2
Artificial Intelligence

McDermott – section on game-playing

Cognitive science is based on the idea that computation is the main thing going on in the brain. Cognitive scientists create computational models of the sorts of tasks that brains do, and then test them to see if they work. This characterization is somewhat vague because there is a wide range of models and testing strategies. Some cognitive scientists are interested in discovering exactly which computational mechanisms are used by human brains. Others are interested primarily in what mechanisms could carry out a particular task, and only secondarily in whether animal brains actually use those mechanisms.

This discipline has been around for about half a century. Before that, psychologists, linguists, neuroscientists, and philosophers asked questions about the mind, but in different ways. It was the invention of the digital computer that first opened up the possibility of using computational models to explain almost everything.

It is possible to approach cognitive science from various points of view, starting from psychology, neuroscience, linguistics, or philosophy, as previous authors have done (Jackendoff 1987; Dennett 1991; Churchland and Sejnowski 1992). My starting point is going to be computer science. The application of computer science to cognitive science is called *artificial intelligence*, or *AI*. AI has been used as a platform for a more general discussion before (Minsky 1986; Hofstadter and Dennett 1981), but rarely in a way that takes philosophical questions seriously.[1]

One misapprehension is that artificial intelligence has to do with intelligence. When the field started, it tended to focus on "intellectual" activities such as playing chess or proving theorems. It was assumed that algorithms

for hard problems like these would automatically apply to other areas requiring deep thought, while "lower level" activities like walking or seeing were assumed to be relatively straightforward applications of control theory. Both of these assumptions have been abandoned. There turns out to be no "General Problem Solver"[2] that can solve a wide range of intellectual problems; and there turns out to be no precise boundary line between lower-level activities and high-level "thinking." There are plenty of boundaries between *modules*, but no difference in computational mechanism between peripheral and central modules, and no clear support for the notion that as you penetrate deeper into the brain you finally reach an area in which you find "pure intelligence."

It would be better, therefore, if AI had a different name, such as *cognitive informatics*. Since we can't wave a magic wand and cause such a terminological revolution, we will keep the old label for the field, but let me repeat: AI will have a lot to say about what role computation plays in thinking, but almost nothing to say about intelligence.

People tend to underestimate the difficulty of achieving true machine intelligence. Once they've seen a few examples, they start drawing schematic diagrams of the mind, with boxes labeled "perception" and "central executive." They then imagine how they would carry out these subtasks, and their introspections tell them it wouldn't be that hard. It doesn't take long for them to convince themselves that intelligence is a simple thing, just one step beyond Windows 98. The truth is that boxes are much easier to label than to fill in. After a while, you begin to suspect the boundaries between the boxes were misdrawn. Fifty years into the AI project, we've become much more humble. No one would presume any longer to draw a schematic for the mind. On the other hand, we have more concrete ideas for solving particular tasks.

Some of the topics in this chapter may be familiar to some readers. I urge them not to skim the chapter too hastily, though, because when I allude to computational mechanisms later in the book, I'm thinking of mechanisms such as the ones I describe here. That's not to say that there won't be plenty of new algorithms and data structures discovered in the future. I'm just not assuming the need for, or predicting the discovery of, any powerful new ideas that will revolutionize the way we think about computation in the brain. There is nothing up my sleeve.[3]

Computer Chess

To get a feel for what artificial intelligence is trying to do, let's look at a particular case history, the development of computer programs to play games such as chess. Playing chess was one of the first tasks to be considered by researchers, because the chessboard is easy to represent inside a computer, but winning the game is difficult. The game has the reputation of requiring brains to play, and very few people ever get good at it. There is no obvious algorithm for playing chess well, so it appears to be a good domain for studying more general sorts of reasoning, or it appeared that way at first. In the 1950s, Allen Newell, Clifford Shaw, and Herbert Simon wrote some papers (Newell et al. 1958) about a program they designed and partially implemented, which used general-purpose symbolic structures for representing aspects of chess positions. However, over the years chess programs have become more and more specialized, so that now there is no pretense that what they do resembles human thinking, at least not in any direct way.

