# Table of Contents

# *Part A - THE PUZZLE*

## Problem Description

Given an empty rectangular frame of length L and width W, and N rectangular tiles. And We need to design and write a GA that arranges the tiles in the Frame in such a way that minimizes the free space

## Methodology

### I. Representation

**Real-Valued Representation** – because the given values are integer numbers and changes are also in integers.

### II. Population and Population Size

A puzzle of N tiles will be generated as population. The initial population will be stored as JSON file **population.json**. The individuals in the population are also JSON Strings with the following keys and values:

| | |
|---|---|
| **Length** | Frame Length (L) |
| **Width** | Frame Width (W) |
| **Pieces** | Number of tiles (N) |
| **Puzzle** | A list of tiles, which are basically the arrangement of tiles in the frame (mentioned below) |

*Table 1*

A tile is a list [x, y, l, w], where (x, y) are coordinates of the bottom left corner of the tile, and (l, w) are the dimensions of the tile.

Therefore, a puzzle is made of a list of tiles = $[[x_1, y_1, l_1, w_1] \ldots\ldots\ldots, [x_N, y_N, l_N, w_N]]$, for a puzzle of N tiles.

Population Size I have taken is **300**.

The final population is stored in the **final_population.json** file.

### III. Fitness Evaluation

Fitness of an individual is calculated by finding the percentage of free space in the frame. A *low* percent means *better* fitness value.

### IV. Parent Selection

**Uniform Random Selection** - It is unbiased, and every individual has the same probability to be selected

### V. Variation Operations

#### Crossover

**Discrete Uniform Crossover** - Each pair of parents give two offspring's by simply swapping x and y coordinates of parents.

#### Mutation

**Nonuniform Mutation** - $x_i' = x_i + N(\mu, \sigma)$
Where $\mu = 0.5,\ \sigma = varies\ on\ popsize\ and\ gensize$

### VI. Survivor Selection

Survival selection is done using explicit approach ***Crowding*** for preserving diversity.

## Results

Below are some of the different approaches I implemented for finding the best parameters.

| | **ALG-1 (Best)** | **ALG-2** | **ALG-3** | **ALG-4** |
|---|---|---|---|---|
| **Symbolic Parameters** | | | | |
| **Representation** | Real-valued | Real-valued | Real-valued | Real-valued |
| **Parent Selection** | Uniform Random | Exponential Rank | Uniform Random | Uniform Random |
| **Recombination** | Discrete Uniform | Whole Arithmetic | Discrete Uniform | Discrete |
| **Mutation** | Non-Uniform | Uniform | Non-Uniform | Non-Uniform |
| **Survivor Selection** | Crowding | Fitness-Proportionate | Crowding | Crowding |
| **Numeric Parameters** | | | | |
| **Population Size** | 300 | 300 | 300 | 150 |
| **Generation Size** | 200 | 200 | 200 | 100 |
| **Mutation Rate** | 0.2 | 0.3 | 0.7 | 0.3 |
| **Rotation Rate** | 0.4 | 0.3 | 0.7 | 0.3 |
| **Crossover Rate** | 1 | 1 | 1 | 1 |
| **Results** | | | | |
| **Minimum Avg Fitness** | 10.5 | 22 | 16.7 | 16.2 |

*Table 2* - The above table has 2 EAs, with 3 instances for one of them. Out of them, ALG-1 has the best performance when compared with others.

Below are the plots for the best ALG-1 where three different inputs are taken.

## Plots

I have plotted for three different inputs, where mainly each input has *Solution* (***fig-a***), *Final minimum fitness solution* (***fig-b***), *Minimum fitness value per each generation* (***fig-c***) and *Average fitness value per each generation* (***fig-d***).

For each plotted figures ***c*** & ***d***, the x-axis is Generations (out of 200) and y-axis is Fitness values.

