EE 496
INTRODUCTION TO COMPUTATIONAL INTELLIGENCE

HOMEWORK
#2

Assigning Students to Courses Using Genetic Algorithm

Ayberk Aydın

1875699

I. INTRODUCTION

In this homework, we are going to assign students to courses that has limited capacity (8) concerning their preferences of priority for the courses. The student group may be crowded or not and the popularity of the courses may be balanced or unbalanced. Genetic algorithm is used to assign the students to the courses in the optimal way in terms of the preference of students. Also, effect of different metaparameters for genetic algorithms are observed

II. METHODS

For using genetic algorithm to student assignment, 4 functions must be defined.

- How an initial population is created, which data structure is used for individuals.
- How the fitness values of individuals are calculated.
- How a mutations occurs.
- How a child is produced by two parents (Crossover)

1. **Creation of Initial Population and Type of Individuals**

   For defining individuals, 20x4 (or 30x4) matrices are used where rows correspond to students and columns correspond to courses; for defining populations, MATLAB cell arrays of these matrices are used. Initial population is created randomly by creating a 20x4 (or 30x4) zero matrix and assigning "1" to a random index of each row. If an individual corresponds to a distribution that exceeds course capacity, it is removed from population. This routine occurs until the initial population fills up to its capacity, which is determined by the user.

2. **Fitness (and Cost) Function**

   In contrast to the convention, MATLAB's Genetic Algorithm toolbox is a **minimization** tool of the fitness function. So, the lower fitness function means better individuals using this toolbox. Hence, fitness function is selected to be sum of all elements of elementwise product of an individual and preference matrix. Moreover, cost function is selected to be sum of every student's distance from its first selection. For example if every student is assigned to its favorite course except 2 students; one is assigned to its second favorite course and the other is assigned to its third favorite course. Then the cost function is (2-1) + (3-1) = 3. For the selection function, tournament selection is used.

```
fitnessValue=fitness(individual,preferenceMatrix):
        a=elementwiseProduct(individual,preferenceMatrix)
        fitnessValue=sumAllElements(a);
        return fitnessValue
```

Figure 1: Pseudo-code for fitness function

3. **Mutation Function**

Mutation is simply small alterations of chromosomes of individuals. In our case, mutation function is determined to be a swap between two students from different courses. By this way, mutation of a proper (not exceeding any course capacity) individual will also be proper.

```
mutated=mutation(individual):
        p1=random(size)
        p2=random(size)
        mutated=individual.swapRows(p1,p2)
        return mutated
        end
```

Figure 2: Pseudo-code for mutation function

4. **Crossover Function**

Crossover is an operation that takes two (or more) individuals as parents and results in a new individual that resembles its parents. In our case, crossover function is single point crossover. First n row of first parent and last 20-n (or 30-n) row of second parent is merged to make a child where n is a randomly selected number. If the child is faulty (exceeds course capacity), the child is omitted and crossover is repeated for different n.

```
offspring=crossover(parent1,parent2)
        p1=random(size)
        offspring(1:p1)=parent1(1:p1)
        offpring(p1:end)=parent2(p2:end)
        return offspring
        end
```

Figure 3:Pseudo-code for crossover function

After determining these functions, genetic algorithm has run and performance as measured with respect to

- How quick the program converged
- How small the value of converged cost function.

III. EXPERIMENTAL RESULTS

For determining the effects of parameters, crossover fraction and population size is altered iteratively. Effect of mutation rate is also observed by this way since the implementation of Genetic Algorithm Toolbox keeps crossover rate and mutation rate dependent of each other. A fraction (crossover rate) of parents crossed over to make new generation members and the other fraction (mutation rate=1-crossover rate) of parents are mutated to make other members of new generation in MATLAB's Genetic Algorithm Toolbox.

Before the experiments, some parameters are predetermined by trial and error to give decent results in reasonable time. Also, the worst case input parameters are used (crowdedness, popularity).The following parameters turned out to be reasonable in terms of time and cost function;

- Tournament size/Population size = 0.1
- Max generations to run = 150
- Crowded students
- Unbalanced popularity

Experimented parameters are;

- 5 Crossover rates(mutation rate) = [0, 0.2, 0.5, 0.8, 1] (1-crossover rate for mutation rate)
- 4 Population sizes = [10, 50, 200, 500]

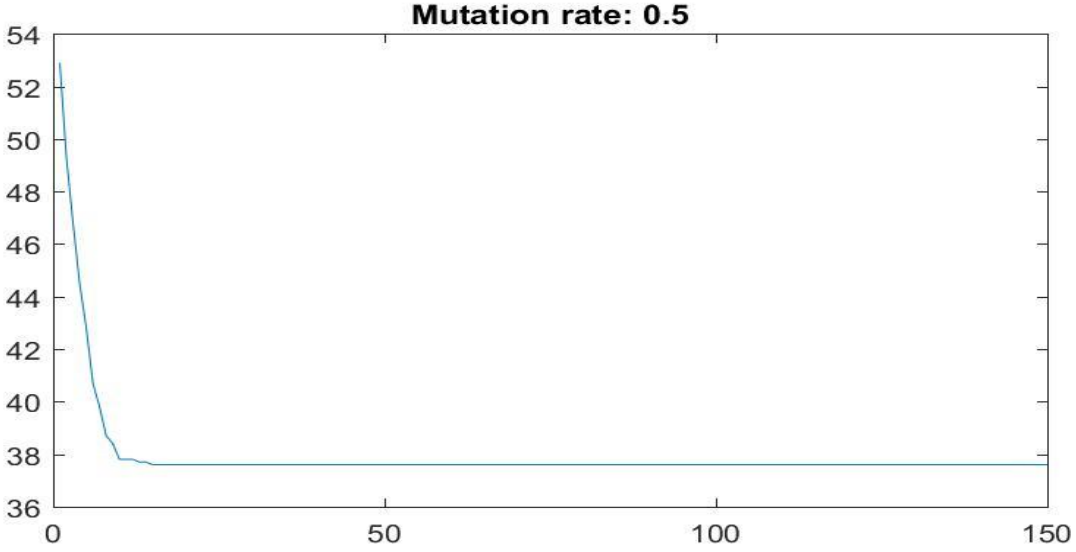
Experiments are done in the following way;

