American University of Armenia

College of Science and Engineering

Artificial Intelligence Project

Solving Sudoku Puzzle

Team: Aram Adamyan, Hovhannes Hovhannisyan, Ararat Kazarian

Summer 2023

Table of Contents

Abstract

In the field of artificial intelligence, Sudoku presents an intriguing and difficult challenge in addition to being a fun and well-known puzzle. It has been a problem in the programming industry for a long time, and several solutions have emerged to address it as effectively as feasible. Larger puzzles continue to be difficult, despite the fact that modern techniques offer effective answers for little ones. In this paper, we have concentrated on some of the existing algorithms for solving the Sudoku puzzle as a Constraint Satisfaction Problem (CSP), a local search problem with different implementations. As a result, we have come up with an implementation that runs faster than those already existing implementations of this report due to the MRV(Minimum Remaining Value) heuristic and is guaranteed to find the solution. The purpose of this paper is to compare the results of various algorithms and implementations to figure out the best one among those.

Keywords: Sudoku, Constraint Satisfaction Problem, algorithm, implementation, MRV heuristic, algorithm, implementation, search

2. Introduction

A Sudoku problem is composed of a 9x9 grid that is split into nine 3x3 subgrids, or "boxes." The objective of the problem is to fill the grid with numbers from 1 to 9 while adhering to certain guidelines:

All of the digits from 1 to 9 must appear precisely once in each row. There must be exactly one instance of each digit from 1 to 9 in each column. From 1 to 9 must appear precisely once in each 3x3 box.

The player's job is to fill in the remaining vacant cells in a valid Sudoku problem in a fashion that satisfies the aforementioned requirements. A legitimate Sudoku puzzle begins with some cells pre-filled with numbers. Initially, the minimum number of values that can be preassigned is 17, which is considered to be hardest to solve among 9x9 sudokus. In general, 9x9 sudokus are the most famous ones, but there can be any n^2xn^2 grid, with nxn subgrids.

3. Literature Review

3.1 Hill Climbing

The heart of the algorithm lies in the generic_algorithm function. It begins by generating a population of randomly generated Sudoku puzzles, each representing a potential solution. The population size can be adjusted using the pop_size parameter and you can control the number of generations the algorithm will run using the max_generations parameter.

At each generation, the algorithm evaluates the fitness of each individual in the population by counting the number of conflicts in their Sudoku grid. A lower conflict value indicates a better solution. The algorithm then uses elitism to retain the best-performing half of the population and generate the next generation by performing hybridization and mutation operations. During hybridization, two original meshes are selected from the top half of the population and an intersection point is applied to create two subnets.

After intersecting, the algorithm feeds random mutations into some grids. Mutation involves modifying the value of certain cells while leaving the previously affected cells intact.

The process of assessing fitness, selecting parents, performing hybridization, and introducing mutations is repeated for several generations until a solution that does not conflict (Sudoku grid has been resolved) or has reached a maximum number of generations. In the end, the algorithm that returns the best-performing instance, which is the Sudoku grid with the fewest conflicts, is the solution to the puzzle.

To use the algorithm, you can feed your Sudoku grid as a 9x9 list with predefined values (0 for empty cells) and call the algorithm_genetics function with this grid. The solved Sudoku grid will be printed using print_grid and the elapsed time will also be displayed.

3.2 Simulated Annealing

For a satisfiability problem, it is sensible to equate the cost to the number of constraint violations. Each step of annealing operates by proposing a move to a new randomly chosen state. Proposed steps that decrease cost are always accepted.

Proposed steps that increase cost are taken with a high probability in the early stages of annealing when the temperature is high and with a low probability in the late stages of annealing when the temperature is low. Inspired by models from statistical physics, simulated annealing is designed to sample the state space broadly before settling down at a local minimum of the cost function.

The proposal stage randomly selects two different cells without clues. To ensure that the most troublesome cells are more likely to be chosen for swapping, we select cells non-uniformly with probability proportional to exp(i) for a cell involved in i constraint violations. Let B denote a typical board, c(B) its associated cost, and n is the current iteration index.

At temperature τ, we decide whether to accept a proposed neighboring board B by drawing a random deviation Uniformly from [0, 1]. If U satisfies $U \leq \min \{\exp([c(Bn) - c(B)] / \tau n), 1\}$, then we accept the proposed move and set $Bn+1 = B$.

