2.1 Solving Problems

[00:00] In trying to think about the ways in which machines and human minds or animal minds are both similar and different, a reasonable idea is to take a sample domain for human thinking and to see if we can model it using computers. Now, there are lots of domains, lots of sample problems that we could imagine like how do we see, the vision problem, or how do animals navigate, the navigation problem, or how do we make judgments about certain situations. But a reasonable place to start, and in fact historically, this was a place that cognitive science kind of started, is to look at a fairly constrained narrow set of problems that humans solve. The problems that I have in mind are represented here. I'm thinking of things like puzzle problems. Now, why are those good examples to begin with?

[01:10] First of all, the idea is that we're trying to take fairly narrow, constrained, but still interesting problems that people solve, and to see if we can get some ideas about how people solve those problems by trying to model the process using computer programs. Puzzle problems are a very natural start for this. I'll talk a little bit about why that's true. But the things that I have in mind here, you see a Rubik's cube or the puzzle at the upper right there is the 15-16 puzzle, where the goal is to slide those little tiles. You can only slide a tile into the blank space. So, for example, in the 15-16 puzzle that you see at the upper right there, I could slide the 14 down into the blank space or the 4 across into the blank space, and you can keep making moves like that. Your goal is to try and arrange the 15 tiles so that they start at the upper left, and they're in numerical order: 1-15. The puzzle at the bottom right is the rush hour puzzle, where you have little plastic cars and trucks on a grid, and your job is to move the cars and trucks around so that the red car, which you see buried in the mass there, can drive all the way out of the grid through that little gap at the right. That's the goal of the Rush Hour puzzle. The moves are that you can move trucks and cars back and forth in rows and columns. Of course, there's the famous, by now kind of addictive, Sudoku puzzle.

[02:59] I'm also going to mention another puzzle because I'm going to use it as an example to think with here. So it's the college student's letter home. It's an arithmetic puzzle; 'send more money'. And the idea here is that each unique letter stands for a unique digit somewhere between 0 and 9. If you see the letter more than once, like the letter 'E' for example, that will stand for the same digit throughout. And the result is that this is a legitimate - it's a correct addition problem. So if you add the numbers representing 'send' and 'more', you get the number representing money. So this is one of these standard arithmetic type puzzles. Oh, I should mention one other interesting constraint on this puzzle. The numbers are written in standard fashion meaning that you're not going to write a number with a leading zero. So right off the bat, for example, you know in this puzzle that 'S' and 'M' do not stand for zero. That's one of the things you know. In any event, all these are the kinds of problems that seem like they might be good first examples to think with for modeling human minds with machines.

[04:41] Now, there are strengths and limitations of these kinds of puzzles. First of all, they are characterized by not only conscious, but as I say here, effortful thought. It seems to take effort to solve them. You have to work at solving them. What I mean by conscious is that at

least much of the problem solving process, perhaps not all of it, but much of the problem solving process could be vocalized. You could say what you're trying to do. You're aware of what you're trying to do. That's not always the case as we've seen in thinking about problems. We don't really know how it is that we recover three dimensions from two when we open our eyes and look at objects in the world. We're not really conscious of that. There's much that we do, of course, where we have tremendous talents like being able to tie our shoes, or ride a bicycle, or walk around a room without bumping into things, where it's very hard to make conscious what it is we're doing. In the case of these puzzles though, they have at least a substantial element of conscious thought about them.

[05:53] As I say, they're evolutionary recent, meaning, I guess animals don't do them; animals don't seem to be terribly interested in abstract puzzles. So this is something that human beings do. Again unlike a problem like vision or walking on two legs, which some other animals are able to do, or navigating a space, or things like that. Those are evolutionary older venerable problems, and we might look to animals' solutions to these problems in thinking about how to model them. But puzzles are human problems, and in some ways, that makes them easier for us to think about because we don't have to think about a long evolutionary history to this. Whatever equipment we have, whatever talents we have at solving these puzzles, we've developed pretty recently.

[06:58] They're not especially complicated by things like cultural or affective elements. I say that a little carefully. Naturally, there are some cultural elements to all the puzzles that I've shown. Not all cultures are equally into different puzzles. Moreover, it's not quite accurate to say that there's no affective or no emotional element to these things. You want to solve them; you're curious about solving them. So you do have some interest in solving this. Otherwise, no one would attempt them at all; if there was absolutely no affective component. On the other hand, it is fair to say that these are reasonably abstract, and they're reasonably not culture dependent. That is, you get the feeling that for many of these puzzles, you could explain them to people from a wide range of cultures, even people who have never seen the puzzle before, and they would understand what the puzzle means. So there's not a strong dimension of cultural or emotional baggage to these things, which again, you might hope might make them easier to model. They're fairly self-contained, as I say, which is that they don't require a great deal of background knowledge. They require some. You have to kind of understand the ideas of arithmetic, or you have to understand the basic physics of sliding tiles, or perhaps what it means to handle a Rubik's Cube. But nonetheless, these are problems that don't require scads of outside factual knowledge about the world. So all of these things make them pretty good candidates for an attempt to model the human problem-solving process by a machine; how do we solve these problems, how could you get a machine to solve these problems.

[09:11] Those same advantages are also in a sense disadvantages. These problems, because they are abstract and because they're fairly narrow, don't partake of some of the qualities that we might also be interested in: problems that animals also solve or problems where we find it hard to vocalize or make conscious our solution. So those are especially interesting kinds of problems, and we're ruling them off limits for the time being. So we'll look at these.

