An Evolutionary Approach to solve Sudoku Problems

Sanjeev Shrestha
University of Idaho
shre6177@vandals.uidaho.edu

ABSTRACT

Genetic Algorithm is a branch of evolutionary computation that deals with solving a computational problem by means of natural operators such as natural selection, mutation and crossover. Sudoku is a combinatorial number-placement puzzle in which we try to fill a 9 X 9 grid in such a way that each digit from 1 to 9 appears exactly once in the entire row, column and the 3 X 3 sub grid. Some of the numbers are typically given in the Sudoku puzzle. Apart from them, some other numbers can be correctly predetermined using placement techniques known as 'Hidden' Singles and 'Naked' Singles. These techniques are used to make the computational space smaller. In this paper, I describe an approach to solve the remaining Sudoku problem using the Genetic Algorithm. The selection used is tournament selection, uniform crossover is used between grids and mutation used is random mutation inside the sub-grid. Fitness function is evaluated on the basis of conflicts for a cell in the row and conflicts. Even though the GA would run into local optimum at times, the run would converge and sometimes give you with global optimum. Hence, the fitness function and other genetic operators were effective in finding a solution.

Keywords

Genetic Algorithm, Tournament Selection, Uniform crossover, Swap Mutation, Fitness function, Sudoku

1. INTRODUCTION

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. Genetic Algorithm is typically comprised of a number of individuals collectively called a population. The generated individuals are then subjected to some selection mechanism on the basis of their fitness value. Evolution may take place in two ways. 'Generational Model' is an approach in which typically discard the whole old population (may be select a few best individuals using elitism) and generate a new population on the basis of old population. Another approach is to use the 'Steady State Model' in which the new generated individuals are appended and the worst individuals are removed to make a constant population [1].

Genetic Algorithm is basically characterized by its operators such as selection, fitness function, mutation and crossover. Selection process comprises of selecting best or worst individual from the population based on their fitness value. The fitness function gives you a measure of how good the individual actually performs for a particular problem. Mutation is a 'small' change in the gene of an individual. Crossover deals with the interchange of information from one individual to another i.e. the parents exchange their genetic information to result in offspring.

Table 1. Description of the EA for the Sudoku Solver

Algorithm	Steady State Model
Population size	100
Representation	Permutation from digits 1-9 in each 3 X 3 Sub grid
Selection method	Tournament selection (having tournament size 6)
Board Size	A 9 X 9 cell array filled with digits 1-9 and one corresponding cell Type = {VALUE, PREDETERMINED, GIVEN}
Crossover method	Uniform Crossover
Crossover rate	20% in between two grids of two individuals
Mutation method	Sub grid Mutation (type of swap mutation for entire 3 X 3 sub grid)
Mutation rate	10% for each sub grid
Fitness function	• Fitness = (45 – sum of elements in row) + (45 - sum of elements in column) + # of conflicts in row and column
Stop Condition	If solution has been found (fitness = 0) or Up to 10000 MAX_ITERATIONS

Sudoku is a classical logic-based, combinatorial number placement problem in which we have to fill a 9 X 9 grid with digits from 1-9 where the digits are exactly repeated once in each row, column and the 3 X 3 sub-grid [3]. The problem is partially filled with some numbers and each problem has one unique solution. Sudoku is often regarded as a constraint-satisfaction problem as well due to its constraint rules in row, column and sub grid [2].

Sudoku problems may have different difficulty standards such as easy, medium, hard & difficult. However, some methods do exist for solving the Sudoku problem up to a certain point. The methods used for predetermining some numbers of the puzzle are given by techniques such as 'Naked Singles' and 'Hidden Singles' method [4].