Firstly; 10 random preference matrix is created and for each experimented parameter value pair. Then, genetic algorithm is run 10 times for each preference matrix and best individual vs generation plot is obtained with respect to the average of this iterations for each crossover rate, population size pair, which means genetic algorithm is run for;

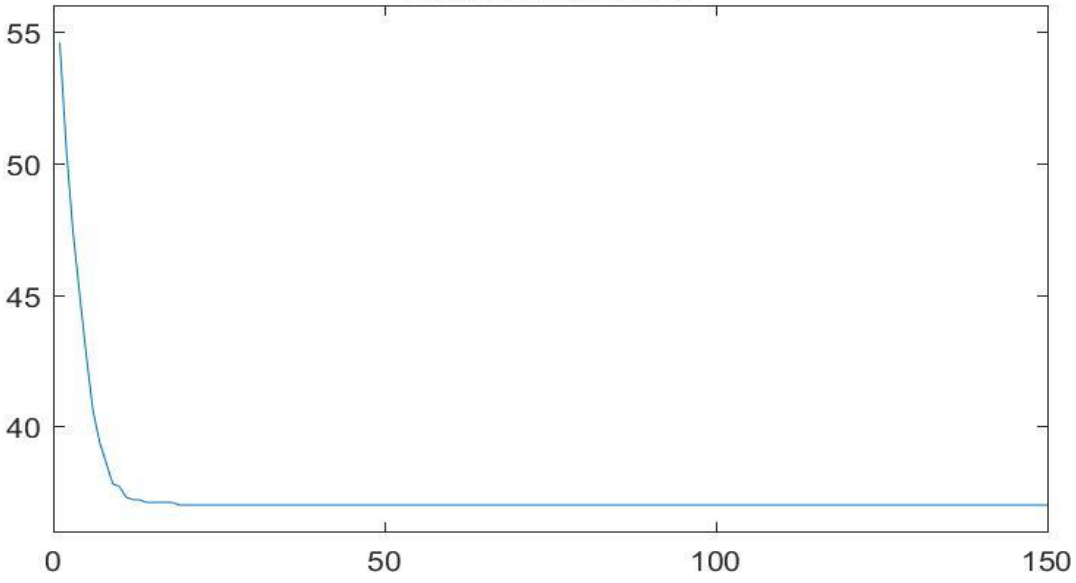
5(crossover rates) x 4 (population sizes) x 10(iterations).

times. Figures 4(a-u) show the results of the experiments and Figure 5 shows an input-output pair for the best obtained parameters from the experiment.

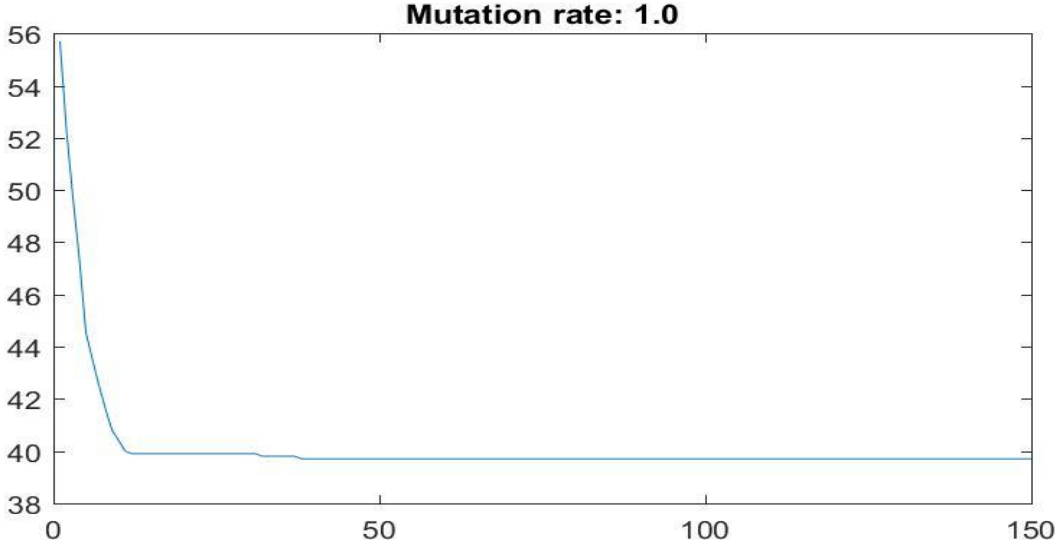
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 37.60.
Population Size: 500
Crossover rate: 0.5
Mutation rate: 0.5



