Week 1

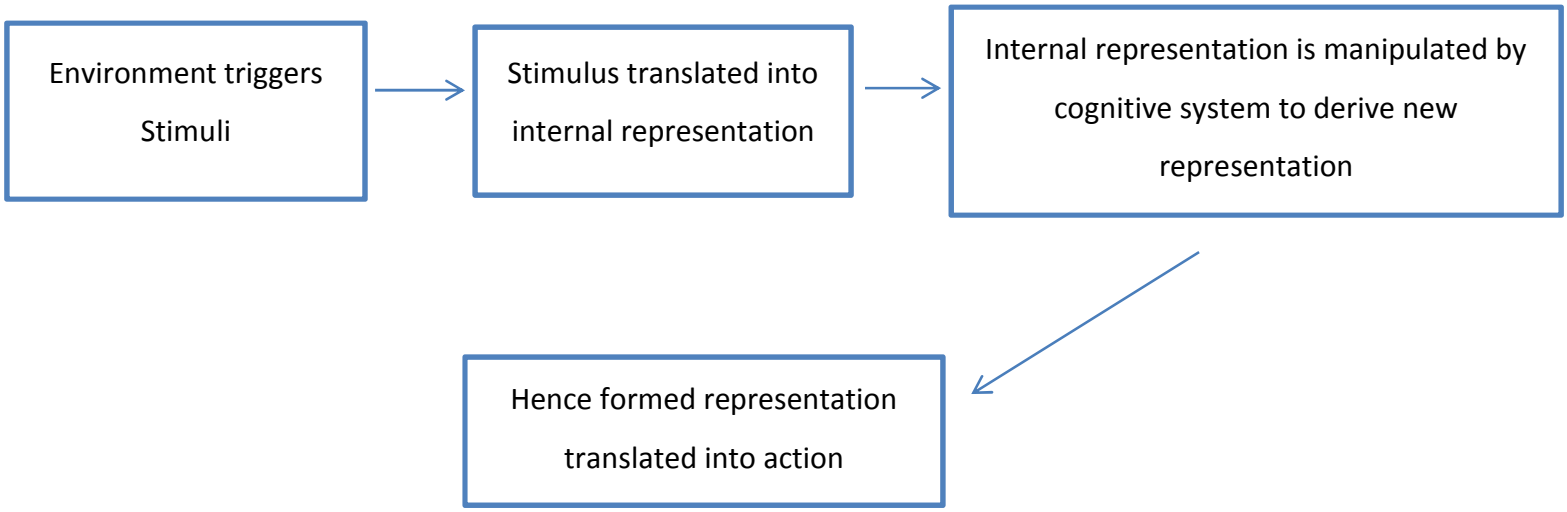
From AIND >

What is intelligence?

Basically anything which could acquire and deduce information and can retain it as knowledge to apply and see it in relation with previously acquired knowledge, and all these considered within a specific domain or an environment. This was remarkably explained by that example of AI based room cleaner which is intelligent in relation to its environment i.e. cleaning

How we can emulate such behavior in machines?

How far can we can we take it?



Modern day Ai/ML algorithms all emulated from as simple approach as described by Craick, 1942 on broader sense that how our brains perceive and act to stimuli from environment, this approach purported quite an approach used in naïve neural network.

Building an intelligent Sudoku agent

The convention used is:

1 box -> single box in Sudoku.

1 unit -> group of 9 boxes with a property that each one of them should hold a unique single value between 1 and 9.It could be a diagonal, column, row, or a 3*3 square.

Peers of box -> elements of all units it is part of.

The Sudoku agent is taught in lectures is built using Uniformed search as base.

Uninformed means we *only* know:

- The goal test
- The ***childcreater*** () function that creates child

(But ***not*** which non-goal states are better)

Uniformed search can be defined as:

> "A problem determines the graph and the goal but not which path to select from the frontier. This is the job of a search strategy. A search strategy specifies which paths are selected from the frontier. Different strategies are obtained by modifying how the selection of paths in the frontier is implemented."

For example: BFS (one of **uninformed** search method). We just generate all successor state (child node) for current state (current node) and find if there is a goal state among them, if there isn't we will generate one of child node's successor and so on. Because we don't have information, so just generate all.

The Sudoku solving agent described in the course work as.

- For all the blank spaces, we fill them with the all the possible values they could take in. i.e. for a given blank space we have look through all its peers and fill in the values not contained in any of them.
- Then for every unit we evaluate if there is any number which is contained in only one unfilled block. If some block contains such value the value in the block is replaced by that number.
- We now again apply the repetition of steps 1 and 2 until the puzzle is solved (we apply this repetition as for example consider some unit on which we worked in step 2, for the newly assigned no to the block we cleared that unit of that number but other units for which it is part of may still

contain that number, so we again need to apply the step 1 to completely remove that no from its peers).

As this approach may many times doesn't lead to solved puzzles. So we now bring in notion of uninformed search.

