Analysis and Solving of a Sudoku using Constraint Satisfaction, Backtracking and Genetic Algorithm

Aditya Vinod Kumar, Parth Shah, Richa Sharma[1]

***Abstract*— This paper details three different implementations of solving the classic Sudoku[1] puzzle, their metrics over test cases and their merits, demerits.**

I. INTRODUCTION

The objective was to implement and compare the efficiency of different algorithms against a range of Sudoku problems differing in complexity. After the implementation of these algorithms, they were run on a set of Sudoku problems which were categorized as easy, hard and extreme. The running times of these algorithms was documented and the results follow. Bench-marking was performed on an Intel i5 dual core processor operating at 2.9GHz.

II. APPROACH

1) Constraint Satisfaction: This solution initially starts with assigning all possible values to each cell and based on constraint propagation and value elimination of the cells, reaches a solution. We start by assigning the possible values one through nine for each of the cells. This algorithm uses two key concepts in its approach, if a grid cell has only one particular value, then that value must be removed from its peers and when a unit of grid cells can have only one possible place for a value, then the value can be placed in the respective grid cell. We first begin with the pre-filled cells, and remove the value of those cells from its peers. At some point in the flow when a particular unit has only one possible place for a value, we assign that value to its grid cell and recursively eliminate the value in the other cells of that unit. In this way the constraints are propagated throughout the table of grid cells.[2]

2) Backtracking Algorithm: The backtracking algorithm uses recursion. It determines a number in a cell by arbitrary choice (usually sequentially for numbers one through nine) and then recursively fills the rest of the cells till a solution is found or there is a collision.[3] This recursion ensures backtracking.The bulk of the time in this algorithm is spent in evaluating a particular solution and checking for collisions. The complexity of this algorithm is of the order of

$$O((81 - c)!)$$

for a nine by nine problem where c is the number of positions already filled and the basic operation in evaluating a given solution.

[1]In alphabetical order as there was equal contribution. For Artificial Intelligence (UE17CS325) with Dr. Soma Dhavala, PES University

3) Genetic Algorithm: The genetic algorithm is implemented for a Sudoku in the following way. A mask is created on the problem to preserve the integrity of the pre-filled cells. Each chromosome of an individual in a generation corresponds to one possible solution of the Sudoku. The size of the generation is pre-defined. While initializing the generation, each solution is a random permutation of a vector of one through nine following which the mask is applied to it. The fitness of an individual in a generation is calculated by counting the number of collisions in each row, column and sub-grid. A lower fitness corresponds to a more correct solution. The population is sorted according to its fitness. Elitism is performed where the top ten percent of the generation moves onto the next one. From the top fifty percent of the population, two parents are chosen. Now, randomly, a row is selected and two numbers (which are not part of the mask) are selected to be swapped in both parents. The new parent with a lower fitness is added to the next generation.[4] The genetic algorithm is observed to not be a correct algorithm for a Sudoku as it may never converge upon reaching a local minima. Moreover, a mating procedure is undefined as the integrity of the original puzzle must be preserved. For completeness, the algorithm ends after a pre-defined number of generations.

III. RESULTS AND DISCUSSIONS

1) Constraint Satisfaction: This algorithm works well for easy problem scenarios but in complex scenarios, the algorithm fails to converge and ends up with a number of values for one or more grid cell and ceases to continue the constraint propagation. In such a scenario, we employ a search strategy such as the depth first search, which tries to assign a value to the grid cell with minimum number of possible values and hopes to find a solution. If the process fails, the next value is assigned and this procedure continues recursively until the first solution is found.

2) Backtracking Algorithm: For small problem sizes, ie. where the number of cells to fill are small in number, backtracking performs well.
The figure shows running times for different hard test cases. This graphic clearly shows correlation between specific problems and running time of the algorithm. This is supported by data in table two. Therefore, backtracking is not reliable in terms of running time. The backtracking algorithm was not run on extreme test cases due to large running time.

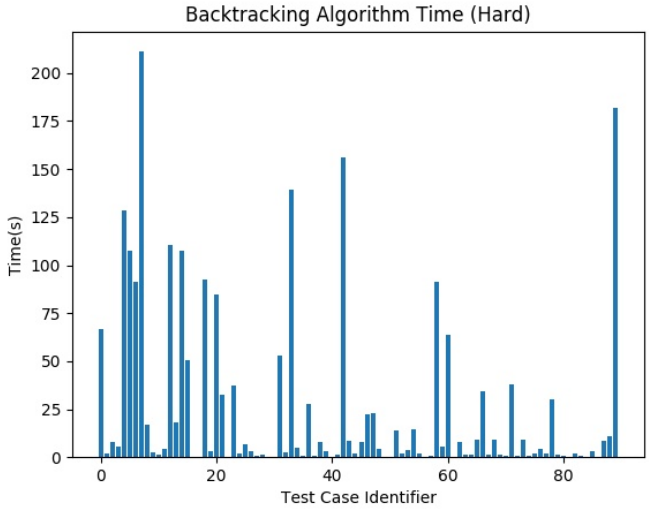

TABLE I
BENCH-MARKING PER TEST CASE IN SECONDS (EASY)

Algorithm	Mean	Variance
Constraint Satisfaction	7.89×10^{-4}	1.029×10^{-8}
Backtracking Algorithm	0.553	1.196

TABLE II
BENCH-MARKING PER TEST CASE IN SECONDS (HARD)

Algorithm	Mean	Variance
Constraint Satisfaction	8.26×10^{-4}	5.89×10^{-8}
Backtracking Algorithm	24.69	1960.03

TABLE III
BENCH-MARKING PER TEST CASE IN SECONDS (EXTREME)

Algorithm	Mean	Variance
Constraint Satisfaction	8.39×10^{-4}	1.09×10^{-8}

IV. SUMMARY

The constraint satisfaction algorithm performs the best for test cases of varying difficulties. The backtracking algorithm does not solve certain test cases in finite time. The genetic algorithm does not converge if it reaches a local minima.

1) Constraint Satisfaction: We find that for most hard problems, employing constraint propagation along with a search strategy delivers the best results. This approach is the best artificially intelligent solution closest to a human solving. Improvements can be made to this by employing a different search strategy which considers a penalty per branch taken to generate the fastest solution.

2) Backtracking Algorithm: The backtracking algorithm is essentially a brute force approach at solving the Sudoku puzzle. The worst case can be generally avoided by randomly picking the next number for a specific position. Optimizations can be made by pre-processing the grid to assign weights to each number at each position according to their frequencies and picking numbers with higher weights for a specific position first.

3) Genetic Algorithm: As discussed, the genetic algorithm is not a correct or complete algorithm as it is a variation of the non-deterministic Monte Carlo algorithm. However, another approach could be to arrive at a partial solution and then solve using one of the above mentioned algorithms. Improvements to convergence can be made by introducing a method of pencil marking.[5]

REFERENCES

[1] Sudoku is a logic based, combinatorial number placement puzzle. More details at this **link**.
[2] Constraint Propagation for Sudoku is discussed at this **link**. The author has explained the design of the problem solving approach extensively.
[3] The backtracking algorithm is discussed at this **link**. A sample impementation is provided as well.
[4] This process isn't mating as two genetically modified parents are contesting to be a part of the next generation.
[5] Pencil marking is discussed in the report at the following **link**. Some techniques were borrowed from the same with respect to the genetic algorithm modelling and hyper-parameter tuning.