ARTIFICIAL INTELLIGENCE COURSE PROJECT

REPORT 2

TOPIC: SUDOKU

CS22B2015 HARSHITH B
CS22B2022 PRANAAV PRASAD

---

1. INTRODUCTION

Before we start explaining how we are using AI and what techniques and topics of AI we are using to solve Sudoku, let's recall what the problem is again.

The problem statement we have chosen is one of the most exciting puzzles in the world, Sudoku, which has a size of $n^2 \times n^2$ (where $n = 1, 2, 3, 4, \ldots$).

In the last report, we mentioned how AI plays a role in solving the Sudoku problem and how vast the puzzle state space can be. In the upcoming sections, we have mentioned how we are solving Sudoku using AI.

2. METHODOLOGY OF SOLVING

2.1. STATE SPACES.

State space in AI usually tells us the information of the state of the puzzle or puzzle at any particular stage of the problem-solving stage. Any data structure or a combination of data structures can be used to represent a state space of a problem.
In the present topic of SUDOKU, we choose an array of $n^2 \times n^2$ to represent the state space of the puzzle.

2.2. GENERATION OF THE STATE SPACE TREE.

The Sudoku Solver program receives an initial state of the Sudoku puzzle and makes it the root node. This initial state is passed as a parameter to the generate_next_states function.

In FIGURE-1 in the next page we can notice that there are only 3 children for the tree node i.e only 3 possible values were possible to be generated by generate_next_states function. This is because the function is following certain constraints to generate the next space discussed in the next part.

