**Genetic Algorithm Based Sudoku Solver**

Genetic algorithms are commonly used to generate high-quality solutions to [optimization](https://en.wikipedia.org/wiki/Optimization_(mathematics)) and [search problems](https://en.wikipedia.org/wiki/Search_algorithm) by relying on biologically inspired operators such as [mutation](https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)), [crossover](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)) and [selection](https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)). Here we are trying to solve the Sudoku, which is a Number/Character placement puzzle that has achieved remarkable popularity in the past few years.

This program reads in an initial grid configuration from a Json file, given as a command-line argument.

This we will provide the index of the numbers we need. Other Genetic algorithm parameters are specified as arguments, including the number of initial solution strings and dimension of the Grid. The maximum number of generations, percent of current population used for reproduction, and mutation rate are hardcoded as below.

**Initialized Genetic variables as follows**:

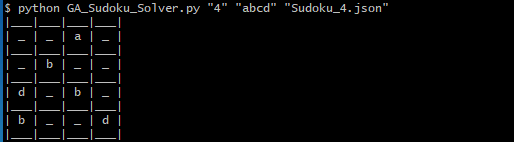
Nc = 1000 # Number of candidates (i.e. population size).

Ne = int (0.05 \* Nc) # Number of elites that is the individuals in the generation with the best fitness values.

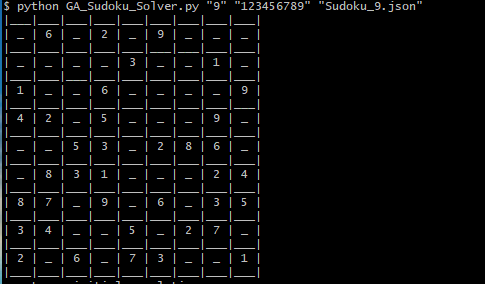
Ng = 10000 # Number of generations.

Nm = 0 # Number of mutations.

A **sample** input for 2x2 look likes below:



A sample input for 3x3 look like below



We will do the below validation for the input

1. Check Nd is proper square
2. Length of input is equal to Nd
3. Input contain proper Digit/Characters (No white spaces)
4. Input contain duplicate character or 0
5. Whether the Json file exists.

This solver work as below

* Solver has below algorithm:

1. Randomly initialize the Population:

Here we will initialize the vector of int (Passing index of the given word as word to as numbers are faster than Strings). Initially we will update a NdxNd array of vector with zeros and the we will find the permutation of 1 to Nd and randomly store those in chromosomes. This is done by the Population.seed() function.

1. Determine the Fitness

The fitness function will find which solution is good. We will update the fitness of candidate and sort them. The fitness of a candidate solution is determined by how close it is to being the actual solution to the puzzle. The actual solution (i.e. the 'fittest') is defined as a Nd x Nd grid of numbers in the range [1, Nd] where each row, column and sqrtVal x sqrtVal block contains the numbers [1, Nd] without any duplicates if there are any duplicates then the fitness will be lower.

1. Until convergence repeat:
2. Parent (Tournament) Selection :

Here we will select n random population and perform a tournament among them. Select best from this N population in a stochastic way.

1. Crossover (Generate new population)

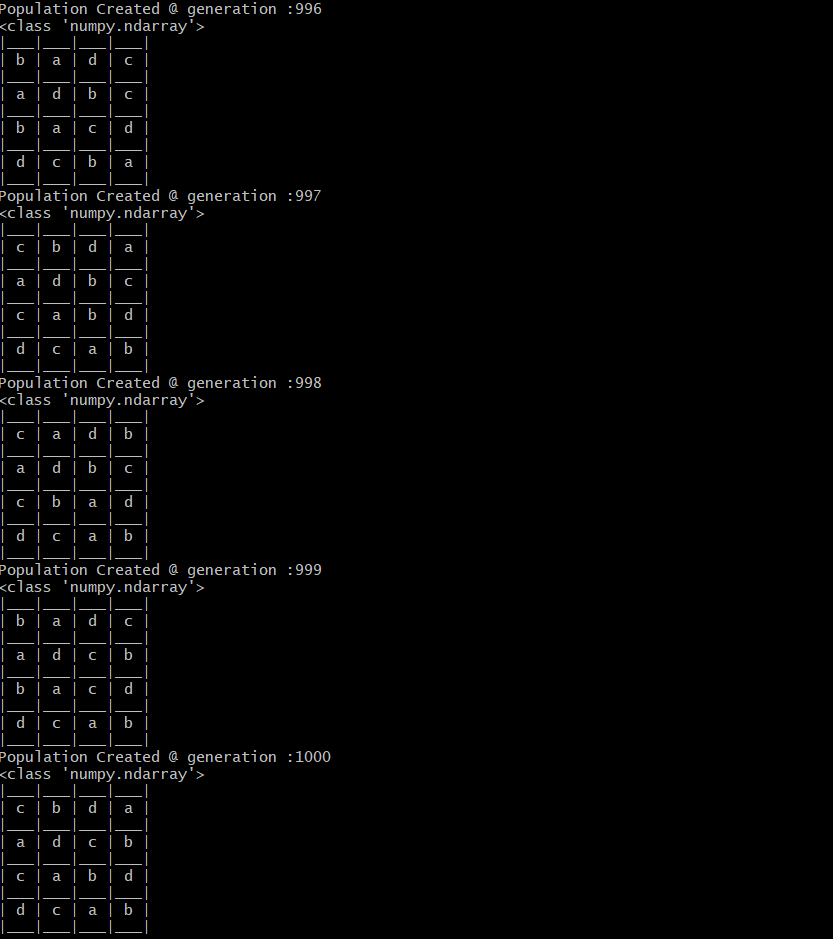
Creating 2 child’s from the selected parents from step 3. Here completely new gene get created from the parents by mutating

1. Perform Mutation on child

Here we will insert random string in population to maintain diversity.

1. Calculate fitness child

Sample population for generation will be as below (There will be 1000 entries as we initialed Ng=1000. Below snippet of last few. Basically we don’t need 1000 generation when Nd is 4, but for others it is required.):



Sample outputs for the inputs given above.

