame system, were forced to discover its advantages and limitations, and design their software accordingly. This experience is probably closer to what many of them might do after graduation than writing custom code. It also gave them a chance to use a much more complex tool than they could have written, and served as a contrast with the mini-deductive retriever that they wrote for project 3.

The frame system used was FrameKit, version 2.0, from Carnegie Mellon University (Nyberg 1988). I received FrameKit as part of the VANTAGE modeling system that I use for my research. FrameKit is a fairly complete frame system, but does not have the nice additions (*e.g.* graphical specification) that many commercial systems have. On the other hand, it did not require special hardware to run, and was already available on the systems used in class. I would be interested in evaluating other representation systems based on Common LISP (so that students can add their own demons to frames).

Project 5 – Neural Nets

The final project was a hands-on experience with neural networks. The students were given a complete database of all possible tic-tac-toe games and their outcomes, and the network was to learn to recognize a game won by X. The students used a backpropagation simulator, and were asked to experiment with different network topologies and training / test sets to see how the learning was affected. Students were also asked to analyze the activities of the hidden nodes to try to determine what features were being learned.

This project was done at the end of the semester, when most of the students were at their busiest, and there was not much time for the assignment. Thus, the goal was to give as much hands-on experience with neural networks as possible, without overloading the students. By structuring the project more as a lab exercise than a programming project (but still with a design component), the goal was met. The students developed a good sense of how neural networks learn, and not incidentally, how long it can take! Like the frame project, this one gave them a sense of how AI technology is used.

In-Class Computing

For the Fall of 1993, Introduction to Artificial Intelligence was taught in one of the classrooms at Rensselaer that has a built-in UNIX workstation and projector. Although this classroom was one of the first, an increasing number of classrooms are being equipped in this manner. The classroom equipment was invaluable, especially when teaching LISP at the beginning of the semester, and when introducing each of the programming projects.

In the Fall of 1994, the class is again scheduled in the same classroom. This year, we plan to increase the use of the projection facilities to include in-class demonstrations of existing AI programs and interactive tools that students can later use on their own to help understand concepts presented in the course. An example of an interactive tool is described in (Vastola & Walker 1994).

In-Class Introduction of LISP

The in-class workstation was a perfect tool for the introduction of LISP. Instead of using handouts showing what to type and what would happen, I was able to open a window on the workstation, start up LISP, and run through a live sequence of examples. The class sessions basically followed the sequencing of the recommended LISP textbook (Hennessey 1989), but was able to vary based on students' interaction during class. This format was much more lively than my traditional overhead slides, and students were much more involved in the class. Even typographical errors were chances to introduce facets of LISP such as the error-handling facilities and the debugger.

A log of each classroom session was saved in the class directory for students to review at their leisure. Example files (both those loaded during a session and those created during a session) were also saved in the class directory. Thus, students were freed from taking notes during the demonstrations, so they could more fully participate in class. To emphasize important points and help to organize the log, I typed LISP comments.

I spent the same amount of classroom time on LISP as in previous semesters, but found that with the live presentation I could cover more material and the students were able to retain more. This semester there were fewer comments about the overload of material during the LISP unit than in past semesters.

Support for Programming Projects

Except for the first programming project, which is a self-contained practice problem for getting started with LISP,

each programming project included significant code, and the students needed only to write the portion that was most relevant to the technique being studied. The in-class workstation was invaluable for introducing the "given" portions of each problem.

On the day each program was introduced, the relevant code was presented in-class. When each project was introduced, the relevant software was explained and demonstrated in class. For the game-playing project, the demonstrated software included "dummy code" for the controller so that actual games could be demonstrated. Other software, including the unification procedure and Framekit, was demonstrated using sample data. In all cases, the sample data or dummy code was made available in the course directory for students to peruse on their own.

Additional Program Demonstrations

This year, I plan to add two more uses of the in-class workstation. The first is to include demonstrations of some of the increasing number of artificial intelligence systems and tools that are becoming available over the Internet, as well as demonstrating examples of current research. (One of my goals in attending this workshop is to learn more about what software is available for demonstration.) The second use of the workstation is to incorporate interactive tools that help students understand concepts presented in class. The first example of such a tool will be used this year (Vastola & Walker 1994).

The Course Newsgroup

One feature provided to all courses at Rensselaer is a newsgroup in the rpi.courses hierarchy. The Introduction to Artificial Intelligence newsgroup, rpi.courses.ai, was used by the professor and TA for distributing late-breaking information on the course such as errors in handouts, extensions on projects, and changes in office hours. The newsgroup also provided a forum for students to ask questions about course topics and for students to contact each other for course-related purposes, such as setting up project teams and study groups. At the end of the semester, the newsgroup archive was a record of these discussions, allowing for better planning for future semesters.

Conclusions

The students' experience in the course Introduction to Artificial Intelligence is enhanced by the use of computers, both in the classroom and for special projects. By allowing the students to incorporate existing code in their own designs, the projects can be both interesting and do-able in the time allotted. The in-class workstation and class repository allow programming-oriented instruction in a more enjoyable and intensive manner, as well as allowing live demonstrations of existing artificial intelligence applications and pedagogical tools.

References

Hennessey, W.L. 1989. *Common LISP.* New York, NY: McGraw Hill.

Information Technology Services. 1993. ITS Computing: A guide, Internal Publication, Rensselaer Polytechnic Institute.

Nyberg, E.H. 1988. The FrameKit User's Guide Version 2.0. Technical Report, CMU-CMT-88-107, Center for Machine Translation, Carnegie Mellon University.

Rich, E. and Knight, K. 1989. *Artificial Intelligence (2 ed.).* New York, NY: McGraw Hill.

Steele, G.L., Jr. 1990. *Common LISP the Language.* Bedford, MA: Digital Press.

Vastola, D.A. and Walker, E.L. 1994. Interactive Learning Tool for Statistical Reasoning with Uncertainty. In these proceedings.