Paper information:

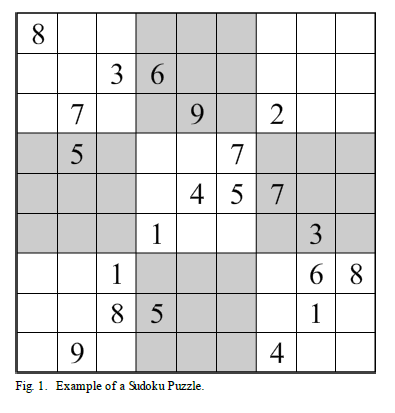
Authors: Rory Douglas

Published year: 2014

Publisher: University of Derby

Representation:

Chromosomes consist of 81 byte values. These are split into 9 sets of 9 byte values to represent 3x3 sub-grids.





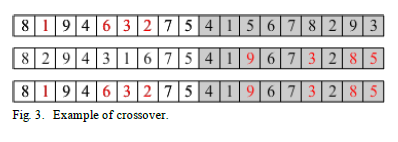
Fitness Function:

The fitness function in this algorithm simply checks whether the rows and columns have duplicates and increments a cost value for every time there is a duplicate. Squares are not checked as by design they will always be valid. Fitness will be measured between 0 and 1, with 1 being ideal fitness.

Crossover:

Crossover operation has been explained vaguely as bellow:

Crossover is performed by selected a 3x3 sub grid from either of the parent chromosomes based on the crossover rate.

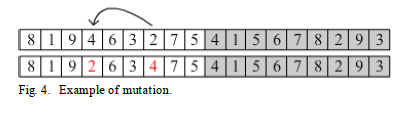


Mutation:

Mutation operation also has been explained vaguely as bellow:

and problem is that didn't mention that mutation perform for each block or just for one block.

Mutation is performed by swapping two bytes values within a 3x3 sub-grid.



Parent and survivor selection:

Tournament selection as parent selection mechanism.

Elitism 80% for survivor selection.

Paper Conclusions:

The algorithm developed in this paper was able to solve simple Sudoku puzzles, however it is fairly safe to say it does not excel at solving Sudoku. Two major problems with the algorithm are:

* the size of the search space
* the fitness calculation.

Fitness Implementation:

For Implement fitness function we present two way:

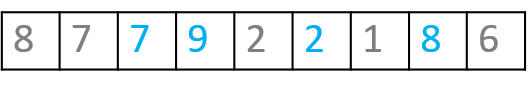
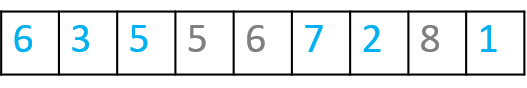
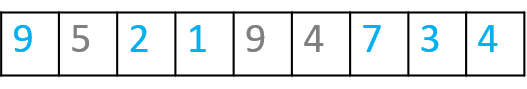
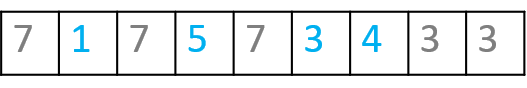
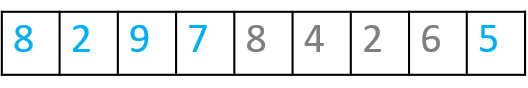
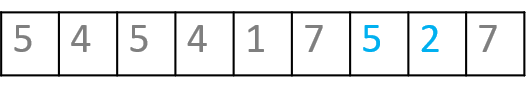
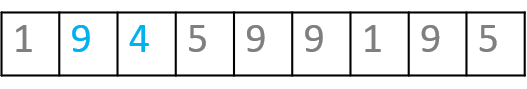
* Count duplication of number in row and column as for occur n>1 we add n-1 to fitness of chromosome.
* Or another way as beside doing the first way, for each absent number in row or column we add 1 to fitness

Proof accurate of fitness:

Output of algorithm for this initial chromosome.



row duplication calculation. It doesn’t need to check the rows in order.