Two Uniformed search strategies that can be used here are:

- Depth First Search with Backtracking.
- BFS

(Both DFS and BFS use STATE Based Models)

State-based Models:
- **Model task as a graph of all possible states**
  Called a "**state-space graph**"
- A state captures all the relevant information about The past in order to act (optimally) in the future
- Actions correspond to transitions from one state to Another
- Solutions are defined as a sequence of steps/ Actions (i.e., a path in the graph)

We usually prefer DFS in game solving agent like these cause we want to completely visualize the future a move may lead on to, if it solves it ok, otherwise we try other move,

"For example in games like Chess, tic-tac-toe when you are deciding what move to make, you can mentally imagine a move, then your opponent's possible responses, then your responses, and so on. You can decide what to do by seeing which move leads to the best outcome. Only some paths in a game tree lead to your win. Some lead to a win by your opponent, when you reach such an ending, you must back up, or backtrack, to a previous node and try a different path. In this way you explore the tree until you find a path with a successful conclusion. Then you make the first move along this path. "

And also DFS is memory efficient compared to BFS.BFS requires us storing all the possible states but in DFS with backtrack the function backtracks from unsolvable states and the memory is hence released. BFS can be used here if we were to find problem tacking shortest path to the solution, though this isn't an issue if Sudoku have a unique solution there we can use DFS but some Sudoku don't have unique solution and there if we need shortest path BFS need to be used.

We have used DFS as:

- We have combined our step 1 and step 2 into single step .Let us call it step A. Which returns False if given Sudoku isn't solvable other returns the Sudoku.
- We call step A in DFS. If the tree isn't solved we create Children on some block by (taking into consideration each of the numbers inside it one by one). We create children with a constraint that

the block with the minimum no of NUMBERS will be provided with single number. This is because we will have fewer cases to consider at the upper parts of tree hence will be having lesser breadth at the starting of the tree and hence the tree will be comparatively smaller and faster and so the time will be comparatively lesser**. If step A return false** we return False. **If children return False** we traverse other children.

THE SUDUKO PROJECT

The Naked Twin problem,

The statement: If in a unit there is an occurrence of 2 blocks each having same NUMBERS and each of length 2 .Now as we know that there are 2 numbers and 2 blocks to hold them so there could be possible 2 cases i.e. blockA holds Number1 and Block B holds the Number2 or the other way round. The central idea is that no other box/block in the unit can hold these two numbers other than them. So we remove these numbers from all other Boxes from unit. There could be problem if we removed these numbers before calculating all suck pairs .For example,

Consider in 1st COLUMN unit the boxes be holding numbers as in figure, and consider 7th row to be as in figure, now we were 1st column we processed 56 and 56 as Naked twins now we were to strip all other boxes in the unit of 5 and 6, then 54 in 2nd row changes to 4 and 46 in G1 changes to 4. Now later when we come across 7th row we can't deduce naked twin for 46, cause 46 in G1 is now just 4.So here an anomaly.

9							
47							
17							
56							
17							
56							
46	9	8	**46**	3	**75**	2	**75**
3							
2							

Stripping everyone off 5 and 6 in column 1.Leads to…

This may lead to unsolvable Sudoku because we have skipped the case with 6 the G1 and 4 in G4.

9							
47							
17							
56							
17							
56							
4	9	8	**46**	3	**75**	2	**75**
3							
2							

Another astounding example of Naked twin not covered in course but can be seen as

9							
4							
78							
567							
1							
568							
46	9	8	**46**	3	**75**	2	**75**
3							
2							

Here, 567 and 568 are in fact naked twins ,cause the box D1 and F1 both are the only ones that can only take 5,6 no other box takes on 5 or 6 ,So 7 and 8 here could be ignored and the problem could be seen as Naked Twin Problem.

Naked Twin problem.

Other optimizations which could further speed up our Sudoku solving:

➢ Triplets Finding

Triplets can occur in rather different fashions, they have the Naked Twin Analogous though. Suppose one box in the Unit may contain numbers like XYZ, the other boxes may contain pairs of XYZ in any fashion, but they ***should not be twins*** and the ***union of other two boxes should produce all the three entries.***

One possible example is:

475	9	8	**46**	3	**47**	2	**75**

⇨ ***Same triplet from previous in different form. Other than these there also be different numbers than triplets in the box (that condition could make a bit more complex, though can be implemented).***

475	9	8	**46**	3	**475**	2	**75**

475	9	8	**46**	3	**47**	2	**475**

475	9	8	**46**	3	**475**	2	**475**

475	9	8	**46**	3	**75**	2	**45**

(There can be many permutations of these.).The elimination and rest steps remain similar to those in naked Twin.

Quadruple Problem

The steps of detection remain analogous to those in Triplets detection; it's just that here could be even more permutation's to look through.

When to apply naked twins or triplets etc. stripping?

Just before creating the child in DFS we furnish our current Sudoku and then send in new values to the child. This is cause so there are lesser permutations of Sudoku to consider from and hence lesser the breadth of the tree to deal with and with same depth so better performance.

Diagonal Problem

There isn't much there in this problem we just got to create 2 new units with diagonal elements for both the diagonals, and it to our units list and the program does the rest.

When considering naked twins or triplets or anything we just got to look through our newly created diagonal units too.

As the Wikipedia article on Solving Sudoku says,

"There are many 40000 symmetries and related clues for solving them have been discovered."

Further optimization can be done by these. Or we can apply a Variant of DFS and BFS combined too for solving this gives better results for some cases.