Thus, the greater the increase in the number of constraint violations, the less likely the move is made to a proposed state.

(Chi & Lange, 2013)

3.3 Genetic Algorithm

Evolutionary algorithms can be seen as variants of stochastic beam search that are explicitly motivated by the metaphor of natural selection in biology: there is a population of individuals (states), in which the fittest (highest value) individuals produce offspring (successor states) that populate the next generation, a process called recombination. There are endless forms of evolutionary algorithms, varying in the following ways:

• The size of the population.

• The representation of each individual. In genetic algorithms, each individual is a string over a finite alphabet (often a Boolean string), just as DNA is a string over the alphabet

ACGT. In evolution strategies, an individual is a sequence of real numbers, and in genetic programming, an individual is a computer program.

• The mixing number, ρ, which is the number of parents that come together to form offspring. The most common case is $\rho = 2$: two parents combine their “genes” (parts of their representation) to form offspring. When $\rho = 1$ we have stochastic beam search (which can be seen as asexual reproduction). It is possible to have $\rho > 2$, which occurs only rarely in nature but is easy enough to simulate on computers.

• The selection process for selecting the individuals who will become the parents of the next generation: One possibility is to select from all individuals with probability proportional to their fitness score. Another possibility is to randomly select n individuals ($n > \rho$), and then select the ρ most fit ones as parents.

• The recombination procedure. One common approach (assuming $\rho = 2$), is to randomly select a crossover point to split each of the parent strings, and recombine the parts to form two children, one with the first part of parent 1 and the second part of parent 2; the other with the second part of parent 1 and the first part of parent 2.

• The mutation rate, which determines how often offspring have random mutations to their representation. Once an offspring has been generated, every bit in its composition is flipped with probability equal to the mutation rate.

• The makeup of the next generation. This can be just the newly formed offspring, or it can include a few top-scoring parents from the previous generation (a practice called elitism, which guarantees that overall fitness will never decrease over time). The practice of culling, in which all individuals below a given threshold are discarded, can lead to a speedup (Baum et al., 1995).

3.4 Backtracking Search

Rollback systematically extends a partial solution until it becomes a complete solution or violates a constraint. In the latter case, it falls back to the next allowed partial solution and starts the growth process again. The advantage of backtracking is that a wide range of potential solutions can be rejected in bulk. Backtracking begins by building for each empty Sudoku square (i, j) a Lij list of compatible digits. This is done by scanning rows, columns, and subgrids of cells. The empty cells are then sorted by the number of |Lij | lists.

Backtracking is also known as deep search. In this implementation, the strings are treated as the nodes of the tree, as shown in Figure 2.1. Creating lexicographically ordered strings constitutes a tree traversal that systematically removes subtrees and moves up and down along branches. Since pruning large subtrees is more efficient than pruning small ones, sorting the plots by number forces the decision tree to have fewer branches at the top. We use C code from Skiena and Revilla implements backtracking on the Sudoku grid. Backtracking has the advantage of finding all solutions when several solutions exist.

4. Methods

1. Hill Climbing

The algorithm starts by generating a random initial Sudoku grid with some cells filled in and others empty (represented by 0). Then it iterates, trying to improve the solution at each step. In each iteration, the algorithm uses the Hill Climb technique, which involves evaluating the "energy " or " cost" of the current Sudoku grid. The energy here is defined as the number of filled cells in the grid. The lower the energy, the closer the grid is to a valid solution. To improve the solution, the algorithm generates a neighborhood grid by randomly selecting an empty cell and assigning it a valid number. A valid number is one that does not violate Sudoku constraints in the current row, column, or 3x3 box. The selection of the next move is done using the Monte Carlo method, where certain moves are probabilistically favored based on their impact on the energy (cost) of the network. If the new configuration has a lower energy (reducing the number of filled cells) than the current grid, the algorithm will accept the move and continue. On the other hand, motion is still accepted with some probability based on energy differences and temperature parameters. This probabilistic acceptance allows the algorithm to escape the local optimization and explore different paths in the solution space. The iterations continue until a valid solution is found (i.e. all cells are filled and constraints are satisfied for each row, column, and 3x3 box) or until the maximum number of iterations is reached. Multi. The latter prevents the algorithm from running indefinitely if it does not find a valid solution within a reasonable amount of time.

