# Solving Sudoku Puzzles with AI

## CS7IS2 Group Project (2021/2022)

Rakesh Nair, Mohammad Eslami, Misbah Rizaee, Congqin Yan, and Tapabrata Mukhopadhyay

ranair@tcd.ie, meslami@tcd.ie, rizaeem@tcd.ie, yanc@tcd.ie, mukhopat@tcd.ie

**Abstract.** Sudoku puzzle is a classic problem in which missing digits are to be placed into a 9x9 grid of squares that are split into 3x3 boxes so that the numbers 1 through 9 appear only once in every row, column, and box. Although every Sudoku puzzle has solution, they involve different levels of hardship. In this paper, we introduce, implement and analyze some of the AI algorithms suitable including uniformed-search, genetic algorithms, constraint satisfaction and convolutional neural network to solve any given Sudoku puzzles. The results are compared with respect to the time taken and memory consumed by each algorithms.

**Keywords:** Sudoku puzzle, artificial intelligence, uninformed search, constraint satisfaction, convolutional neural network, genetic algorithm, backtracking algorithm

## 1 Introduction

Number games and grids like Sudoku have been around for hundreds of years but the modern game, as we know it today was originated by Howard Garns. Howard's first Sudoku puzzle was published in 1979 by Dell magazines and it was called Number Place but today we know it as Sudoku [3]. The famous Swiss mathematician, Leonhard Eular also wrote about similar concepts but the idea of these types of number games in general dates back ancient china [4].

From our experience, Sudoku games fall into two categories, they are either so easy which sometimes they become boring or they are too hard and we stop solving them. The motivation behind our research on Sudoku game is to find how well different algorithms can solve any Sudoku puzzles in few seconds. The objectives of our research are to compare and contrast the performance of these algorithms and examine which one is faster. Maybe there are some puzzles which takes minutes to be solved so we are going to discover such cases as well.

### 1.1 The Rules of Sudoku

The objective of the Sudoku game is to correctly fill in all the empty squares, in a grid of squares, with the correct numbers. As it was mentioned earlier, the classic Sudoku game involves a grid of 81 squares which are divided into 9 blocks, each block contains 9 squares. Each of the blocks, rows and columns must contain all the numbers from 1 through 9 within its squares without any duplicates.

## 2 Related Work

Whenever we are doing problems like Sudoku, there are always many ways to approach them. Starting with the backtracking algorithm, this is one of the best algorithms to solve such problems. We have made solving Sudoku problem fantastically simple using this algorithm. The idea behind backtracking algorithm is very easy to understand. Any time we have a problem that can be solved by a series of decisions, we might make a wrong decision but when we realise that we have made a wrong decision, we can backtrack to the place where we have made a decision and we can try something else. What was tricky about backtracking is not the concept but figuring out how to implement it in the code and apply it in program. Backtracking is most often associated with recursion but the way backtracking happens in a recursive solution is tough to see and that can make it difficult at first to write recursive backtracking solutions.

Search algorithms can be used to tackle artificial intelligence problems. Uninformed search and informed search with heuristics are two types of search algorithms. Uninformed search is a search method that is able to only differentiate between a target state and a non-goal state and has no knowledge about how distant the goal state might be from the present state [2]. In other words, uninformed search techniques are strategies which use a tree-traversing algorithm to search the sequence through which an agent could reach to the goal state without using any additional information. Depth-first search (DFS) and Breadth-First search (BFS) are among the uniformed search strategies.

As it is apparent, there is a set of constraints that are to be satisfied in every given Sudoku puzzle. Hence, Sudoku is recognized as a classic example of constraint satisfaction problem. Helmut in his paper [1] found a method using quasi-group completion concepts by which the Sudoku solution is obtained without performing any search algorithms. This can significantly reduce the time and memory usage and indicates that constraints satisfaction can be an efficient choice for solving Sudoku puzzles.

