...Sudoku Solver...

... 1- project idea ...

2) A Sudoku Solver using Differential Evolution AND the Backtracking Algorithm.

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. For example, the (left) figure demonstrates a typical Sudoku puzzle, and its solution (right).

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

==========

… 2- Main Functions …

(1) Sudoku Solver using the Backtracking

1- board

It’s 2Darray = initial board of Sudoku

```
# Main board in project PDF
'''
mainBoard = [[5, 3, 0, 0, 7, 0, 0, 0, 0],
             [6, 0, 0, 1, 9, 5, 0, 0, 0],
             [0, 9, 8, 0, 0, 0, 0, 6, 0],
             [8, 0, 0, 0, 6, 0, 0, 0, 3],
             [4, 0, 0, 8, 0, 3, 0, 0, 1],
             [7, 0, 0, 0, 2, 0, 0, 0, 6],
             [0, 6, 0, 0, 0, 0, 2, 8, 0],
             [0, 0, 0, 4, 1, 9, 0, 0, 5],
             [0, 0, 0, 0, 8, 0, 0, 7, 9]]
```

-

2- CheckValidCellRow

*Aim: check can but the number in this row or not?

*take: integer number in range [1,9] , index of row and the board

*How to work: if number in this row return false (can’t but this number) else return true (can but this number)

*return: true if valid else return false

```
43
44  def CheckValidCellRow(board, number, rowIndex):
45      column = 0
46      while column < 9:
47          if board[rowIndex][column] == number:
48              return False
49          column += 1
50      return True
51
```

-

3- CheckValidCellColumn

*Aim: check can but the number in this column or not?

*take: integer number in range [1,9] , index of column and the board

*How to work: if number in this column return false(can't but this number) else return true (can but this number)

*return: true if valid else return false

```
34
35  def CheckValidCellColumn(board, number, columnIndex):
36      row = 0
37      while row < 9:
38          if board[row][columnIndex] == number:
39              return False
40          row += 1
41      return True
42
```

-

4- CheckValid3X3Cells

*Aim: check can but the number in this box or not?

*take: integer number in range [1,9] , index of column , index of row and the board

*How to work: first calculate index of start ceil in this box then check if number in this box return false(can't but this number) else return true (can but this number)

*return: true if valid else return false

```
52
53  # function --> Check If Valid Number In 3x3 Cells
54  def CheckValid3X3Cells(board, number, rowIndex, columnIndex):
55      startRow = 3 * math.floor(rowIndex / 3)
56      startColumn = 3 * math.floor(columnIndex / 3)
57
58      for i in range(startRow, startRow + 3):
59          for j in range(startColumn, startColumn + 3):
60              if board[i][j] == number:
61                  return False
62      return True
63
64
```

-

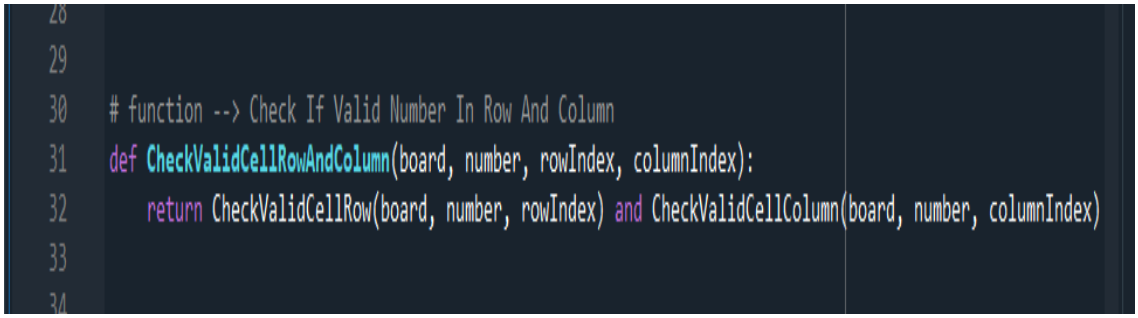
5- CheckValidCellRowAndColumn

*Aim: check can but the number in row and column or not?

*take: integer number in range [1,9] , index of column , index of row and the board

*How to work: call function CheckValidCellColumn and function CheckValidCellColumn

*return: true if valid else return false



```
28
29
30  # function --> Check If Valid Number In Row And Column
31  def CheckValidCellRowAndColumn(board, number, rowIndex, columnIndex):
32      return CheckValidCellRow(board, number, rowIndex) and CheckValidCellColumn(board, number, columnIndex)
33
34
```

6- BacktrackingSolution function

*Aim: find the solution of Sudoku

*take index of column initialize zero, index of row initialize zero and the board

*How to work: make iteration from 1 to 9 and but the numbers in the empty ceil then call function CheckValidCellRowAndColumn and function CheckValid3X3Cells to check if valid number or not then call itself again by column+1 (recursive) if column reach to 9 (end of columns) or end row return True else if it is end of column increase row by one then make column = 0

*return: return true if you find answer else return Null

```python

# function --> Backtracking Algorithm Solution
def BacktrackingSolution(board, row=0, column=0):
    if row == 8 and column == 9:
        return True
    elif column == 9:
        row += 1
        column = 0

    if board[row][column] == 0:
        for generatedNumber in range(1, 10):
            if CheckValid3X3Cells(board, generatedNumber, row, column) and \
                    CheckValidCellRowAndColumn(board, generatedNumber, row, column):
                # print(generatedNumber, '->', row, ',', column)
                board[row][column] = generatedNumber
                if BacktrackingSolution(board, row, column + 1) is None:
                    # print(0, '->', row, ',', column)
                    board[row][column] = 0
                else:
                    return True
        if board[row][column] == 0:
            return None
    else:
        return BacktrackingSolution(board, row, column + 1)
```

-

(2) Sudoku Solver using Differential Evolution

1-copyBoard

*Aim: copy from board

*take: This function take the board of Sudoku

*return: create copy from this board in 2darray (copyTo) then return it

```
64
65  #-----------------------------------------------------
66
67  def copyBoard(copyFrom):
68      copyTo = []
69      for row in range(0, len(copyFrom)):
70          copyTo.append(copyFrom[row].copy())
71      return copyTo
72
73  #-----------------------------------------------------
74
```

-

2-population

*Aim: generate population

*take: This function take the board of Sudoku and number of population you want to create