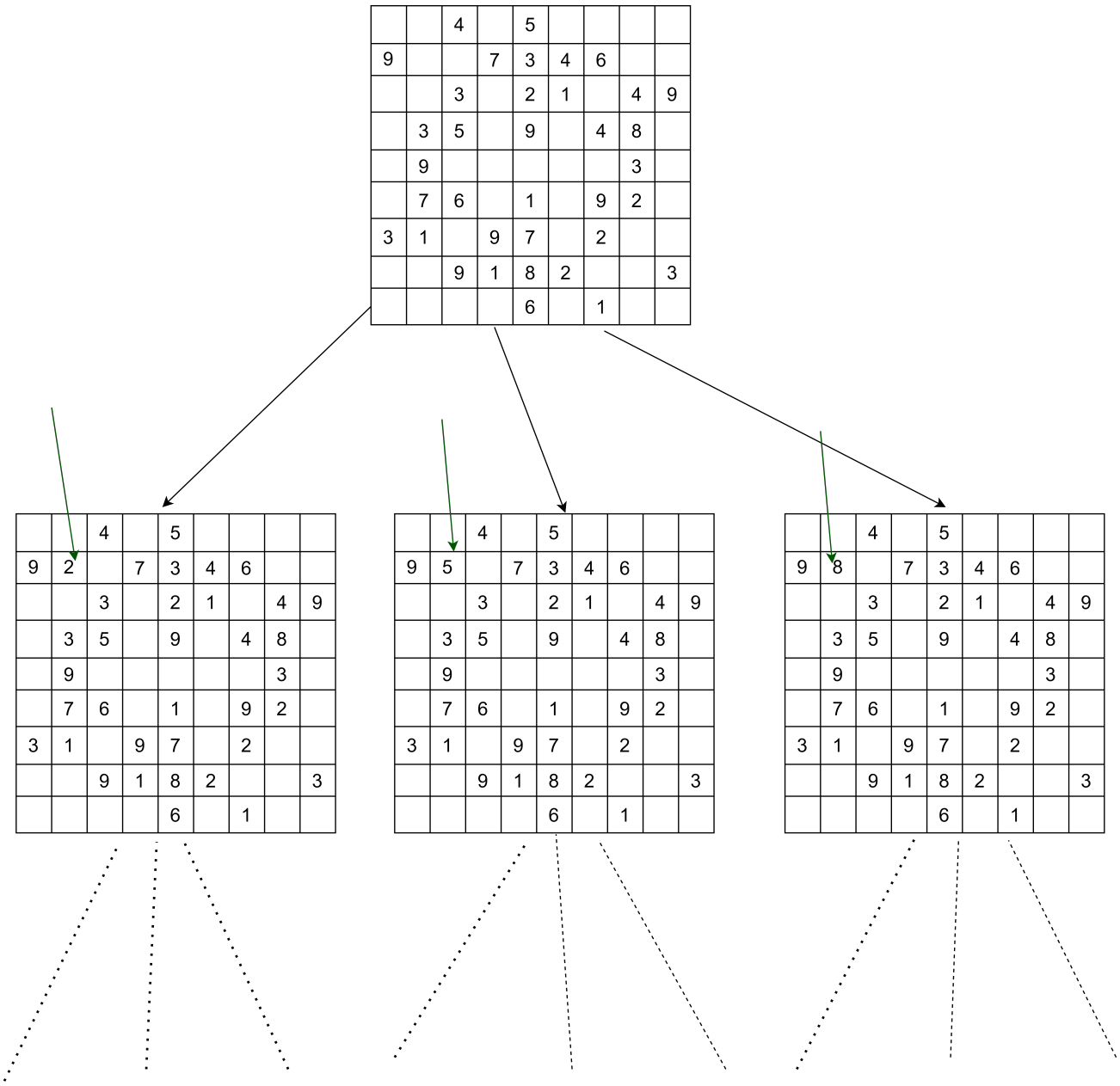

FIGURE 1. STATE SPACE TREE OF SUDOKU

2.3. **CONSTRAINTS.**

The constraints to solve Sudoku can also be in other words said that we are following the standard rules of Sudoku which are:

(1) **SUBGRID UNIQUENESS RULE**: Each $n \times n$ subgrid of the puzzle shall have only one unique values that belong to $[1, n]$.
(2) **ROW AND COLUMN UNIQUENESS RULE**: Similar to the above rule for any box in $n^2 \times n^2$ the value it contains should not only be unique to the subgrid but also to its particular row and column.

Based on the above constraint or rules we can see in the image why we were able to get only 3 possible values for the position/index (2,2) which are 2,5, and 8. Using this and another topic called heuristics we enable the function to generate new states smartly instead of blindly generating some superficial states which we know can generate wrong sudoku solutions or slow down the process to solve the problem.

2.4. HEURISTICS.

Heuristics in AI gives a rough estimate of the cost to achieve a target or destination child in the state space tree. In this particular project we have applied the following heuristics to traverse or generate the state space efficiently.

(1) **SUBGRID HEURISTICS**: In this heuristics we quantify the sum of all the filled values or the amount of non filled spaces in each subgrid 1 to n. Then we choose the subgrid with least non filled spaces to choose a space to fill the next value or generate the next state.
(2) **ROW/COLUMN SUM HEURISTICS**: From the above heuristic to choose a space out of n-k available spaces(k is the number of filled spaces) we calculate the space which has the row and column with least empty spaces i.e max(non-empty row spaces + non empty column spaces).
(3) **SUM OF RELEVANT VALUES**: In this heuristic we calculate the sum of number of options per subgrid + number of options per row + number of options per column.

2.5. GRAPH TRAVERSAL.

From the above section we have determined the constraints, heuristics and how the next state is being generated. Now to actually find the optimal path out of this vast puzzle state space we need to implement graph traversal. We have many graph traversal algorithms from the most basic Depth First Search, Breadth First Search, Iterative Deepening Depth First Search to little more constraint optimised A* and DFBB algorithms.

In this particular sum though using the heuristic 3 in the above subsection, A* can be achieved by quantifying h(n) but since each new state space will have only one value to be inserted and a sudoku puzzle will have only 1 or atmost a very limited amount of solutions A* isn't an appropriate choice to solve this puzzle.Therefore we choose to solve the sudoku puzzle using the standard DFS and BFS.