Genetic algorithm is a search heuristic method to solve the problem, self-discovery and make judgement quickly and efficiently. this algorithm is inspired by Darwinism, which is a biological evolution theory that all species arise through natural selection. A genetic algorithm generates a patch of generations with possible solutions for the research problem and selects the fittest individuals which are parents to reproduce the next child generation. Evaluate them to get the fittest children as the next parents to get the finalized best solution.

## 3 Problem Definition and Algorithm

Sudoku puzzle is a relatively old classic problem. Hence, many approaches have been developed to solve Sudoku. As the focus of this project has been on AI methods, we have tried to utilize some of the classic and interesting AI methods to solve Sudoku. For instance, the solution of a Sudoku puzzle can be found using search algorithms. The more efficient the search algorithm is, the more

quickly the solution is found. Sudoku is also recognized as a classic constraint satisfaction problem. The essence of Sudoku problem is based on the constraint that we have to satisfy. These constrains (see section 1). We also apply some relatively new approaches such as Convolutional Neural Network (CNN) and Genetic Algorithm to solve the Sudoku puzzles which are discussed in the following subsections respectively.

### 3.1 Uniformed Search Agent (Baseline)

Search algorithms can be used to tackle artificial intelligence problems. Uninformed search and informed search with heuristics are two types of search algorithms. Uninformed search is a search method that is able to only differentiate between a target state and a non-goal state and has no knowledge about how distant the goal state might be from the present state []. In other words, uninformed search techniques are strategies which use a tree-traversing algorithm to search the sequence through which an agent could reach to the goal state without using any additional information. Depth-first search (DFS) and Breadth-First search (BFS) are among the uniformed search strategies. The difference is in the data structure by which they queue the nodes of the tree to visit throughout the tree-traversing process.

**Depth-first Search** The Depth-fisrt search (DFS) is a popular method to be used in search problems. When a dead-end occurs in any iteration, DFS method traverses a tree in a depth-ward motion and utilizes a stack to remember to acquire the next vertex to start a search [tutorial point dfs]. We first define the terminology of a search algorithm in Sudoku context.

- Node: every state of a Sudoku puzzle is represented as a node in the tree search. Each node may contain a set of cells which are already assigned a value and a set of empty cells to be filled.
- Successor: A successor of a node inherits the current state of its parent. In addition, it has one more filled cell than its parent. That is, every state of the puzzle that its next empty cell has been filled with a valid digit which satisfies the constraints of the parent node is considered the successor of the parent node.
- Constraint Check: To assign a value to an empty cell to create a successor, the agent must check if the new value satisfies the requirements and the constraints of the current state of the puzzle (see section 1).
- Goal State: The goal state is a state in which all cells of the puzzle have a valid digit and all of the constraints of the puzzle have been met.

The process is started by finding the first empty cell to which we have to assign a value. We then try all values from 1 through 9 in the cell. Among all possible values, we select the values which satisfies the constraint of the Sudoku puzzle (see section 1). For each valid candidate value, we make a copy

of the current state of the puzzle, assign the value to the cell and return it as a successor. It is obvious that a node can have multiple successors. We then recursively repeat the same process for all successors of the node until we reach the goal state. Figure (1) illustrates the idea behind the DFS in Sudoku context:

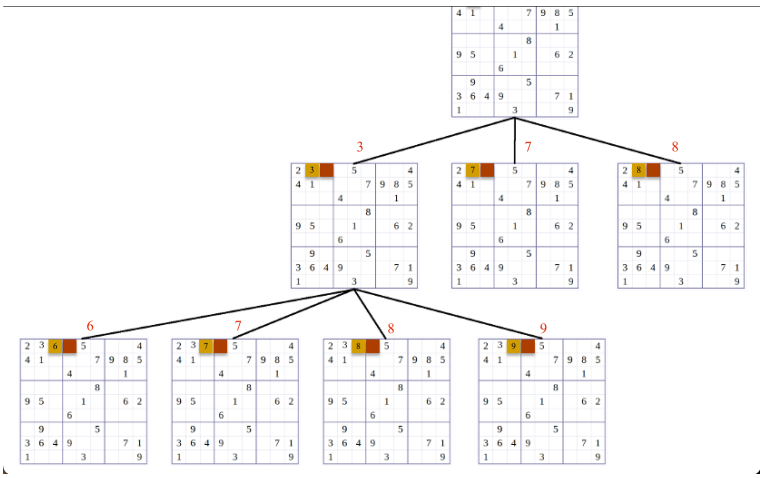

**Fig. 1.** Depth-first Search in Sudoku puzzle

### 3.2 Backtracking

Now let's talk in more details about how we would actually approach Sudoku problem using Backtracking algorithm. Backtracking algorithms attempt to solve a problem one step at a time. If it becomes obvious at any point along the way that the present path will not lead to a solution, it will return to the previous step or backtrack and take an alternative option. This is known as recursive function which is used in backtracking algorithms. A recursive function is one that keeps calling itself until a certain task is completed. It tries to solve a problem by examining all potential solutions until one is identified. If a solution is not discovered each time a path is examined, the algorithm goes back to try another potential way, and so on, until a solution is found or all of the paths have been evaluated.

### 3.3 Constraint Satisfaction

To solve the sudoku puzzle, we will use PuLP package in Python. PuLP is a python package that may be used to solve linear programming. As mentioned earlier, The essence of Sudoku puzzle is based on constraint satisfaction. It is worth mentioning that there is no optimal solution to a Sudoku problem, which

means we should not look for maximization or minimization of any equations. To solve Sudoku using constraint satisfaction techniques, we first need to define the variable of the problem. These variables depend on the nature of the constraint present in the problem. Therefore, we first look into the constraints that we have to satisfy in a Sudoku puzzle.

**Constraint 1: Each cell can only have one value** Only one value can be present in each cell, meaning that the other values will be absent. As a result, we may consider a binary variable with a value of 0 if the number is not there and 1 if it is. The variables looks like below:

$$(\text{value, row, col}) = 0 \text{ or } 1$$

We should maintain the row and column variables as constant while varying the value from 1 to 9. This guarantees that every cell has just one value. Since only one parameter would be equal to 1, the rest has to be 0, the total of the binary digits should be 1.Because the row and column both have a range of 1 to 9, there are 81 permutations (row, col). There will be 9 binary variables in each combination. As a result, there are 729 variables. Only 81 of the 729 distinct pairings will have a value of 1, while the others would have a value of 0.

**Constraint 2: Each row must have all values from 1 to 9, no repeated value** We can maintain the value and row unchanged while changing the col from 1 to 9 in this situation.

**Constraint 3: Each column must have all values from 1 to 9, no repeated value** Instead of maintaining value and row constant, we keep value and col constant in this constraint, which is quite similar to our prior constraint.

**Constraint 4: All values from 1 to 9 must be present in each of the 9 boxes, no repeated value.**

**Constrain 5: Set the variables for occupied cells to 1** In Sudoku puzzles, there are usually some cells which are already assigned a value. Each value which is present at each cell is considered as a new constraint to our problem. Therefore, we set the variables (value, row, col) of already occupied cells to 1.

Finally, we solve the problem Pulp.solve() using method which solves a linear program with the defined variables and returns the solution. The linear equation created in this process is written into a file called Sudoku.lp.

### 3.4 Convolution Neural Network (CNN)

An artificial neural network is used for modelling. It is comprised of node layers, consisting of input layer, one or more hidden layer and output layer, which delivers the final output. Each node connecting to one another has a weight and threshold associated with it. If output of any individual node is above threshold, data is sent to the next layer else no data is passed along within the network. This help us to learn which notable features are there in the data to produce the output. Each of this node has its own linear regression model, consisting of data, threshold, weight, and an output.

Convolutional neural networks, also called ConvNets are used to recognize a pattern present in images. It is a class of deep learning, feed-forward artificial

neural network technique which uses a variation of multilayer perceptions designed because of which it will require minimal preprocessing. It is remarkably similar to ordinary neural network; they are made of neurons that have learnable weights and biases. Unlike NN, layers in ConvNet have neurons arranged in 3 dimensions: **height, width, and depth**. Neurons in a layer is only connected to the small region of the previous layer, instead of all the neurons in a fully connected manner. At the end of the ConvNet architecture, the full image is reduced to a single vector of class scores arranged along with the depth. A simple ConvNet is a sequence of layers, and there are three types of layers in a CNN: convolutional layer, pooling layer, and fully connected layer.

