Problems for Minds and Machines

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.

Selecting a problem for mind and machines:
Puzzle problems are good examples to begin with. There are strengths and limitations of these puzzles.

1. They are characterized by not only being **conscious**, but iIt seems to take **effort** to solve them.
2. There's not a strong dimension of cultural or emotional baggage to these things. They're self-contained, which is that they don't require a great deal of background knowledge, they require some.

Representing a problem:
A natural way is to think of these problems in terms of **problem space** i.e. **graph**.

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 moves in the puzzle.

> *So if we were to move in this 15-16 puzzle, 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.*

How hard is the problem:
It can be determined by how big is the problem space i.e. 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 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.

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 how do we reach the goal state.

Reaching 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. This can be referred to as a **search problem**.

We search 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.

Ways to search a graph:

1. **Weak methods**

   They don't assume any real knowledge of the problem beyond the problem space description.

   a. **DFS (Depth First Search)**
   b. **BFS (Breadth First Search)**

   They're useful methods and they're often the foundations of more complex methods.

   Problems with these algorithms:

   DFS:

   We may run into a problem if the problem space is infinite. You may find yourself, for example, looking down a path of potential moves that never ends.

   BFS:

   Sometimes problems expand exponentially. So if you look at a tree like this, you might see one node in the top layer, three nodes in the second, nine 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.

2. **Human-like methods**

   a. **Hill climbing**
   b. **Best-First Search**
   c. **Means-ends Analysis:**

   It was one of the early techniques for problem-solving studied both in human beings and in writing computer programs to solve puzzles. It does not work in every situation. This was investigated early on by the way by **Alan Newell** and **Herb Simon** in the early years of artificial intelligence.

   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 posea new problem which is getting to the point where I can attack it and try to solve that problem by means-ends analysis. It's an inherently recursive idea.

Issues faced while searching graph:

1. **Completeness of the problem / appropriate search algorithm**

   We'd like to have a search technique that is guaranteed to find a solution if there is one. So taking for example are 15-16 puzzles, 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. 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.

2. **Time to find the solution**

   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. We might want to find an approximate solution,or see if we can find a better search algorithm that will take less time. So memory is 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.

3. **Finding the best solution**

   Our algorithm may find a solution, but it may not find the best solution. This could conceivably be a meaningful issue to you where not only do you want the program to find a solution, you wanted to find by some measure the best solution.

Thinking of problems in search space formalism:

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.

Chess memory experiment:

Done by **Chase** and **Simon** in the **early 1970s**. Designed to test that question: *“Do chess experts have better memories than chess novices?”*

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, then they asked both the chess novice and the chess expert to see how well they could recreate the board from memory. Chess experts were able to recall the position of the pieces relatively well as compared to

novice players. But, when the pieces were arranged randomly, both the experts as well as the novice players were at an equal stage of recalling the positions.

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.

Another issue is that sometimes it's tricky to determine how to think of a problem space.

Chess-Domino experiment:

If we are given a chess board(8x8) and we have to tile the board with 32 dominoes in such a way that the dominoes are covering the entire board and they don't overlap. We can easily do that by taking the chessboard and then placing four dominoes in each row or four dominoes in each column.

Now, in a variant of this problem(**mutilated chessboard problem**), we take away the opposite corner squares of the chessboard leaving a board with only 62 squares on it i.e., we've taken the upper left and the bottom left square out of the chessboard. Now we cannot solve the problem. Because every domino that you placed 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, can't be done.

This is, in fact, it's 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.

So these are instances of the limitations of the unthinking problem space approach, in order to look at a problem in a creative way, we may have to think about the problem space in ways that are non-obvious.