2.5.1. ***DFS:***. Depth First Search uses a Stack data structure. We maintain two arrays Open and closed and after visiting the first element a1 in Opened we push it to closed and push all the nodes reachable through a1 into Opened to the top of the stack/front of the array.

2.5.2. ***BFS:***. Breadth First Search uses a Queue data structure. Similar to DFS we maintain two arrays but instead of pushing the nodes reachable through a1 to the top of the stack/ front of the array we push it to the last of the stack.

2.6. CHOICE OF GRAPH TRAVERSAL.

In the above section we discussed about only two of the graph traversal techniques and in the Heuristics subsection we also mentioned why A* is not an appropriate choice. In this section let us understand why other graph algorithms other than DFS are not the best choice.

Sudoku is a problem where we already highlighted the fact that it has only 1 or very limited number of solutions because it comes under the category of CSP's(Constraint Satisfaction Problem). In solving CSP's we assign value to each state one at a time to satisfy the constraints and go on to solve the entire problem step by step. Therefore in tightly constrained problems with few possible solutions, such as sudoku backtracking algorithms are the best and return the best results. And only DFS comes under the backtracking algorithms category.

2.7. COMPARISONS.

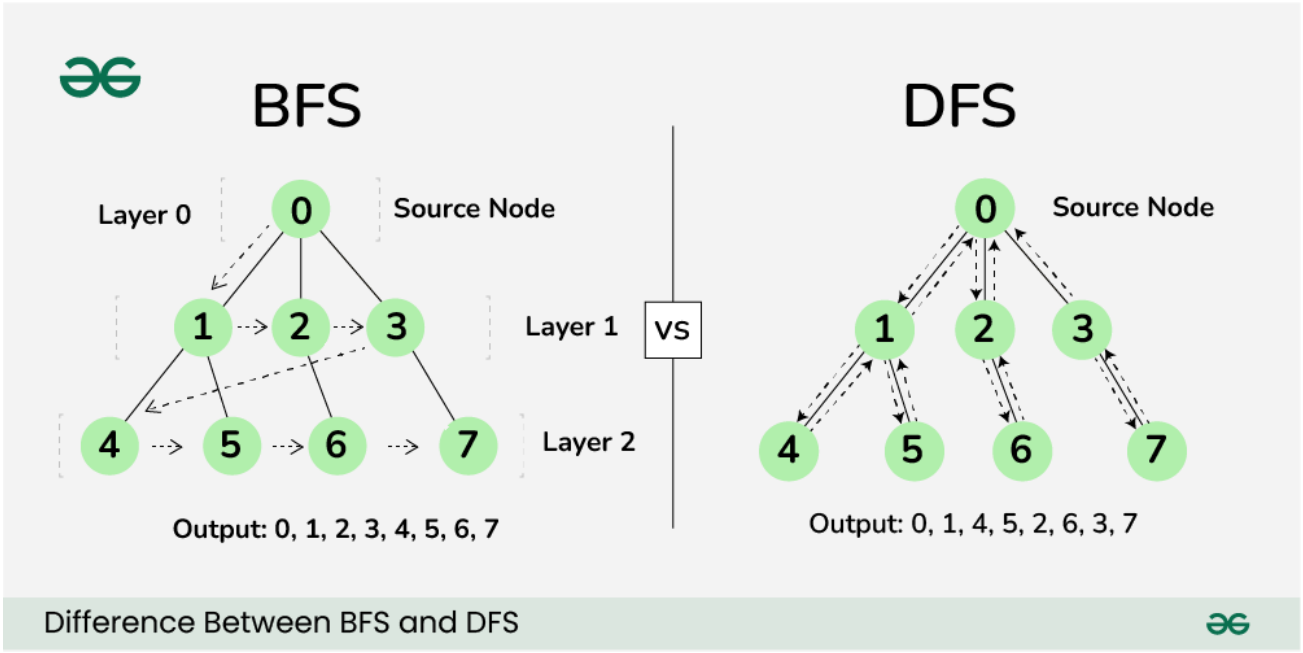

FIGURE 2. BFS vs DFS (Source : GeeksForGeeks)