### *Input 1*

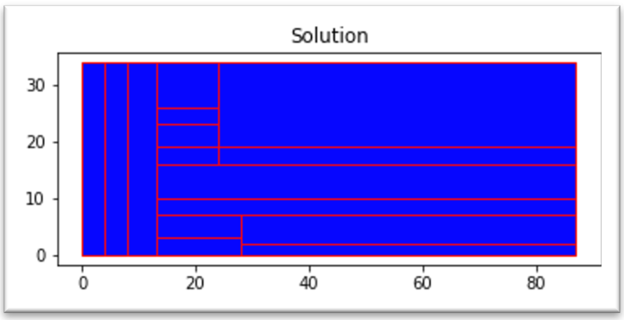


*fig-a*

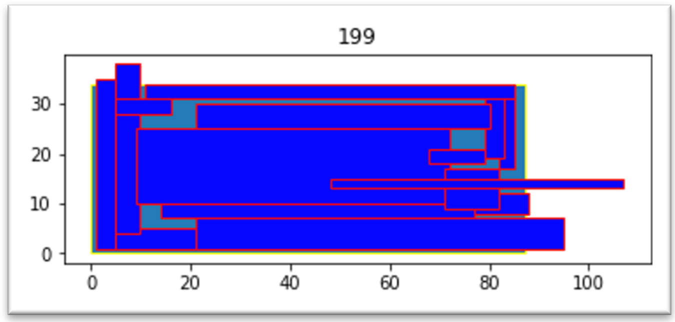


*fig-b*

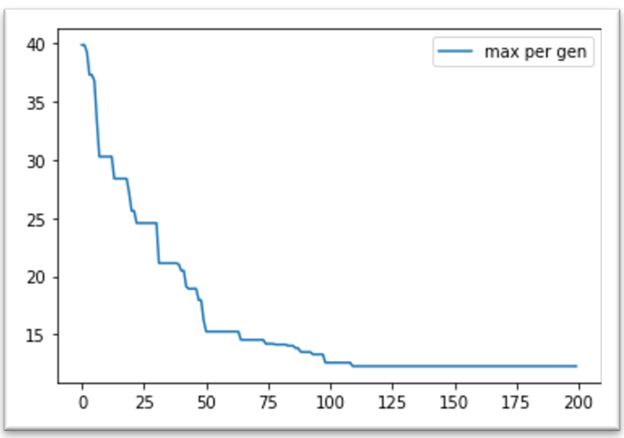


*fig-c*

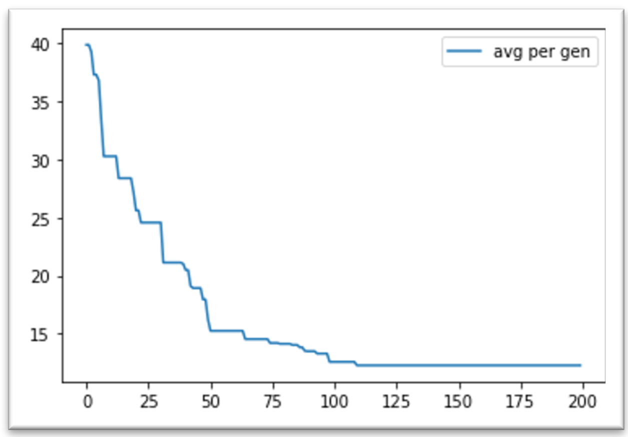


*fig-d*

### *Input 2*

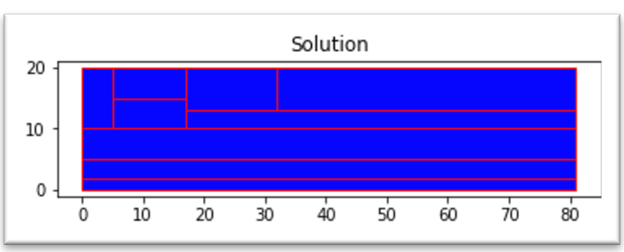


*fig-a*

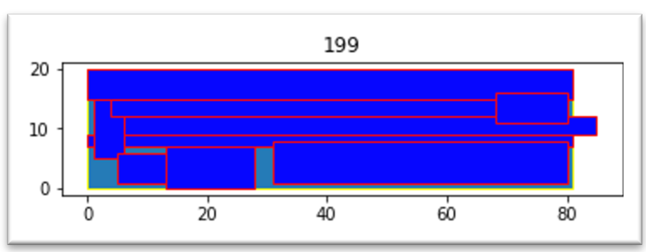


*fig-b*

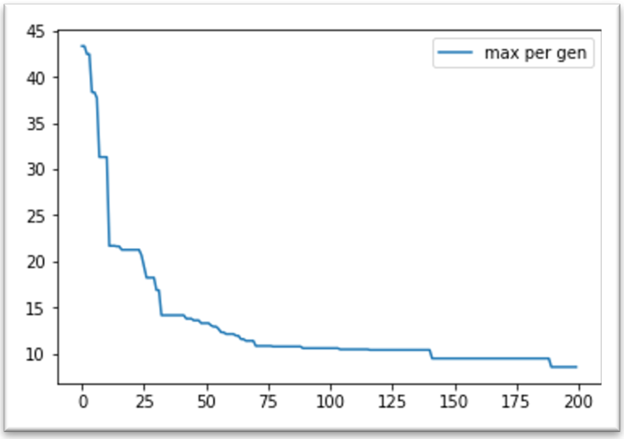


*fig-c*

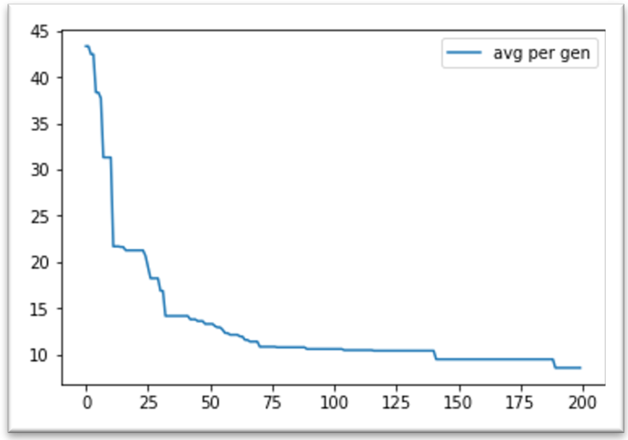


*fig-d*

## ***Input 3***

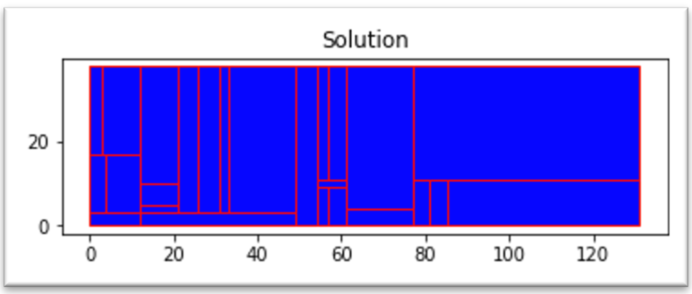


*fig-a*

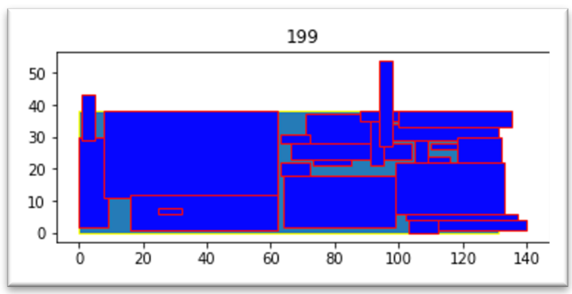


*fig-b*

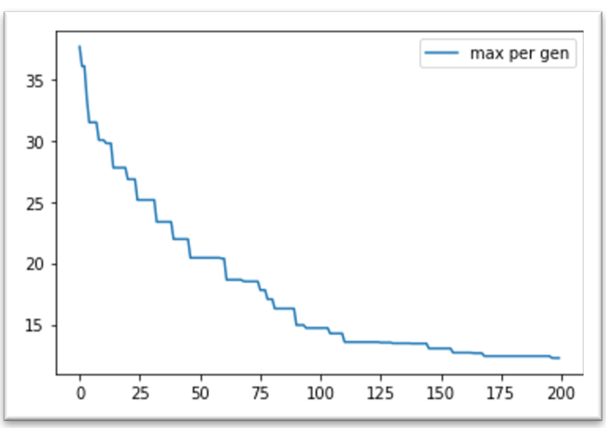


*fig-c*

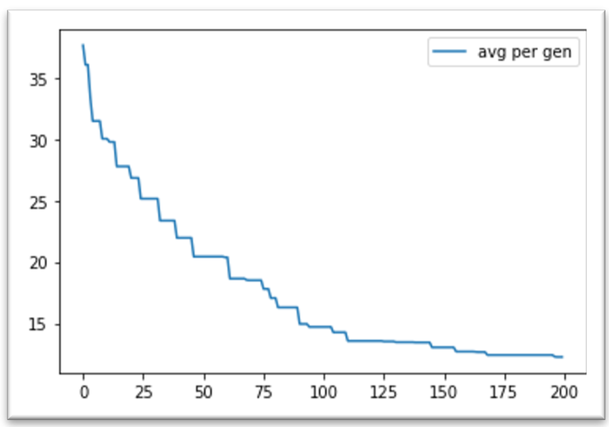


*fig-d*

These are the mean and standard deviation of both ***average*** and ***maximum*** fitness of a 350-individual population over 200 generations.

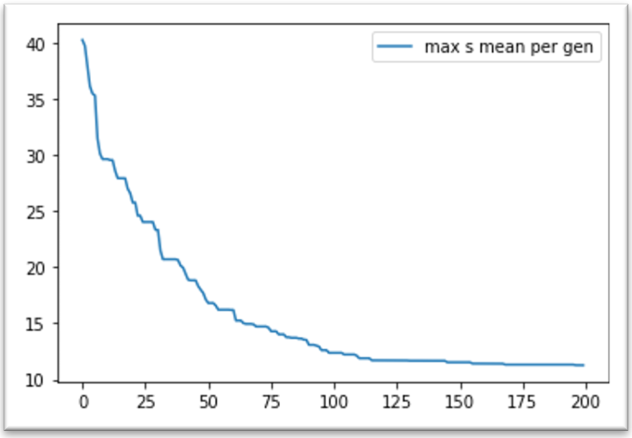


*Fig 1 – Mean of Minimum fitness values of 3 different inputs*

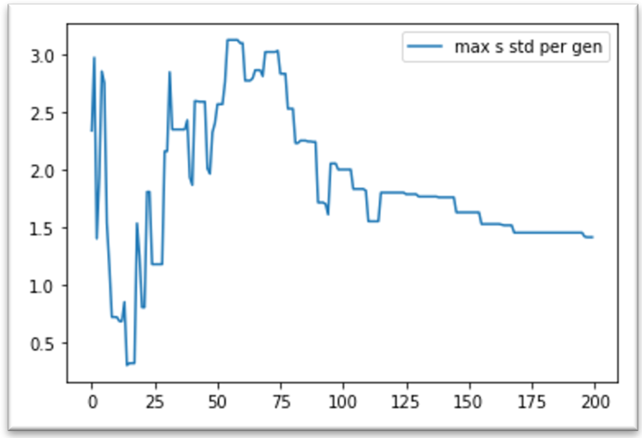


*Fig 2 – Standard Deviation of Minimum fitness values of 3 different inputs*