Almost all chess programs work along the lines suggested in early papers by Claude Shannon and Alan Turing (Shannon 1950*a*, 1950*b*; Turing 1953), which build on work in game theory by von Neumann and Morgenstern (1944). A key feature of chess is that both players know everything about the current state of play except each other's plans. In card games a player is usually ignorant of exactly which cards the opponent has, which adds a dimension we don't need to worry about in chess, where both players can see every piece. We can use this feature to construct a simple representation of every possible continuation from a chess position. Imagine drawing a picture of the current position at the top margin of an enormous blackboard. Assuming it is your turn to move, you can then draw all the possible positions that could result from that move, and join them to the original position by lines. Now below each such position draw a picture of every position that can be reached by the opponent's next move. Continue with your move, and keep going until every possible position reachable from the original position has been drawn. (This had better be a really big blackboard.) The process will go on for a long time, but not forever, because at any position there are only a finite number of available moves and every chess game must come to an end. The resulting

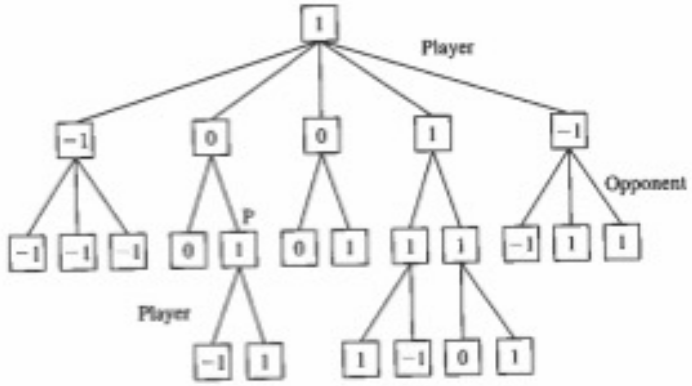

Figure 2.1
Game tree

drawing is called a *game tree* (figure 2.1). Each position is indicated by a square. Pretend the square contains a chess position reached at a certain point in a game; I'll explain the numbers shortly. The *root* of the tree is at the top and the *leaves* at the bottom. It looks more like a tree if you turn the picture upside down, but it's called a tree even though it's usually drawn with the root at the top. The root is the original position, and the leaves are the final positions, where the game has ended and no further moves are possible.

We now label the leaves with 0, 1, or −1, depending on whether the game is a draw, a win for you, or a win for your opponent, respectively. To describe what happens next I need to introduce another definition. Let the *children* of a position B refer to the positions reachable by making one move in B. (I apologize for the mixing of a family-tree metaphor with the botanical-tree metaphor.) Now pick a position P that has only leaves as its children, that is, a position in which no matter what move is made the game is over. Suppose that at that position it is your turn to move. If there is a child position labeled 1 (meaning "you win"), then you can win in position P by making the corresponding move. Hence P can be labeled 1 as well. If no child is winning, then if there is a child labeled 0 (draw), you can at least draw, so P should be labeled 0. Otherwise, no matter what you do you will lose, so P is lost, and should be labeled −1. Actually, this is impossible in chess; the last person to move can't lose. We include this possibility for completeness, because we're going to see the same pattern elsewhere in the tree: at any position where it's your turn to move, the position should be labeled 1 if any child is labeled 1; 0 if no child is labeled 1 but some child is labeled 0; and −1 if every child is labeled −1. In other words, at every position where it's your turn to move, you should label it with the maximum of the labels attached to its children. At positions where it's the opponent's turn to move, you should, by a similar argument, place a label equal to the *minimum* of the labels attached to the children.

If we continue in this way we will eventually label every position. The original position will be labeled 1 if you can force a win, −1 if your opponent can, and 0 if neither of you can. This claim may not seem obvious, so let's consider in more detail the strategy you can follow if the label is 1. There must be some child position labeled 1, so pick one and make the move that gets you there. At this position it is the opponent's turn to move, so if it's labeled 1 then every child position must be labeled 1. So no matter what the opponent does, you will be in a situation like the one you started with: you will have a position with at least one child position labeled 1. You will be able to make the corresponding move, and then wait for the opponent to pick among unappetizing choices. No matter what happens, you'll always have a position labeled 1 and can therefore always force a situation where the opponent must move from a position labeled 1. Sooner or later you'll get to a position one step from a leaf, where you have a move that wins.

