# Sudoku Solver Using Differential Evolution Algorithm

## (1) Project idea:

A Sudoku Solver using Differential Evolution.
'Sudoku Puzzle' is a number game pioneered by the great Swiss mathematician Leonhard Euler in the year 1783, it is a 9 X 9 square that is divided into nine, 3 X 3 sub squares. In the beginning, there are some static numbers (called givens) in the puzzle. The game is to fill all non-givens such that each row, column, and sub square contains each integer from 1 to 9 once and only once. The difficulty level of the Sudoku puzzle is determined not only by the number of givens, but also it dependents on about 20 factors.
The objective of this project is to present a new Retrievable Genetic Algorithm based on a new model of the fitness function for solving a Sudoku puzzle. Sudoku of varied difficulty levels are solved.

## (2) Main functionalities:

1- Create Initial Sudoku Board
   1. create 2d empty array with numpy -> array of zeros **zero means cell is empty**
   2. fill the created board with random values to start
   3. save locations that have starting values to make them never change

2- Create Initial Population generation zero
   Now we have ready board to work with. let's say our population is 100 solutions.
   1. create random solutions to be the first generation

3- Evaluation applying fitness function on each individual in the population
   1. apply fitness function on each individual in the population **Fitness Function**
   2. sort individuals depending on results of fitness function

4- Update Children --> mutation
   1. mutate each individual to be better

5- Cross Over to get Children
   1. cross over two parents and create new children **crossover function**

6- Select the Best depending on Evaluation
   1. select the best individuals from population to continue living **Selection Function**

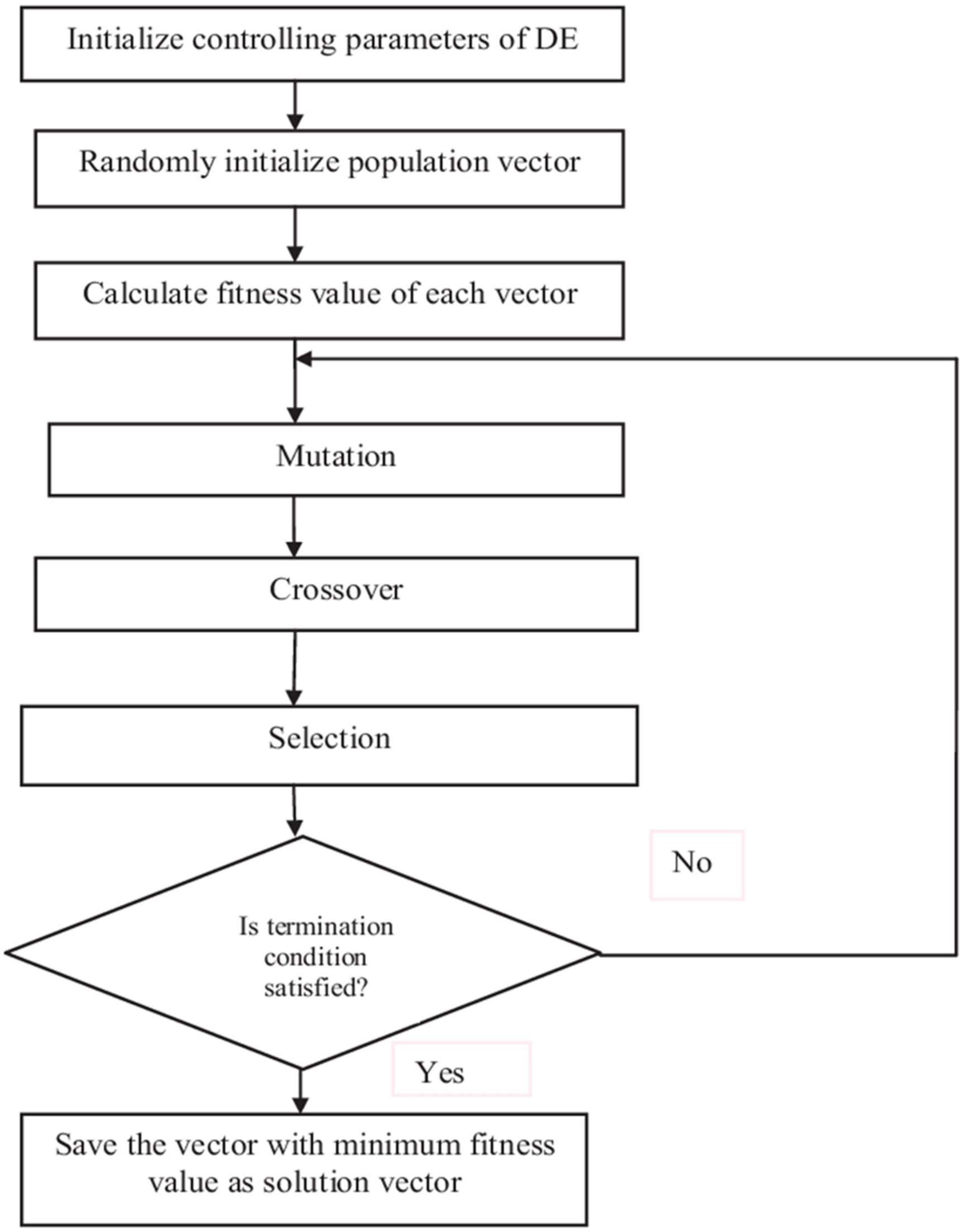
## (3) Similar applications in the market:

There are a lot of applications in the market that solve the Sudoku puzzle such as:
Mobile applications:
**OkayCode Sudoku Solver (https://andauth.co/liaHQU)**
**Snap Solve Sudoku (https://andauth.co/X1nqXR)**
**Sudoku Solver Camera (https://andauth.co/l9vkG5)**
**Sudoku Solver by Shai Alkoby (https://andauth.co/1G9lfG)**
Desktop application:
**Sudoku Solver (https://www.microsoft.com/en-us/p/sudoku-solverfree/9nblggh0jvj2)**

## (4) A literature review of Academic publications:

1- http://acsr.wi.pb.edu.pl/wp-content/uploads/zeszyty/z9/Boryczka,Juszczuk-full.pdf
2- http://www.cs.uccs.edu/~jkalita/work/cs571/2012/DifferentialEvolution.pdf
3- https://link.springer.com/article/10.1023/A:1008202821328
4- https://machinelearningmastery.com/differential-evolution-from-scratch-in-python/
5- https://www.semanticscholar.org/paper/Solving-the-sudoku-with-the-differential-evolution-Boryczka-Juszczuk/7300912959903dd5227e04506d3988da29f943a9
6- https://ieeexplore.ieee.org/abstract/document/4632146/

## (5) Details of the algorithm(s)/approach(es) used:


Initialize controlling parameters of DE
Randomly initialize population vector
Calculate fitness value of each vector
Mutation
Crossover
Selection
Is termination condition satisfied?
No
Yes
Save the vector with minimum fitness value as solution vector

**Evolutionary Computation (EC):**

o Evolutionary Computing (or Evolutionary Algorithms “EA”) is a field of science and engineering that tries to apply some phenomena that appears in the nature to the optimization.

o Most notable is the adaptation of Charles Darwin’s evolution theory to solve difficult search and optimization tasks.
o The method is universally applicable, since it has been successfully applied to almost all thinkable search & optimization problems in engineering, science & people’s everyday life.

**Motivation for Introducing Differential Evolution (DE):**

o DE was designed to be a stochastic direct search method. In other words, it satisfies the first requirement since it can be used to minimize any function for which a global minimum exists.
o DE uses a population of vectors where the stochastic perturbation of the vectors can be done independently. In other words, parallelization is easy.

**Differential Evolution: Basic Components ..**

o DE is a nature-based algorithm that belongs to the class of genetic algorithms since it uses selection, crossover, and mutation operators to optimize (using a global-search metaheuristic) an objective function over the course of successive generations (Suresh et al., 2009).
o DE is a parallel population-based direct search method where the population is comprised of NP vectors each of dimension D and is primarily suited for numerical optimization problems.
  o NP does not change during the algorithm’s lifetime.
o The initial population is chosen randomly and should cover the entire parameter space.

**Differential Evolution: Basic Components ..**

o Mutation: DE generates new individuals in the population by adding weighted difference between two population vectors to a third vector (the mutated vector).
o Crossover: The mutated vector’s parameters are then mixed with the parameters of another pre-determined vector, the target vector, to yield a trial vector.
o Selection: If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the next generation.
o Each population vector must serve once as the target vector so that NP competitions take place in one generation.

**DE (Differential Evolution): the Algorithm**

o Select four parents (p1, p2, p3, p4)
o Calculate the difference vector between two parents: p1 - p2
o Add the difference to the third parent vector with weight F:
o p3 + F( p1 - p2 )
o Crossover between the new vector and fourth parent with gene selection probability “crossover rate” CR o If ( Math.random() > CR ) xi = xi ( p4 ) o else xi = xi ( p3+ F (p1- p2) )
o Compare the new child with fourth parent (p4), the more fit will reach the next generation.

o Start with NP randomly chosen solution vectors.
o For each i in (1, .., NP), form a ‘mutant vector’
o vi=xr1+ F .( xr2- xr3)
o Where r1, r2, and r3 are three mutually distinct randomly drawn indices from (1, .., NP), and also distinct from i, and $0 < F \leq 2$ (F is a real and constant value).