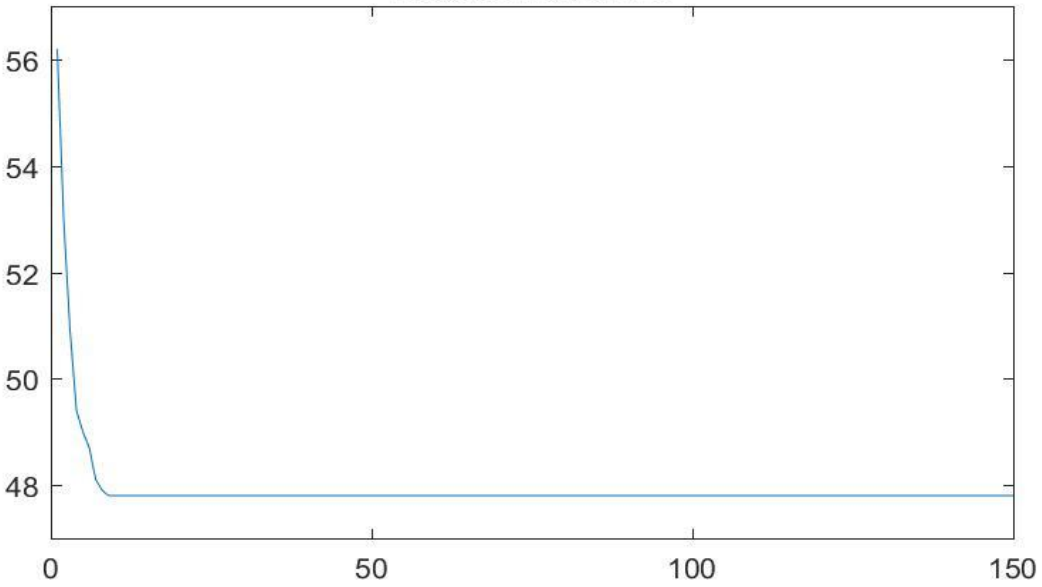
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 37.00.
Population Size: 500
Crossover rate: 0.2
Mutation rate: 0.8



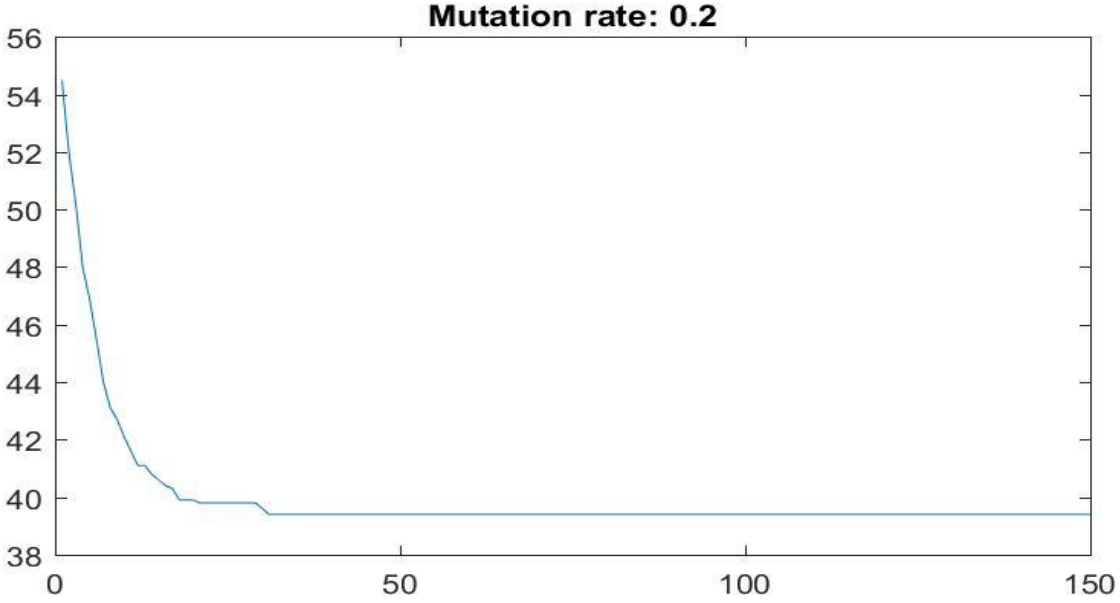
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.70.
Population Size: 500
Crossover rate: 0.0
Mutation rate: 1.0



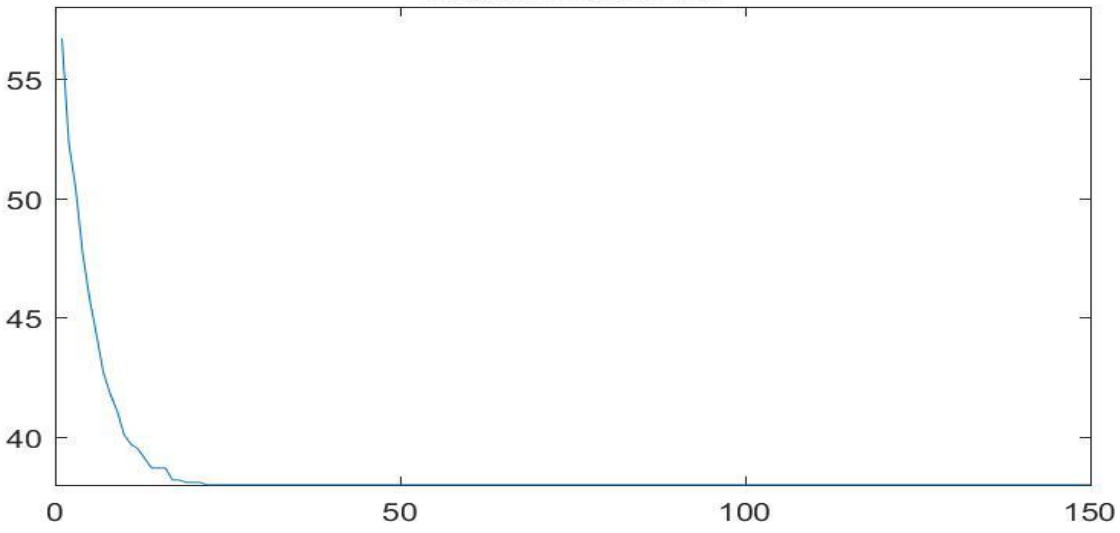
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 47.80.
Population Size: 200
Crossover rate: 1.0
Mutation rate: 0.0