[09:48] A natural way that when people started to model the human problem-solving process for these puzzles - a natural way was to think of the problem in terms of what computer scientists call a problem space. And what they mean by a problem space is a graph. I'm not talking about a graph like a graph of a function, rather a graph that consists of lots of vertices, which are always represented by little circles. So vertices are represented by circles, and edges are represented in this case by arrows. The idea is the vertices of this graph are meant to represent particular states of the puzzle. So, for instance, in the 15-16 puzzle, a state of the puzzle could be an arrangement of the 15 numbers and the blank space. So some arrangement of the 15 numbers and blank space within that four by four grid represent a state of the puzzle, and each state, each unique state, is designated by a vertex in the graph. Edges between vertices represent what we naturally think of as moves in the puzzle. So if we were to move in this 15-16 puzzle that I show here, if we were to move the 14 tile one space down, that would change our state from one particular state in the problem space to a nearby state in the problem space.

[11:35] This is a very general technique for representing puzzle type problems. It's often used to try to represent other kinds of problems, but it has a natural affinity, it has a certain natural quality for representing puzzle type problems. So you could imagine if you think about this how you would represent a puzzle like the 15-16 puzzle, the Rubik's Cube puzzle, the Rush-Hour puzzle. It takes a little more ingenuity perhaps to represent the arithmetic puzzle this way, but once you've got the hang of translating problems into problem spaces, there are some natural candidates for those kinds of things too. So a problem space is a good general purpose abstraction for representing a lot of these problems. Once you've decided to think about puzzles in terms of problem spaces, other insights emerge from that metaphor right away.

[12:46] First of all, we could talk about how hard a problem is. What does it mean? We've been speaking, and we've thought about this question of what makes a problem hard. Well, one of the questions about a puzzle is: is it a hard puzzle or an easy puzzle? One way to reframe that question, not the only way, it's not the only factor, but one way to reframe that question is to think about how big is the problem space and by that I mean, how many nodes, how many vertices are there? How many distinct states are there in the problem? That's at least a rough measure of how difficult the problem is. It fits our intuition because we kind of sense that if you take a similar puzzle, but have smaller or larger versions of it, the larger versions intuitively are going to be more difficult than the smaller versions. You could imagine cases where that might not be true, but still that's our intuition, and it's a good intuition. So the 15-16 puzzle is harder than the 8-9 puzzle, where you're moving only eight numbers around in a three-by-three grid. Presumably if we made a 5-by-5 grid, the 24-25 puzzle, that would be harder to solve than the 15-16 puzzle. And similarly with Rubik's Cube, the market versions, they actually market a two by two by two version and three by three by three, and four by four by four and so forth. So each of those puzzles, they get progressively harder as the cubes get larger. Why is that? Well, we could actually put a numeric measure to that by saying, how many distinct states are there in each of these puzzles? Again, our intuition is that the larger the graph, the more the possible states, then the harder the puzzle.

[14:50] Let's take a couple of examples. For instance, in the 15-16 puzzle, how many

distinct states are there? That'll tell us something about how difficult the puzzle is. Well, I'm making a rough estimate here. In fact, it's not a perfect estimate, but a reasonable estimate as to how big the problem space is for the 15-16 puzzle, would be to say that in the upper left corner there, we could have any of 16 distinct things: 15 numbers or a blank. So there are 16 choices for what's in the upper left corner; the tile just to the right of the upper side, to the upper left corner, which in this case, is occupied by five. Once we've chosen one of the 16 possibilities for the upper left corner, there are now 15 remaining possibilities for that tile. Once we've chosen that tile, there are 14 remaining choices for the space, which is in this photo occupied by 12. So it doesn't take a whole lot beyond that to see that the number of possible states in the 15-16 puzzle is 16 choices times 15 choices times 14 times 13 times 12. Finally, when you get to the last space, you have only one choice ,and that's written as 16 factorial. So there's 16 factorial as a first estimate;16 factorial distinct states in the 15-16 puzzle. In fact, that's not quite right, but I'm not going to go into the details. This is a pretty good estimate. 16 factorial is a big number. So a lot of distinct states in this puzzle.

[16:48] How many states are there in the 8-9 puzzle? Well, by the same reasoning, there are nine factorial states in the 8-9 puzzle. That's not a small number, but it's considerably smaller than 16 factorial. If we wanted to do a 24-25 puzzle, we would need a graph of size 25 factorial, which is a very big number. So our intuition is telling us that the larger the particular version of a puzzle, the harder it gets. That intuition is matched by what we see in the size of the problem space for each of these things. Nine factorial is not a small number, but it's considerably smaller than 16 factorial, which itself is considerably smaller than 25 factorial. The counting states for the Rubik's Cube takes considerably more work, but again, it turns out to be the case that the three by three by three cube has quite a few distinct states, has a very large problem space, but it's smaller than the four by four by four cube, which itself is a good deal smaller than the five by five by five cube. So again, the sizes of the problem space gets bigger.

[18:13] How about this Send More Money puzzle? How big is its problem space? Well, we can make a pretty good estimate of that too. We'll say that each of these, as it turns out, there are eight distinct letters in this puzzle, S-E-N-D M-O-R and Y. So there are eight distinct letters in this puzzle. By the same kind of reasoning that we used with the 8-9 puzzle, we could say well, we can assign one of 10 digits to ‘S’: anything from 0-9. Once we've made that assignment, we only have nine digits left that we could assign to ‘E’. Once we've made that assignment, we only have eight digits left that we could assign to ‘N’. So a first reasonable estimate is 10 - actually, it's not quite 10 factorial because we only have eight letters - so we end up going down to 10 times 9 times 8 times 7 times 6 times 5 times 4 times 3. Another way of writing that is 10 factorial over 2. So we have 10 factorial over 2 distinct assignments of numbers to letters here. We may in fact want to play with our representation of the problem space, but that's not a bad first estimate anyway at the size of the problem space. And it too is pretty big.