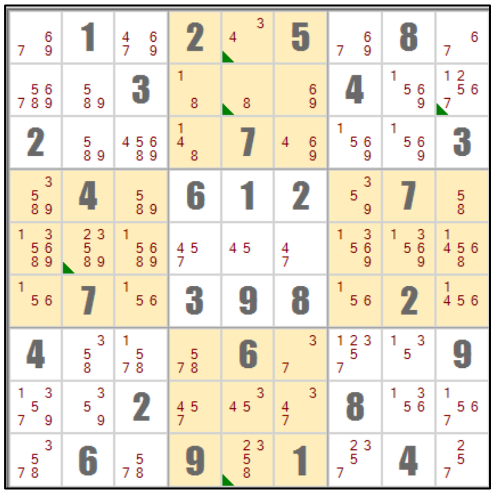


Figure 1. An initial Sudoku Puzzle, containing 28 given points

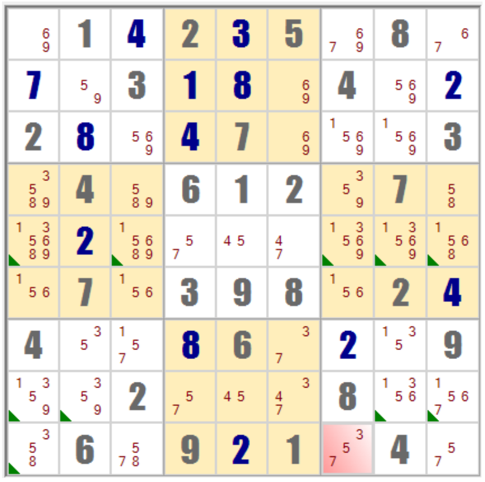


Figure 2. Sudoku Puzzle filled with 41 predetermined points after applying 'Hidden' Singles and 'Naked' Singles techniques

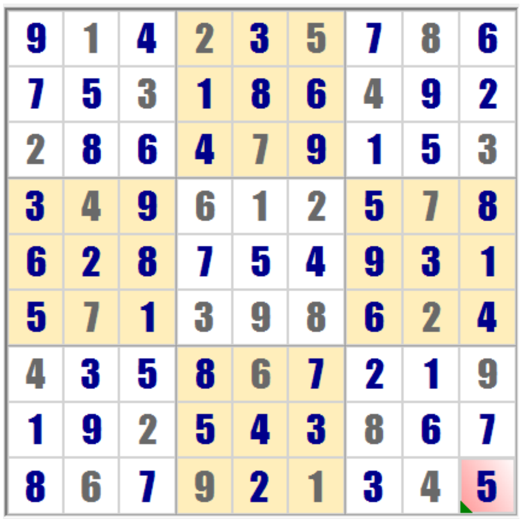


Figure 3. Solution to the Sudoku Problem in Figure 1

The primary motivation for this experiment was to test how the Sudoku problem could be solved using a Genetic Algorithm and defining a good fitness function which completely solved Sudoku problem irrespective of time constraints. The technique to solve the initial Sudoku problem was proposed in [4]. This technique has been implemented in this project as well. Mantere et al. proposed a fitness function having sum of each row and each column as 45 [3]. The fitness function has been extended to add the number of conflicts in row or column for a particular cell if duplicate results are detected.

In this paper, Section II presents a brief description of the experimental setup. Section III deals with what the results of the experiment were. Lastly, Section IV gives the overall conclusion of the paper.

2. Experiment Description

A genetic algorithm was programmed in Java to solve the Sudoku problem using a genetic algorithm. The initial problem configuration was read from a file which contained the initial given Sudoku numbers. A 9 X 9 2-D array of Cells was constructed to store the board configuration. The board was particularly made up of 'Cell' elements. The Cell element was composed of a CellValue (an integer value) for storing the given number from the Sudoku mapping file and a CellType (enum) which described what kind of cell it was, i.e., {GIVEN, PREDETERMINED, VALUE}.

This board was subjected for calculation of 'Naked' Singles and 'Hidden' Singles for 50 iterations. Any values found during this operation were of cell type 'PREDETERMINED'.