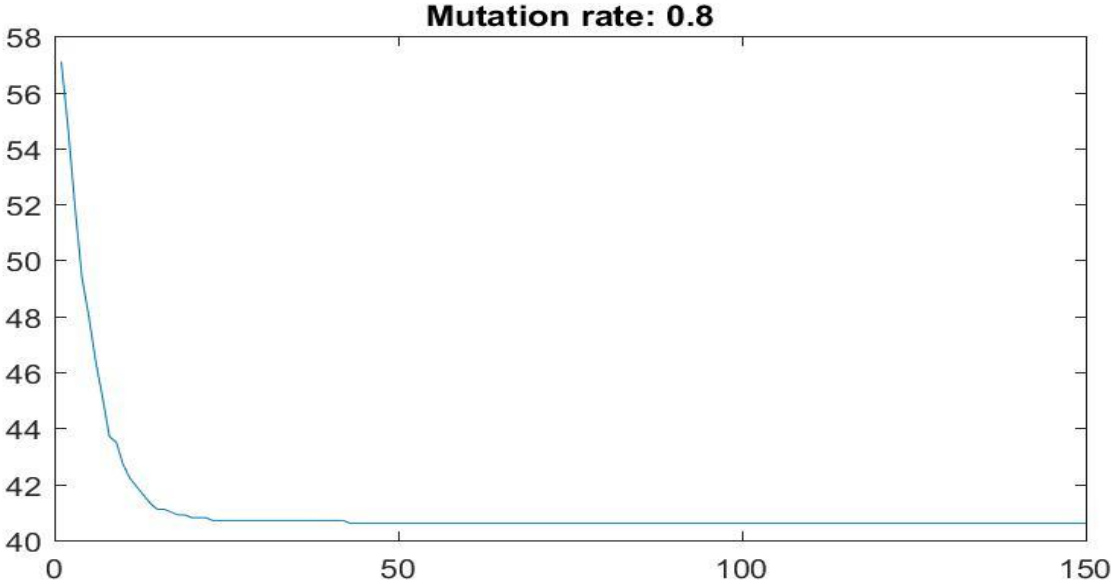
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.40.
Population Size: 200
Crossover rate: 0.8
Mutation rate: 0.2



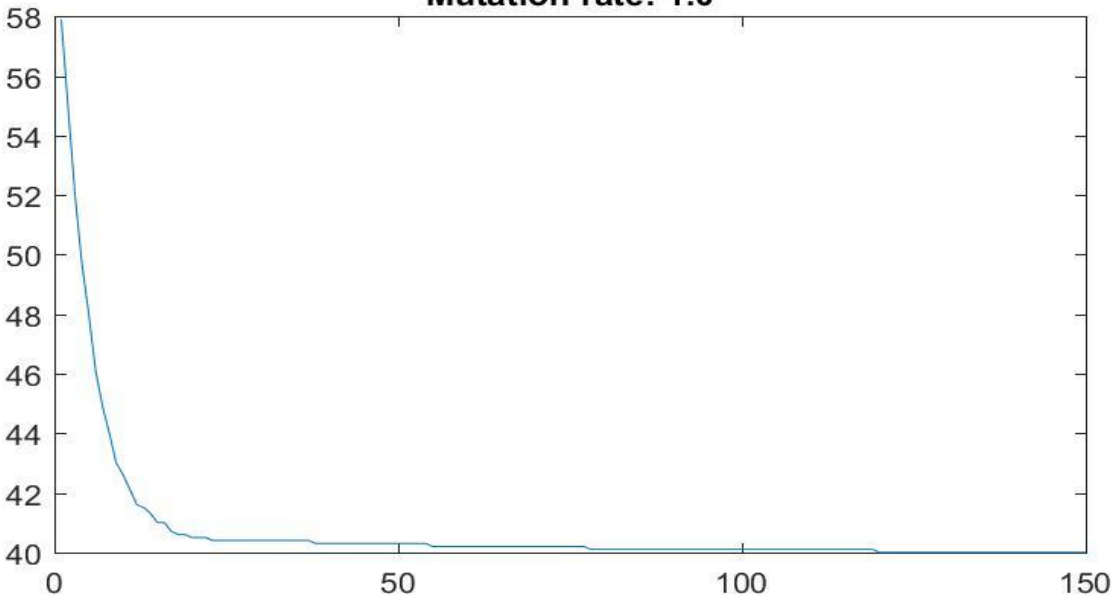
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 38.00.
Population Size: 200
Crossover rate: 0.5
Mutation rate: 0.5



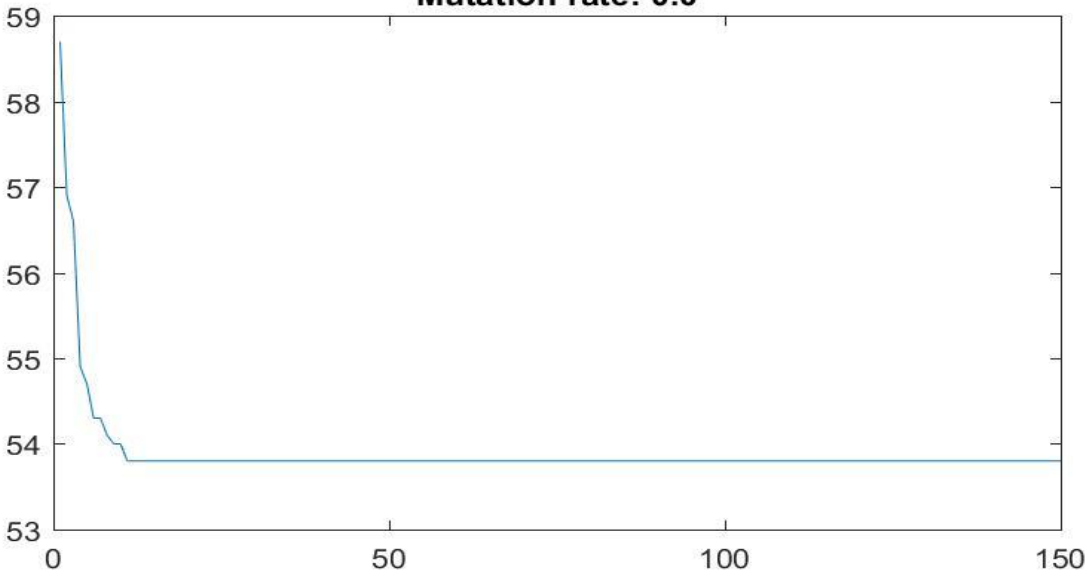
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 40.60.
Population Size: 200
Crossover rate: 0.2
Mutation rate: 0.8