2. Simulated Annealing

Simulated Annealing is a version of hill climbing that has additional parameters initial temperature and cooling rate, which can help not to get stuck in the local extremum(minimum in this case) points. Increasing the initial temperature allows the function to increase and evade local minimums. On the other hand, it makes the convergence slower than regular hill climbing.

As to the cooling rate, the higher it is, the faster the convergence, and the lower the probability of getting stuck into local minimums. By trying different values for those parameters, we have chosen the initial temperature to be 1, and the cooling rate to be 0.995 to take better results from this trade-off.

3. Genetic algorithm

It begins by generating a population of randomly generated Sudoku puzzles, each representing a potential solution. The population size can be adjusted using the pop_size parameter and you can control the number of generations the algorithm will run using the max_generations parameter. At each generation, the algorithm evaluates the fitness of each individual in the population by counting the number of conflicts in their Sudoku grid. A lower conflict value indicates a better solution. The algorithm then uses elitism to retain the best-performing half of the population and generate the next generation by performing hybridization and mutation operations. During hybridization, two original meshes are selected from the top half of the population and an intersection point is applied to create two subnets. After intersecting, the algorithm feeds random mutations into some grids. The process of assessing fitness, selecting parents, performing hybridization, and introducing mutations is repeated for several generations until a solution that does not conflict (Sudoku grid has been resolved) or is reached. In the end, the algorithm that returns the best-performing instance, which is the Sudoku grid with the fewest conflicts, is the solution to the puzzle. To use the algorithm, you can feed your Sudoku grid as a 9x9 list with predefined values (0 for empty cells) and call the algorithm_genetics function with this grid.

4. Backtracking search(regular)

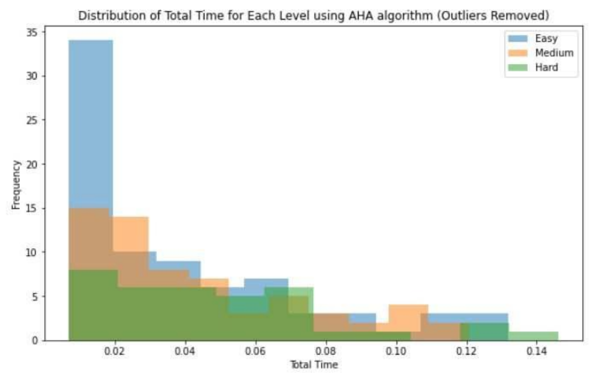
Using Backtracking, we follow a step-by-step procedure to solve the Sudoku puzzle. We search the grid for an empty cell first. When a cell is empty, we attempt to fill it with a digit from 1 to 9. We do a safety check beforehand to make sure the transfer is authorized. We check that the selected digit does not already exist in any of the related rows, columns, or 3x3 boxes. If it's a safe move, we place the digit in the cell. The procedure of attempting digits and determining their validity is then repeated in the next vacant cell. Backtracking enters the picture at this point. We go back and try a different number for the preceding cell if we find that a digit we previously attempted leads to an incorrect solution (violates the Sudoku rules). When we run out of options or come across a valid digit, we go backward once again.

5. AHA

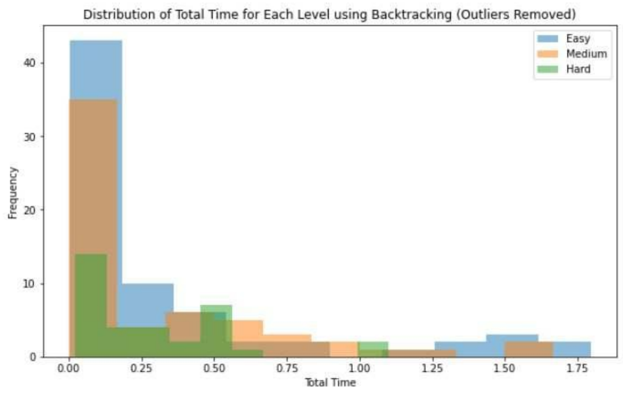
AHA is nothing more than a backtracking algorithm with a Minimum Remaining Value (MRV) heuristic value. The worst-case complexities of regular backtracking search and AHA are exactly the same, as in the worst case, both of the algorithms have to search throughout the entire tree. However, using the MRV heuristic can help to save a significant amount of time on average, as the assignment process starts from the cell, with the smallest possible assignments. It decreases the branching factor, which leads to failure earlier, in case the chosen value is not the correct one.