The convolution layer is the first layer of the convolution network which can be followed by more convolution layers or pooling layers, the fully connected layer is the final layer.

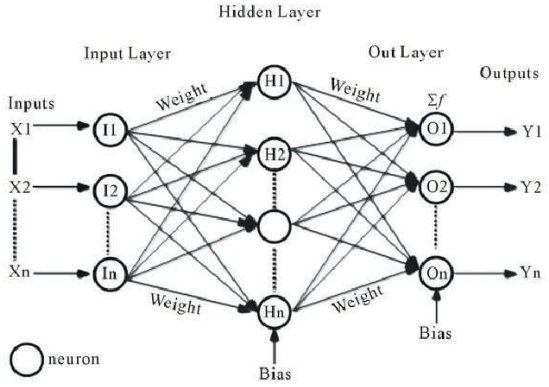


**Fig. 2.** A schematic of an artificial neural network

**Convolution Layer** Convolution layer is the core of the CNN which does most of the computational heavy lifting. it requires a few components, which are input data, a filter, and a feature map. Convolution layer applies a convolution operation to the input, passing the result to the next layer, which is a linear operation which involves the multiplication between an array of input data and two-dimensional matrix of weights called a kernel (kernel is smaller than input data and the dot product is used for performing multiplication between them). The final output from the series of this dot products from the input and filter is known as activation map/feature map or convolved feature. CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map after each convolution operation for introducing nonlinearity to the model. Two hyperparameters were used to control the size of the output volume – depth and zero-padding.

- Depth corresponds to the number of filters that we would like to use which will affect the depth of the output.
- Zero-padding allow us to control the spatial size of the input volume, so that the shape of the input and output volume are the same. There are three types padding – Valid, Same and Full. In our case, we are using Same padding.

**CallBacks** Callback functions are defined to perform actions at various stages of training, in our case the training is stopped if there is drop in accuracy.

### 3.5 Genetic Algorithm

**Generate Initial Populations:** In the population, there is a batch of generations, which have a number of chromosomes. The generating processing begins with generating chromosomes for each generation, each chromosome is the possible solution for Sudoku, generating the chromosome base on the given puzzle of Sudoku, chromosomes are every available element that can take place on the zero elements on a given puzzle. Evaluate initial populations by using the fitness score to assess each chromosome's performance. Based on the Sudoku roles, every column and row should only have nine unique numbers from 1 to 9, and each 3X3 block of Sudoku should follow the same pattern. Recording every unique number as 1 in every column and blocks, if one of the numbers is duplicated, then plus record to 2. The more duplicated number, the more weight it contains in the records. the fitness score is the average duplicated column rate times duplicate block rate.

**Selecting Elite Generations:** Choose the number of generations, which have better fitness scores. They include more valuable chromosomes or solutions to create new children by using the crossover. Maintain a certain amount of generations from the last population, and generate the rest of the generations as the next population.

**Parents' Selection for Crossover:** Selecting the best fittest chromosome (solution) as parent generation to crossover and pass their genes to the child generation in order to get a better solution. Two chromosomes are chosen based on their fitness scores, selecting the parent with a higher score or a low score equally, which means it has a 50 per cent of peaking the one with the higher fitness score and 50 per cent of choosing the one with a lower fitness score.

**Crossover:** Crossover is the most important step to generating get optimized generation. There are many swapping algorithms for crossover functions, such as single value crossover, row crossover, column crossover, and block crossover. In our program, we used the row crossover method on parents' chromosomes, Swapping each element in every row in chromosomes, choosing two random rows as swapping objects to exchange their elements.

**Mutation:** The mutation is another operation that optimizes the elements in chromosomes and maintains the diversity of chromosomes by checking three criteria on chromosomes that have already crossover. There are no duplicated elements in the column, no duplicated elements in the row, and no duplicated elements in the block. If two elements satisfy these criteria, then swap two elements and add children to new generations.