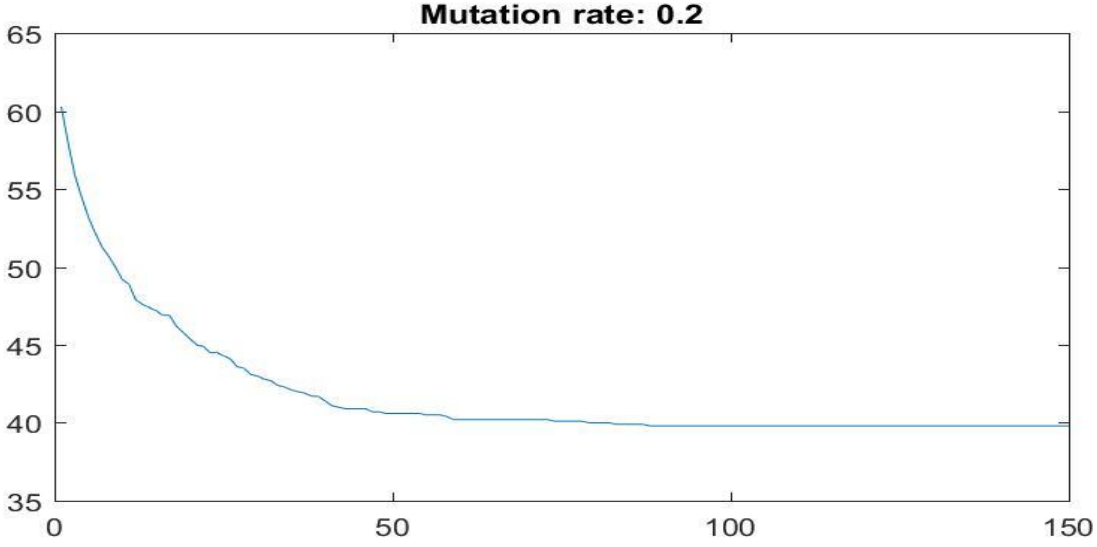
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 40.00.
Population Size: 200
Crossover rate: 0.0
Mutation rate: 1.0