*return: list of the population

```
73  #-----------------------------------------------------
74
75  # function --> Genetic Algorithm
76  def population(startedBoard, length):
77      pop = []
78      for i in range(0, length):
79          person = copyBoard(startedBoard)
80          for row in range(0, 9):
81              for column in range(0, 9):
82                  if person[row][column] == 0:
83                      person[row][column] = int(random.uniform(1, 9))
84          pop.append(person)
85      return pop
86
87  #-----------------------------------------------------
```

-

3-fitnessRow

*Aim: measure row efficiency

*take: This function take the board of Sudoku

*return: sum of wrong ceil in each row

(wrong ceil) is : ceil was repeated more than once in one row

```
86
87  #-----------------------------------------------------
88
89  def fitnessRow(board):
90      fit = 0
91      for row in range(9):
92          setRow = set()
93          for column in range(9):
94              setRow.add(board[row][column])
95          fit += 9 - len(setRow)
96      return fit
97
98  #-----------------------------------------------------
```

-

4-fitnessColumn

*Aim: measure column efficiency

*take: This function take the board of Sudoku

*return: sum of wrong ceil in each column

(wrong ceil) is : ceil was repeated more than once in one column

```
97
98  #-----------------------------------------------------
99
100 def fitnessColumn(board):
101     fit = 0
102     for column in range(9):
103         setColumn = set()
104         for row in range(9):
105             setColumn.add(board[row][column])
106         fit += 9 - len(setColumn)
107     return fit
108
109 #-----------------------------------------------------
110
```

-

5-fitness3X3Cells

*Aim: measure one [box 3*3] efficiency

*take: This function take the board of Sudoku

And index of row ,column

*return: number of wrong ceil in the box

(wrong ceil) is : ceil was repeated more than once in a box start by startRow and startColumn

```
118
119  def fitness3X3Cells(board, rowIndex, columnIndex):
120      startRow = 3 * math.floor(rowIndex / 3)
121      startColumn = 3 * math.floor(columnIndex / 3)
122      setNumbers = set()
123      for i in range(startRow, startRow + 3):
124          for j in range(startColumn, startColumn + 3):
125              setNumbers.add(board[i][j])
126
127      return 9 - len(setNumbers)
128
```

-

6-fitness9x9

*Aim: measure all [box 3*3] efficiency

*take: This function take the board of Sudoku

it call function fitness3X3Cells and give it all boxes in board

*return: sum of wrong ceil in each box

(wrong ceil) is : ceil was repeated more than once in one box

```
#-------------------------------------------------

def fitness9x9(board):
    fit = 0
    # 0 3 6 each iteration increase by 3
    for row in range(0, 9, 3):
        for column in range(0, 9, 3):
            fit += fitness3X3Cells(board, row, column)
            #(0,0)(0,3)(0,6)
            #(3,0)(3,3)(3,6)
            #(6,0)(6,3)(6,6)
    return fit

#-------------------------------------------------
```

-

7-fitness

*Aim: measure Board efficiency

*take: This function take the board of Sudoku

it call function fitness9x9 , fitnessColumn and fitnessRow

*return: sum of wrong ceil in each box and each column and each row

-You want to minimize the fitness function

```
133
134 #------------------------------------------------
135
136 def fitness(board):
137     return fitnessRow(board) + fitnessColumn(board) + fitness9x9(board)
138
139 #------------------------------------------------
140
```

-

8-grade

*Aim: measure all population efficiency

*take: This function take the population.

it measure fitness function to each person in population and calculate the average of them

*return: Average fitness of all population

```
139 #------------------------------------------------------
140
141 def grade(pop):
142     total = [fitness(member) for member in pop]
143     return sum(total) / len(pop) * 1.0
144
145 #------------------------------------------------------
```

-

9-clamp

*Aim: make value in the range 1 to 9

*take: value (integer number)

*return: if number < 1 return 1 else if value > 9

Return 9 else value > 0 and value<=9 return value

```
145  #------------------------------------------------------
146
147  def clamp(value):
148      value = int(value)
149      return 1 if value < 1 else 9 if value > 9 else value
150
151  #------------------------------------------------------
```

-

10-makeMutantVector

*Aim: make mutation in board3 by the rate of mutateRate and generate new child

*take: three boards and the origin board and mutation rate …. it calculate the new child by Subtract board1 and board2 and sum of it board3

*note: The mutant vector is created by equation (V = b3 - W(b2 + b1)) where the W is the weight or the mutant value.

*return: new child after mutation

```
#-----------------------------------------------

def makeMutantVector(board1, board2, board3, orgBoard, mutateRate):
    mutantVector = copyBoard(orgBoard)
    for row in range(9):
        for column in range(9):
            if mutantVector[row][column] == 0:
                mutantVector[row][column] = clamp(
                    board3[row][column] + (mutateRate * (board1[row][column] - board2[row][column]))
                )
    return mutantVector

#-----------------------------------------------
```

-_____--_____

11-evolve

*Aim:

Population need to evolve to advance the next generation

*How to work:

For each individual in population do that :

1) Select three unique parent from population and work on them mutation then it return new individual (female)
2) Make crossover by crossoverRate and generate new child caused by mixing the parent(male) and individual(female) generated by mutate function
3) Selection : if fitness of the child better than the fitness of parent then Replace the parent with the child

4) If fitness function for individual equal zero (the optimal solution) then sort the population by fitness function and return it

*take: 1-population 2-original board 3- mutateRate 4- crossOverRate

*return: population after advance the next generation

```
164
165  def evolve(parents, orgBoard, mutateRate=.3, crossOverRate=.3):
166      global finalSolution
167
168      for targetMemberIndex in range(len(parents)):  # [0 -> N]    0 -> Target Board
169          # Select 3 Random Members
170          parentsLength = len(parents)
171          x1 = x2 = x3 = None
172          while x1 == x2 or x1 == x3 or x2 == x3:
173              x1 = parents[random.randint(0, parentsLength - 1)]  # 0
174              x2 = parents[random.randint(0, parentsLength - 1)]  # 1
175              x3 = parents[random.randint(0, parentsLength - 1)]  # 2
176
177          # Mutate Board
178          #return new child
179          mutantVector = makeMutantVector(x1, x2, x3, orgBoard, mutateRate)
180
181          # Crossover
182          targetBoard = copyBoard(orgBoard)
183          for row in range(9):
184              for column in range(9):
185                  if orgBoard[row][column] == 0:
186                      if crossOverRate >= random.random():# number from 0 to 1
187                          targetBoard[row][column] = mutantVector[row][column]
188                      else:
189                          targetBoard[row][column] = parents[targetMemberIndex][row][column]
190
```

```
190
191          # Selection
192          mutantVectorFitness = fitness(targetBoard)
193          targetMemberFitness = fitness(parents[targetMemberIndex])
194          if targetMemberFitness > mutantVectorFitness:
195              parents[targetMemberIndex] = targetBoard
196
197          if fitness(parents[targetMemberIndex]) == 0:
198              #sort population by fitness function
199              graded = [(fitness(member), member) for member in parents]
200              parents = [x[1] for x in sorted(graded)]
201              finalSolution = parents[targetMemberIndex]
202              return parents
203
204      return parents
205
```