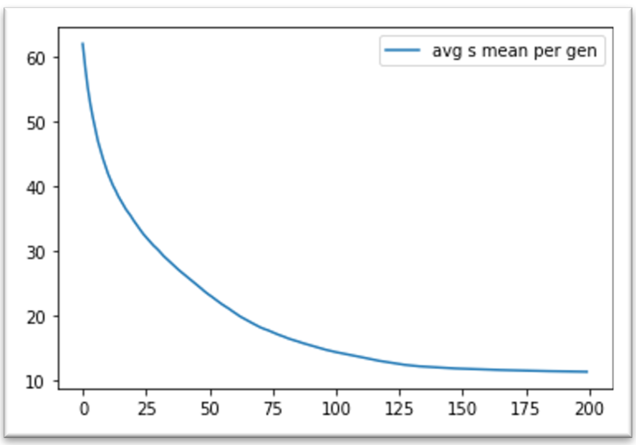


*Fig 3 – Mean of Average fitness values of 3 different inputs*

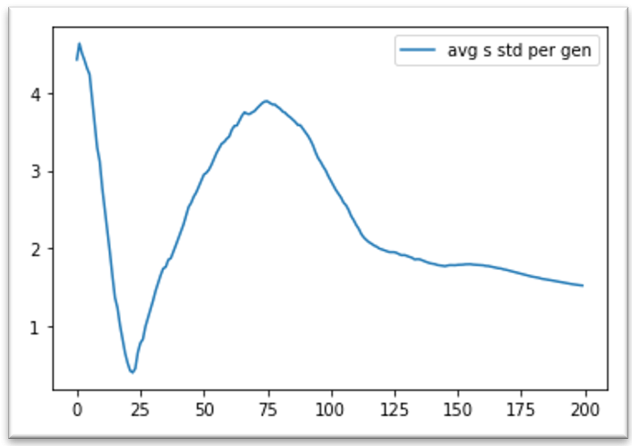


*Fig 4 – Standard Deviation of Average fitness values of 3 different inputs*

From the *mean* plots we can say that the average fitness we are achieving around 10 which is **90%** of the given rectangle is occupied with the pieces. And from the decreasing *standard deviation* plots we can say that the variation between individuals is going low.

I'm also implementing parameter control by varying the sigma in nonuniform mutation depending on the generations, population size and fitness values.

## Suggestions

### *Modifications*

The changes I would like to do is in the mutation where I'm already using non-uniform mutation but I'm expecting some improvement in self-adaptive mutation because as this is more mutation-oriented problem and the values in the individual are more for computing the best fitness value. So, if we vary mutation depending the learning rate of the sigma's we may achieve better solution.