4 duplication + 4 absence = 8

2 duplication + 2 absence = 4

4 duplication + 4 absence = 8

2 duplication + 2 absence = 4

2 duplication + 2 absence = 4

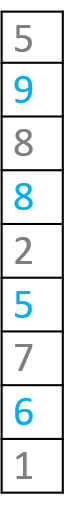
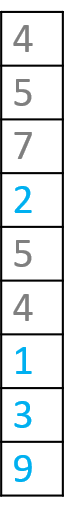
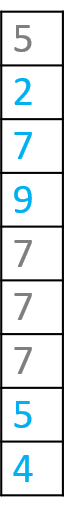
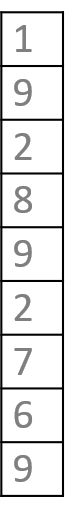
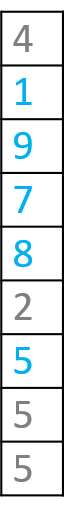
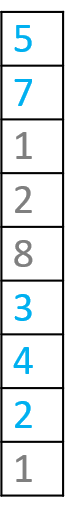
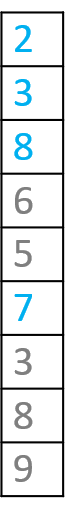
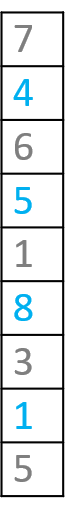
2 duplication + absence = 4

3 duplication + 3 absence = 6

2 duplication + 2 absence = 4

5 duplication + 5 absence = 10

fitness calculation per column (duplication + absence)



4

4

4

6

6

4

8

4

4

Fitness calculation:

As paper mentioned fitness value must be in (0,1].

For achieving that we determine worst and best solution as boundary.

And by using expression: 1- (fitness/144) final fitness achieve.

* Worst case is a puzzle with same value for every cell:
  + Worst Fitness = 144 -> 1-(144/144) = 0
* Best case is a sudoku answer:
  + Best Fitness = 0 -> 1-(0/144) = 1

Mutation Implementation:

For Implement mutation function we present two way:

* Swap mutation inside each sub-grid(block) of chromosome.
* Randomly choose one block of chromosome to perform swap mutation inside.

Mutation Implementation 1:

[[8, 4, 2, 9, 1, 2, 9, 8, 7],[9, 3, 4, 1, 4, 8, 9, 3, 2], [5, 2, 1, 7, 3, 4, 8, 8, 2],

[8, 2, 9, 3, 2, 9, 5, 5, 8], [7, 8, 2, 8, 9, 6, 2, 4, 9], [9, 7, 5, 4, 1, 9, 3, 7, 8],

[6, 1, 4, 6, 3, 5, 2, 9, 4], [5, 1, 3, 7, 9, 7, 2, 9, 5], [4, 7, 7, 2, 8, 1, 6, 7, 4]] **fitness: 92**

block index: 2 first\_index: 5 second\_index: 2

[[8, 4, 2, 9, 1, 2, 9, 8, 7], [9, 3, 4, 1, 4, 8, 9, 3, 2],[5, 2, 4, 7, 3, 1, 8, 8, 2],

[8, 2, 9, 3, 2, 9, 5, 5, 8], [7, 8, 2, 8, 9, 6, 2, 4, 9], [9, 7, 5, 4, 1, 9, 3, 7, 8],

[6, 1, 4, 6, 3, 5, 2, 9, 4], [5, 1, 3, 7, 9, 7, 2, 9, 5], [4, 7, 7, 2, 8, 1, 6, 7, 4]] **fitness: 92**

Mutation Implementation 2:

[[8, 4, 2, 9, 1, 2, 9, 8, 7],[9, 3, 4, 1, 4, 8, 9, 3, 2], [5, 2, 1, 7, 3, 4, 8, 8, 2],

[8, 2, 9, 3, 2, 9, 5, 5, 8], [7, 8, 2, 8, 9, 6, 2, 4, 9], [9, 7, 5, 4, 1, 9, 3, 7, 8],

[6, 1, 4, 6, 3, 5, 2, 9, 4], [5, 1, 3, 7, 9, 7, 2, 9, 5], [4, 7, 7, 2, 8, 1, 6, 7, 4]] **fitness: 92**

block index: 0 first\_index: 0 second\_index: 7 block index: 1 first\_index: 8 second\_index: 7 block index: 2 first\_index: 0 second\_index: 5

block index: 3 first\_index: 5 second\_index: 0 block index: 4 first\_index: 2 second\_index: 3 block index: 5 first\_index: 2 second\_index: 6

block index: 6 first\_index: 2 second\_index: 6 block index: 7 first\_index: 4 second\_index: 7 block index: 8 first\_index: 2 second\_index: 1