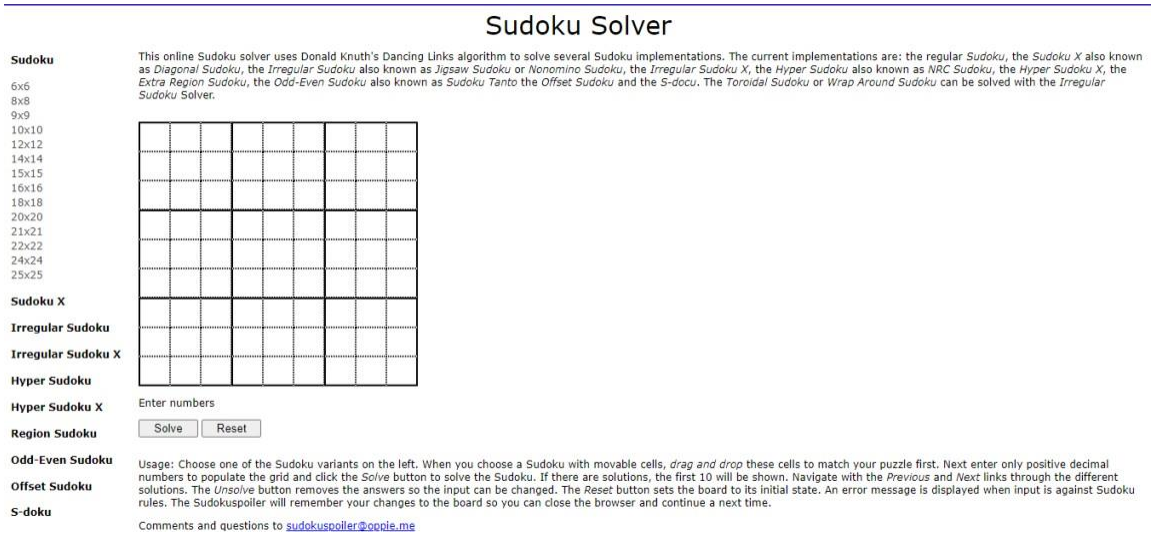
==========

3- ....Similar_applications_in_the_market.....

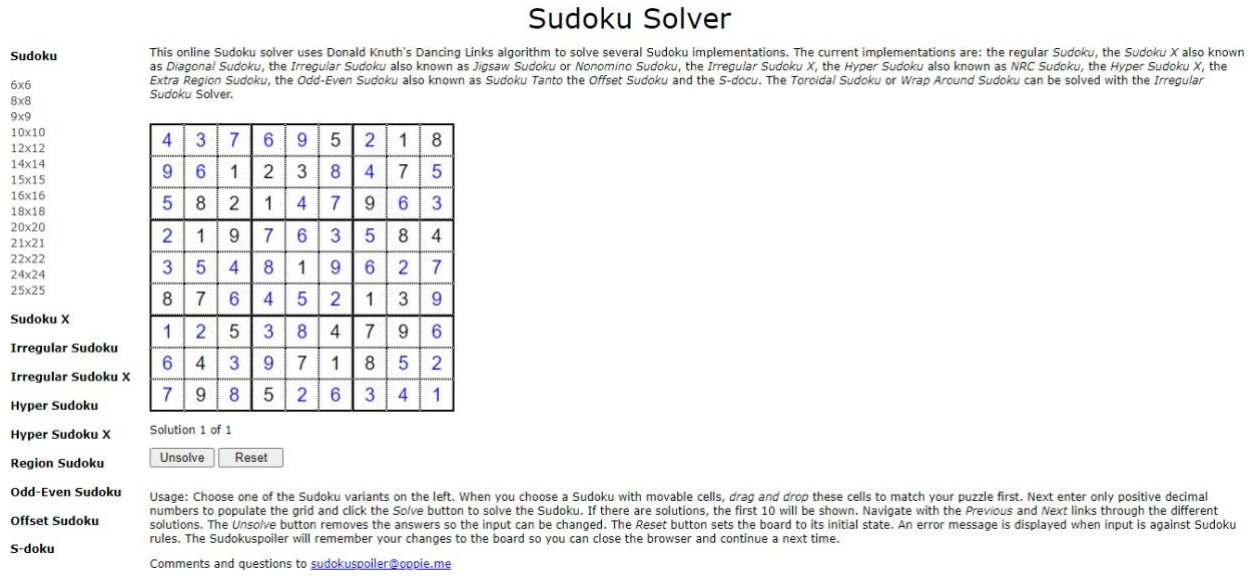
(1) ***Sudoku solver website***

Link: https://sudokuspoiler.azurewebsites.net/

This application takes any grid and solve it immediately

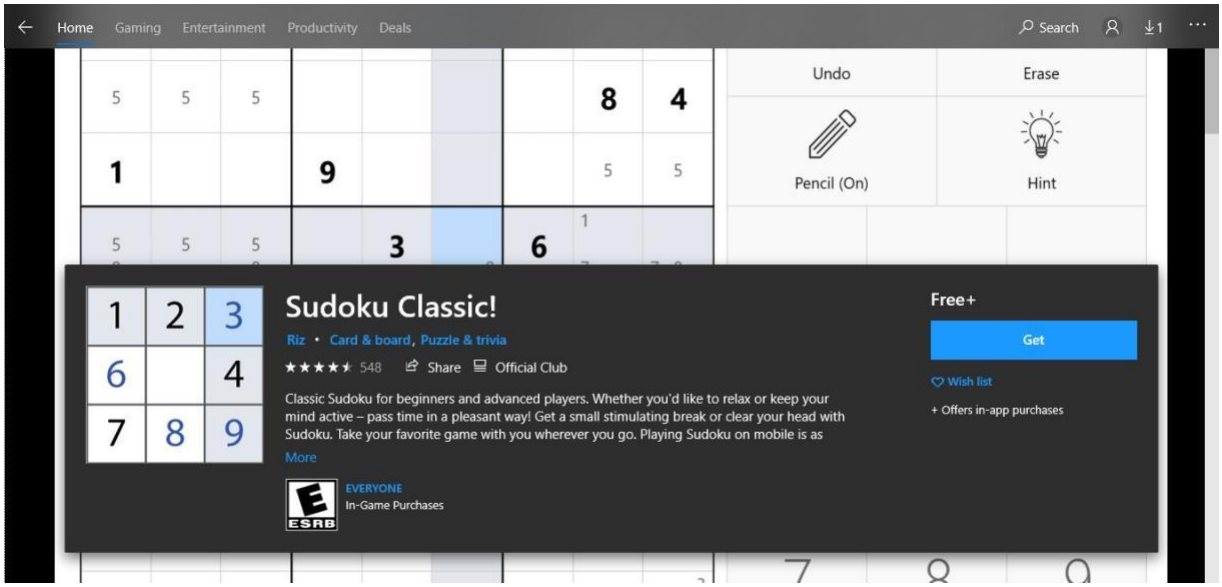


This is the website when you first open it, it gives you an empty grid to fill and provide you with the answer you need as simple as that:



(1) Sudoku Classic!

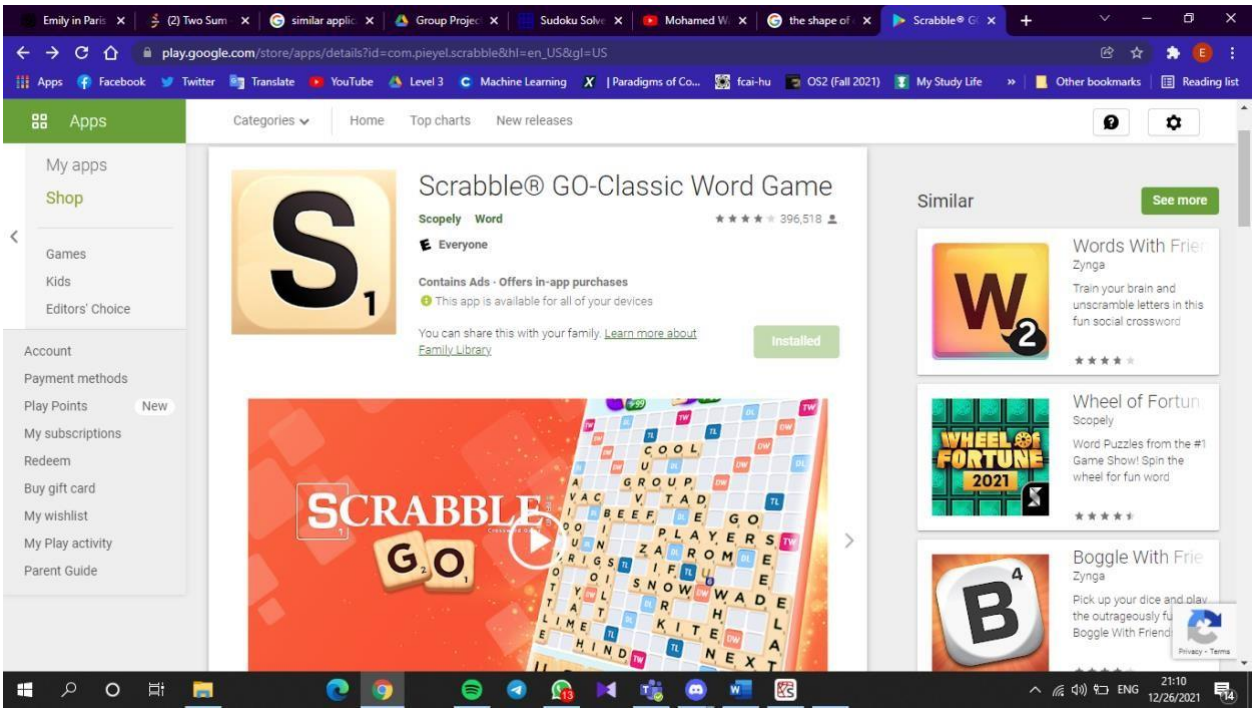
Sudoku is also available in google play and Microsoft store for people to install and play with many different levels:



I think it is implemented by backtracking because when I tried to play it, it counts the number I place in the grid as wrong number even if it satisfies all conditions but it's not in the ideal place, the program runs the grid to the end to see if it is ideal position or not and counts the move as a mistake if it's not or when it needs to backtrack.

(2) Scrabble® GO-Classic Word Game

Scrabble game also can be implemented with both backtrack and differential evolution just like our sudoku game



==========

4- ….An initial literature review of Academic publications (papers)

Research Paper Number 1

Research Paper Link : https://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group6Alexander/final/Patrik_Berggren_David_Nilsson.report.pdf

There are multiple algorithms for solving Sudoku puzzles. This report is limitedto the study of three different algorithms, each representing various solving approaches. Primarily the focus is to measure and analyze those according to their solving potential. However there are also other aspects that will be covered in this thesis. Those are difficulty rating, Sudoku puzzle generation, and how well the algorithms are suited for parallelizing. The goal of this thesis is to conclude how welleach of those algorithms performs from these aspects and how they relate to one another. Another goal is to see if any general conclusions regarding Sudoku puzzlescan be drawn. The evaluated algorithms are backtrack, rule-based and Boltzmann machines. All algorithms with their respective implementation issues are further discussed in section 2 (background).

Definition:-
Box: A 3x3 grid inside the Sudoku puzzle. It works the same as rows and columns, meaning it must contain the digits 1-9.
Region: This refers to a row, column or box.
Candidate: An empty square in a Sudoku puzzle have a certain set of numbersthat does not conflict with the row, column and box it is in. Those numbers are called candidates or candidate numbers.
Clue: A clue is defined as a number in the original Sudoku puzzle. Meaning thata Sudoku puzzle have a certain number of clues which is then used to fill in new squares. The numbers filled in by the solver is, however, not regarded as clues.