This idea was first made explicit by von Neumann and Morgenstern. It sounds like a surefire recipe for winning at chess (and any other board game with complete information about the state of play and no random element such as dice or roulette wheels). Of course, no person can play the game this way, because the equipment (an enormous blackboard) is too unwieldy, and it would take way too long to work out the whole game tree. But computers can get around these problems, right? They can use some kind of database to represent the tree of positions, and use their awesome speed to construct the tree and label all the positions.

No, they can't. There is no problem representing the board positions. We can use a byte to represent the state of a square, and 64 bytes to represent the board.[4] It's not hard to compute the set of all possible moves

from a position. There are a finite number, and there are unambiguous rules that determine how each piece can move. The problem is that there are just too many possible positions. At the start of the game White (who always moves first) has 12 possible moves, and Black has 12 possible replies. As the game progresses, the number typically grows, then shrinks again when lots of pieces are off the board. A game can go on for a long time, but some games are over quickly. An accurate estimate of how big the game tree is would have to deal with these factors of varying width and depth of the tree, but we will neglect all complexities in the interest of getting a quick but crude estimate. For simplicity, let's just assume there are 10 possible moves at each position and that the entire tree is 100 ply deep. A *ply* is a move by one player, so a 50-move game corresponds to 100 ply. The total number of positions after one ply is 10; from each of those we can reach 10 more, so the number reachable in two ply is 10×10, or 100. After three ply the number is $10 \times 10 \times 10 = 1000$. Past this point writing out the multiplications gets tedious, so we use the abbreviation 10^3, or, after n ply, 10^n. This number, written out, is 1 followed by n zeroes. The number of leaves in the tree is, therefore, about 10^{100}. This is a huge number. If the computer could examine a billion positions (10^9) a second, and it was started when the earth first formed about 5 billion years ago, it would have examined only about 10^{26} positions so far. If it lasts another 50 billion, it will get to 10^{27}. Furthermore, the result of the search would have to be stored somehow. We need to know the first magic move, and then for each possible reply we have to store at least one winning move. That means that at positions where the opponent moves we must represent all the possibilities, while at positions where you move we need represent only one. That cuts the stored tree down to a size of only 10^{50}. A ten-gigabyte hard disk can store 10^{10} bytes. We will need 10^{40} ten-gigabyte disks to store all the positions in the tree. Suppose each hard drive is 1 cm on a side. The volume of the earth is about 10^{12} cubic meters, so if the earth were hollowed out and used to hold hard drives it could hold 10^{18} of them. So we'll need 10^{22} earth-sized planets to hold all the data. I belabor all this because it is easy to misjudge just how large these numbers are. Once you grasp their size you realize why it is that no two chess games are alike; there is always a large amount of uncharted territory that one or both of the players can push the board into.

However, there are only 13 ways a square can be (un)occupied, & 64 squares, so there are less than 13^{64} configurations (less, because most squares might be unoccupied & other reasons). So with say 13 numbers per position, not more than 13^{65} positions.

It is unlikely that a computer could ever be built that could construct more than a small fragment of the game tree for chess. But perhaps a small fragment is all that is necessary. As first suggested by Shannon, a computer could construct all the positions reachable in, say, five ply. At that point it could apply a *position-evaluation function*, which would look at factors like the number of remaining pieces belonging to each player, pawn structure, piece mobility, king safety, and so on in order to come up with an estimate of which player is winning. The numbers wouldn't have to be 1, 0, or −1. We could use numbers like 0.3 to indicate a slight advantage for White, or −0.9 to indicate a big advantage for Black. The computer can then use the same labeling idea, taking the maximum of child labels at a position where it is to move, the minimum where its opponent is to move.