Parameters like population size, generation size etc. has to change based on number of pieces and size of pieces. Because if we have large rectangles, we minimize the free space within few generations. But if have many pieces with small sizes then we need train with large population size.

We need to change the representation as trees where may improve current solution.

### *Fitness Evaluation Improvement*

We may improve the fitness function because at present the fitness value by the amount of free space left the main rectangular frame. But if we some have incorporated the overlapping minimizing value in the fitness function then we may get better results.

### *Diversity*

I am measuring the diversity by using the following function which calculates the distance between all individuals:

$$D = \sum_{i=1}^{N} \sqrt{\sum_{j=1}^{i-1} d(g_i, g_j)}$$

Where $d(g_i, g_j)$ is the distance (I'm using Euclidean) between individuals $g_i$ and $g_j$.

| | **INITIAL DIVERSITY** | **FINAL DIVERSITY** |
|---|---|---|
| **INPUT 1** | 238.9717755 | 31.03613401 |
| **INPUT 2** | 257.942318 | 0.0 |
| **INPUT 3** | 581.714802 | 253.17 0.0 |

*Table 3 – Diversity measured using above formula for the 3 inputs which I used in Results*

We can see that the diversity decreases from initial population to population generated from final generation. And also, for the *input 2,* final diversity is zero which implies that the all the individuals are same.

I'm getting better results because of preserving diversity in survival selection using explicit approach ***Crowding***.

## PSEUDO CODE

```
SET pop size = 300
SET gen size = 200
SET seed size = 3

##################################################################################
#                               MAIN PROGRAM
##################################################################################

FOR each s in range seed size

  CALL gen.generate by passing pop size
   solution = Load "solution.json"
   main population = "population.json"

  FOR each g in range gen size

    #update poplation with fitness values
    population = CALL calFitness by passing main population

    sorted dict= CALL sort.sortPopulation by passing population, pop size

    all popupulation = copy sorted dict

    # ------------------- Parent Selection----------------------
    # Choose any selection method
    Sampled pop = CALL parentSel.uniformParentSelection by passing all popupulation, pop
                  size
    # ------------------- Crossover--------------------------
    #Choose any selection method
    Crossover pop = CALL cros.discreteCrossover by passing sampled pop, pop size
    cross_cal_pop = CALL calFitness by passing crossover pop to update fitness values
    # --------------------- Mutation --------------------------
    SIGMA = pop size / (g+(pop size/10)) #Variable SIGMA
    IF g is greater than gen size//1.5 and 3 out of last 10 elements in fit_max_gen list THEN
      SIGMA =5
    #Choose any Mutation method
    Mutated pop = CALL mut.nonUniformMutation by passing cross_cal_pop, SIGMA,
                   pop size
    mut_cal_pop = CALL calFitness by passing mutated pop to update fitness values

    # --------------------- Survival Selection ----------------------
    #Choose any survival selection method
    survival_pop = CALL survivalSel.survivorSelONCrowding by passing population,
                   mut_cal_pop, pop size
```

```
        sur_sort_pop = CALL sort.sortPopulation by passing survival_pop, pop size to sort based
                       on fitness
        main_population = COPY sur_sort_pop

    DUMP in "final_population.json" JSON file

##############################################################################################
#                                  Uniform Random Parent Selection
##############################################################################################

DEF uniform Parent Selection (all population, pop size)

    Shuffle the parents and store in same variable
    RETURN all population

##############################################################################################
#                                  Discrete Uniform Crossover
##############################################################################################

DEF discrete Crossover (sampled pop, pop size)

    Select Two individuals from the sampled pop
    And based on the probabilities interchange x and y with corresponding x, y of the orther
individual based on the l, w.
    Store it in pop cross variable
    RETURN pop cross

##############################################################################################
#                                           Non-Uniform Mutation
##############################################################################################

DEF Non-Uniform Mutation (population, SIGMA, pop size)

    SET Count = 0
    WHILE count less THAN pop size
        ind = randomly select an individual from population
        SET rot_rate = 0.4
        SET mut_rate = 0.2
        IF (SIGMA == 5):
            SET rot_rate = 0.2
            SET mut_rate = 0.4
        SET Mean = 0.5
        FOR p IN pieces
            IF rot_rate Greater than random value between (0,1)
                Change l and w in that piece
```

```
    Population = Updated the changed piece
    FOR p IN pieces
      IF rot_rate Greater than random value between (0,1)
        x= Add random using (mu, sigma) and round to integer
        y= Add random using (mu, sigma) and round to integer

    Population = Updated the changed piece

    INCREMENT count by 1
  RETURN population

################################################################################
#                          Survival Selection Using Crowding
################################################################################

DEF survivor Selection based on Crowding (parents pop, offsprings pop, pop size)
  SET pop count = 0
  WHILE pop count LESS THAN pop size
    SORT ALL Puzzle pieces based on the l^2 + b^2 values and STORE them different parent
    and offspring variables
    direct_dis1 = Find distance between parent1_puzzle and offspring1_puzzle
    direct_dis2 = Find distance between parent2_puzzle and offspring2_puzzle
    cross_dis1 = Find distance between parent1_puzzle and offspring2_puzzle
    cross_dis2 = Find distance between parent2_puzzle and offspring1_puzzle

    IF (direct_dis1+direct_dis2) LESS THAN (cross_dis1+cross_dis2)
      IF parent1 fitness GREATER THAN offspring1 fitness
        STORE offsprings_pop1
      ELSE
        STORE parents_pop1
      IF parent2 fitness GREATER THAN offspring2 fitness
        STORE offsprings_pop2
      ELSE
        STORE parents_pop2

    ELSE
      IF parent1 fitness GREATER THAN offspring2 fitness
        STORE offsprings_pop2
      ELSE
        STORE parents_pop1
      IF parent2 fitness GREATER THAN offspring1 fitness
        STORE offsprings_pop1
      ELSE
        STORE parents_pop2
    INCREMENT pop count BY 2
  RETURN survivor pop
```

# Part B – AUTOMATA MACHINE

## Problem Description

Given a randomly generated set of rules, an 8-bit initial state, and an 8-bit goal state. We need to design and write a GA that finds the set of rules that will transform the initial state to the goal state after some number of passes.