A Sudoku game consists of a 9x9 grid of numbers, where each number belong tothe range 1-9. Initially a subset of the grid is revealed and the goal is to fill the remaining grid with valid numbers. The grid is divided into 9 boxes of size 3x3.
Sudoku has only one rule and that is that all regions, that is rows, columns, and boxes, contains the numbers 1-9 exactly once.[2] In order to be regarded as a properSudoku puzzle it is also required that a unique solution exists, a property which can be determined by solving for all possible solutions.

The level of difficulty is not always easy to classify as there is no easy way of determining hardness by simply inspecting a grid. Instead the typical approachis trying to solve the puzzle in order to determine how difficult it is. A common a misconception about Sudoku is that the number of clues describes how difficult itis. While this is true for the bigger picture it is far from true that specific 17-clue puzzles

are more difficult than for instance 30-clue puzzles.[11] The difficulty of a puzzleis not only problematic as it is hard to determine, but also as it is not generally accepted how puzzles are rated. Puzzles solvable with a set of rules may be classified
as easy, and the need for some additional rules may give the puzzle moderate or advanced difficulty rating. In this study difficulty will however be defined as the solving time for a certain algorithm, meaning that higher solving times implies a more difficult puzzles. Another interesting aspect related to difficulty ratings is thatthe minimum number of clues in a proper Sudoku puzzle is 17.[2] Since puzzles generally become more difficult to solve with an decreasing number of clues, due to the weak correlation in difficulty, it is probable that some of the most difficult

The backtrack algorithm for solving Sudoku puzzles is a brute-force method. This can be viewed as guessing which numbers go where. When a dead end is reached,the algorithm backtracks to an earlier guess and tries something else. This means that the backtrack algorithm does an exhaustive search to find a solution, which means that a solution is guaranteed to be found if enough time is provided. Even thought this algorithm runs in exponential time, it is plausible to try it since it is widely thought that no polynomial time algorithms exists for NP-complete problem such as Sudoku. One way to deal with such problems is with brute-force algorithms provided that they are sufficiently fast. This method may also be used to determine
if a solution is unique for a puzzle as the algorithm can easily be modified to continue searching after finding one solution. It follows that the algorithm can be used to generate valid Sudoku puzzles (with unique solutions), which will be discussed.

There are several interesting variations of this algorithm that might prove to be more or less efficient. At every guess, a square is chosen. The most trivial method would be to take the first empty square. This might however be very inefficient sinceThere are worst case scenarios where the first squares have very many candidates.
Another approach would be to take a random square and this would avoid the above mentioned problems with worst case scenarios. There is, however, a still better approach. When dealing with search trees one generally benefits from having as few branches at the root of the search tree. To achieve this the square with least candidates may be chosen. Note that this algorithm may solve puzzles very fast provided that they are easy enough. This is because it will always choose squares with only one candidate if such squares exists and all puzzles which are solvable by that method will therefore be solved immediately with no backtracking.

Research Paper Number 2

Research Paper Link : https://www.researchgate.net/publication/303553939_Recursive_Backtracking_for_Solving_99_Sudoku_Puzzle

CONCLUSION AND FUTURE WORK In this paper we used a backtracking algorithmfor solving Sudoku puzzles with different numbers of clues. e have implemented the algorithm using java and also compared the results.The results reveals that the application program developed by us performs well for solving 30 puzzles of size 9*9 with various clues.

Future work includes studying neural network and develop an algorithm using Neuralnetwork to solve a 9*9 sudoku puzzles.
Overall there is space for larger studies with more algorithms for Solving SudokuPuzzles of other sizes.
Also implementation of Sudoku Puzzles in Data security remains as a future work.

Research Paper Number 3

Research Paper Link : https://en.wikipedia.org/wiki/Sudoku

Nowadays Sudoku is a very popular game in the world and it appears in different media ,including websites, newspapers and books, mobile apps. There are numerous methods or algorithms to find Sudoku solutions and Sudoku generatingalgorithms.

This paper explains possible number of valid grids in a 9*9 sudoku and developed a programming approach for solving a 9*9 sudoku puzzle and the results have been analysed in accordance with various number of clues for (9*9) sudoku.

A Sudoku consists of a 9×9 square grid containing 81 cells.
[2]The grid is subdivided into nine 3×3 blocks. Some of the 81 cells are filled in with numbers from {1,2,3,4,5,6,7,8,9}.

These filled-in cells are called givens or clues. The goal of the player is to fill in the whole grid using the nine digits so that each row, column and block contains each number exactly once and this constraint on the rows, columns, and blocks is knownas One Rule.

The solution of a Sudoku Puzzle requires that every row, column and block containall the numbers in the set[1,2,….9] and every cell will be occupied by only one number.

Research Paper Number 4

Research Paper Link :
https://www.math.uci.edu/~brusso/DengLiOptLett2013.pdf

The selection operator, crossover operator and mutation operator of the IGA arefurther improved with respect to the drawback of IGA.
First, the novel HGA uses group as a division for priority level in selection operator, thereby impairing the similarity of the selected chromosome and optimal chromosome, so that the probability of chromosome having more abundant genes forselection is reasonably enlarged; secondly, the crossover operator has been

endowed with dual effect of self-experience and population experience based on the concept of combining particle swarm optimization, thereby making the whole iterationdirectional; the secondary probability of mutation will increase along with decrease offitness, particularly at a later stage of iteration, a reasonable adjustment for mutation probability is conducted according to the fitness value of the optimal chromosomes inthe current population, upon which the algorithm reliability could greatly be
improved.

Finally, under the circumstance of no better chromosome achieved after 4 times of iteration it should be replaced by a near-optimal chromosome in time so as to changethe evolution direction of population timely, thereby to avoid falling local problems.

Simulation experimental results showed that not only the novel algorithm can accurately solve a global optimal solution, but also significantly enhance the convergence rate and stability of the algorithm, becoming one of the effective globaloptimization methods. Though the HGA algorithm has very good results for easy and challenging Sudoku puzzles, it is not so ideal for difficult and super difficult Sudoku puzzles. This awaits our deeper study on this algorithm in the future work to make it more suitable to solvethe difficult and super difficult Sudoku puzzles.