[19:52] The fact that we have at least a beginning metric, a beginning way of thinking about the difficulty of a problem, is helpful. It isn't the whole story. What makes a problem difficult is not just the size of the problem space, although, it's natural that that would be a contributing factor. It may include other things like, well, let's think about this. Once we have a

graph representation of a puzzle, we can start to think about some other things about this representation. When we receive a puzzle - like an unsolved Rubik's cube or the Send More Money puzzle or The Rush-hour puzzle with all the cars scrambled around the grid - when we get that puzzle, we are being given a starting state in this graph. Think of it as one of the distinct states of the graph, and that's where we're going to start. So you've given us a puzzle, and it happens to correspond to this vertex in the graph.

[21:01] Now, our job is to move about the graph so that we can take specific moves from one state to another until we get to a goal state. A goal state is, there may be more than one in the graph or there may be just one, but a goal state is a vertex which represents the solved puzzle. So in the case of say the 15-16 puzzle, there would be one state in the graph that represents the arrangement of tiles from 1-15 followed by a blank. That's the goal state. Our job is to find a path in the graph starting at our start state and moving over edges because those are legal moves. So we're moving over edges in the graph to get to the goal state. Computer scientists refer to this as a search problem. We're searching a graph starting at a particular vertex and looking for a particular target vertex, and we're doing a search of the graph to look around and see if we can get to that goal or target vertex. This is a search problem that itself gets us to think about certain algorithms or certain computational ideas that we could employ to do a graph search. And it's the thing that computers are really good at doing.

[22:37] So we've taken the situation this far. We've said, okay, there are certain kinds of puzzles, and we can represent the puzzle as a graph, and we can represent our attempt to solve the puzzle as searching the graph. Now, even in saying that, we're already making some big assumptions: is this really what people do? Is that really a good model of how people think about these puzzles? Even again, now we might even think, well we don't do this consciously, but maybe in a sense if we're vocalizing our solution, you could make some matches between. You could make some correspondence between the things that we say we're thinking about and the tasks involved in searching a graph. We could start to explore that. Certainly, we could start to explore writing computer programs to solve these puzzles, but we're not guaranteed when we do that that the computer programs are particularly representative of what humans are actually doing when we solve the puzzles. It's not a bad start though. So we'll continue with our discussion of these kinds of problem solving models in the next session.

2.2 Problems for Minds and Machines

[00:00] We've been talking about using computers to solve problems, and for the time being, the problem that we have in mind is mostly the standard puzzles that we saw in a previous lecture. Which is to say, think about things like the 15-16 sliding tile puzzle, as an example, or Rubik's Cube or various other kinds of puzzles. At least to a first approximation, those are the kind of problems that both we, as humans, are able to solve (sometimes with a great deal of difficulty) and that computers are able to solve because they fit well into the programming style of problem solving. When computers are given the task of solving puzzles like this, as we described, we think of the puzzle in terms of a space or a graph that the computer is going to search. We're going to write a computer program to systematically search a graph which represents the available states of the puzzle. Our initial situation, the puzzle that we're given, is a start state in the graph. Then we're going to move from node-to-node in the graph. Looking, as we go, where each move from node-to-node represents a move in the puzzle like a twist of a Rubik's cube or a slide of a tile. But we're going to move from node-to-node in the graph, searching in some systematic way for the goal state of the puzzle.

[01:59] Now, there are many many techniques in computer programming for graph search. There are a few basic techniques, and I'm going to describe a couple of those today. But one can spend a tremendous amount of time studying the ins and outs of searching graphs via computer. For our purposes, we're only going to describe some major ideas. But of course, there's much more detail that you could go into if you're interested.

[02:35] To begin with, we're going to think of searching our puzzle graph or searching our problem graph in terms of what are called weak methods. Now, they're called weak methods not because they're useless or not helpful, but because they don't assume any real knowledge of the problem beyond the problem space description. That is, once you've framed the problem in terms of a graph, you can use weak methods to try and find a solution, a goal in the problem. Let me be a little specific. Imagine we're doing something like solving the 15-16 puzzle. We represent our initial state of the 15-16 puzzle. I'll just do this one time so you can see what a state is supposed to mean. So imagine putting in numbers in each of these little squares and there's the empty space. So our initial state of the puzzle is represented by this node. Everywhere that we can get to in one move, that is one sliding tile, is written as a new node. In this case, there are three next moves because we could move this little tile in here or this one up here down or this one below upwards. So there are three next moves that we could go to. From each of those, we could go to still other states of the puzzle. I don't know if three is the right number of arrows to be drawing here; maybe I could draw two in one case or four in another. But in any event, think of this as the start state of the puzzle. These are all the states you could get to within one legal move. The next layer is all the states you could get to within two legal moves. The next layer is all the states you can get to using three legal moves and so forth.