## Methodology

### I. Representation

**Bit String Representation** – The individual we are dealing are having binary numbers.

### II. Population and Population Size

For the desired population size n, an initial state, a goal state, and n sets up rules table will be generated. The initial population will be stored as JSON file **automata-population.json**. The individuals in the population are also JSON Strings with the following keys and values:

| | |
|---|---|
| **Initial State** | 8-Bit binary number |
| **Goal State** | 8-Bit binary number |
| **Rules Table** | 5-bit truth table with output values of 0, 1, 2 or 3 |

*Table 4*

Population Size I have taken is **80**.

The final population is stored in the **final_automata-population.json** file.

### III. Fitness Evaluation

Fitness of an individual is calculated by *Minimum Edit Distance* (***MED***) between the final and goal state. A *lower* MED means a *better* fitness value.

### IV. Parent Selection

**Uniform Parent Selection** – It is unbiased and every has the same probability to be selected

## V. Variation Operations

### Crossover

**Uniform Crossover** – Each gene randomly changed based on the probabilities for one child and the other child is inverse copy of gene with the first.

### Mutation

**Uniform Mutation** - By altering each rule independently with a probability.

## VI. Survivor Selection

**Fitness based selection** – Selecting the best individuals on fitness values for parent population size.

# Results

Below are some of the different approaches I implemented for finding the best parameters.

| | ALG-1 (Best) | ALG-2 | ALG-3 | ALG-4 |
|---|---|---|---|---|
| **Symbolic Parameters** | | | | |
| **Representation** | Bit-String | Bit-String | Bit-String | Bit-String |
| **Parent Selection** | Uniform Random | Exponential Rank | Uniform Random | Uniform Random |
| **Recombination** | Uniform | n-point | Uniform | Uniform |
| **Mutation** | Uniform | Uniform | Uniform | Uniform |
| **Survivor Selection** | Fitness-Proportionate | Fitness-Proportionate | Fitness-Proportionate | Fitness-Proportionate |
| **Numeric Parameters** | | | | |
| **Population Size** | 80 | 80 | 100 | 50 |
| **Generation Size** | 50 | 50 | 50 | 30 |
| **Mutation Rate** | 0.2 | 0.2 | 0.9 | 0.2 |
| **Crossover Rate** | 0.8 | 0.8 | 0.1 | 0.5 |
| **Pass Size** | 5 | 10 | 10 | 1 |
| **Results** | | | | |
| **Minimum Avg Fitness** | 0 | 0 | 1 | 1 |
| | | | | |

*Table 5* - The above table has 2 EAs, with 3 instances for one of them. Out of them, ALG-1 has the best performance when compared with others.

Below are the plots for the best ALG-1 where three different inputs are taken.

## Plot

I have plotted for three different inputs, where mainly each input has *first individual of the initial population Input* (***Table-a***), *first individual of the final population generated* (***Table-b***), *Solution using table-b during each pass and step* (***Table-c),*** *Minimum fitness value per each generation* (***fig-e***) and *Average fitness value per each generation* (***fig-f***).
For each plotted figures ***e*** & ***f***, the x-axis is Generations (out of 50) and y-axis is Fitness values. Rules for ***Table-c*** are there in the pseudo code.

### *Input 1*

**FIRST INDIVIDUAL OF THE INITIAL RADOMLY GENERATED POPULATION**

| | |
|---|---|
| **INITIAL** | '00111000' |
| **GOAL** | '10100001' |
| **RULES** | [['00000', 3], ['10000', 0], ['01000', 2], ['00100', 1], ['00010', 1], ['00001', 2], ['11000', 3], ['10100', 2], ['10010', 2], ['10001', 3], ['01100', 2], ['01010', 2], ['01001', 1], ['00110', 1], ['00101', 0], ['00011', 2], ['11100', 0], ['11010', 0], ['11001', 3], ['10110', 3], ['10101', 3], ['10011', 3], ['01110', 0], ['01101', 0], ['01011', 0], ['00111', 0], ['11110', 2], ['11101', 0], ['11011', 3], ['10111', 0], ['01111', 1], ['11111', 1]] |
| **FITNESS** | 62 |

*Table - a*

**FIRST INDIVIDUAL OF THE FINAL POPULATION OF LAST GENERATION**

| | |
|---|---|
| **INITIAL** | '00111000' |
| **GOAL** | '10100001' |
| **RULES** | [['00000', 2], ['10000', 1], ['01000', 3], ['00100', 2], ['00010', 2], ['00001', 1], ['11000', 0], ['10100', 1], ['10010', 2], ['10001', 0], ['01100', 1], ['01010', 1], ['01001', 0], ['00110', 0], ['00101', 0], ['00011', 0], ['11100', 0], ['11010', 1], ['11001', 2], ['10110', 1], ['10101', 2], ['10011', 0], ['01110', 0], ['01101', 2], ['01011', 2], ['00111', 1], ['11110', 2], ['11101', 2], ['11011', 0], ['10111', 2], ['01111', 0], ['11111', 1]] |
| **FITNESS** | 0 |