5. Evaluation

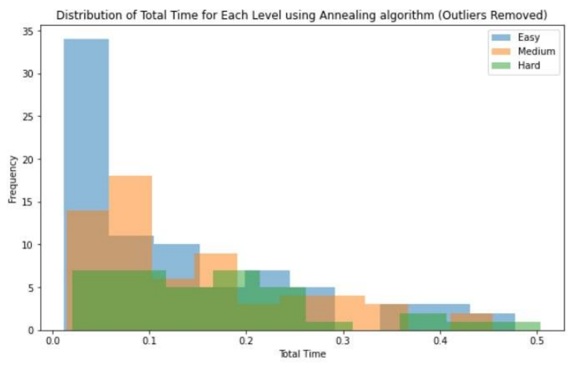
As mentioned earlier five distinct algorithms were implemented and used in the scope of this project: the AHA Algorithm (custom), the Backtracking Algorithm, the Genetic Algorithm, the Simulated Annealing Algorithm, and the Hill Climbing Algorithm. Each algorithm was assessed based on its efficiency, and speed in solving Sudoku puzzles.



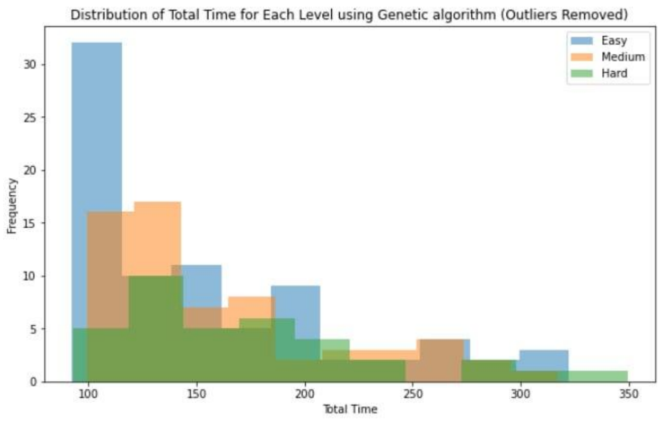
The custom AHA Algorithm demonstrated a remarkable ability to efficiently explore the solution space and quickly converge to a guaranteed correct solution, making it the top-performing choice for solving Sudoku puzzles.



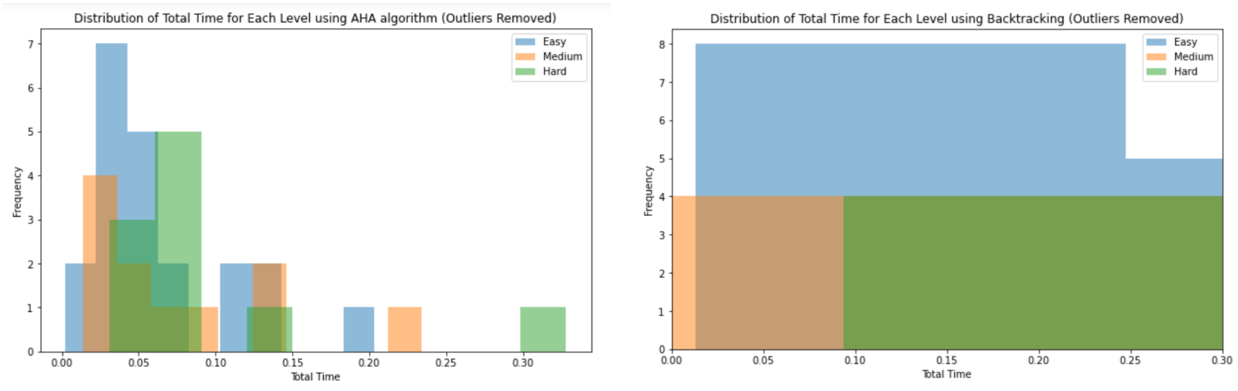
The Backtracking Algorithm proved to be the second-best option in the evaluation. As it has also a guarantee to converge, while not as fast as the custom AHA Algorithm, Backtracking effectively traversed through possible solutions and reliably found correct answers. It demonstrated solid performance, especially on moderate-level puzzles.