[04:57] Now, I'm sweeping some stuff under the rug here. For one thing, it's clear that for many puzzles this graph format will be enormous. We'll deal with that. I mean, that's part of the difficulty of solving puzzles in general. I've also structured this graph in a form that computer

scientists call a tree. Meaning, in this case, there's more definition to give, but from our standpoint what it means is there is a designated state called the root, which is where we started the puzzle. Then there are branches from the root going to all the moves, all the places, the states of the puzzle we can get to within one move. And then from these there are branches to all the states we can get to via two moves and so forth. So the structure of this graph will look like a spreading tree as it moves downwards, as we go to four and five and six moves and so forth.

[06:01] So with this in mind, searching a graph becomes searching a tree. And the standard weak methods, the two that are always introduced first in this context, are called depth-first search and breadth-first search. Depth-first search, we could go into much greater detail, but I'll just give you an informal description of depth-first search. Depth-first search means that you follow a certain path of moves. Imagine that you make one move and then you make a second move and a third move and a fourth move and so forth. You're following down, in depth, a particular path to try to solve the puzzle. If you run out of options or you reach a dead end, then you move back up to the last place that you were able to make another choice and try that. So qualitatively, the feel of depth-first search is that it looks down a tree for a long way. If it runs into a dead end, it backs up and tries some other branches. If those don't work, it backs up some more and tries some other branches. So you get this pattern of working up and down the tree until you find, hopefully, a solution to your problem.

[07:29] Breadth-first search, by contrast, is a systematic way of searching a tree in which you first look to see if this is the goal state. If not, you look at all the states you can get to within one move, and see if they contain a goal state. If not, you look at all the states you can get to in two moves, see if they contain a goal state and so forth. So breadth-first search is almost like reading the tree across as a text. You're seeing if the goal state is here, then in the next layer, the next layer, the next layer. As I say, these are methods that don't depend on knowing much about the puzzle other than the fact that you've set it up as a tree. They're useful methods, and they're often the foundations of more complex methods. But they're also highly flawed in the sense that there are different reasons why either one can prove to be a huge problem.

[08:39] In the case of breadth-first search, we've noted that problems have this nasty habit of expanding, sometimes exponentially. So if you look at a tree like this, you might see 1 node in the top layer, 3 nodes in the second, 9 in the next, 27 in the next, 81 in the next. If your solution takes something like 25 moves, you will be looking at an extraordinary number, an unwieldy number of problem states in this tree. Depth-first search runs into other problems in that sometimes a problem space is infinite. You may find yourself, for example, looking down a pathway of potential moves that never ends. The 15-16 puzzle is not infinite and so you can set up depth-first search so that it will eventually find a solution. But it still may take an extraordinarily long time.

[09:48] As I say, weak methods are called that because they assume very little knowledge. In that respect, they're not very human-like. To some extent, we do things when we're solving puzzles that can look a little bit like depth-first or breadth-first search, but in their extreme versions, we don't really think that way. You may reflect on the way in which you solve some puzzles, and see if there's some elements of your own problem-solving that feel either a

little bit like breadth-first or depth-first search. More human-like, that is, as we're moving from machine solving puzzles to minds solving puzzles.

[10:34] There are machine techniques that feel a little closer to the human style. They're still not perfect in every respect, but they feel a little closer. One is called hill climbing, and the idea is you think of the problem space as laid out on a terrain in which the closer you are to a solution, the higher the terrain is. So in effect what you're trying to do is wander around the terrain on this graph looking for the highest peak. Think of the solution to the problem as a mountaintop: the towers above the rest of the graph. So you're moving around on this terrain looking for the highest peak of the graph.

[11:26] Now, hill climbing is not a bad idea if in moving around the graph you're always trying to move upward. So it's as though you were walking on a terrain, that geographical terrain, and always seeking to move upward as you go. In certain situations that can work very well. In other situations, you may run into problems where, for example, you may be looking for the highest peak in your graph, and you may run into a situation where there are two standard kinds of problems. These aren't the only ones. But one problem is you move to the top of a little hill, and it is not the ultimate solution to the problem, but every move that you can make from that little hill seems to be a move downward. That can be an issue because you're stuck. It's what could be called a local maximum, meaning you've moved to a place where every move from that particular situation seems to make your situation worse. However, this little hill is not the ultimate solution to your problem. So it's hard to know where to go from here.

[12:47] Another kind of problem with hill climbing is that you may be on extraordinarily flat terrain, a plateau, in which there's no clear reason to move in one direction or another. Those are the kinds of problems that come up with hill climbing. But in some respects, it feels a little bit closer to a humankind of reasoning. We're looking for a peak, and in order to find that peak, we'll see if we can find it by just moving upward wherever possible.

[13:17] Best-first search is an idea where you're searching a tree, but every time you consider the next move, you look to see which move appears to be the best. Now, notice that this idea, just like hill climbing, assumes that you have some metric for the goodness of a state. It assumes that when you're considering these three different moves you have a reason to prefer one to the other two. That represents knowledge about the problem. So the only time you could use an idea like best-first search is when you feel that you can rank order moves in the puzzle in a sensible way.

