**SUDOKU**

**GOAL**

Given a partially filled 9\*9 grid the goal is to assign digits from 1 to 9 to the empty cells so that every row, column, and sub grid of size 3\*3 contains exactly one instance of the digits from 1 to 9.

**IMPLEMENTATION**

The problem is solved using two Algorithms

1. BACKTRACKING
2. GENETIC ALGORITHM

**BACKTRACKING**

Backtracking is an algorithmic-technique that often makes it to possible to solve at least some large instances of difficult combinatorial problems. In this project we are implementing sudoku using backtracking. Backtracking is basically an improvement over exhaustive search. The exhaustive-search technique suggests generating all candidate solutions and then identifying the one (or the ones) with a desired property. sudoku also we can solve by using exhaustive search by generating all possible configurations of numbers from 1 to 9 to fill the empty cells and then trying every configuration one by one until the correct configuration is found. But it is computationally very intensive. This is where backtracking comes into picture. Unlike exhaustive search, they construct candidate solutions one component at a time and evaluate the partially constructed solutions. If no potential values of the remaining components can lead to a solution, the remaining components are not generated at all.

The concept behind backtracking is based on the construction of a state-space tree whose nodes reflect specific choices made for a solution's components. Each time a particular component of the solution is created it is evaluated as follows.If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining componentneed to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

Like all other Backtracking problems, we can solve Sudoku by one by one assigning numbers to empty cells. Before assigning a number, we check whether it is safe to assign. We basically check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, we assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then we try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, we return false.

**GENETIC ALGORITHM**

Following phases are considered for implementing sudoku using genetic algorithm.

* Creation of initial population:

Initial population is created by determining the legal values that each empty square can take and randomly choosing numbers from these legal values. 1000 such candidates are considered for each generation.

* Fitness function:

The fitness of a candidate solution is determined by how close it is to being the actual solution to the puzzle. In order to find the fitness of each individual the number of unique values in each column and each block is calculated (weighted sum with weight=81) and final solution will give a fitness value of 1 since every column, row and block contains unique values.

* Selection:

Out of 1000 candidates in each population 50 fittest candidates are preserved for next generation and remaining candidates are created by crossover.

* Crossover:

Two new child candidates are created by crossing over parent genes using cycle crossover method. The crossover function requires two parents to be selected from population pool. Tournament method is used to select parents.

* Mutation:

Mutation is performed by picking a row and then picking two values from that row to swap.

The algorithm will terminate either when fitness of a individual becomes 1 or number of generations becomes 10,000.

**TIME COMPLEXITY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of the game/Algorithm | Easy | Medium | Hard | Expert |
| Backtracking | 369701 ns | 543228 ns | 15824717 ns | 25861594 ns |
| Genetic Algorithm | 94206084 ns | 613529582 ns | 1345779906 ns | 14432111474 ns |

From the above table it is clear that the time taken by the backtracking algorithm is very less compared to the genetic algorithm for all categories of the game. This is mainly because genetic algorithm involves creation of 1000 candidates for each population, fitness calculation for each candidate and 10,000 such generations are considered for the termination of the algorithm which makes the algorithm inefficient in terms of time.

**SPACE COMPLEXITY**

Backtracking algorithm is using a recursion stack which is N\*N step deep where N=9. Hence space complexity is O(N\*N).

In Genetic Algorithm space complexity is O(N) where N represents the number of candidates in the population.

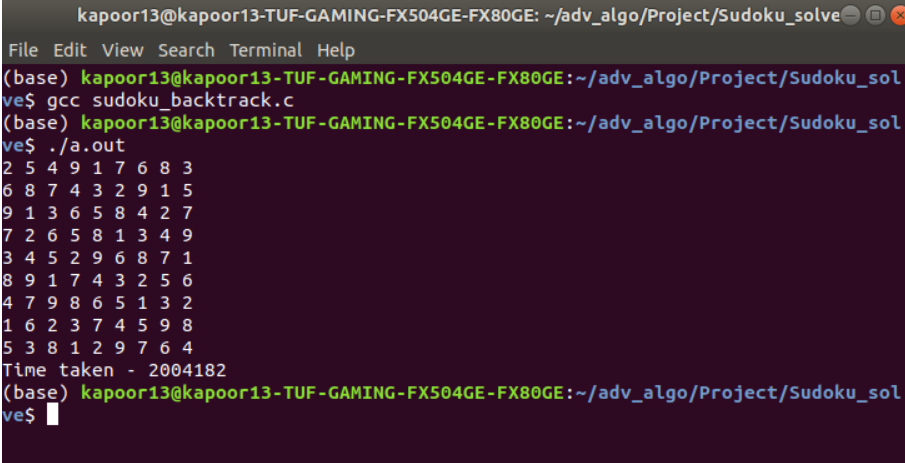
**CONCLUSIONS**

For solving sudoku Backtracking is a better algorithm when compared to genetic algorithm in terms of time and space.

Genetic algorithm being a probabilistic and stochastic algorithm convergence may not happen for some input.

**RESULTS**

Backtracking:



Genetic algorithm:

