Project documentation

Project idea

Sudoku problem:

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", or "regions") contain all the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

At the beginning of game the grid only contains some of the „start values" - selected cells from 9 by 9 matrix. These are then used to fulfill remaining empty cells. Also set of start squares is determined in such way, that it provides a unique solution to the Sudoku. Rules of this game are very simple and the objective is clear. But in the elementary 9x9 puzzle version of the game there are about 6.671x1021 valid grids, and only one appropriate solution for some given cells is satisfied. Moreover the general case of the problem has been proven to be the NP-Complete [21], which makes it one of the most interesting problems, that can be solved by approximate algorithms.

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

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

The Sudoku puzzle can be solved with several algorithms we use two algorithms in our project:

1-Backtracking algorithm

2-Differential evolution

❖ First Algorithm| Backtracking

- Main functionalities:

1) Empty_place
2) Valid_number
3) Solve_Sudoku

- Details of Algorithm:

1 - Pseudocode:

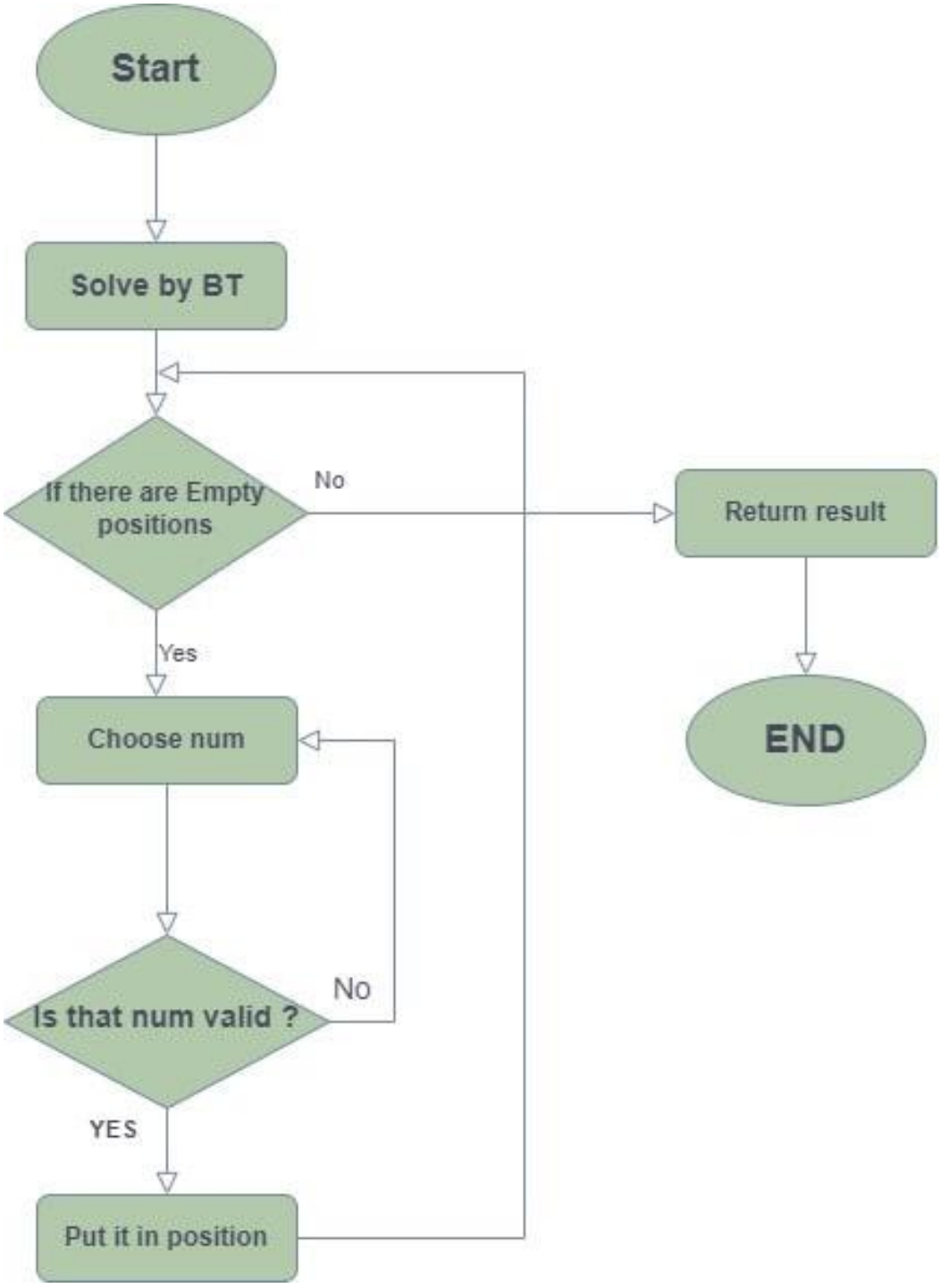
1. Start
2. We check empty places in the board
3. We start a loop to guess a number
4. For checking whether if this is a valid number or not , we check if there are any numbers in this row or column or grid equal this number
5. If number is not valid backtracking is done to the previous state of the board
6. Repeat the steps 3-5 until the board is filled
7. Return the board
8. Stop

2-dedails of functions:

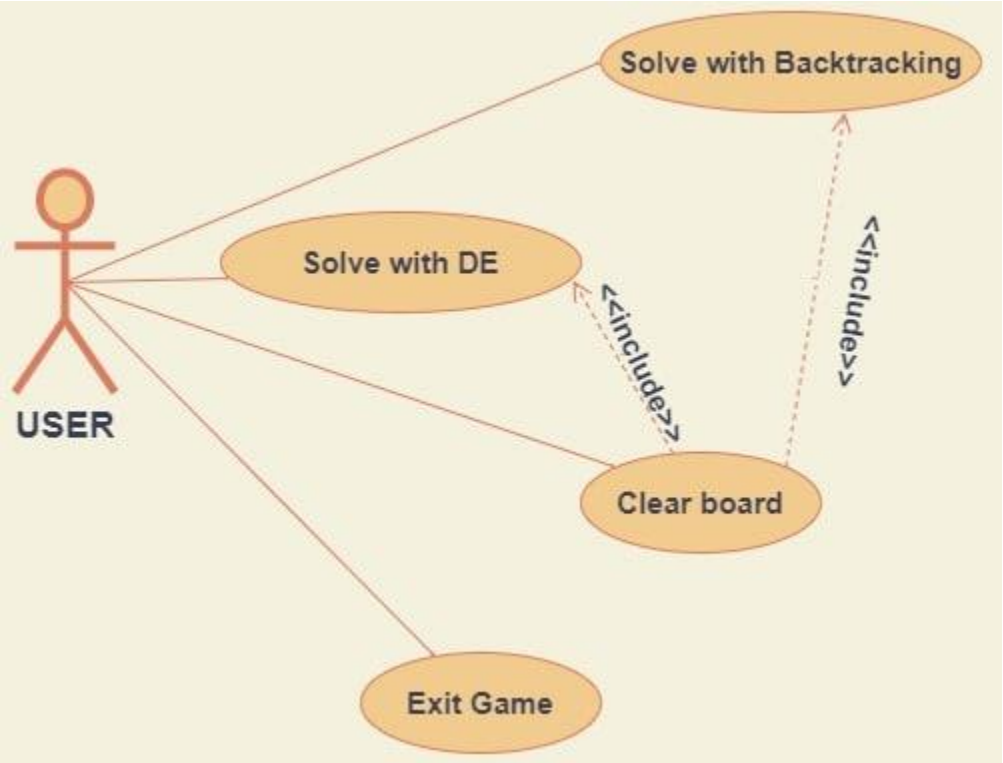
1) Empty_place:
    a) Check empty places
2) Valid_number:
    a) Check if there are any number on the same column or row or grid equal the number is guessed
3) Solve_Sudoku:
    a) Guess number and put the valid one in the empty place and backtracking if it not valid until the board is filled

3- Applied Algorithm using Diagrams:

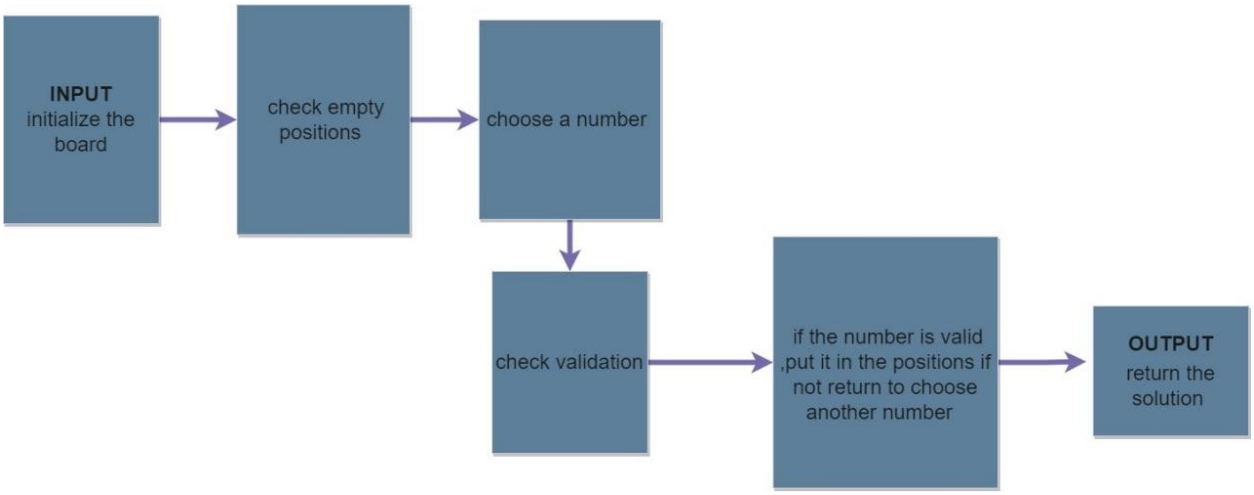
flowchart:



Usecase:



Block Diagram:



❖ Second Algorithm| Differential Evolution

• Main functionalities:

1. Is_valid
2. Fitness
3. Select_pop
4. Select_indv
5. evolve

• Details of Algorithm:

1 - Pseudocode:

1. Select population of solutions by selecting a random number and put it in the first empty place
2. Measure the fitness of each individual of population
3. If fitness equal 0 return solution
4. If not choose the individuals with highest fitness
5. Repeat step 1-3 and take random individual of population and replace them with the individuals with highest fitness.
6. choose three individuals of population randomly , calculate
   The difference between two of them and calculate the equation
   =>mutate=third ind.+.8*difference
7. Choose a random number
8. Choose another individual of population and measure fitness of it
9. If this number <CR , trial =mutate ,if number >CR ,trial =the fourth individual
10. Measure fitness of trial
11. If fitness of it<fitness of fourth indv. Take trial in next generation else take the individual.
12. Repeat steps 6-11 for all population
13. Repeat steps 5-12

2-dedails of functions:

1) Is_valid:

a)check if number is valid or not

2) fitness:

a)measure the fitness of individual

3)select_pop:

a)select population of solution

4)select_indv:

a) select random number and put it in the first empty place

b) check if number is valid in this place or not

c) if not select a random number again and repeat this step 100 times until finding valid number

5) evolve :

a) choose three individuals of population randomly

b) calculate the difference between the first two of them

c) calculate the equation =>mutate = third individual+.8*the difference

d) choose a random number and measure the fitness of it

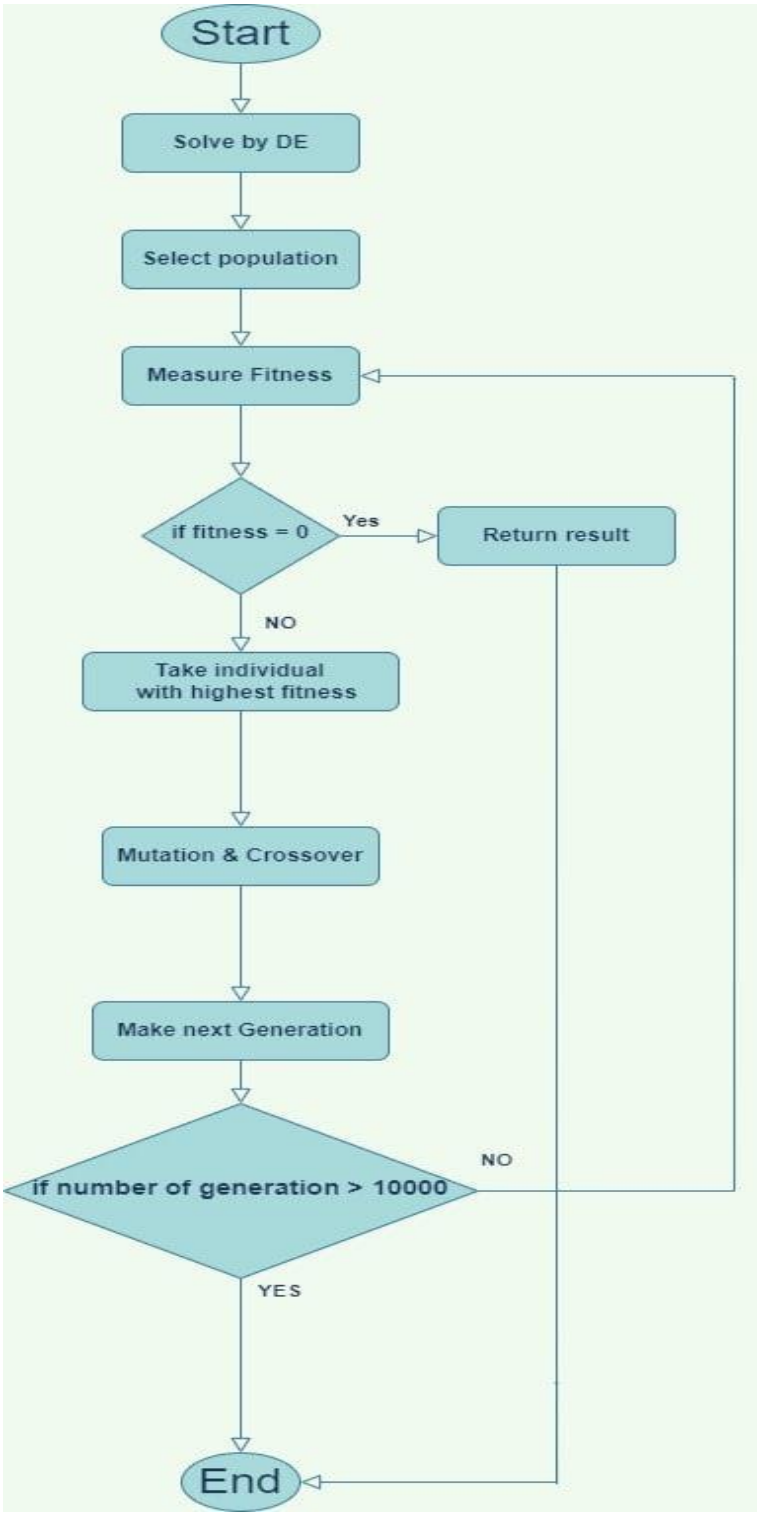
e) choose another individual from population

f) if this number < CR (0.09) , trial = mutate else, trial = 4^{th} individual

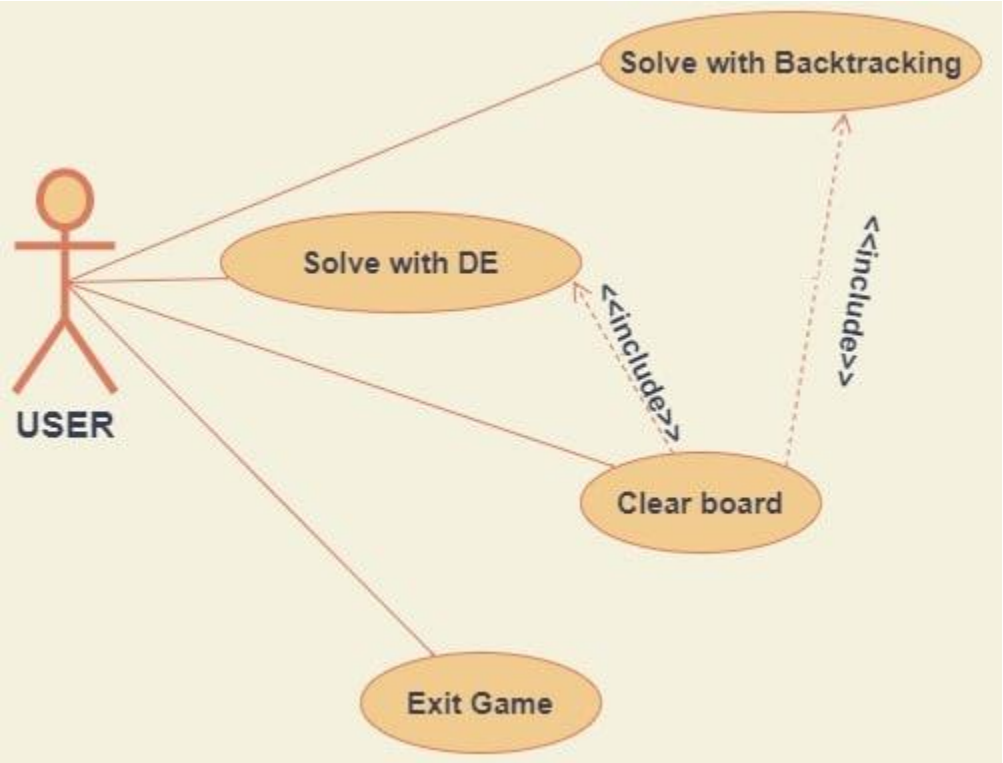
g) measure fitness of trial if fitness of trial < fitness of 4^{th} individual , take trial in next generation else take the individual