Research Paper Number 5

Research Paper Link : http://acsr.wi.pb.edu.pl/wp-content/uploads/zeszyty/z9/Boryczka,Juszczuk-full.pdf

Difficulty rating	Cells given	Minimum number of iterations	Maximum number of iterations	Average number of iterations	Median	Std dev
New	0	6831	40802	20261.4	17462	11870.7
Easy	34	6251	15421	9111.2	7462	4397.7
Easy	30	4918	12572	6472.6	5150	4056.1
Medium	30	6253	26255	13921.4	7124	10559.5
Medium	26	7171	33498	19846.6	17463	9409.9
Hard	26	11563	21462	16312	16857	3987.2
Hard	24	10831	40802	20261.4	17462	11870.7
New generated 1	30	7513	24524	16598.4	16356	6578.3
New generated 2	30	8457	18436	11427	9135	4242.9
New generated 3	30	3476	31673	15062.6	12452	10409.1
New generated 4	30	4339	28463	13036	9687	9753.8

After the analysis of all test sets, the following conclusions may be reached:

– The Differential evolution is capable of solving very difficult Sudoku puzzles. For example „hard” instances in 15 thousand iterations.

– The Proper mutation schema is very difficult to implement even with strong theoretic analysis of geometric operators.

– The Simple swap mutation brings very good results.

– The Algorithm is far better with finding solutions close to the optimal solution (for example the fitness function equal

To summarize it should be observed as well that the Differential Evolution and other evolutionary algorithms are capable not only to find solutions for Sudoku puzzles but also to generate new puzzles from scratch. The new mutation schema seems to be avery effective tool for solving discrete problems.
Our next goal is to adapt similar mutations into the continuous optimization problem.One of the most difficult things will be the description of the mutation schema, whichwill modify only a small part of the genotype even in the continuous optimization problems.
The key issue should be the proper representation of the individual.
We try to show that appropriate problem transformation is one of the main problemsfor the described mutation schema.

==========

...5- Dataset (Examples of input/output)...

)...

Examples of input/output

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

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

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

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

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

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

==========

...6-Details of Algorithm...

- Analysis of the results, What are the insights? The graph increasing by decreasing the grade of the populations.
- What are advantages? When grade of the populations decreases, Then the evolution is better so we are closer to the target
- What are disadvantages? When grade of the populations is stopped decrease or slow in

decreasing, Then the evolution is not growing enough to the target

- Why did the algorithm behave in such a way? The Differential Evolution is finding the global max, Where every member is in local max. In initialization, It randomly creates number of boards as population, and we try to make mutation, crossover and selection to evolve the population. As it takes time to reach the global max
- What might be the future modifications? By tuning hyperparameters, Like Mutant Factor, Crossover Rate, Number of members in population, Try to make better fitness function.
- Backtracking

  Aim: find the solution of Sudoku

  take index of column initialize zero, index of row initialize zero and the board

  How to work: make iteration from 1 to 9 and but the numbers in the empty ceil then call function CheckValidCellRowAndColumn and function CheckValid3X3Cells to check if valid number or not then call itself again by column+1 (recursive) if column reach to 9 (end of columns) or end row return True else if it is end of column increase row by one then make column = 0

- *return: return true if you find answer else return Null
-

==========

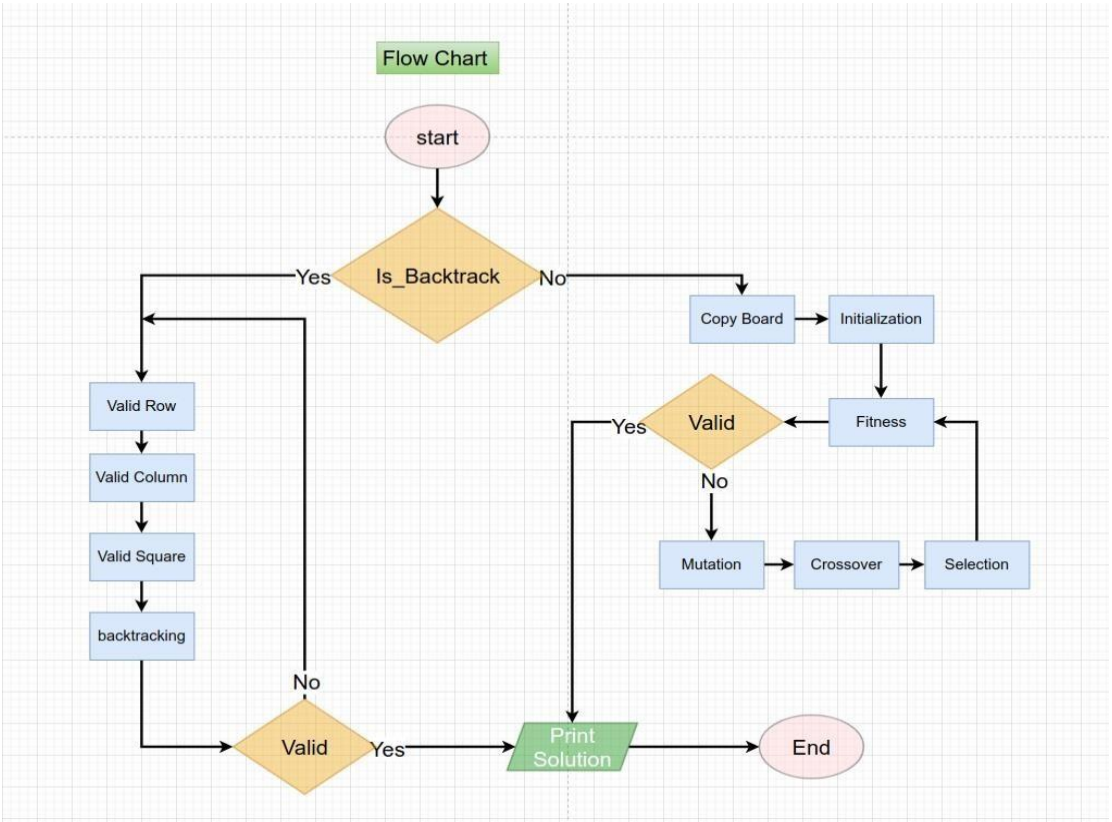
7-….Development platform…..