A short pseudo code of the algorithm is presented as follows:

```
Create a list, permute = {1, 2, 3, 4, 5, 6, 7, 8, 9}
for each cell in the board{
    if cell.type at row r, column c = = type.value{
        check & remove elements in permute that are in row
        check & remove elements in permute that are in column
        check & remove elements in permute that are in subgrid
        if(permute.size = = 1){
                fillValueAtPosition(r,c, permute.value, type.PREDETRMINED)
        }
        else{
        check & remove for Hidden Single row(r,c,permute)
        check & remove for Hidden Single column(r,c,permute)
        }
        permute = {1, 2, 3, 4, 5, 6, 7, 8, 9}
    }
}
```

The individuals for the GA were then created from the resulting board and a permutation list of {1, 2, 3, 4, 5, 6, 7, 8, 9} with value places filled with numbers that may not violate the sub grid consistency. The corresponding technique is outlined below:

```
Create a list, permute = {1, 2, 3, 4, 5, 6, 7, 8, 9}
for each cell in the board{
    if cell.type at row r, column c = = type.value{
        check & remove elements in permute that are in subgrid
        fillValueAtPosition(r,c,permute.value, type.Value)
        permute = {1, 2, 3, 4, 5, 6, 7, 8, 9}
    }
}
```

Note that, at any iteration, the sub grid consistency always holds. However, the row and column consistency may not be valid for all individuals.

The generated individuals are then subjected to natural evolutionary operations such as selection, mutation and crossover. Tournament selection was done on the individuals. 10 random individuals were selected from the population and the best fit individual was selected as parent. Uniform crossover was done for each pair of parents in between the sub grid of the parents with 20% chance of crossover. Mutation was carried out in the sub grid as well. For each selected sub grid, the values were randomly placed again for cells having cell.type = 'VALUE'. The mutation rate was 20 % for each sub grid.

For the first part of the experiment, the following fitness function was chosen [3].

Fitness function, f(x) = (45 – sum of values for entire row for each row) + (45 – sum of values for entire column for each column) **… (a)**

The experiment was conducted by integrating other terms from [3]. However, they did not prove to be very effective. Hence, after some error the fitness function was rectified as below:

Fitness function, f(x) = (45 – sum of values for entire row for each row) + (45 – sum of values for entire column for each column) + (number of conflicts in a row + number of conflicts in a column) **… (b)**

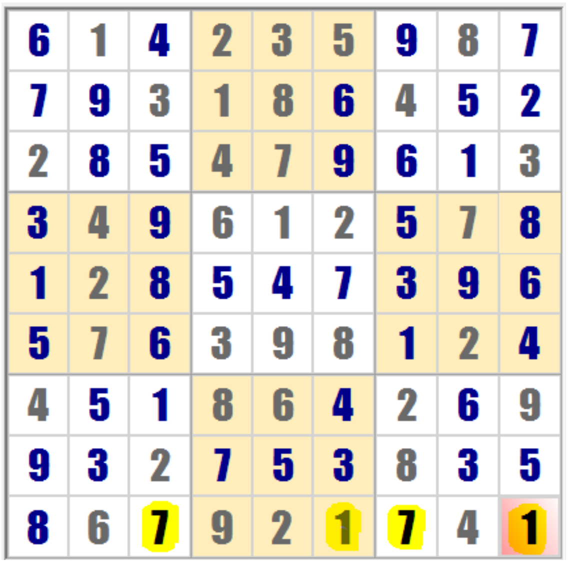
The population was evolved for 10,000 iterations or stopped if a solution was found early during the run.

3. Results

The results were collected over a number of 25 trials. High mutation rate had a very negative effect on the convergence of the population. The GA got stuck at local optima at times and only during some lucky trials would it be able to find the correct solution.

The main intriguing part of the project for me was the fitness function. For fitness function (a) the idea was the individual with fitness value '0' would satisfy all the constraints. However, the resulting individual was of the following format as in figure 4.

The sum of all the rows and the columns equate to a perfect 45 for each row and each column. The evolutionary algorithm thus cleverly, found a different 'solution' with all numbers summing up to give the perfect 'wrong' solution (different than desired one).