Simulated Annealing Algorithm and Hill Climbing Algorithm produced comparable results in the evaluation. Both algorithms showed good accuracy in solving Sudoku puzzles, but they fell short in terms of speed compared to the top-performing AHA and Backtracking algorithms, and based on the chosen parameters we can have cases where the solution is not found. However, Simulated Annealing Algorithm displayed a slight advantage over Hill Climbing, as it exhibited a bit better performance due to its capability to escape local optima and explore the solution space more effectively.



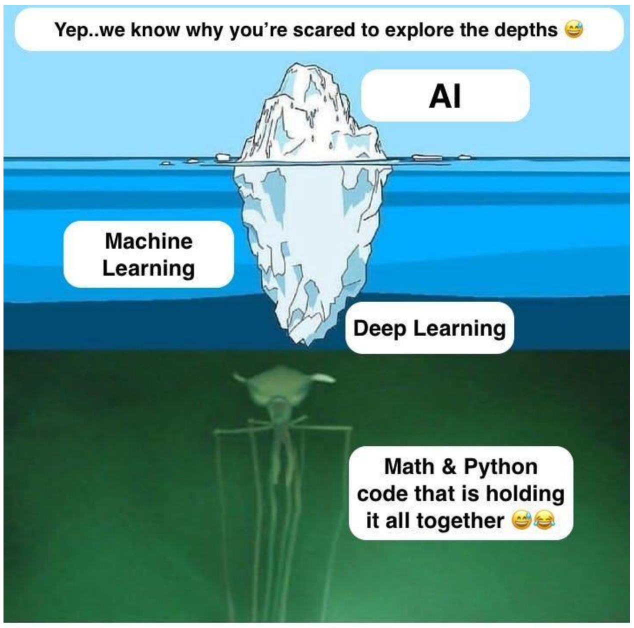
On the other hand, the **Genetic Algorithm**, while conceptually promising, proved to be the slowest among the five and most of the time couldn't converge to a solution. Its approach of using evolutionary principles to find solutions to Sudoku puzzles resulted in longer computation times and limited its ability to compete with the other algorithms. As a result, the Genetic Algorithm's speed constraints placed it last in the evaluation.



Comparison of AHA algorithm and Backtracking Search algorithm within a specific time range

6. Conclusion

As we have seen above, AHA works the fastest among all the mentioned ones and it guarantees the solution. However, to be able to make this statement for sure, we have to find the best results, which can result from simulated annealing and genetic algorithms, as there are some machine learning/deep learning algorithms, with the help of which, we may find values of the parameters to lead to better results than AHA does. Although they are not guaranteed to find the solution, on average they can run significantly faster and find the solution in almost any case.


Yep..we know why you're scared to explore the depths 😅
AI
Machine Learning
Deep Learning
Math & Python code that is holding it all together 😅😂

7. Bibliography

Chi, E. C., & Lange, K. (2013, May 16). *TECHNIQUES FOR SOLVING SUDOKU PUZZLES*. Retrieved November 14, 2022, from https://arxiv.org/pdf/1203.2295.pdf

Chi, E. C., & Lange, K. (2013, May 16). *TECHNIQUES FOR SOLVING SUDOKU PUZZLES*. Retrieved November 14, 2022, from https://arxiv.org/pdf/1203.2295.pdf

Innovative Techniques and Applications of Artificial Intelligence. pp. 3749. 2007. Lewis, Rhyd. “Metaheuristics can solve sudoku puzzles.” Journal of Heuristics. Vol 13, (2007): 387401. 06 April 2009.

Chang, M. W. (2021). *Artificial Intelligence A Modern Approach* (4th ed., p. 134). Y Pearson Education.

C. CHI, E., & LANGE, K. (2013). TECHNIQUES FOR SOLVING SUDOKU PUZZLES. 2-3. 1203.2295.pdf (arxiv.org)