**Evaluate New Generations and Repeat:** Check the best fittest chromosomes in generations, if the new generations have too many repeated chromosomes, it will reduce the diversity of the chromosomes, then regenerate chromosomes to avoid that. And then repeat the process from selecting elites generations until it gets the final solution, which is the fittest chromosome.

## 4 Experimental Results

### 4.1 Methodology

For our experiment, we test and compare the performance of all our algorithms over 3 difficulty-level based categories of datasets [5]. The difficulty levels are Easy, Medium, and Hard. Each of these categories are based on labelling by a human and each category has 10 puzzles.

We use the following methodology and evaluation criteria for the algorithms:

1. We create the agents based on 4 algorithms: Backtracking, Convolutional Neural Networks, Constraint Satisfaction, and Genetic Algorithm.
2. We run all the above mentioned agents on our difficulty-level based categorised datasets.
3. We record the time taken to solve the puzzles in each category of datasets.
4. We record the memory consumption for solving the puzzles in each category of datasets.
5. We run and record the time taken and memory consumption for our baseline algorithm (Depth First Search) as well.
6. We create tables for time and memory usage to compare the performance of all the algorithms against the baseline.

**Note:** The memory consumption calculation is done based on [8]. It breaks down the memory usage into two categories:

1. **RSS**: Resident Set Size, which is the physical memory used by a process (i.e. Stack + Heap memory).
2. **VMS**: Virtual Memory Size, which is the virtual memory used by a process.

### 4.2 Results

The following tables illustrate the results of our time and memory analysis by pitting the 4 of our chosen algorithms against the baseline of DFS algorithm:

| | Algorithms | | | | |
|---|---|---|---|---|---|
| | CNN | BT | DFS | CSP | GA |
| Level | Average Time (s) | | | | |
| Easy | 2.7611 | 5.81683 | 96.63875 | 0.10059 | 25.49593 |
| Meduim | 2.9476 | 1.86427 | 244.681863 | 0.120845 | 24.50787 |
| Hard | 3.39929 | 4.91056 | ~1200 | 0.1107175 | 37.576396 |

**Fig. 3.** Average Time Taken to Solve Sudoku Puzzles of Various Difficulty Levels

| | Algorithms | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | CNN | | BT | | DFS | | CSP | | GA | |
| Level | Average RSS (kb) | Average VMS (kb) | Average RSS (kb) | Average VMS (kb) | Average RSS (kb) | Average VMS (kb) | Average RSS (kb) | Average VMS (kb) | Average RSS (kb) | Average VMS (kb) |
| Easy | 3866 | 38380 | 7.782 | 6.554 | 73731 | 73563 | 1854 | 1490 | 169.393 | 131 |
| Meduim | 29.08 | 446.46 | 5.325 | 10.24 | 99051 | 56769 | 1853 | 1490 | 188.47222 | 131 |
| Hard | 464 | 1970 | 0 | 0 | ~200000 | ~90000 | 1869 | 1490 | 176.614 | 970 |

**Fig. 4.** Memory Used to Solve Sudoku Puzzles of Various Difficulty Levels; RSS = Stack+Heap, VMS = Virtual Memory

### 4.3 Discussion

**Backtracking:** In our experiment, we discovered that certain problems are more difficult to be solved than others, even though the difficulty level is the same. We observed an anomaly as well. As can be seen in Fig. 5, it took 40 seconds to solve an easy Sudoku puzzle (more than any other puzzles with hard difficulty level) while most of the other easy puzzles took about one second to solve. A possible explanation for the anomaly is that since the backtracking algorithm strictly follows its process, what might be easy for a human being might not be easy for the algorithm. The easy, medium, and hard puzzles we have used were labelled with those difficulty levels by a human being. The backtracking algorithm, limited by its quirks, took the long route to solving the first "Easy" problem.