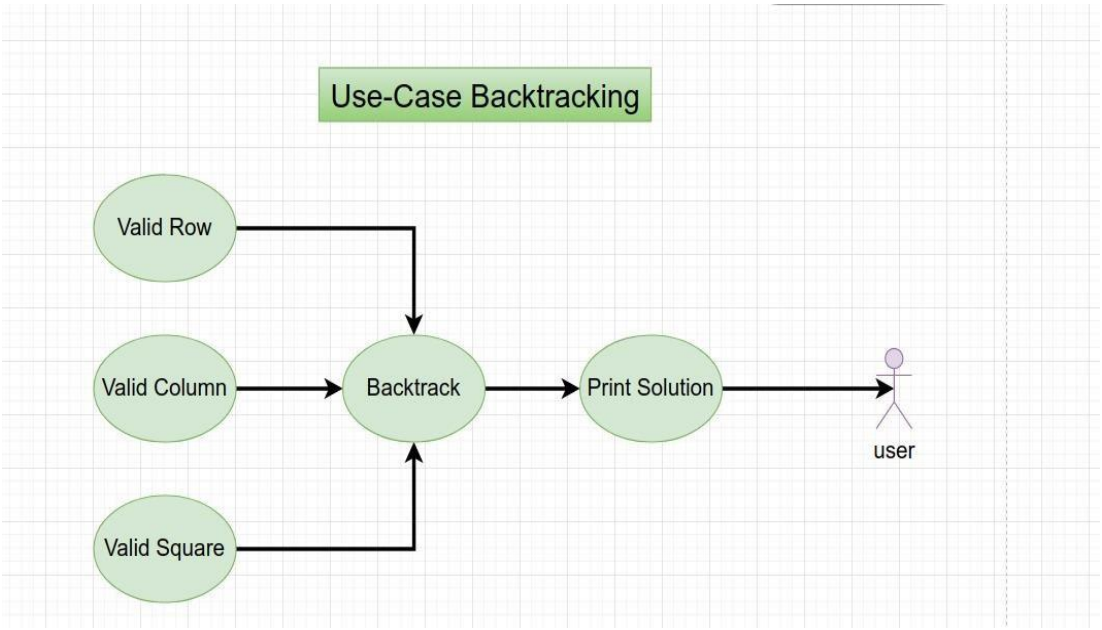
all packages used in the project: pygame, math, random, matplotlib.pyplot coding

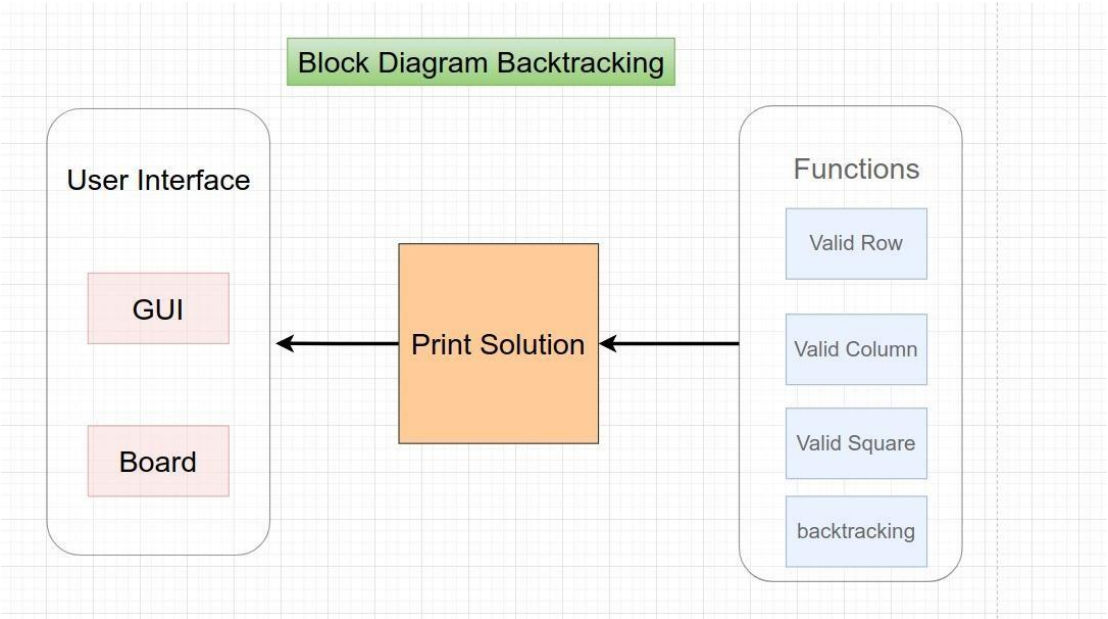
platforms: spyder and pycharm language: python 3.9