[14:08] I'm going to describe yet another technique called means-ends analysis. This has an interesting history in cognitive science because it was one of the early techniques for problem-solving studied both in human beings and in writing computer programs to solve puzzles. It too does not work in every situation. In fact, it is fairly constrained in the kinds of situations that it applies to. But it's readily representable in a computer program, and it applies to certain kinds of puzzles or problems that we sometimes run into. I'm not going to give you a coded version of means-ends analysis, but I'll try and give you the flavor and pros. This was investigated early on, by the way by Alan Newell and Herb Simon, in the early years of artificial intelligence. So the basic idea behind means-ends analysis is you are in a particular state of a puzzle or problem. You want to get to a goal state. You first look at what appears to be the

biggest difference between you, your current state, and the goal state. If you can do something that will reduce that difference right off the bat, you do it. On the other hand, if you can't find anything to do to reduce that difference, you see if there's something else that you can do that will reduce the difference between your current state and a position in which you can attack the biggest difference. So there's a recursive element to this means-ends analysis. Let me say this again. The idea is you look at your current state, you look at the goal, you say what's the biggest difference between me and the goal. If I can attack that right now, let me do it. If I can't, let me pose a new problem, which is getting to the point where I can attack it, and try and solve that problem by means-ends analysis. It's an inherently recursive idea, and in coded form it's actually a beautifully elegant program.

[16:34] It's close to certain kinds of puzzles such as the rush hour puzzle, where you're trying to maneuver that red car out of the grid, and the only way you could do that is by moving cars that are in front of the red car, so that you can eventually get it out of the grid. So when you try to solve the rush hour puzzle, you find yourself saying things like if I could move the car directly out I would do that. But I can't because there are a couple of cars in the way. So my new problem is getting those cars out of the way. Let's see if I can get those cars out of the way. If I can't, it's presumably because they're being blocked by still other cars. So let's see if I can get those out of the way. That's a technique for solving rush hour, at least for some puzzles, and it has the flavor of means-ends analysis.

[17:37] Now, for computer scientists, when we try to write algorithms for searching a problem space graph, there are standard issues that always come up. They're not exactly the kinds of issues that always come up when we think about us as human beings trying to solve problems, but they resonate a little bit at least with some of the human issues. One problem is we'd like to have a search technique that is guaranteed to find a solution if there is one. So taking for example our 15-16 puzzle, if there is a solution, then because it's a finite graph, if we implement either depth-first or breadth-first search in a reasonable way, and again, I'm sweeping a little bit under the rug, but if we do this with reasonable cleverness, either of those two weak methods is guaranteed to find a solution (assuming there is one). Some search algorithms may not be complete in that sense. They may be interesting for other reasons, and they may be promising, but they may not be guaranteed to find a solution. Obviously, if we had an algorithm that was guaranteed to find a solution, that's of some value to us.

[19:08] A second issue is how long is this going to take to run? If we are going to find a solution, is it going to take a minute, a year, a century? That too is an issue about creating the search algorithms. We may have a vast problem space, and searching that problem space for a solution, we may be guaranteed to find a solution, but it may take years. In which case, we have other problems on our hands. We might want to, for example, find an approximate solution or see if we can find a better search algorithm that will take less time.

[19:50] Similarly, in computer science, along with time, there are space considerations. So memory is a good deal cheaper than it used to be, but it's still finite in any computer. And so we don't want to write a program that will take an infinite or an astronomical amount of memory. We would like to write a search program that will only use up a relatively small or measurable amount of memory as the algorithm runs. That too can sometimes be a problem.

[20:23] Finally, there's an interesting issue that deserves mention here. Our algorithm may find a solution, but it may not find the best solution. Take for instance solving Rubik's cube. You may be given a scrambled Rubik's cube, and it may turn out that the quickest solution to this scrambled Rubik's cube takes four moves. You could unscramble it in four moves. Your computer program, your search algorithm, finds a solution to the Rubik's cube, but it takes 20 moves. Well, maybe you don't care. I mean maybe all you wanted was a path to solve the Rubik's cube. But this could conceivably be a meaningful issue to you where not only do you want the program to find a solution, you want it to find by some measure the best solution.

[21:24] These are all things that computer programmers care about when we write search algorithms. And they have resonance with certain kinds of human problem-solving activities, but the connection is not always straightforward.

[21:44] There are other kinds of issues about solving problems that are worth mentioning. I don't want to finish the discussion of solving problems quite yet, but in this particular session or lecture, I would like to just talk about two additional issues about thinking of problems in this search space formalism. One is that the way that human beings solve problems or puzzles seems to involve more than just search or more than just immediate search. It seems to involve other kinds of things like looking for patterns in problems that we're solving.

[22:27] Let me give you an example of what I mean by that. This is a famous experiment that was done by Chase and Simon in the early 1970s. Here was the nature of the experiment. It was actually a memory experiment; they were looking to ask a question about the memories of chess experts versus chess novices. It has sometimes been suggested that one reason to study chess is that it'll improve your mental functioning: perhaps even improve your memory. So the Chase and Simon experiment was designed to test that question. Do chess experts have better memories than chess novices? The way that they did this test was they showed both chess experts and chess novices sample chessboards with pieces on them, like the one in this slide. So they showed both the expert and the novice a chessboard like this one with pieces upon it, and they only showed it to them for a limited amount of time, a matter of seconds I believe. Then they asked both the chess novice and the chess expert to see how well they could recreate the board from memory. Now, in principle, if a chess expert has a far better memory than a chess novice, he or she will be able to recreate the board much more accurately. Here's what they found; it was quite interesting. What they found is that, if the chessboard had an arrangement of pieces that was taken from an actual game, from real gameplay, then indeed the chess expert was much much better at recreating the board than the novice. If, however, the chessboard just had a random scattering of pieces in various places, the kind of situation that you would never actually see in a real game, then the chess expert and novice were equally bad at recreating the chessboard. They were comparable.