**Convolution Neural Network:** An already available Sudoku dataset [6] of one million puzzles was used to train the CNN model. We split the dataset with a 95%:5% training data to test data ratio. Our CNN model records/saves the model after each epoch as a checkpoint and stops the epoch if the accuracy of the model appears to be decreasing compared to the previous epochs. We trained the model for 10 epochs and plotted the accuracy and loss for our trained model, shown in Fig. 6. Most of the puzzles from the "Hard" category took the highest time to reach the solved state.

From Fig. 6, we can deduce that the accuracy of the model is quite lower than expected, but that is most likely due to the reason that the Sudoku puzzles can be solved in multiple ways but we had only one answer per puzzle available in the dataset.

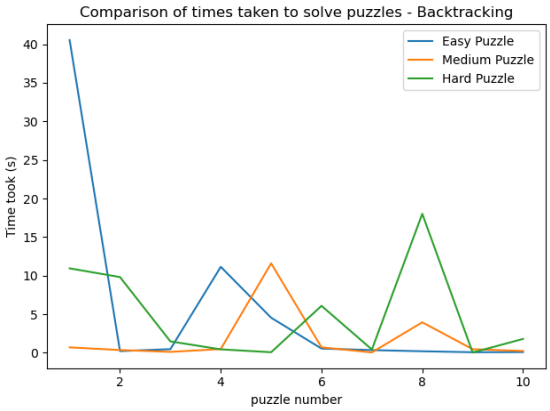


**Fig. 5.** Time Taken to Solve Each Difficulty-level of Sudoku Puzzles Using Backtracking

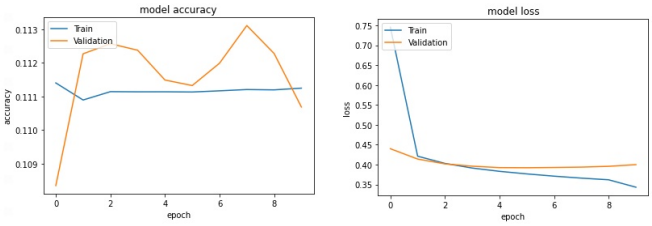


**Fig. 6.** Accuracy and Loss plot for the CNN trained model

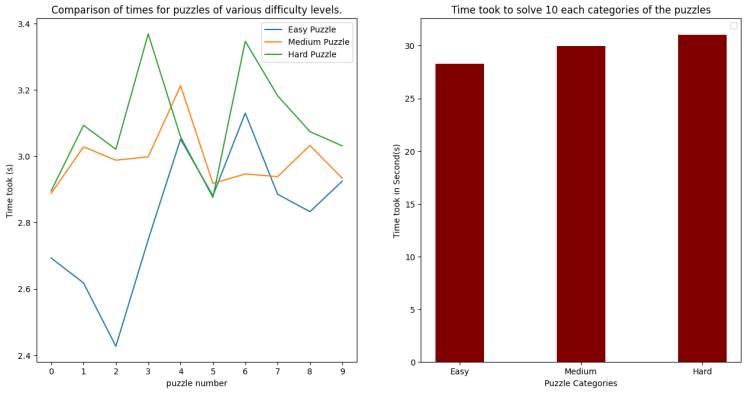


**Fig. 7.** Time took for solving each categories of the puzzles

From Fig. 7. we can observe that the CNN model took least time in solving the "Easy" category of puzzles with lesser time gap when compared to "Medium" and "Hard" puzzles.

**Constraint Satisfaction:** As mentioned earlier, Sudoku puzzle is fundamentally a constraint satisfaction problem. Using Pulp package, we defined the variables and the constraint of the given Sudoku puzzle. We then formulated the problem using linear programming. As it can be seen in Fig. 3 and Fig. 4, this method outperforms all the other algorithms insofar as the time taken to solve the puzzles is concerned. The Constraint Satisfaction agent has been able to solve any given Sudoku puzzle in the order of milliseconds. The memory usage is also relatively low compared to most of the other proposed agents. Given the fact that this agent does not require to be trained as opposed to CNN, this approach turned out to be the most efficient algorithm to be adopted to solve Sudoku puzzles.