[[8, 4, 2, 9, 1, 2, 9, 8, 7], [9, 3, 4, 1, 4, 8, 9, 2, 3],[1, 2, 4, 7, 3, 5, 8, 8, 2],

[9, 2, 9, 3, 2, 8, 5, 5, 8], [7, 8, 8, 2, 9, 6, 2, 4, 9], [9, 7, 3, 4, 1, 9, 5, 7, 8],

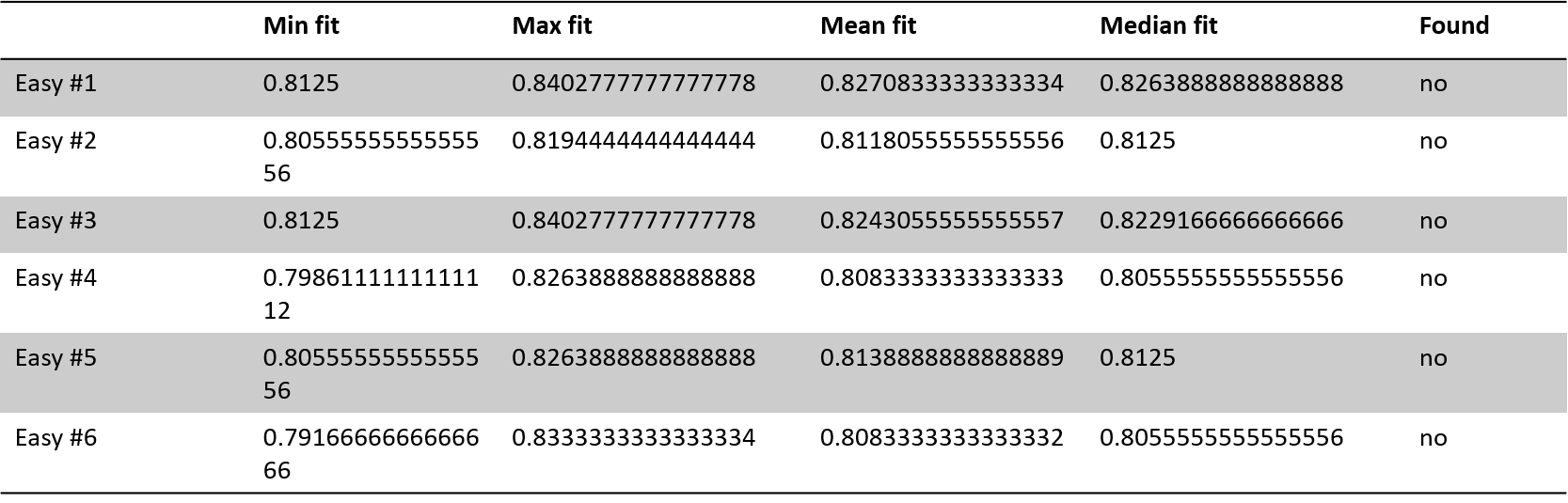
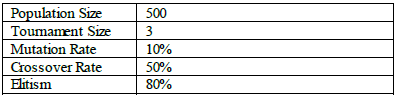
[6, 1, 2, 6, 3, 5, 4, 9, 4], [5, 1, 3, 7, 9, 7, 2, 9, 5], [4, 7, 7, 2, 8, 1, 6, 7, 4]] **fitness: 92**

Crossover Implementation:

Crossover function take two parents from population and create two children by uniform method.

We test uniform and orderone xover.