[24:46] The interpretation of this experiment is that the chess master has better memory of real chess boards because when he or she is playing a game, they're not just interpreting the game position as a random collection of pieces, they're interpreting the board in terms of larger concepts of chess play. Maybe a pawn arrangement, or whether a king has been castled, or who's getting control of the center, or which pieces are threatened. Those are larger scale notions that inform the way that a chess master looks at a board; they're not available to a

chess novice. But when a chess master has to recreate a board, he or she is thinking in terms of these larger scale patterns. That too would be applicable to other kinds of problems. For example, this has been studied in problem solving in physics, where physics experts are able to look at a problem and grasp certain concepts in the problem: larger patterns that are unavailable to the novice. So that's one issue. We would like, as we're thinking about writing machine models of problem solving, we would like to think of ways of grasping higher level patterns about certain problems.

[26:20] Another issue, and this is the last one I'll talk about, is that sometimes it's tricky to determine how to think of a problem space. This is a fantastic little puzzle that way. Because it's a fantastic puzzle, I'm going to describe it to you at enough length that you can think about it. Here's the idea, you take a standard chessboard, eight by eight chessboard, and you're given the problem of tiling that chessboard with 32 dominoes. That isn't very hard. What you could do, for instance, is take the chessboard and then place four dominoes in each row or four dominoes in each column. When I say tile the board with dominoes, what I mean is place the 32 dominoes on the 64 squares in such a way that the dominoes are covering the entire board and they don't overlap. There are all kinds of ways of doing this, and you can see that there are all kinds of ways of doing it. That, as it turns out, is not a hard problem.

[27:27] Now, we set a slightly variant problem. This is called the mutilated chessboard problem. We take away the opposite corner squares of the chessboard, leaving a board with only 62 squares on it. So as you see in the slide here, we've taken the upper left and the bottom left square out of the chessboard. Now there are 62 squares left in the chessboard. The question is can you tile this mutilated chessboard with 31 dominoes? So can you take 31 dominoes and lay them out on this chessboard in such a way that they'll cover the entire thing, and they won't overlap? Now, this is interesting - when I pose this question to computer scientists what they often do is they imagine writing a program that will systematically search all the possible ways of laying down dominoes on a 62-sized mutilated chessboard, and they'll try to imagine ways of doing that systematically and trying every possibility. You could do that, I mean, such a program could be written. However, there's an immediate way of seeing that the answer is no - it cannot be done. The reason for that answer, when you think about it, is this: every domino that you place on this chessboard has to cover one black and one white square. Whether you place it vertically or horizontally, it's going to be covering one black and one white square. Therefore, if we put down 31 dominoes, they should cover 31 black squares and 31 white squares. In this mutilated chessboard, however, we've subtracted two white squares. So the board, as left, has 32 black squares and 30 white squares. Once we place 30 dominoes, therefore, we will have covered all the white squares and there will still be two black squares leftover. It can't be done. This is what could be called a parity problem because every domino has to reduce the size of the available puzzle by one black square and one white square. And so we can't do that.

[30:10] In order to see that though, we have to think of the problem space in different terms. The sort of automatic brute force way of thinking of the problem space would lead us to this vast search algorithm. A different way of looking at the problem space would lead us to an immediate solution. So this is an interesting instance of the limitations of the sort of unthinking

problem space approach. In order to look at a problem in a creative way, you may have to think about the problem space in ways that are non-obvious.

2.3 Hard Problems for Computers

[00:00] For many problems, computers are extremely good at solving them. They're good models of how to go about solving problems. In some cases, they solve certain types of puzzle problems more effectively than people do. Because they make use of a problem space representation, they're systematic and search, and they can often solve certain kinds of, what I might call, stereotyped puzzle problems much more quickly. In some cases, more effectively, more quickly, and finding optimal solutions in ways that people find difficult to do. But not all problems have this flavor of being able to be translated into a problem space. That is to say, some problems are difficult in interesting ways both for people and for machines. It's interesting to take a look at these kinds of problems. Both because they compel us to ask questions about our own techniques for solving problems, and also they compel us to look at computer programs and think about new and creative ways of implementing problem-solving strategies in computers. So rather than stick with the tried and true problems space approach, let's look at some examples that in one fashion or another challenge that approach.

[01:43] Here's an example. We've talked about if you've watched some of the previous discussions, we've talked about techniques by which computers can solve certain kinds of puzzle problems like Rubik's Cube, or the famous 15-16 puzzle, or the Rush Hour puzzle, other kinds of things like that. This is another sort of puzzle that you can get at a puzzle shop. These are topological puzzles. They come in various forms: rings, or wires, or things like that. In general, the structure of these puzzles is that you have to take them, they're physical objects, and then you have to manipulate them in some way to separate pieces or reconnect pieces. They're often quite difficult. It's an interesting question to ask how you would get a computer to go about solving a problem like this. It doesn't seem to fit terribly naturally into the problem space formalism. That is, it's often not clear when you pick up one of these puzzles, what constitutes the distinct states of the puzzle and what constitutes the legal moves. In a case like Rubik's Cube or the 15-16 puzzle, it's very clear what the distinct states of the puzzle are. They are the arrangements of the cubes; they are the arrangements of the tiles in the box. You can pretty much say what it takes to make a distinct move. Slide a tile in the 15-16 puzzle or rotate one plane 90 degrees counterclockwise or clockwise in the case of Rubik's Cube. All well and good. With a puzzle like this though, it's a little tricky to say what are the distinct states. If I take one of these objects and move one of the rings just, by say, half a degree in space, just rotated around a vertical axis, is that a new state of the puzzle? Have I executed a legal move?