*Table - b*

| PASS | STEP | RULE | STATE |
|---|---|---|---|
| **INITIAL STATE** | | | '00111000' |
| **0** | 0 | 1 | '00111000' |
| | 1 | 0 | '00101000' |
| | 2 | 0 | '00100000' |
| | 3 | 0 | '00100000' |
| | 4 | 1 | '00100010' |
| | 5 | 2 | '0010001' |
| | 6 | 1 | '1010001' |
| | 7 | 0 | '1010001' |
| | | | '1010001' |
| **1** | 0 | 1 | '1010001' |

| | | | |
|---|---|---|---|
| | 1 | 3 | '10100001' |
| **GOAL STATE** | | | '10100001' |

*Table - c*

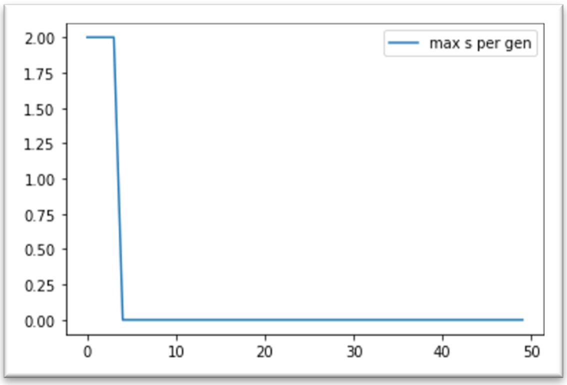


*fig-e*

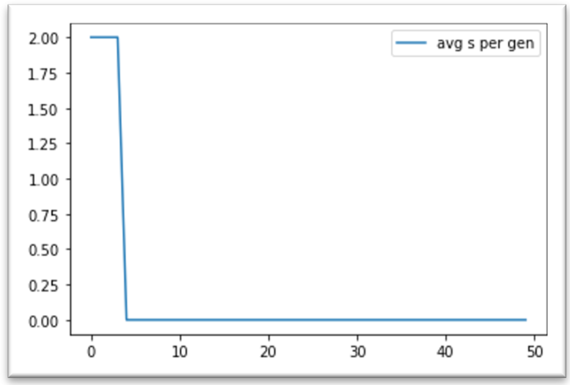


*fig-f*

## ***Input 2***

**FIRST INDIVIDUAL OF THE INITIAL RADOMLY GENERATED POPULATION**

| | |
|---|---|
| **INITIAL** | '11010111' |
| **GOAL** | '11101100' |
| **RULES** | [['00000', 2], ['10000', 2], ['01000', 1], ['00100', 3], ['00010', 2], ['00001', 1], ['11000', 0], ['10100', 2], ['10010', 3], ['10001', 2], ['01100', 1], ['01010', 1], ['01001', 2], ['00110', 3], ['00101', 1], ['00011', 1], ['11100', 3], ['11010', 0], ['11001', 0], ['10110', 2], ['10101', 2], ['10011', 1], ['01110', 0], ['01101', 0], ['01011', 0], ['00111', 2], ['11110', 0], ['11101', 2], ['11011', 2], ['10111', 3], ['01111', 3], ['11111', 0]] |
| **FITNESS** | 4 |

*Table - a*

**FIRST INDIVIDUAL OF THE FINAL POPULATION OF LAST GENERATION**

| | |
|---|---|
| **INITIAL** | '11010111' |
| **GOAL** | '11101100' |
| **RULES** | [['00000', 1], ['10000', 0], ['01000', 3], ['00100', 1], ['00010', 2], ['00001', 0], ['11000', 2], ['10100', 3], ['10010', 2], ['10001', 2], ['01100', 3], ['01010', 3], ['01001', 2], ['00110', 2], ['00101', 0], ['00011', 3], ['11100', 1], ['11010', 3], ['11001', 2], ['10110', 2], ['10101', 1], ['10011', 3], ['01110', 2], ['01101', 3], ['01011', 3], ['00111', 1], ['11110', 1], ['11101', 1], ['11011', 1], ['10111', 0], ['01111', 0], ['11111', 1]] |
| **FITNESS** | 0 |

*Table - b*

| PASS | STEP | RULE | STATE |
|---|---|---|---|
| **INITIAL STATE** | | | '11010111' |
| **0** | 0 | 3 | '110010111' |
| | 1 | 1 | '110010111' |
| | 2 | 3 | '1100100111' |
| | 3 | 0 | '1100100011' |
| | 4 | 0 | '1100100001' |
| | 5 | 1 | '1100100001' |
| | 6 | 1 | '1100100001' |
| | 7 | 1 | '1100100001' |
| | | | '1100100001' |
| **1** | 0 | 2 | '110100001' |
| | 1 | 2 | '11100001' |
| | 2 | 1 | '11100001' |
| | 3 | 3 | '111000001' |
| | 4 | 0 | '111000001' |
| | 5 | 0 | '111000001' |
| | 6 | 3 | '1110000001' |
| | 7 | 1 | '1110000001' |
| | 8 | 2 | '111000000' |
| | 9 | 1 | '111000000' |
| | | | '111000000' |
| **2** | 0 | 1 | '111000000' |
| | 1 | 2 | '11100000' |
| | 2 | 0 | '11100000' |
| | 3 | 1 | '11101000' |
| | 4 | 1 | '11101100' |
| **GOAL STATE** | | | '11101100' |