6	1	4	2	3	5	9	8	7
7	9	3	1	8	6	4	5	2
2	8	5	4	7	9	6	1	3
3	4	9	6	1	2	5	7	8
1	2	8	5	4	7	3	9	6
5	7	6	3	9	8	1	2	4
4	5	1	8	6	4	2	6	9
9	3	2	7	5	3	8	3	5
8	6	7	9	2	1	7	4	1

Figure 4. Invalid Solution with fitness funtion (a)

Adding the other terms from fitness function of [3] did not prove so 'effective' as well. I think this was mainly due to the improper scaling between different terms. Fitness function (b) proved to be a much better alternative. However, convergence to local optimum could not be avoided at times.

Another feature that I found interesting in Sudoku was when I assigned certain numbers to just one of the partially solved puzzles as in (2) the puzzle seemed to be completely solved using discussed "predetermination" techniques. Hence, simultaneously solving the puzzle using GA and "predetermination" techniques at each step might prove to be effective.

The results below are from the best run of the evolutionary algorithm.

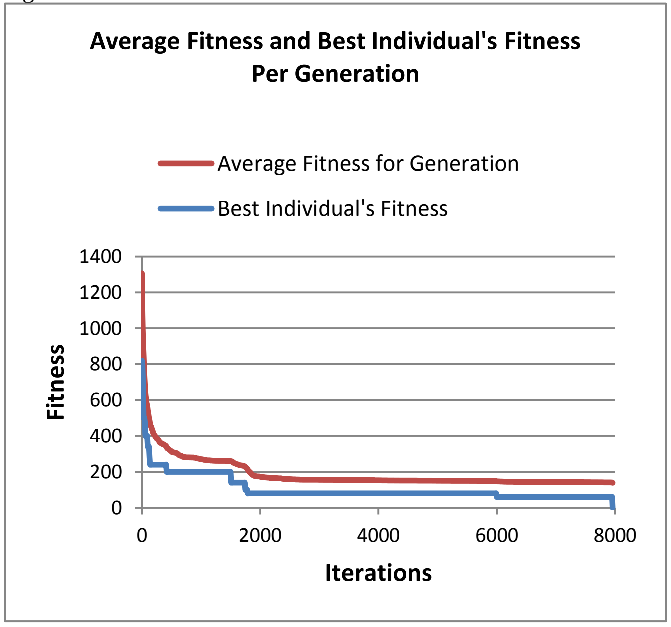


Figure 5. Graph showing average fitness and best fitness evolving over time for solved puzzle

4. Conclusion

In conclusion, the Sudoku problem was solved using a Genetic Algorithm. Even though the GA got stuck at local optima, it successfully solved at some ‘lucky’ iterations. The GA performed very well in terms of speed and convergence rate was very high during preliminary runs. After a while the convergence rate slowed down uniformly as shown in the results section. It was analyzed that high mutation rate was very harmful for the working of the GA for the Sudoku problem. It was also found that the task of choosing a ‘good’ fitness function was vital for the working of the GA.

Some future work that may be pursued for this problem could be to use a database to keep a track of all the permutations of a grid and generate individuals incrementally and by avoiding generation of same individuals by using a table of all generated individuals. The incremental generation refers to generation of each block (comprising of 3 sub grids) so that they may satisfy the row constraint and sub grid constraint. Then a combination all the three blocks would probably give a better solution. Using the table to store all the generated individuals will be used to verify the notion proposed in class for ‘No Free Lunch Theorem’.

In conclusion, the Sudoku problem was successfully solved using the Genetic Algorithm using the new fitness function and random sub grid mutation strategy.

5. REFERENCES

[1] Eiben, Agoston E., and J. E. Smith. "Genetic Algorithms." *Introduction to evolutionary computing*. New York: Springer, 2003. 37-67. Print.

[2] Simonis, Helmut. "Sudoku as a constraint problem." *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*. Vol. 12. 2005.

[3] Mantere, Timo, and Janne Koljonen. "Solving and rating sudoku puzzles with genetic algorithms." *New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP*. 2006.

[4] Weiss, John M. "Genetic Algorithms and Sudoku." *MICS*. 2009.