3- Applied Algorithm using Diagrams:

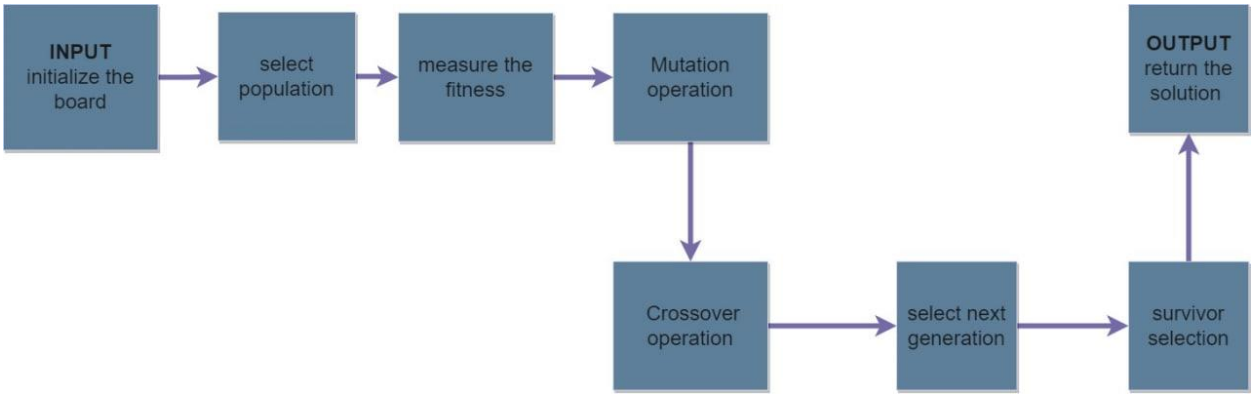
flowchart:


Start
Solve by DE
Select population
Measure Fitness
if fitness = 0
Yes
Return result
NO
Take individual
with highest fitness
Mutation & Crossover
Make next Generation
if number of generation > 10000
NO
YES
End

Usecase:



Block Diagram:



An initial literature review of Academic publications (papers):

first paper: Recursive Backtracking for Solving 9*9 Sudoku Puzzle

Research paper represents rules of the game and how to solve using backtrack algorithm that they developed to improve efficiency of the old one by implementing it through java language.
To get an idea of how back tracking algorithm performs it is suitable to plot solving times. The backtrack algorithm is an efficient algorithm in its performance.
The backtrack algorithm was tested on 30 puzzles with different number of clues in the time limit of 20 seconds and according to the experiment's results Among the 30 puzzles they have taken for doing the experiment, it have been found out the average solving time is 20.365 s.
representing it by the graph and indeed the time for solving would have continued to increase for the other unsolved puzzles.
They have also talked about the Boltzmann machine algorithm models Sudoku by using a constraint solving artificial neural network problems

second paper: SUDOKU SOLUTION WITH BACKTRACKING ALGORITHM

In that paper the researchers start representing how famous and popular this game.
first time it was created and describe how it is important to cultures and there are a lot of world-wide competitions for that game . Another objective of this study is to investigate to what extent the inclusion of distractors in the puzzle, which are not necessary to make an inference, affects the use of tactics.
The paper represents sequential algorithm and backtrack algorithm in solving sudoku .
sequential solution algorithm developed for Sudoku was compared with recursive return algorithm and performance analysis was performed.
According to the results of the study; the sequential solution algorithm has a much faster solution than the recursive return algorithm in performance. With this solution, sequential solution algorithms seem to be more efficient for Sudoku riddles.

third paper: A Comparative Study on The Performance Characteristics of Sudoku Solving Algorithms

In this paper the researcher aims to find out the fastest algorithm in terms of least time consumption in solving Sudoku by deploying randomly selected puzzles with different difficulty levels.

The study used FIVE methods (algorithms) to solve sudoku problem as Brute-force method (Backtracking), Simulated Annealing, Genetic Algorithm, Harmony Search Algorithm and Graph Referencing Algorithm.

in the experiment the algorithms are fed with a set of 30 randomly chosen Sudoku puzzles from a collection of easy, medium and hard level puzzles.