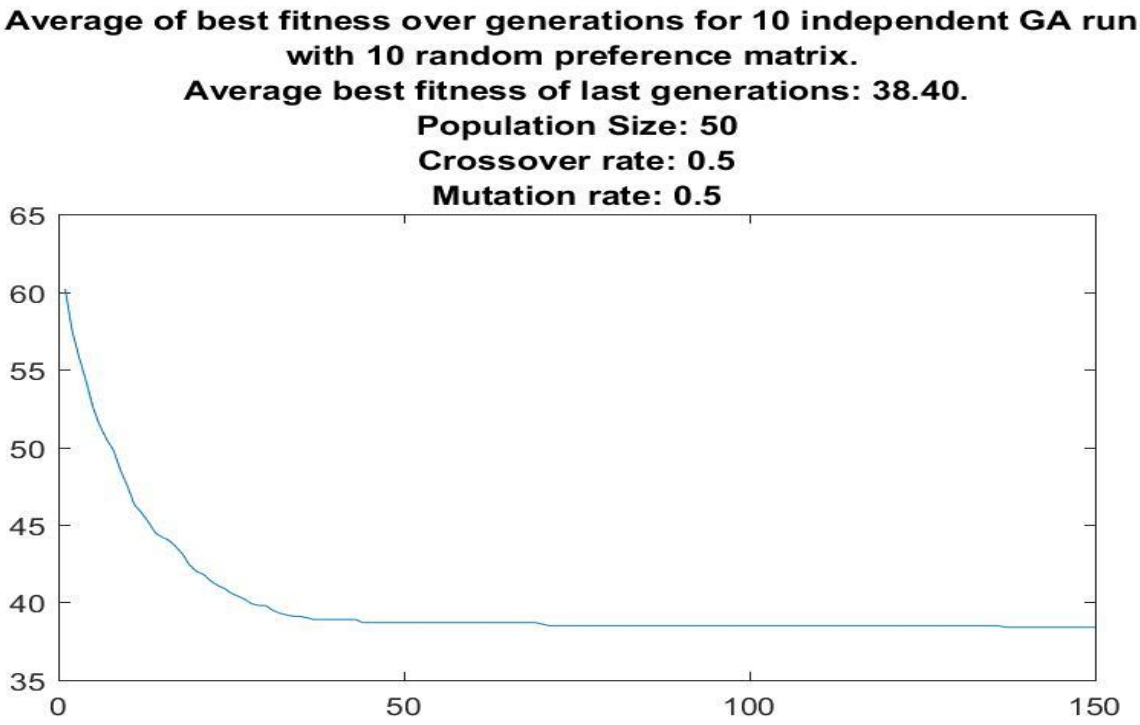


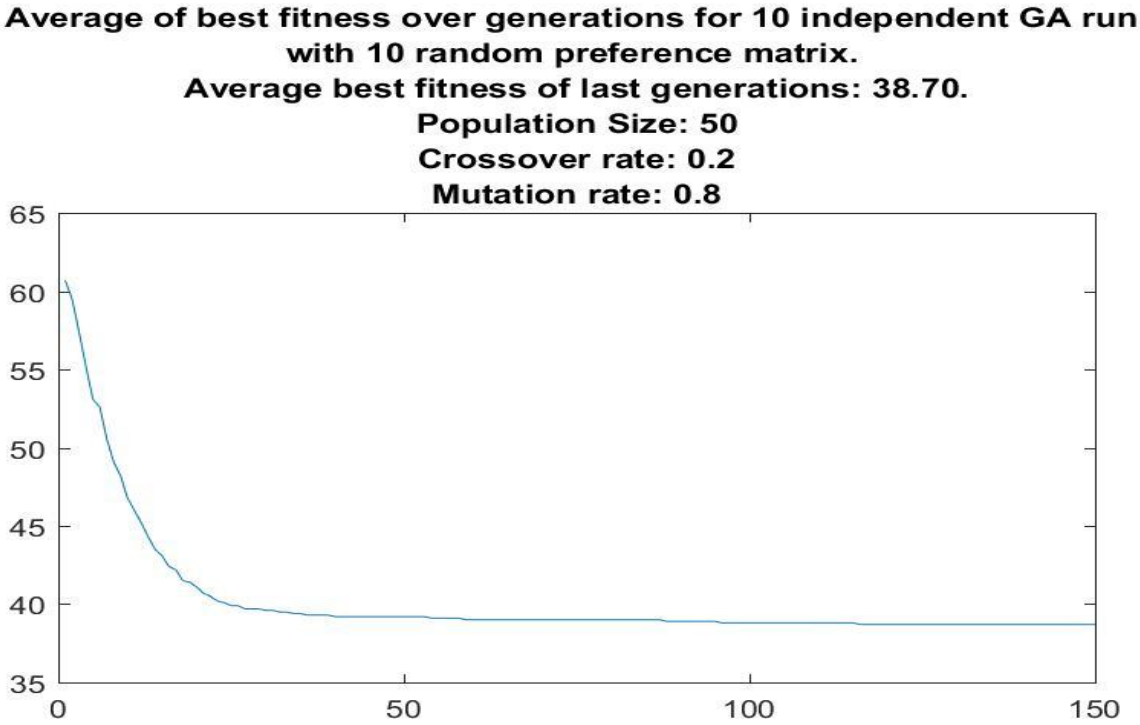
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 53.80.
Population Size: 50
Crossover rate: 1.0
Mutation rate: 0.0