*Table – c*

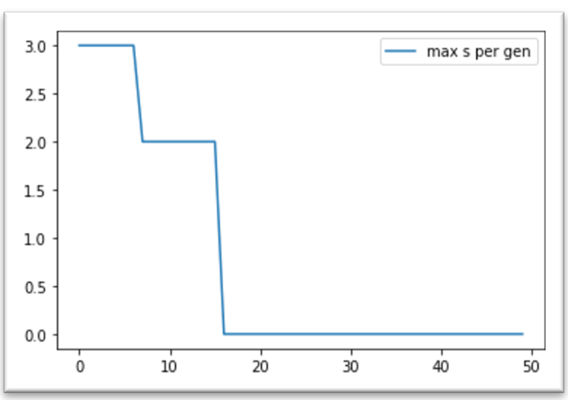


*fig-e*

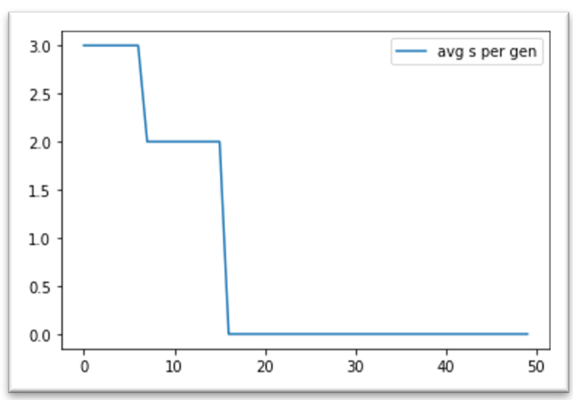


*fig-f*

## *Input 3*

**FIRST INDIVIDUAL OF THE INITIAL RADOMLY GENERATED POPULATION**

| | |
|---|---|
| **INITIAL** | '10001011' |
| **GOAL** | '11000111' |
| **RULES** | [['00000', 3], ['10000', 0], ['01000', 1], ['00100', 2], ['00010', 3], ['00001', 1], ['11000', 0], ['10100', 1], ['10010', 2], ['10001', 3], ['01100', 0], ['01010', 3], ['01001', 0], ['00110', 2], ['00101', 0], ['00011', 2], ['11100', 1], ['11010', 3], ['11001', 0], ['10110', 0], ['10101', 1], ['10011', 2], ['01110', 0], ['01101', 3], ['01011', 3], ['00111', 3], ['11110', 2], ['11101', 0], ['11011', 0], ['10111', 3], ['01111', 1], ['11111', 1]] |
| **FITNESS** | 24 |

*Table - a*

**FIRST INDIVIDUAL OF THE FINAL POPULATION OF LAST GENERATION**

| | |
|---|---|
| **INITIAL** | '10001011' |
| **GOAL** | '11000111' |
| **RULES** | [['00000', 1], ['10000', 2], ['01000', 3], ['00100', 0], ['00010', 0], ['00001', 3], ['11000', 2], ['10100', 3], ['10010', 0], ['10001', 0], ['01100', 2], ['01010', 2], ['01001', 2], ['00110', 0], ['00101', 3], ['00011', 0], ['11100', 1], ['11010', 3], ['11001', 3], ['10110', 2], ['10101', 3], ['10011', 0], ['01110', 1], ['01101', 3], ['01011', 1], ['00111', 1], ['11110', 3], ['11101', 1], ['11011', 0], ['10111', 2], ['01111', 3], ['11111', 0]] |
| **FITNESS** | 0 |

*Table - b*

| PASS | STEP | RULE | STATE |
|---|---|---|---|
| **INITIAL STATE** | | | '10001011' |
| **0** | 0 | 0 | '10001011' |
| | 1 | 0 | '10001011' |
| | 2 | 3 | '100011011' |
| | 3 | 1 | '100011111' |
| | 4 | 2 | '10001111' |
| | 5 | 1 | '10001111' |
| | 6 | 1 | '10001111' |
| | 7 | 2 | '1100111' |
| | | | '1100111' |
| **1** | 0 | 3 | '11000111' |
| **GOAL STATE** | | | '11000111' |

*Table - c*

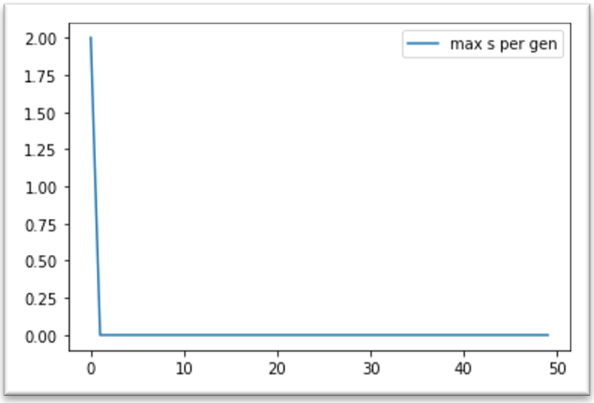


*fig-e*

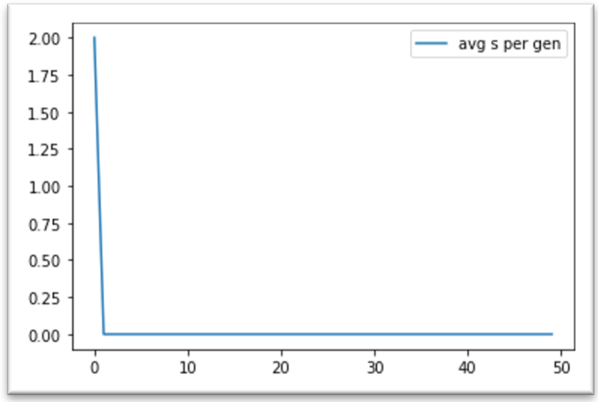


*fig-f*

These are the mean and standard deviation of both ***average*** and ***maximum*** fitness of an 80-individual population over 50 generations.