Results:

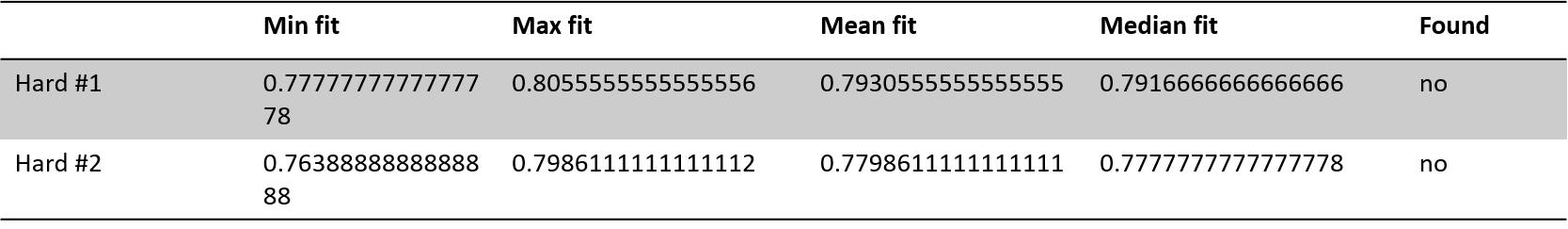
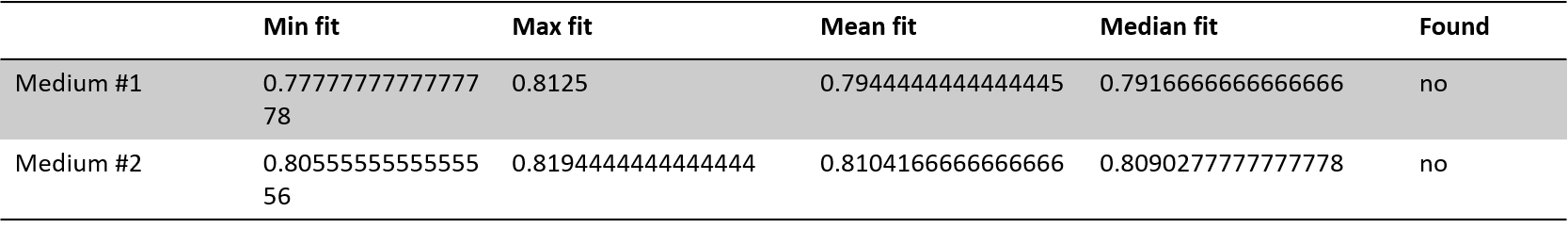


