An Undergraduate Introductory AI Course

Richard E. Korf
Computer Science Department
University of California, Los Angeles
Los Angeles, Ca. 90024
korf@cs.ucla.edu

Abstract

We describe the course content, programming assignments, and syllabus for Computer Science 161, "Fundamentals of Artificial Intelligence", developed and taught at UCLA for the past eight years.

Introduction

Computer Science 161, "Fundamentals of Artificial Intelligence" was first taught in 1986 at UCLA, and has been taught roughly twice a year since then. It is an upper division undergraduate course, taken by about 50 juniors and seniors per offering. A small number of graduate students also take the class. The course has a strong programming component, with the students writing programs weekly in LISP. It is a one-quarter course, with ten weeks of instruction, and four hours of lecture and two hours of recitation each week. Grades are based on the programming assignments, and a midterm and final exam.

The course is a restricted elective in both our computer science and also computer science and engineering undergraduate majors. Students must take either this course, or one of two other more specialized undergraduate AI courses in natural language processing or computer vision. For the graduate students, it serves as a basis for a set of graduate courses in specialized areas of AI, such as problem solving and search, probabilistic reasoning, natural language processing, computer vision, and neural networks. While this course is not a prerequisite of these graduate classes, it provides an overview of the field and places these other courses in perspective.

Below we describe the content and organization of the course, and the programming assignments. The course syllabus is also included.

Course Content

Anyone teaching a survey of artificial intelligence must adopt a taxonomy of the field. What makes this interesting in AI is that there is no generally agreed upon taxonomy in the community. For some, by default, the taxonomy is very shallow, consisting of a one-level listing of different topics with no further structure. This approach was evident in the early AI texts.

In contrast to this approach, we adopt a deeper hierarchy. The top level consists of search algorithms, knowledge representation techniques, and specialized applications. This taxonomy has gradually emerged, and is adopted by a number of recent AI texts. We discuss each of these areas in turn.

We further subdivide the area of search algorithms into a discussion of problem spaces and the problem-space hypothesis, brute-force search algorithms, heuristic search techniques, and planning methods. In the context of problem spaces we describe three different classes of problems: single-agent problems, two-player games, and constraint-satisfaction problems. The brute-force search algorithms include breadth-first search, depth-first search, depth-first iterative deepening, backward chaining, and bidirectional search. The heuristic algorithms covered include best-first search, uniform-cost search, pure heuristic search, A*, iterative-deepening-A*, and depth-first branch-and-bound. We also cover alpha-beta minimax search for two-player games, and chronological backtracking and heuristic repair for constraint-satisfaction problems. In planning, we discuss subgoaling and the General Problem Solver.

Next, we move on to knowledge representation techniques. For the simple problems typically discussed in the context of search algorithms, states are represented by specific data structures designed for each problem. For more complex problems, we don't want to have to design special-purpose representations from scratch, but would like to encode the necessary knowledge in more general-purpose formalisms.

The knowledge representation we spend the most time on is predicate calculus, with resolution theorem proving as the inference mechanism. We show that search techniques must be used to do the reasoning. We also discuss non-monotonic reasoning, using default logic as the main example. As an alternative to a non-monotonic logic, we cover probabilistic reasoning and Bayes theorem. Next we consider produc-

tion systems as a representation of knowledge, and finally structured representations such as semantic nets, frames, and scripts.

The last high-level group of topics in the class are the specialized applications of AI. These include expert systems, natural language processing, speech understanding, computer vision, and robotics. To some extent the presentation of the knowledge-representation techniques overlap with the treatment of the applications. For example, expert systems are discussed along with production systems, and semantic nets, frames, and scripts are treated in the context of natural-language understanding.

For completeness, we also briefly discuss neural nets as an alternative to the symbolic processing paradigm.

Programming Assignments

Programming in LISP is a large part of the course, and indeed the activity that the students spend most of their time on. The rationale is that real understanding of algorithms and data structures or representations comes from programming them. In practice, a course in LISP is a prerequisite to this course.

There are weekly programming assignments, which the students are required to write in pure applicative LISP. The reason for this latter requirement has less to do with AI than with general computer science considerations. This course requires students to program recursively rather than iteratively, a style they are much less familiar with.

The first assignment asks the student to give different implementations of the Fibonacci function, including both exponential and linear-time versions, and to analyze their complexity. The main point of this assignment is to drive home the point of combinatorial explosion. They realize that even computing the 50th Fibonacci number could takes days using the naive algorithm, and that the 100th would take billions of years.

The next assignment is to implement several different brute-force search algorithms, including breadth-first, depth-first and depth-first iterative deepening. The inputs to these functions are explicit trees represented as LISP lists, and the output is a list of the atoms in the tree in the order they would be visited by each algorithm.

The students then implement the A* algorithm, using a problem such as the Five Puzzle, a two by three variant of the familiar Eight Puzzle. The smaller puzzle requires less memory for the open list, and allows an inefficient implementation in interpreted LISP to run in a reasonable amount of time.

Next the students implement minimax search with alpha-beta pruning. To eliminate the overhead of implementing a real two-player game, the input to their function is a tree represented as a list, with the atoms being the numeric static evaluations of the associated frontier nodes.

The N-queens problem is then solved using chronological backtracking, with each solution represented as an N-element list giving the column positions of the queens in each row.

Another assignment concerns using the ideas of the General Problem Solver to solve the Towers of Hanoi Problem. What makes this problem more interesting than the standard solution is that the student's program must find an optimal solution for any pair of legal initial and goal states.

The next assignment is a resolution theorem prover for propositional logic. A related assignment is to write a program to transform an expression in propositional logic into clause form. The solution to this problem typically involves a separate function for each step of the transformation, and most of the code of each of these functions performs pattern matching.

We then ask the students to implement a simple production system. As an application of the production system we ask them to write productions to transform propositional logic expressions into clause form. This solution to the problem is much simpler than their first brute-force approach to the problem, with all the pattern-matching code collected together in the interpreter, and each transformation expressed by a simple production that is easier to read and understand. Another application of the production system is to do parsing of simple natural language sentences, with the production rules representing the grammar.

Syllabus

The following is the syllabus for the last offering of CS161 in the Spring of 1994. Each item represents approximately one two-hour lecture.

1. Introduction to course and artificial intelligence, LISP expressions, functions, conditionals.
2. Recursion, LET, list primitives, recursive list functions.
3. Problem spaces, problem types, search trees and graphs, AND/OR trees.
4. Brute-force search, breadth-first, depth-first, depth-first iterative-deepening.
5. Bidirectional search, heuristic search: best-first, uniform cost, pure heuristic search, A*,
6. Iterative-deepening-A*, Depth-first branch-and-bound.
7. Two-player games, minimax search, alpha-beta pruning.
8. Constraint-satisfaction problems, backtracking and heuristic repair.
9. Planning, subgoaling, General Problem Solver.
10. Midterm exam.
11. Propositional and predicate calculus.

12. Resolution in propositional logic, conversion to clause form.
13. Unification, resolution theorem proving in predicate calculus, Prolog.
14. Non-monotonic reasoning, default logic, decidability, multiple extensions.
15. Probabilistic reasoning, Bayes theorem.
16. Production systems, expert systems.
17. Natural language processing, grammars, syntax and semantics.
18. Structured representations, semantic nets, frames, scripts.
19. Perception, speech, vision, line labelling of polyhedral scenes.
20. Neural networks.

Acknowledgements

The preparation of this paper was supported by NSF Grant No. IRI-9119825, and a grant from Rockwell International.