[03:57] It really seems that when you look at puzzles like this the difficulty, the essential difficulty, is finding out what the states are. What constitutes the meaningful states of the puzzle, the positions of the rings, and usually from that point the moves are not so hard if you know what the states are. But knowing the states is the difficult part of this. So this is the kind of puzzle that challenges our usual notion. Thinking about how to get a computer to solve a puzzle like this, some people are exceptionally good at it, but thinking about how to get a computer to solve a puzzle like this would be a really interesting challenge.

[04:40] There are other kinds of problems that are in a sense kind of standard problems, they show up in math books, or they show up in mathematics exams, or they show up in interviews for tech companies. Often these are problems that require a certain amount of creativity and flexibility to solve. If you've ever taken one of these Math Olympiads, or Putnam exams, or things like that, then you know that problems of this sort can be quite challenging. Assuming you can solve them at all, they may take you days to solve. They may take a long time to think about. Again, it's hard to see how they could be easily translated into the problem space formalism.

[05:44] So here's an example I got from a wonderful book called *How to Solve It: Modern Heuristics*. In a moment we're going to see another book called *How to Solve It*, but this is a newer book with a similar title. I'll just read out the puzzle:

“Mr. Smith and his wife invited four other couples for a party. When everyone arrived, some of the people in the room shook hands with some of the others. Of course, nobody shook hands with their spouse, and nobody shook hands with the same person twice.

“After that, Mr. Smith asked everyone how many times they shook someone's hand. He received different answers from everybody.”

The question is: “How many times did Mrs. Smith shake someone's hand?”

[06:32] So it's a well done puzzle because in this sense, it's a little bit like Rubik's Cube, and then it's well constructed. The way that it's written is intrinsically curiosity-provoking. Say you're reading this puzzle, and then you get to the final question and who even mentions Mrs. Smith up until this point? All we know is that she's married to Mr. Smith, but there doesn't seem on the surface to be any information about her. But this is a well-formed problem, and you can solve it. I don't think I'm going to solve it right here on screen; I will let you ponder this.

[07:17] But the kinds of techniques that you would use to solve a puzzle like this were described in a wonderful book that came out in the 1940s by the mathematician George Polya. It was called *How To Solve It*. It's still in print. It's been reissued many many times. It's a classic, and I couldn't recommend a book more highly. There have been follow-ons to this book along similar lines, that is, books that also talk about problem-solving in this way. They often have new contributions to make, but this is a great book to start with if you're interested in the art and craft of solving problems of the kind that I just showed. Now, Polya, I don't know that he invented, but he certainly popularized the word heuristic; meaning a rule of thumb, a strategy to try in solving a puzzle or a problem. Not guaranteed to solve the puzzle, but often helpful. Often the kind thing that you should have in your bag of tricks, that you should have in your repertoire. Some of the kinds of heuristics that he mentions are things like, if you see a problem like the one about Mrs. Smith, do you know a related problem. Maybe if you know a related problem or one that seems similar, and you've already solved that, maybe that solution will help you with this one. Or did you use all the data in the problem? Did you make use of all the things that were presented in the problem?

[09:03] That's a feature in a way of artificial problems like the Mr. and Mrs. Smith problem. In the real world, sometimes we're faced with problems where we may get extraneous data. We may get data that we don't have to use. But in these well-formed math exam type problems, they play fair, and if they give you data, then presumably they're giving it to you because it's going to be needed to solve the problem. Once you solve it, can you use the answer for a related problem?

[09:36] In the case of the Mr. and Mrs. Smith problem, the advice of drawing a figure I think is especially helpful. If you draw a figure, just doodle out a figure that you think might help you solve this problem, and it may do very well.

[09:53] Then Polya mentions some other typical strategies of proofs like reductio ad absurdum or induction or decomposing the problem into subproblems. Dimensional analysis is a very helpful technique for solving physics problems where you can just reason about the dimensions of the quantities involved. But in any event, these are good basic techniques for finding proofs or solving problems and so forth. Some of these, I think, are extremely useful for at least, as I say, at least the “draw a figure” piece of advice, is extremely useful for thinking about the Mr. and Mrs. Smith problem.

[10:42] There are other problems that I think are interesting to think about in this light. Again, they lead to new ideas about how you would program a computer to solve a problem like this. So here's, maybe you could call it a physics problem. It's a problem about objects in the real world. So imagine, you could actually try and do the experiment if you want. But before you do the experiment, think of this as a problem that you can solve in your head, a thought experiment. So imagine two quarters side-by-side as shown in the picture here. Now you take the quarter at the right, the one which has the head side up, and without slipping, carefully you roll it around. So I could reach, roll this quarter around, the quarter with the tail side up. The tail side up quarter remains stationary the whole time. So once you've rolled this head side of the quarter to the opposite, to the left side of the picture, which way will George Washington be facing? Now, that's an interesting question. The way that we tend to solve questions like this is through, well at least the way that I would try to solve a problem like this, is through a mental imagery, a dynamic mental imagery. Mental imagery that executes a sort of animation. That's a very different kind of programming than the programming that would be involved in trying to represent this as a problem space with moves. To solve a problem like this, you would want certainly to write a program to solve a problem like this. You would want to get a computer to be able to take sample animations or sample pictures, and animate them in realistic ways to get an answer. That would be, and that represents a certain kind of research in computer programming, but it represents a really interesting new way of linking human problem-solving with machine problem-solving. Again, should I give you the answer to this? No, I'm not going to give you the answer to this. What I would say is you should try and think about this problem in your mind and then see if you get an answer by trying the experiment on the desk with two quarters.