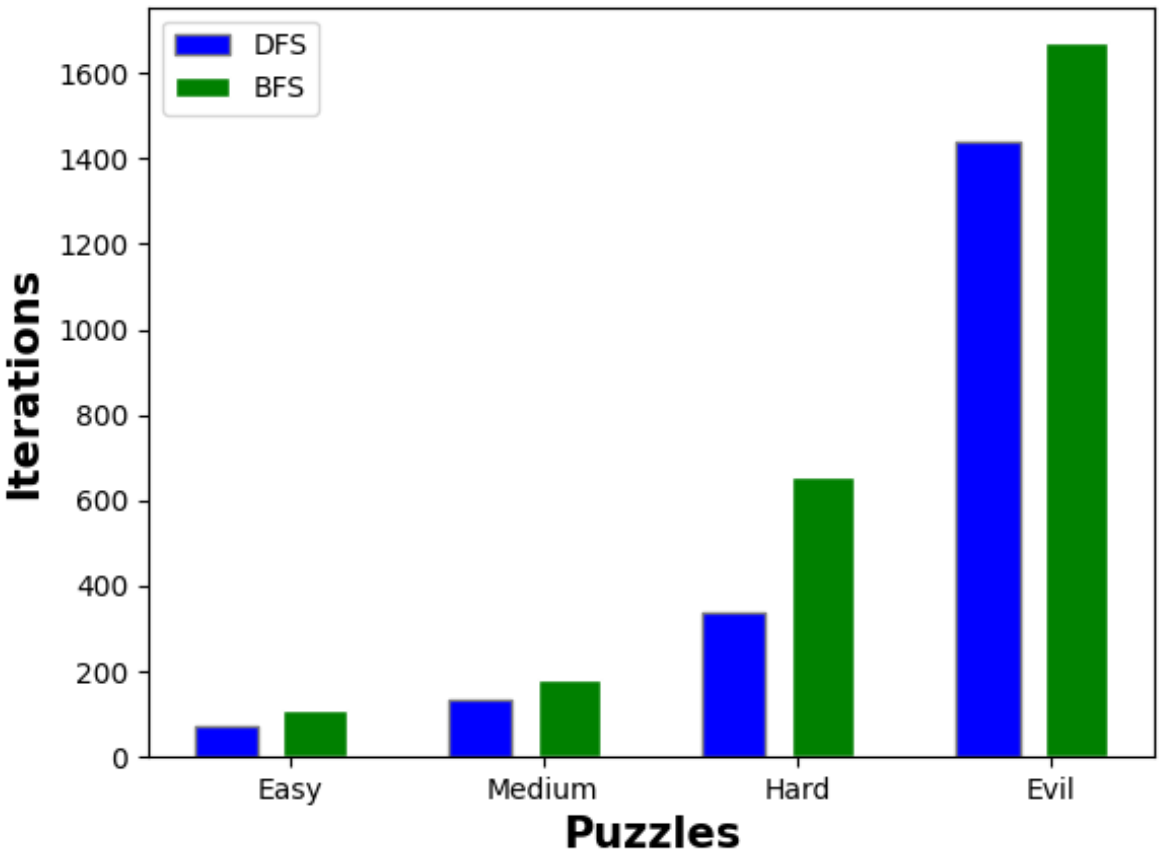

FIGURE 3. DFS VS BFS

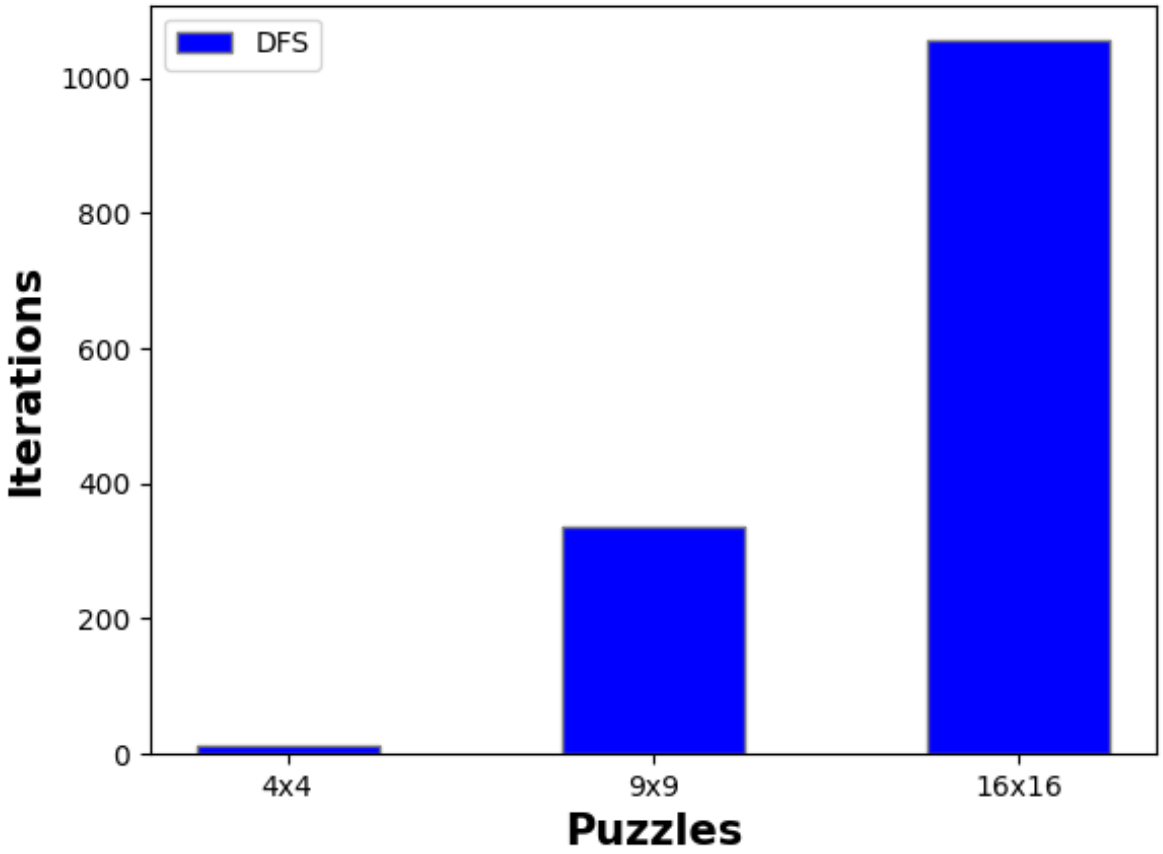

FIGURE 4. DFS TIMES FOR DIFFERENT SUDOKU SIZES

In the above FIGURE 3 we can see a comparison graph between DFS and BFS which tells us the number of iterations or function calls are taking place for a 9x9 Sudoku puzzle of different difficulties easy, medium, hard and evil. We can notice that DFS is clearly faster to solve sums like SUDOKU.

In FIGURE 4 we can see a graph which tells us that the number of iterations for DFS increases exponentially as we increase n value exponentially from n = 2 to n = 4.

3. CONCLUSION:

We have seen how AI solves sums more importantly how we can implement the algorithms and topics of AI into solving a widely known and interesting puzzle SUDOKU successfully using various techniques and comparing which will give us an optimised path to solve a SUDOKU problem.

SPLITTING OF WORK:
CS22B2015 : Reports, Comparisons and Optimal Path Finding
CS22B2022: State space generation, Implementing heuristics and Traversal