This idea underlies almost all chess-playing programs. There are a lot of possible enhancements. The position-evaluation function can be tuned differently for middle game and end game. There are ways of pruning the tree slightly, so that once a good move has been found the computer can skip examining some of the other branches. One can streamline data structures and optimize move-generation algorithms. Good openings can be prestored so that the computer just plays them, doing no game-tree generation at all, just as a grandmaster does. If a position can be reached by two different sequences of moves, the computer can avoid exploring it twice. You can also buy a faster computer. One way to do that is to use several computers to explore the game tree in parallel. However, by itself this step doesn't get you very far, as the numbers above suggest.

Another possible line of attack is to observe how human experts play chess and try to duplicate their skills. This was what Newell and Simon tried. They were interested in questions such as, what do human players look at when they look at a board? What hypotheses do they form about key features of the current position? They thought that answers to these questions might give them ideas they could incorporate into computer programs. However, even though a few things were learned about human play, none of this knowledge is used in the design of today's programs. Nonetheless, these programs play very well, as was shown dramatically in 1997 when a computer, Deep Blue, constructed and programmed by programmers from IBM, beat Garry Kasparov, who is probably the best

human chessplayer. Deep Blue used a combination of parallelism and an excellent position-evaluation function to beat Kasparov.

When this happened, it made headlines all over the world. Every newspaper and magazine ran opinion pieces. Most of the writers of those pieces argued that the computer's victory should not be a threat to our position as the most intelligent species on the planet. There were several main lines of argument. Some pointed to the fact that the methods used by Deep Blue are very different from the methods used by human players. Some argued that Deep Blue's narrowness meant that it wasn't really intelligent; sure it could win the game, but it couldn't even see the board, and when you come down to it, it didn't care whether it won or not. Some believed that the fact that Deep Blue had to be programmed meant that its programmers were intelligent, not the machine itself. And some pointed out that chess is a finite world, so computers were bound to become fast enough to search enough of its ramifications to win.

I don't think these arguments are as strong as they seem at first. But before I address them, let me point out that the evolution of chess programs has been similar to the evolution of AI systems in general. In the early days, researchers in the field tended to assume that there were general algorithmic principles underlying human thought, so that (a) these could be embodied as all-purpose intelligent algorithms, and (b) these algorithms would resemble humans in their style of thinking. As time has gone by, systems have gotten more and more specialized. A researcher in computer vision, which is defined as the attempt to extract information from visual images, deals with completely different issues from those explored by computer-chess researchers, or researchers working on understanding natural language, scheduling resources for companies, and proving mathematical theorems. Within these specialties there are subspecialties. People working in stereo vision (comparing information from two eyes to estimate depth) do not use the same methods as people studying processing images over time. If there are general principles of reasoning, they don't play a role in most of what gets studied nowadays.

In addition, the principles of computer reasoning don't seem much like the principles of human reasoning, at least not those we are introspectively aware of. A chess master will explore a tree of moves, but not every legal move, only those that make sense. Computers usually look at every move

at a given position. This is a pattern that we see a lot: the computer is good at exhaustively sifting through possibilities, while the person is good at insight and judgment. It is this striking difference that many people adverted to in the reassuring editorials that appeared after Deep Blue beat Kasparov.

I think this reassurance is illusory, for the following reasons:

1. Insight and judgment are not the names of techniques used by the brain; they are words used to praise the brain's output.
2. The method used by the brain to arrive at insightful results might, when implemented on a computer, involve a lot of exhaustive sifting.

These points are frequently misunderstood, and explaining them further will be one of my main goals in this chapter.

People often find the digital computer to be a ridiculous model of the human brain. It executes a single instruction at a time, pulling small chunks of data in from memory to a central processing unit where various tiny little operations are done to them, and pushing the results back out to memory, all of this happening repeatedly and repetitively, billions of times a second. What in the brain or mind looks like that? The answer is: nothing. There is no way that a brain can be thought of as a digital computer.

