Conventional AI: Search

Jocelyn Paine

June 3, 1997

1 Introduction

This tutorial will be about an approach to problem solving. Suppose we're given a scrambled Rubik cube, and we want to unscramble it. Humans can do this, some more easily than others; how could a program do likewise? And if we could make such a program, would it be a good imitation of what humans actually do?

The simplest method would be to try each possible configuration in turn, until we find the one we want. This is called "exhaustive search".

We'd start with our original cube. We'd then find all possible configurations which we could get to by rotating one of its layers (rotating one layer of a Rubik cube is the simplest way to get a new arrangement: it's like one move in a game). Are any of the new configurations unscrambled? If so, we're finished. If not, we try generating even newer configurations from the new ones. And so on.

Provided we can generate all other configurations from our original cube, this is guaranteed to succeed. If we could generalise the approach, a program that worked in this way would make a wonderful general-purpose problem solver. All we'd have to do would be to put in a description of the start state and goal states, and a table of legal moves, and we'd get an answer.

Sadly, for most problems this would take an impossibly long time. In playing a game of draughts, it would be necessary to consider some 10^{40} possibilities arising from each move: at the rate of one possibility per microsecond, it would take about 10^{24} centuries to examine them all.

We might try to make search more efficient by making it no longer exhaustive. Consider when we take a cube and list all the possible new configurations. Perhaps two of them look much less scrambled than the others. So perhaps they're most likely to lead to a completely unscrambled cube. So we concentrate on the possibilities arising from each. In general, we pick the least scrambled arrangement at each stage, hoping that it's nearest our goal. This is the idea behind *heuristic search*, where you have an *evaluation function* that ranks some states as nearer to a goal than others.

This method does in fact work for some problems. However, it won't work very well for the cube, as you'll find if you try it. It is possible though to build programs that work more intelligently, and that don't just use distance from the goal as a measure of how to proceed.

One such program was the General Problem Solver of Newell, Simon and Shaw. I'll get on to this next week. Like Evans' program, it's a key stage in AI.

However, to understand GPS, you have to know not only about operators, but also about representing problems as sets of states. It's probably a bit much to cope with that and GPS in one week. So this week, I'll concentrate on states of problems, and on exhaustive and guided search amongst them. It's useful in any case to know about search trees, exhaustive search (and its varieties: depth-first and breadth-first amongst them), and guided search; in practical AI you can fall back on them if less brute-force methods fail.

This week, I won't set an essay. Instead, I'd like you to work through part of one of the Open University course units. It introduces the ideas I've talked about, and does so probably better than the standard textbooks. (In general, Open University course units — not just the Cognitive Science ones — are often better designed than other texts, and may help if you can't find adequate help elsewhere).

So please read and work through *Learning and Problem Solving 3*, Open University Course D303, Block 4, Units 26–28. There are copies in the Psychology library (in the Oversize section, marked AA:O 2–O M), and the RSL (on the Open University bookcase — 2nd bookcase on left as you go in the back entrance to the Lankester Room).

Please read these sections, and do the SAQs (Self-Assessment Questions) in each, except for SAQ 5, which asks about a programming language you won't know (if you want to try it for Prolog, Basic, or whatever you do know, then do).

Ensure that you understand the topics I've listed below. You won't of course have been supplied with the Missionaries and Cannibals puzzle or 15-puzzle the course mentions, but you can make them out of cardboard if you think it necessary).

1 Introduction

2 Exhaustive search

2.1 The Missionaries and Cannibals problem.

2.2 Representing moves and states in the problem.

2.3 Illegal states, loops, backing-up, search trees, nodes, links.

2.4 Depth-first search, loop check, backing-up.

2.5 Breadth-first search.

2.6 Forward versus backward search. Goal-directedness.

2.7 Computers versus humans — hunches, inhibitions, and memory limits.

3 Evaluation functions and heuristic search

3.1 The 15 puzzle. Combinatorial explosion, states.

3.2 Guided search, evaluation functions, heuristics.

3.3 Example of evaluation function.

4 Game playing

4.1 Look-ahead tree.

4.2–4.4 Games and minimaxing.

4.5 The horizon effect.

4.6 Strategic programs.

4.7 Comparison with humans. De Groot, Chase and Simon, Eisenstadt and Kareev (the latter is the reference on representations in Go and Go-Moku I mentioned last time).

2 Other references on search

The OU course will give you all you need to know about breadth-first and depth-first search and evaluation functions. You must also know the difference between blind and guided (heuristic) search. The OU book does introduce these, but in a way which you may find confusing. To make things clearer, read chapter 4 of *Artificial Intelligence* by Winston, and work through an example of *hill-climbing*.

It's as well to know that there are other search methods, such as A*. It's useful to realise why we need so many different search methods, and how their characteristics differ. See Winston for this.

- Article on *Search* in *The Encyclopaedia of Artificial Intelligence*, (RSL: Comp Bd 56) edited by Shapiro. A handy compare-and-contrast of different methods for the computer-minded. It describes depth-first iterative-deepening, a method not mentioned in my other references. This is useful when you want something that has the advantages of breadth-first without its massive space overheads.

- *Artificial Intelligence*, Winston, Chapter 4. Gives algorithms for the various methods. Since they're all in the same style, they're useful if you want to compare how a programmer would solve search problems. Also useful for comparison with GPS next week.

- *Artificial Intelligence* by Charniak and McDermott: RSL stacks; Psychology. Because so many of the problems we discuss are puzzles, you may be left with the impression that these methods are useless for "real-world" problems. Charniak and McDermott give an example of where they can be useful.

3 Further reading — game trees

I omitted this from the readings above. If you're interested, look at the following sections of the OU book:

4 Game playing

4.1 Look-ahead tree.

4.2–4.4 Games and minimaxing.

4.5 The horizon effect.

4.6 Strategic programs.

4.7 Comparison with humans. De Groot, Chase and Simon, Eisenstadt and Kareev (the latter is the reference on representations in Go and Go-Moku I mentioned last time).

There's a nice half-page explanation of minimaxing in *Chess playing programs and the problem of complexity* by A Newell, J C Shaw and H A Simon, in *Computers and Thought* edited by E A Feigenbaum and J Feldman: RSL; Psychology: KH:F.

For a discussion of the limitations of minimaxing, look at the article by Berliner in *Thinking — Readings in Cognitive Science*, edited by P.N.Johnson-Laird and P.C.Wason (CUP, 1977).