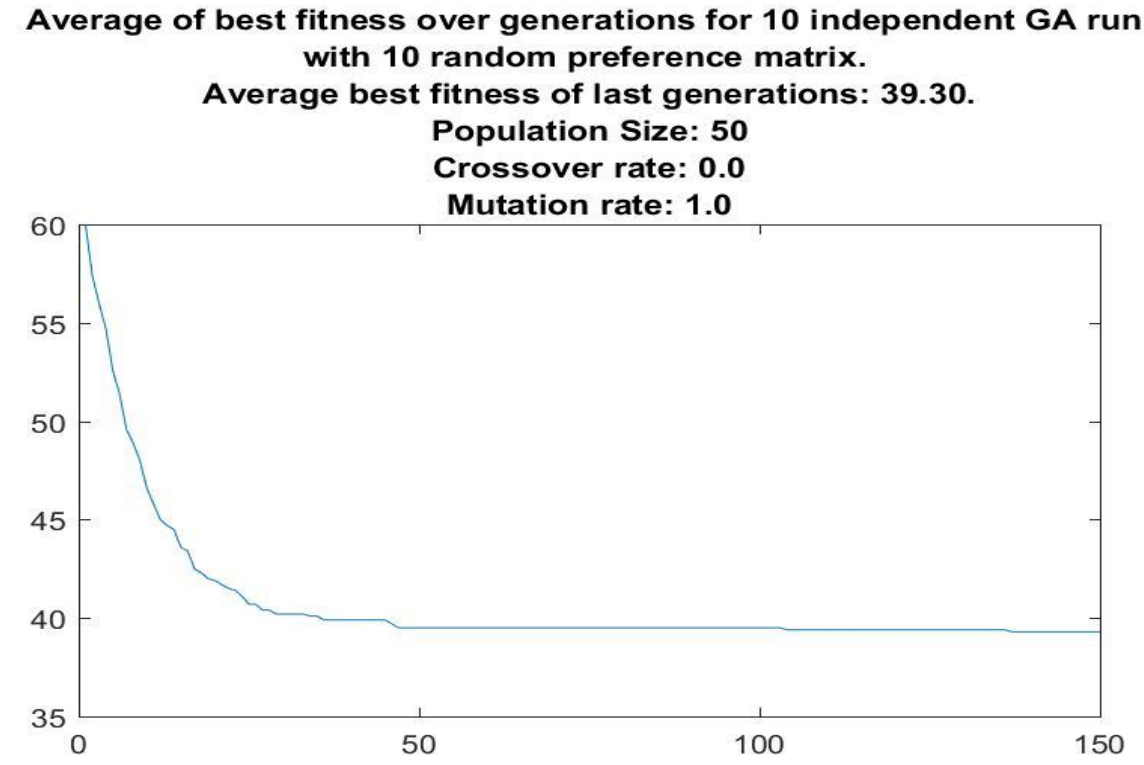


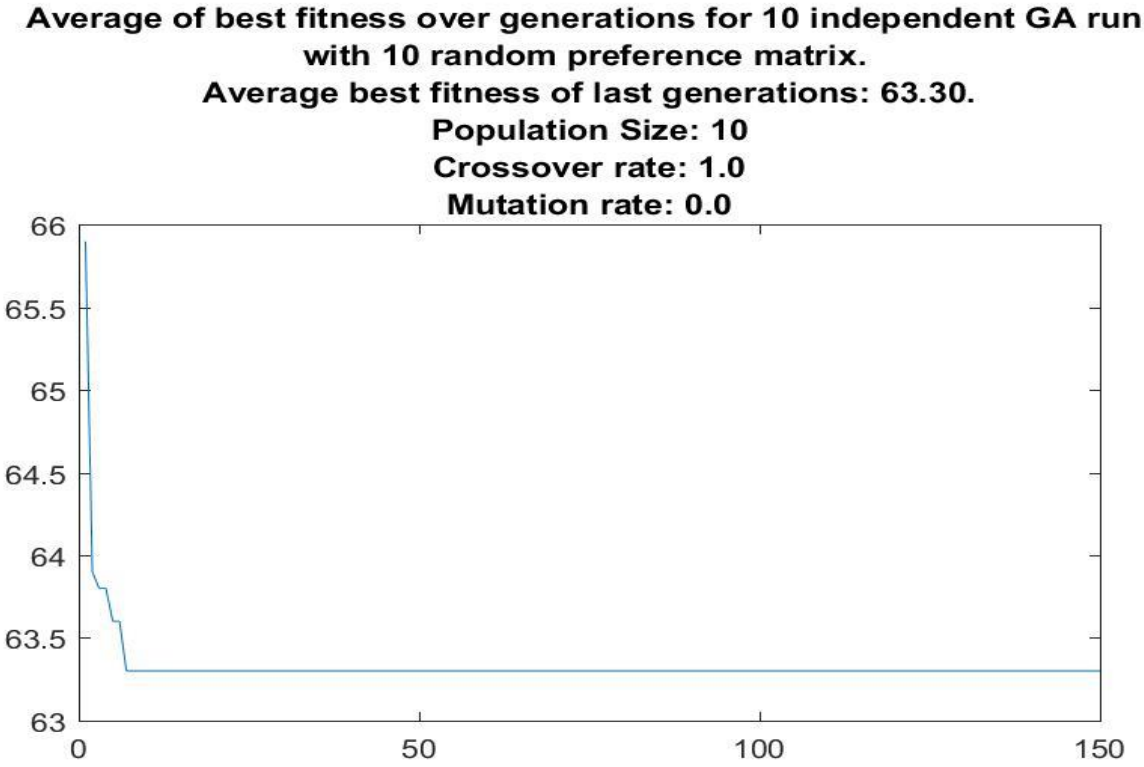
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.80.
Population Size: 50
Crossover rate: 0.8
Mutation rate: 0.2



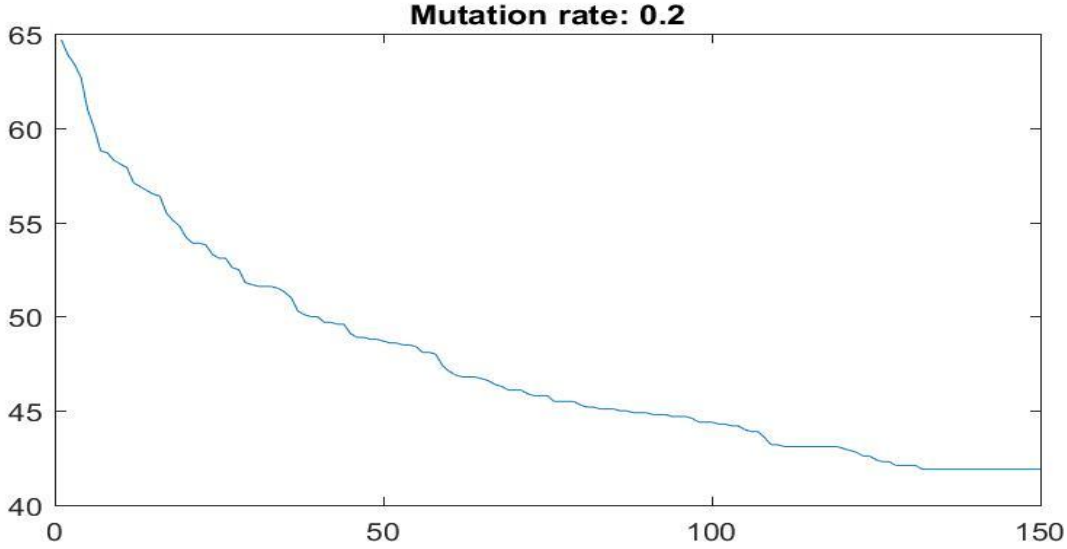

Average of best fitness over generations for 10 independent GA run
with 10 random preference matrix.
Average best fitness of last generations: 38.40.
Population Size: 50
Crossover rate: 0.5
Mutation rate: 0.5
65
60
55
50
45
40
35
0
50
100
150


Average of best fitness over generations for 10 independent GA run
with 10 random preference matrix.
Average best fitness of last generations: 38.70.
Population Size: 50
Crossover rate: 0.2
Mutation rate: 0.8
65
60
55
50
45
40
35
0
50
100
150