This fact is, however, completely irrelevant. Our hypothesis is that the brain is using neurons to do computation. The hypothesis implies that what's important about a neuron is not the chemicals it secretes or the electrical potentials it generates, but the content of the information encoded in those physical media. If you could encode the information in another set of physical properties, and still do the same computation, you could replace a neuron with an equivalent computational device in another medium, and it wouldn't make any difference. You could replace *all* the neurons in a brain with one or more digital computers simulating them, and the brain would be just as good. The point is that the digitalness of the computer is a red herring. Animals need things to be computed in the same sense that they need blood to be circulated. There are many different technologies that can carry out these computations. Each technology will approximate the answer slightly differently, and thereby introduce slightly different errors, but as long as the errors are small these discrepancies will be irrelevant. Neurons are wonderful biological

computers, but there is nothing magical about them. Digital computers are just as good in some ways. Above all, they are much more flexible. We can reprogram them overnight. For that reason they are a superb tool for exploration of hypotheses about what it is neural systems are trying to compute.

The question I raised above is whether the methods used by Deep Blue are intrinsically less insightful and more exhaustive than the methods used by Garry Kasparov and other grandmasters. The answer is that nobody knows. There has been some work on the psychology of chess playing (deGroot 1965; Chase and Simon 1973), but it really doesn't tell us what good chess players are doing. If you ask them, they can give you lots of fascinating advice. Here is the "thinking technique" recommended by Silman (1993):

1. Figure out the positive and negative imbalances for *both sides*.
2. Figure out the side of the board you wish to play on. . . .
3. Don't calculate! Instead, dream up various fantasy positions, i.e., the positions you would most like to achieve.
4. Once you find a fantasy position that makes you happy, you must figure out if you can reach it. If you find that your choice was not possible to implement, you must create another dream position that is easier to achieve.
5. Only now do you look at the moves you wish to calculate. . . .

This may superficially appear to be an algorithm. Each step appears to call a subroutine,[5] and there's even a little loop from step 4 back to step 3. However, this is an illusion. Although Silman gives some further elaboration of each step, such as listing the different kinds of imbalance one is to look for in step 1, it remains true that you can't even begin to follow his advice until you are a pretty good chess player. He's basically telling you how to "get organized." The details are up to you. You might ask, how does a good chess player find "imbalances" in a position? Why does a grandmaster find imbalances better than a mediocre player? Unfortunately, no one really knows. The literature cited above seems to indicate that a grandmaster has a huge store of previously seen positions that he or she can access. Given a new position, two or three relevant positions from this collection come to mind. How are they stored and how are they accessed? No one knows. The key point for us, however, is that looking up these positions in memory requires a lot of work by a lot of neurons. Each neuron is performing a tiny piece of the job. Each piece involves

no insight or judgment at all. If we implemented the process on a digital computer, it would look a lot like exhaustive sifting. So the fact that Deep Blue does a tremendous amount of "brute force" computation is, in itself, irrelevant to the question of whether it is intelligent; we know from our own example that insight can be the result of millions of uninsightful computations.

One can still be reassured by one of the arguments I mentioned above, that Deep Blue is too narrow to be truly intelligent. For the time being, we can expect programs to solve a handful of well-defined tasks and be completely oblivious to other demands of a situation. But the other arguments are mirages. The fact that it took intelligence to program the computer says nothing about the mental capacity of the resulting program. The fact that a game or other problem domain is finite does not imply that eventually computers will become fast enough to solve them. Deep Blue embodied many algorithmic advances, as well as being quite fast. There are plenty of "finite" but enormous domains that computers will never conquer merely by running faster. Finally, the argument that Deep Blue's methods are "exhaustive" in a way that people are not is, as I have shown, based on a quick jump to a conclusion that cannot withstand serious examination.

Neural Nets

I have argued that digital computers are useful for studying mental processes because mental processes are ultimately computational, and digital computers are the Swiss Army knife of computation. This account of where they fit into the explanatory landscape of the brain and its activities seems pretty straightforward to me. However, there is a competing story, which goes like this: There was once a field called Good Old Fashioned Artificial Intelligence (GOFAI, to use an acronym coined by John Haugeland 1985). It was based on a model of mind as a symbol-manipulation system. In this framework, the beliefs of the mind are represented as expressions in a formal language not unlike the language of mathematical logic. The central processor of this mind takes groups of expressions and derives new expressions from them. The expressions have a formal *denotational semantics;* each symbol denotes an object or

Omitted:
Mainly McDermott argues that the supposed distinction between "symbolic" and "subsymbolic" representations is meaningless .