A study has been carried out to find out the solution time and based upon the least consumption in solution time the fastest algorithm is judged.

The comparative analysis has shown the performance of enumerative algorithms and non-heuristic algorithms to be the best. Though heuristic approaches like Simulated Annealing has reflected good performance, the performance is moderate comparable to that of GRA and Brute Force.

Genetic Algorithm might have found extensive application in handling other optimization problems but fails to give satisfactory results in solving Sudoku.

Harmony Search algorithms reflected serious concern on its performance and can be concluded to be the weakest attempt ever made to solve Sudoku problems.

Thus, the comparative performance characteristics study reveals that in terms of least possible time to solve Sudoku problems, GRA establishes its superiority over the others.

fourth paper: SOLVING THE SUDOKU WITH THE DIFFERENTIAL EVOLUTION

In this paper, we present the application of the Differential Evolution (DE) algorithm to solving the combinatorial problem. The advantage of the DE algorithm is its capability of avoiding so-called ,,local minima" within the considered search space. Thanks to the special operator of the adaptive mutation, it is possible to direct the searching process within the solution space. The DE algorithm applies the selection operator that selects from the child population only the offspring with the greater value of the fitness function in comparison to their parents. An algorithm applied to a combinatorial optimization problem: Sudoku puzzle is presented. Sudoku consists of a nine by nine grid, divided into nine three by three boxes. Each of the eighty-one squares should be filled in with a number between one and nine. In this article we show, that the mutation schema has significant impact on the quality of created solution.

fifth paper: ARE EVOLUTIONARY ALGORITHMS REQUIRED TO SOLVE SUDOKU PROBLEMS?

Sudoku puzzles are an excellent testbed for evolutionary algorithms. The puzzles are accessible enough to be enjoyed by people. However the more complex puzzles require thousands of iterations before a solution is found by an evolutionary algorithm. If we were attempting to
compare evolutionary algorithms we could count their iterations to solution as an indicator of relative efficiency. However all evolutionary algorithms include a process of random mutation for solution candidates. I will show that by improving the random mutation behaviours I was able to solve problems with minimal evolutionary optimisation. Experiments demonstrated the
random mutation was at times more effective at solving the harder problems than the evolutionary algorithms. This implies that the quality of random mutation may have a significant impact on the performance of evolutionary algorithms with sudoku puzzles. Additionally this random mutation may hold promise for reuse in hybrid evolutionary algorithm
behaviours.

Similar applications in the market

- ## N-Queen Problem

  -The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. The user should select the number of queens (N). The standard form is the Eight Queens Puzzle, in which one must place eight queens on a standard chessboard such that no queen is attacking any other.
  -using Backtracking algorithm to solve the problem by putting two queen and check if they can attack each other or not and if not put another queen and keep checking by backtracking.

- ## A Simple Maze

  -Simple maze to find a way out from the maze must search until find way out.
  -using backtracking algorithm by checking every possible way to find the right path to get out.
  3-Door Escaping a Maze
  -Door Escaping find a way out by checking every door if not the right one go back and try another one.
  -using backtracking algorithm by checking every possible way to find the right door to get out.

- ## Minesweeper

  click a cell
  – if bomb game over

– if cell that has 1 or more bombs on border, then reveal the number of bombs that border cell
– if a cell that has 0 bombs on border, then reveal that cell as a blank and click on the 8 surrounding cells
5-Knight's Tour
-Given a N*N board with the Knight placed on the first block of an empty board. Moving according to the rules of chess knight must visit each square exactly once. Print the order of each cell in which they are visited.

- Knapsack problem

  – Filling a knapsack. Given a choice of items with various weights and a limited carrying capacity find the optimal load out. 50 lb. knapsack. items are 1 40 lb, 1 32 lb. 2 22 lbs, 1 15 lb, 1 5 lb. A greedy algorithm would choose the 40 lb item first. Then the 5 lb. Load out = 45lb. Exhaustive search 22 + 22 + 5 = 49.

Development platform.

- Pycharm community Edition 2020.2.3
- Tkinter (Python GUI)
- Python language
- Numpy and Random libraries

Resources

- http://acsr.wi.pb.edu.pl/wp-content/uploads/zeszyty/z9/Boryczka,Juszczuk-full.pdf
- https://www.researchgate.net/publication/269239040_Are_Evolutionary_Algorithms_Required_to_Solve_Sudoku_Problems
- https://en.wikipedia.org/wiki/Sudoku_solving_algorithms