**DE: Crossover xi & vi to form a Trial Vector ..**
xi = (xi1, xi2, xi3, xi4, xi5)
vi = (vi1, vi2, vi3, vi4, vi5)
ui= (_, _, _, _, _) ?
o For each component of the vectors, draw a random number in U[0,1], call this $rand_j$.
o CR (Crossover Rate) is the crossover-constant / cutoff which has to be determined by the user. Let $0 < CR < 1$.
o If $rand_j \leq CR$, then $u_{ij} = v_{ij}$, else $u_{ij} = x_{ij}$.
o To ensure at least some crossover, one (or more) component(s) of $u_i$ is/are selected at random to be from vi.
o So, $rnbr_j$ is a randomly chosen index belongs to 1, 2, .., D.
o If $rnbr_j = j$, then $u_{ij} = v_{ij}$

## DE: Crossover xi & vi to form a Trial Vector ..
xi = (xi1, xi2, xi3, xi4,xi5) vi = (vi1, vi2, vi3, vi4,vi5)
So, for example, we may have:
ui = (vi1, xi2, xi3, xi4,vi5)
E.g., Index 1 is randomly selected as a definite crossover.
rand5 <= CR, so it crossed over too.

**DE: the Selection ..**
If the objective-value (fitness-function) COST($u_i$) is
better than COST($x_i$)
then $u_i$ replaces $x_i$ in the next generation
otherwise, we keep $x_i$

**The basic DE algorithm ..**
```
Initialize all agents X with random positions in search space
Repeat
 for each agent X in the population do
   Pick three agents r1,r2,r3
   Pick a random index  between 1 and NP
   Compute the agent’s potentially new position
   for each component of the agent do
     Copy or not copy based on the cases discussedearlier
   end for .. end for
until termination criterion is met (e.g., number of iterations performed, or adequate
fitness reached).
```

Pick the agent from the population that has the lowest cost and return it as the best-found candidate solution.

**The highlights of DE ..**

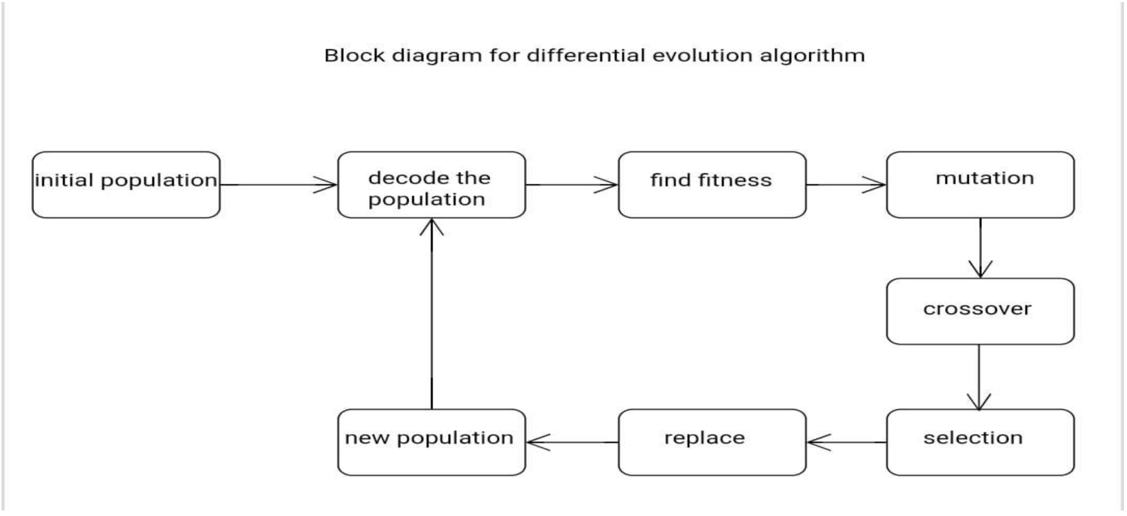
o In the beginning, all candidate solutions are spread uniformly in search space X.
o The differential information (xr2-xr3) is large ⇒ large search steps.
o As the population converges, the candidate solutions are located closer to each other.
o The distances (xr2-xr3) decrease ⇒ the search steps get smaller too.
o Very simple way to perform self-adaptation without needing any additional parameter or operation.
o The step-width is self-organizing from the structure of the population and self adapting along the optimization process.

**The highlights of DE ..**

o A population of NP solution vectors are successively updated by addition, subtraction, and component swapping, until the population converges, hopefully to the optimum.
o No derivatives are used.
o Very few parameters to set.
o Asimple and apparently very reliable method.

**Hybridization of EAs ..**

o The common trend of EAs is hybrids between different evolutionary algorithms.
o These hybrids are relatively easy to do, since most EAs can use the same population table and the same gene presentation.
o Hybrids are mainly done in order to avoid the shortcomings of one algorithm:
o – E.g. In the GA/DE hybrid, GA acts like a global searcher, and DE acts as a local searcher. Thus, GA finds peaks, and DE finds the highest point of that peak.



## (6) Development platform:

Libraries used in the program → Numpy
Random

## (7) Shared folder:

**https://github.com/Boodyahmedhamdy/SudokuSolver**