**Genetic Algorithm:** From the Fig. 8, we see that the time of searching for puzzle solutions on three different types of puzzles shows that the time of searching results on hard puzzles and simple puzzles are quite similar, but the time on the medium puzzle is higher than others. the reason for this may come from generated random collections of solutions with regard to mutation function. And the overall performance of the Genetic algorithm Genetic is time-consuming.

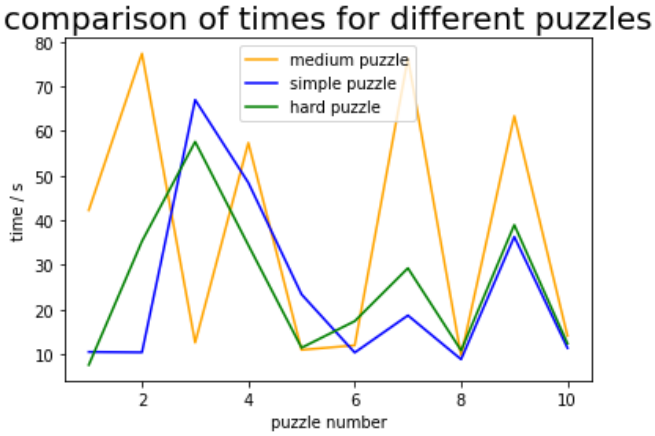


**Fig. 8.** Comparison of times of three level puzzles using Genetic Algorithm

**Comparison with Baseline:** As evident, all the algorithms performed exponentially better than the baseline algorithm of DFS. The Constraint Satisfaction

algorithm did the best in terms of time taken. The Backtracking algorithm performed the best in terms of memory usage.

## 5 Conclusions

In this paper, we investigated a set of AI algorithms to solve Sudoku puzzles. We introduced Sudoku and highlighted the essential constraints involved in the problem. We approached the problem with Backtracking as an enhanced version of classic Depth-fisrt search. We also solved Sudoku with constraint satisfaction using Pulp. We also trained a convolutional neural network using a one-million dataset. As the forth approach, we used genetic algorithm.

We evaluated the results based on the time taken and memory used by each algorithm. We also compared the performance of each method against DFS algorithm as the baseline model. It was observed that all methods outperformed the baseline model. Constraint satisfaction solved every given puzzle in milliseconds and turned out to be the fastest and most efficient method to solve Sudoku puzzles. Backtracking algorithm however emerged as the most miserly in terms of memory consumption.

## References

1. Simonis, H., 2005, October. Sudoku as a constraint problem. In CP Workshop on modeling and reformulating Constraint Satisfaction Problems (Vol. 12, pp. 13-27). Citeseer.

2. Gayatri, P., Performance Analysis of Various Uninformed and Informed Search Strategies on 8 Puzzle Problems-A Case Study.

3. D. NAKAMURA, "The man behind those maddening numbers," 24 May 2009. [Online]. Available: https://www.latimes.com/archives/la-xpm-2009-may-24-adna-sudoku24-story.html (Accessed: 10 April 2022).

4. D. Garisto, "Euler's 243-Year-Old 'Impossible' Puzzle Gets a Quantum Solution," 10 January 2022. [Online]. Available: https://www.quantamagazine.org/eulers-243-year-old-impossible-puzzle-gets-a-quantum-solution-20220110/ (Accessed: 10 April 2022).
5. D. Radcliffe, "3 million Sudoku puzzles with ratings," 2020. [Online]. Available: https://www.kaggle.com/datasets/radcliffe/3-million-sudoku-puzzles-with-ratings (Accessed: 10 April 2022).
6. K. Park, "1 million Sudoku games," 2017. [Online]. Available: https://www.kaggle.com/datasets/bryanpark/sudoku (Accessed: 10 April 2022).
7. robguinness, "How do I profile memory usage in Python?," 14 November 2018. [Online]. Available: https://stackoverflow.com/questions/552744/how-do-i-profile-memory-usage-in-python/53301648#53301648 (Accessed: 10 April 2022).
8. A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese, Wouter Wiggers Faculty of EECMS, University of Twente.