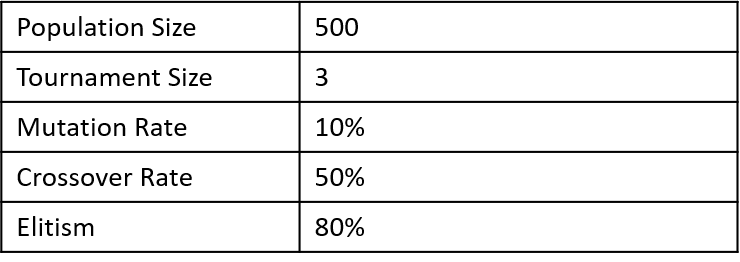
Max generation = 4000

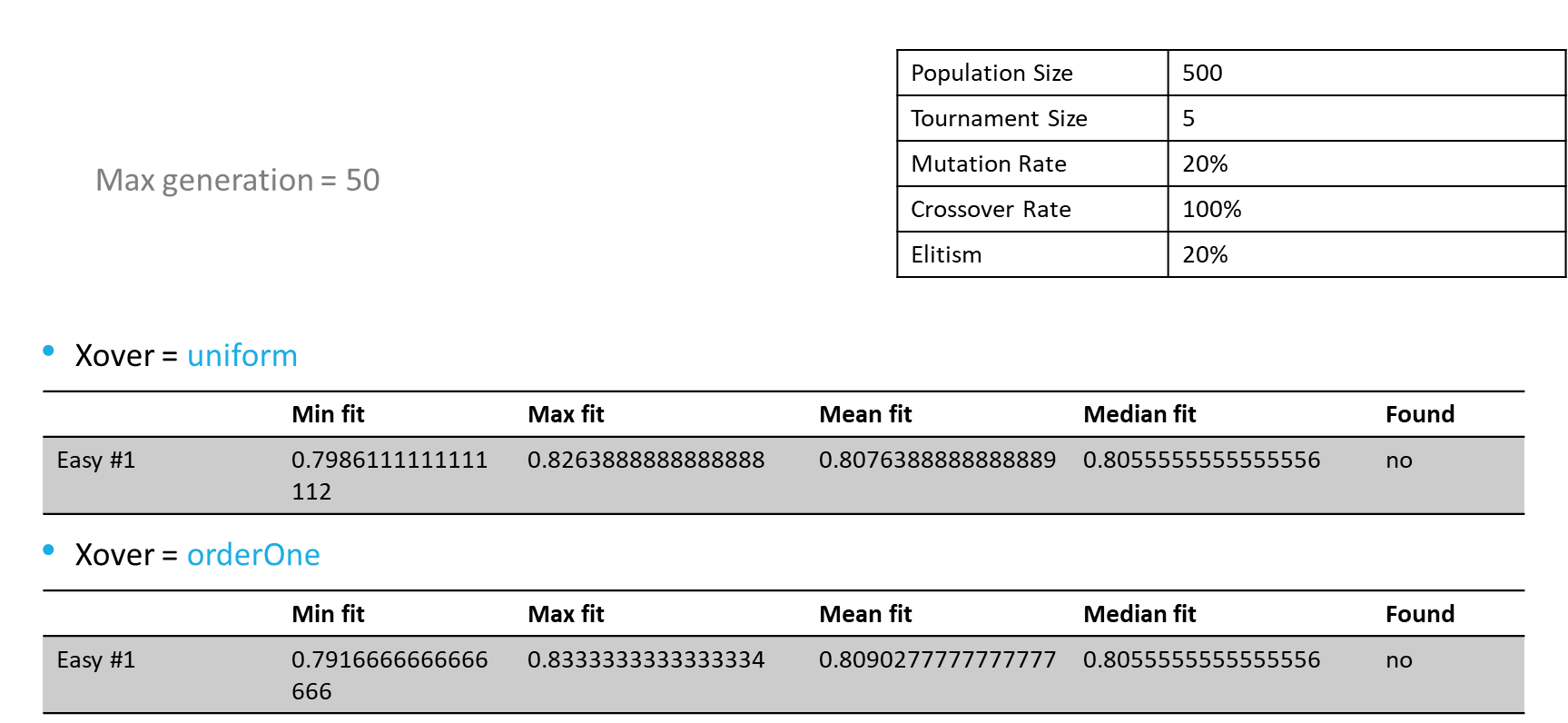
10 times run

Max generation = 4000

10 times run

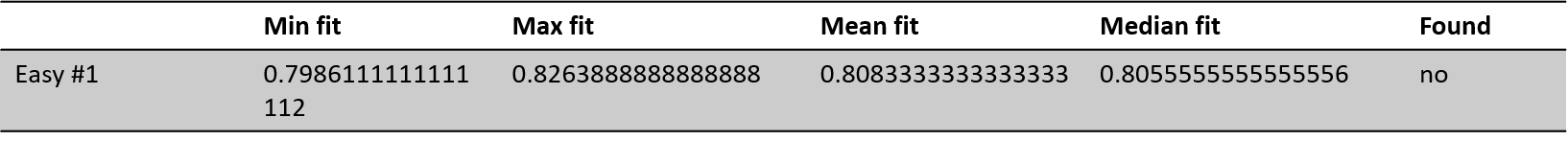


Comparin xover methods:

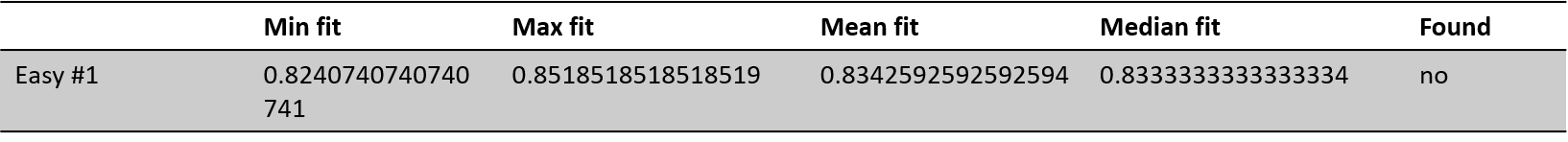


Comparing fitness function methods:

Row and column duplication

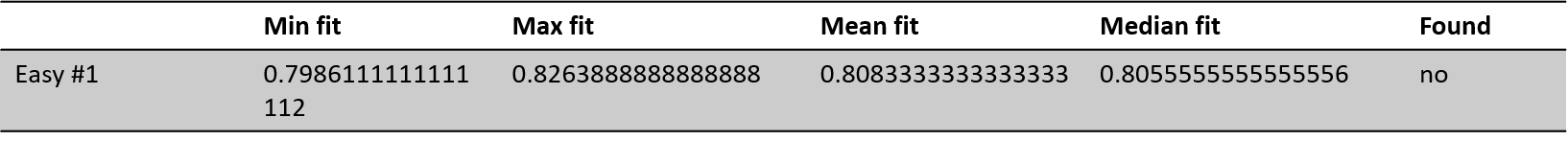


Row and column and sub-grid duplication

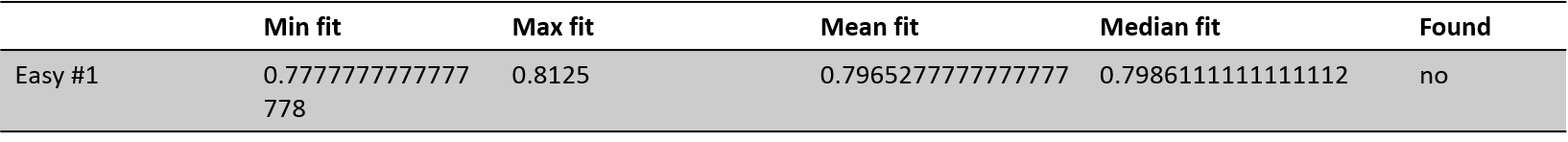


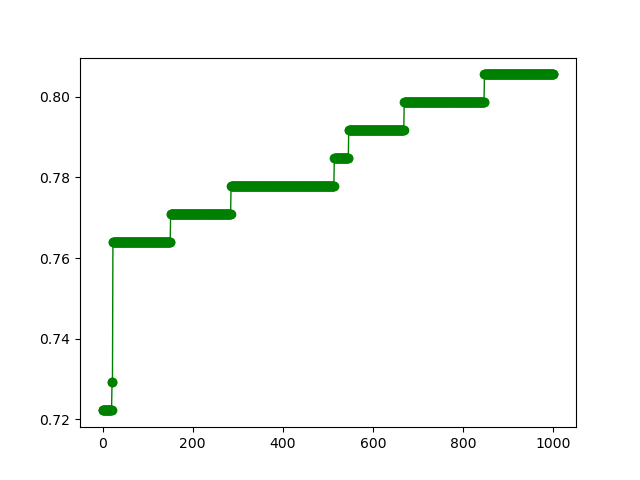
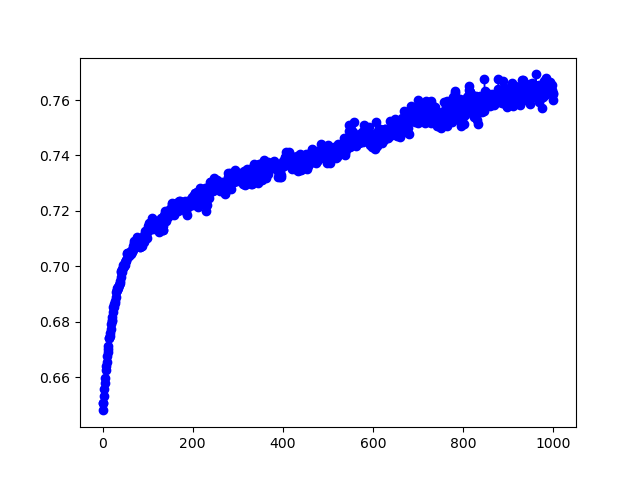
Comparinf mutation mehods:

Multi mutation per block



Single mutation



GA learning process:

Average fitness per generation

Best fitness per generation

Conclusion:

* As paper itself mentioned method has two major problem as size of search area and fitness function
* Cause of ambiguity in method explanation, I think I missed something. but it’s obvious that problem need maybe diversity preserving and somehow get out from local optima.
* More complex fitness function gave better answer.