[13:20] Here's another, this is a more straightforward physics problem. So this is a kind of physics problem that you might get in an introductory physics class. So imagine you've got one of these, physicists apparently love these frictionless tracks and elastic collisions, meaning that energy is conserved and momentum is conserved, and so forth. So imagine that you have a frictionless track, a ping-pong ball rolling to the left, my left, at one centimeter per second, and a bowling ball is rolling to the right at once centimeter per second, and they're going to collide and then after the collision, they will have new velocities. So a totally elastic collision. Now, I don't even know if these are realistic numbers for ping-pong balls and bowling balls, I think a ping-pong ball is well over a gram. But let's just say for the sake of argument that the ping-pong ball is one gram in mass, and the bowling ball is 10,000 grams in mass. So now, you have all the information that you need to solve for the final velocity of both the ping-pong ball and the bowling ball if you use the fact that energy is conserved and momentum is conserved - overall momentum is conserved.

[14:47] Well, you could do it that way. Would take a fair amount of algebra. There's a much more straightforward, a little bit approximate, but much more straightforward way of thinking about the problem. I won't go through the numbers for it. But think about it in these terms. You have the ping-pong ball. Let's see if I can draw this out. You have the ping-pong ball going off like this and the bowling ball going off like this, and this is one gram, and this is 10,000 grams. They're about to collide. Now, think of it from the physical standpoint. From the viewpoint of the ping-pong ball, the bowling ball is much, much bigger. If you were kind of arriving on the ping-pong ball, it would feel to you as though you were smacking into a wall. So if the ping-pong ball were just smacking into a wall, a stationary wall, then we know that - well, in this case, the wall is not going to change its velocity - but the ping-pong ball with energy being conserved would move off at one centimeter per second in the opposite direction after the collision. In other words, if you think of what's going on as a ping-pong ball at one centimeter per second, colliding with a massive wall, it will just bounce off and go at the same velocity. Well, this situation is kind of the same if you think of it from the ping-pong ball's point of view.

[16:41] So one way to reason about this is to reason from the frame of reference of the bowling ball. Imagine that you are in a frame of reference in which the bowling ball is stationary. So you're in a frame of reference that is moving one centimeter per second to the right, but in that frame of reference, the bowling ball is stationary and the ping-pong ball is headed for you at a velocity of two centimeters per second. After the collision, the bowling ball will still be stationary essentially. Yes, it will have a little velocity, but a teeny one. The ping-pong ball will be going off at two centimeters per second to the left in this new frame of reference. So using that kind of thought, you can solve a physics problem like this. But the step of being able to see that you can make that approximation is the interesting thing here. That is to say, what's interesting is not the numerical solution. What's interesting is looking at a situation like this and seeing that you can get a quick, if not perfect answer, by making a little change in the way you represent the problem. That's an interesting kind of human problem-solving, and again, trying to get a computer to be able to do this kind of problem solving would be a creative act. This would take new kinds of directions in computer problem-solving.

[18:19] Finally, you've already kind of seen this slide by accident a little bit, but now I'm

going to unveil it. So this is a wonderful problem from William Poundstone's book, *Labyrinth of Reason*. There are a number of problems like this; where you see problems like this. And this is actually, in a sense, it's a well-formed problem. That is, there is an answer that you can give to this. So let me just read this out:

“A man gets an unsigned letter telling him to go to the local graveyard at midnight. He does not generally pay attention to such things, but complies out of curiosity. It is a deathly still night, lighted by a thin crescent moon. The man stations himself in front of his family's ancestral crypt. The man is about to leave when he hears scraping footsteps. He yells out, but no one answers. The next morning, the caretaker finds the man dead in front of the crypt, a hideous grin on his face.”

The question is: “Did the man vote for Teddy Roosevelt in the 1904 US presidential election?”

[19:28] Now, this is a weird problem, but there are a number of problems of this form. They're fun. You have the information to give a cogent answer to this problem. Unlike the kinds of problems that Polya talks about, there is a fair amount of extraneous data here. There's stuff that is there to not exactly to lead you astray, but stuff within the story that is not relevant to the solution of the problem. There's a little bit that is relevant to the solution of the problem. Even then, solving the problem requires a certain amount of background knowledge of a kind that not everybody has; in other words, to answer this problem in the way that is intended for these kind of problems. By the way, I'm not going to give you the answer again. But if you have some gamesmanship about this kind of problem, you kind of sense that the answer must be no: he didn't vote for Teddy Roosevelt. But there's got to be something in the phrasing of the problem that indicates why the answer is no - like either this had to take place at a different time, the guy couldn't have been alive in 1904, it's taking place outside of America so he couldn't have devoted for Teddy Roosevelt, etc. So there's something going on in the presentation of the problem that is telling you no because I find it difficult to imagine how this information could prove that he did; could prove that the answer is yes. So you kind of know the answer is no, but why the answer is no is tricky. Anyway, solving a problem like this requires a lot of background knowledge. For problems of this kind, the background knowledge may come from all over, may come from all kinds of different domains. Moreover, you have to search within the narrative to see what things are relevant to solving the problem.

[21:53] These kinds of problems that we've talked about in this discussion today are all problems that represent challenges for computational versions of problem-solving. When we talk about machines and minds, different ways and similar ways in which machines and minds operate, these are interesting objects to think with because to get a computer to approach problems like this - and in different cases, people are working on topics related to these ideas - but to get a computer to approach problems like this requires interesting, creative, new approaches beyond that of the problem space.