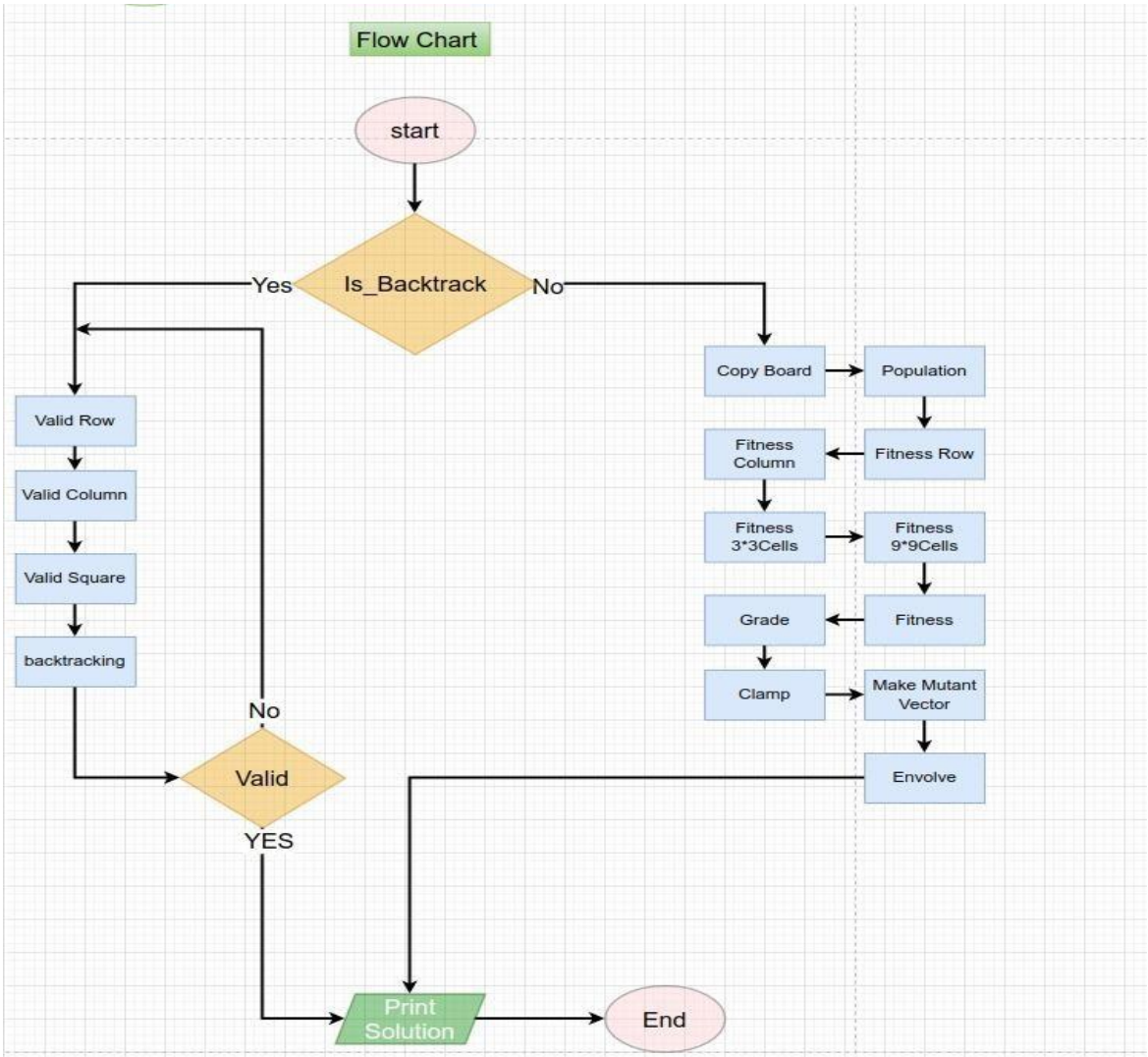
==========

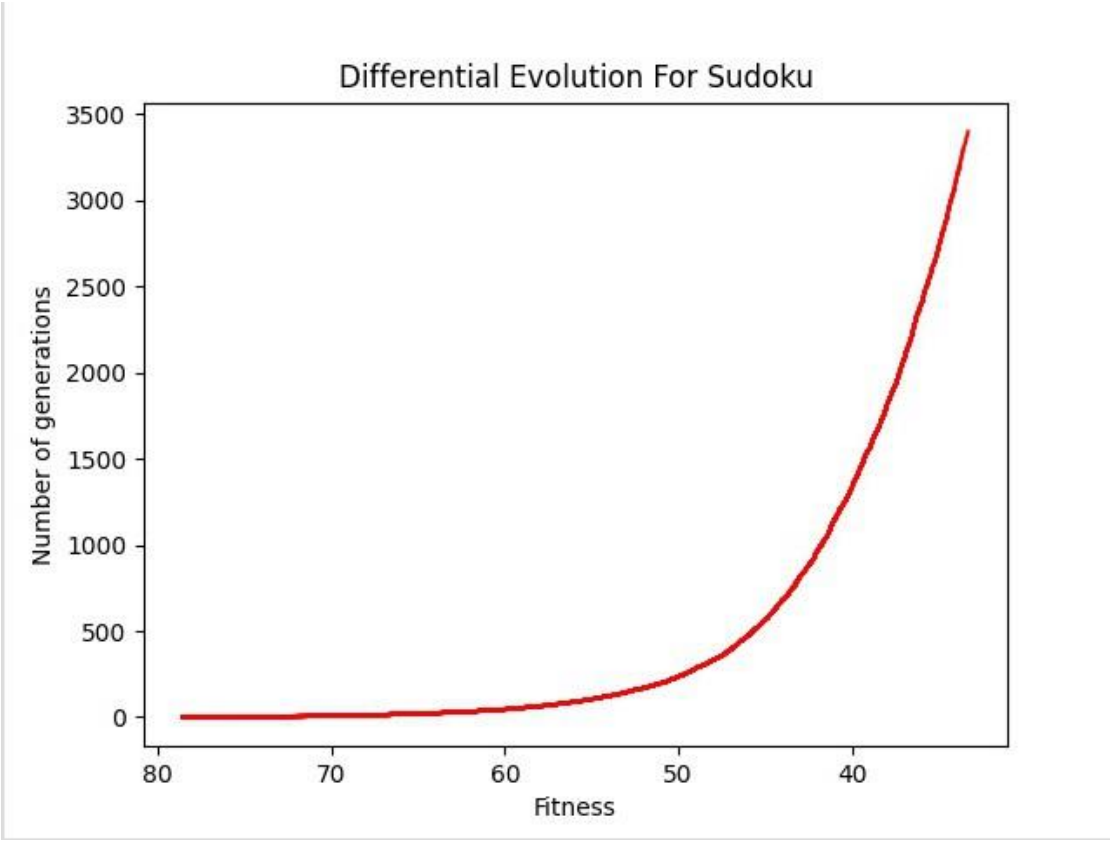
8-….Diagrams and plot…..


Flow Chart
start
Is_Backtrack
Yes
No
Valid Row
Valid Column
Valid Square
backtracking
Valid
No
Yes
Copy Board
Initialization
Fitness
Valid
Yes
No
Mutation
Crossover
Selection
Print Solution
End


Use-Case Backtracking
Valid Row
Valid Column
Valid Square
Backtrack
Print Solution
user


Block Diagram Backtracking
User Interface
GUI
Board
Print Solution
Functions
Valid Row
Valid Column
Valid Square
backtracking


Flow Chart
start
Is_Backtrack
Yes
No
Valid Row
Valid Column
Valid Square
backtracking
Copy Board
Population
Fitness Column
Fitness Row
Fitness 3*3Cells
Fitness 9*9Cells
Grade
Fitness
Clamp
Make Mutant Vector
Envolve
No
Valid
YES
Print Solution
End



Shared code :

https://github.com/Saytara2001/Sudoku/tree/main

==========

Thank You ☺