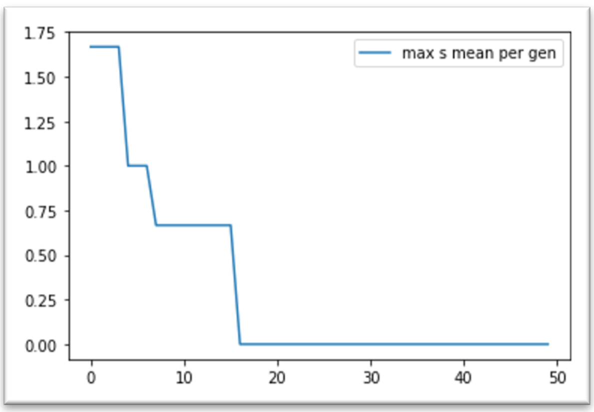


*Fig 5 – Mean of Minimum fitness values of 3 different inputs*

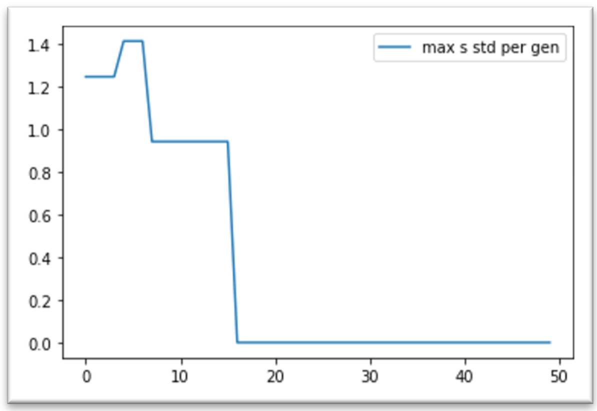


*Fig 6 – Standard Deviation of Minimum fitness values of 3 different inputs*

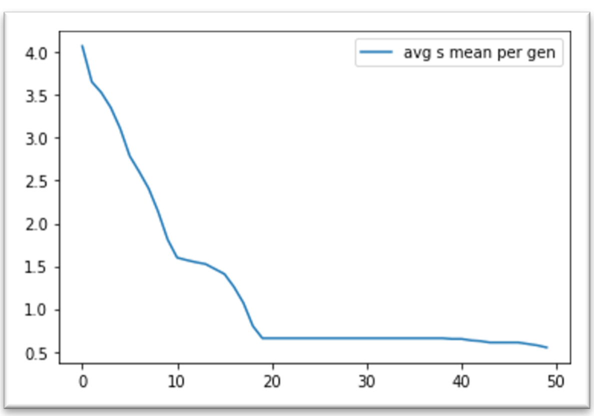


*Fig 7 – Mean of Average fitness values of 3 different inputs*

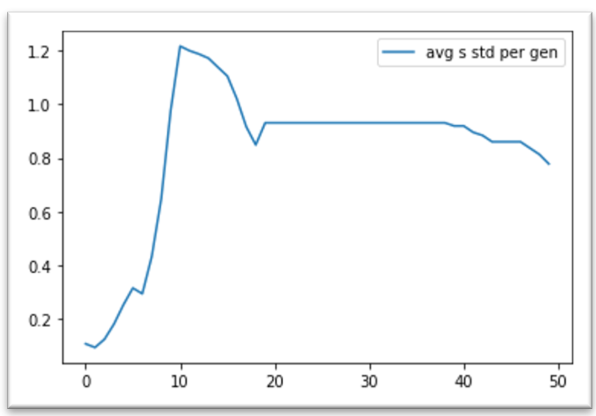


*Fig 8 – Standard Deviation of Average fitness values of 3 different inputs*

From the *mean* plots we can say that the minimum average fitness we are achieving is around 0 which is Goal state is transformed from the initial state with in 5 passes. And from the decreasing *standard deviation* plots we can say that the variation between individuals is going low.

## Suggestions

### *Modifications*

The changes I would like to do is in the Fitness calculation. Now, we are just using MED to find fitness, but we can include passes minimization, i.e. the minimum number of passes or states and steps. By this we can find more precise solution.

### *Diversity*

I am measuring the diversity by using the following function which calculates the distance between all individuals:

$$D = \sum_{i=1}^{N} \sqrt{\sum_{j=1}^{i-1} d(g_i, g_j)}$$

Where $d(g_i, g_j)$ is the distance (I'm using Euclidean) between individuals $g_i$ and $g_j$.

| | **INITIAL DIVERSITY** | **FINAL DIVERSITY** |
|---|---|---|
| **INPUT 1** | 56.88455 | 35.366444 |
| **INPUT 2** | 57.057079 | 30.0580 |
| **INPUT 3** | 56.53926 | 41.29616 |

*Table 6 – Diversity measured using above formula for the 3 inputs which I used in Results*

We can see that the diversity decreases from initial population to population generated from final generation. i.e. the individuals in the final population are somewhat closely related.
By this we increase the generations so that we may get better solution.

## PSEUDO CODE

```
SET pop size = 80
SET gen size =50
SET seed size = 3
SET pass size = 5
SET mut rate = 0.2
SET cross rate = 0.8

############ Rules #############

# 0 - replace the middle value with 0
# 1 - replace the middle value with 1
# 2 - delete the middle value
# 3 - replicate the middle value with left

####################################################################################
#                                              MAIN PROGRAM
####################################################################################

FOR each s in range seed size

    CALL gen.generate by passing pop size
     main population = "automata-population.json"

    FOR each g in range gen size

        #update poplation with fitness values
        Updated population = CALL updatePopulation by passing main population

        Sorted Pop= CALL sort.sortPopulation by passing updated population, pop size

        # ------------------- Parent Selection----------------------
        # Choose any selection method
        Sampled pop = CALL Select.uniformParentSelection by passing all Sorted pop, pop size
        # ------------------- Crossover--------------------------
        #Choose any selection method
        Crossover pop = CALL cros.UniformCrossover by passing sampled pop, pop size
        updatedPopulationFitness1 = CALL updatePopulation by passing crossover pop to update
                                            fitness values
        # --------------------- Mutation --------------------------
        Mutated pop = CALL mut.Mutation by passing updatedPopulationFitness1, pop size
        updatedPopulationFitness2 = CALL updatePopulation by passing mutated pop to update
                                            fitness values
        # --------------------- Survival Selection ----------------------
        #Choose any survival selection method
```

```
        Survival pop = CALL select. survivorSelONFitness by passing population,
                      updatedPopulationFitness2 , pop size
        main_population = COPY Survival pop
    DUMP in "final-automata-population.json" JSON file

####################################################################################
#                        Uniform Random Parent Selection
####################################################################################

DEF uniform Parent Selection (all population, pop size):

    Shuffle the parents and store in same variable
    RETURN all population

####################################################################################
#                              Uniform Crossover
####################################################################################

DEF Uniform Crossover (sampled pop, pop size, cross rate)

    Select Two individuals from the sampled pop
    And based on the probabilities interchange rules from individual to other and vise-versa
    Store the new rules in pop cross variable
    RETURN pop cross

####################################################################################
#                              Uniform Mutation
####################################################################################

DEF Mutation (Population, mut_rate, pop size)

       Randomly SELECT an individual from population
       Based the mutation rate and probabilities change the rules
       STORE back in population
        RETURN Population

####################################################################################
#                      Survival Selection Based on Fitness
####################################################################################

DEF survivor Selection ON Fitness (parents pop, offspring pop, pop size)

    UPDATE the parent population with offspring's population
    SORT them based on the fitness values
    SELECT top pop size and UPDATE orginal population
    RETURN population
```

# THE END