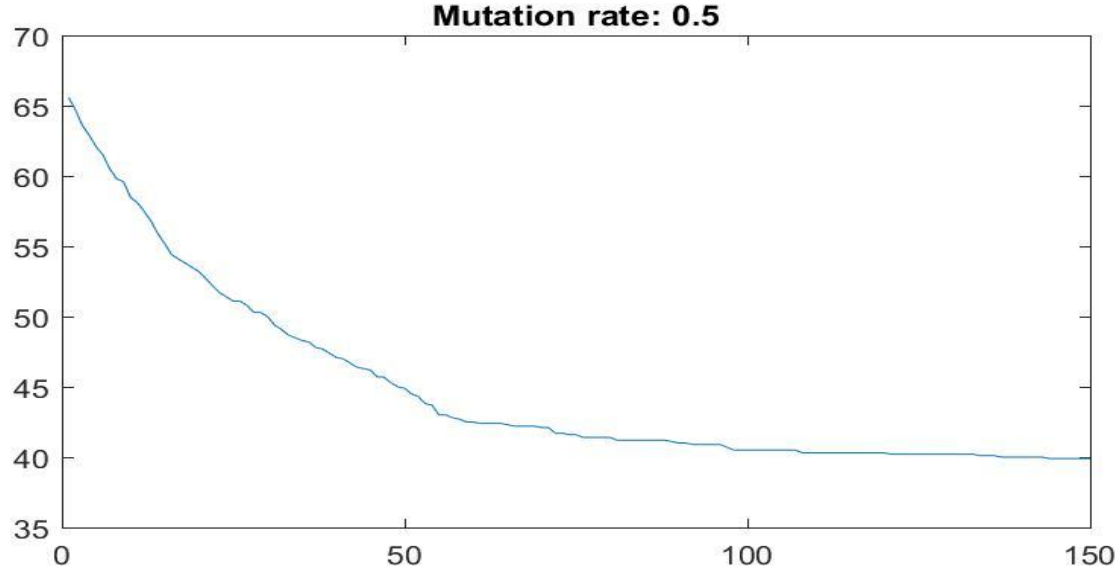

Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.30.
Population Size: 50
Crossover rate: 0.0
Mutation rate: 1.0
60
55
50
45
40
35
0
50
100
150


Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 63.30.
Population Size: 10
Crossover rate: 1.0
Mutation rate: 0.0
66
65.5
65
64.5
64
63.5
63
0
50
100
150

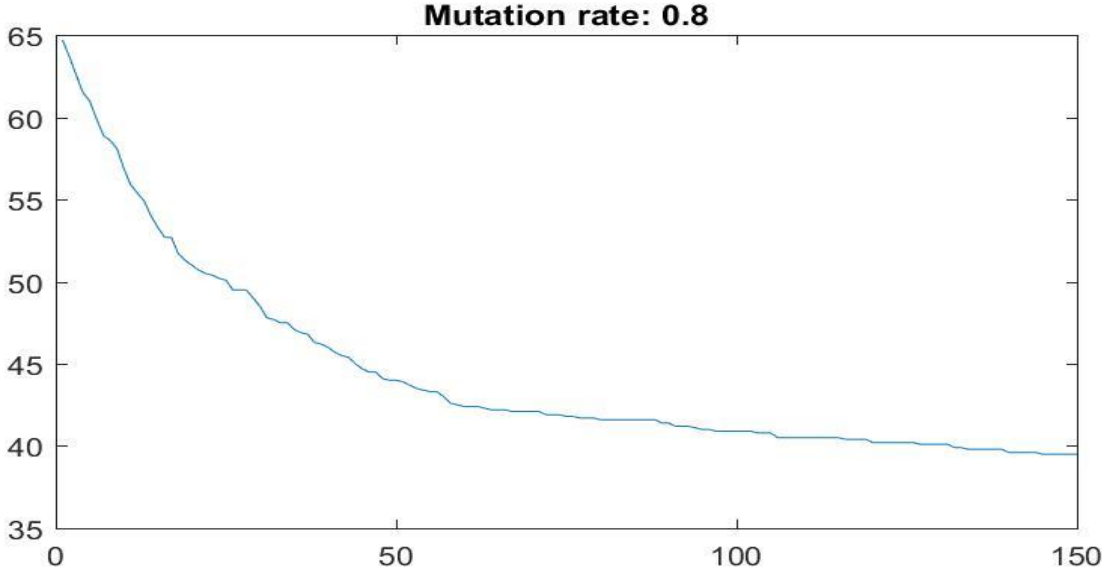
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 41.90.
Population Size: 10
Crossover rate: 0.8
Mutation rate: 0.2



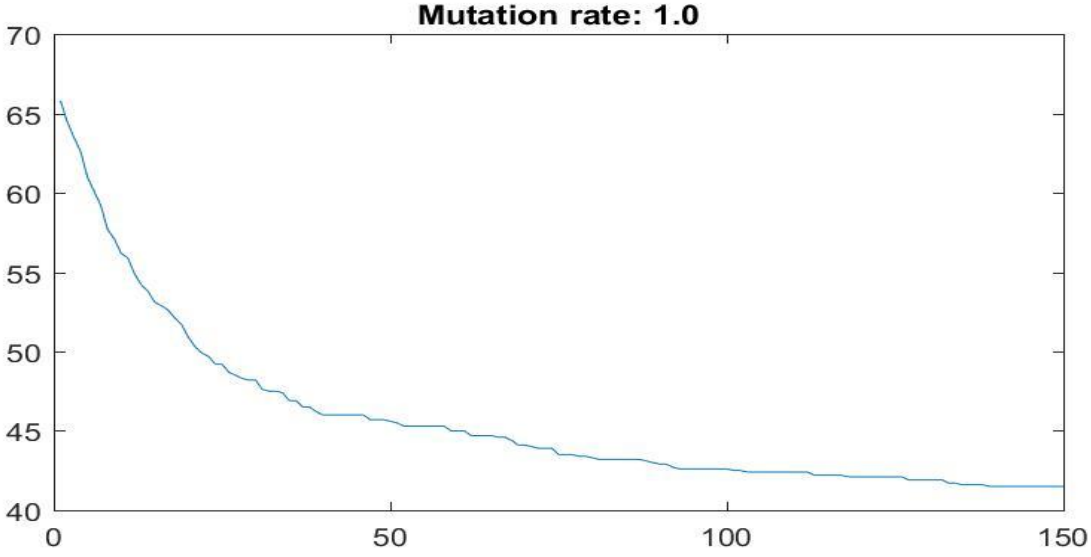
Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.90.
Population Size: 10
Crossover rate: 0.5
Mutation rate: 0.5

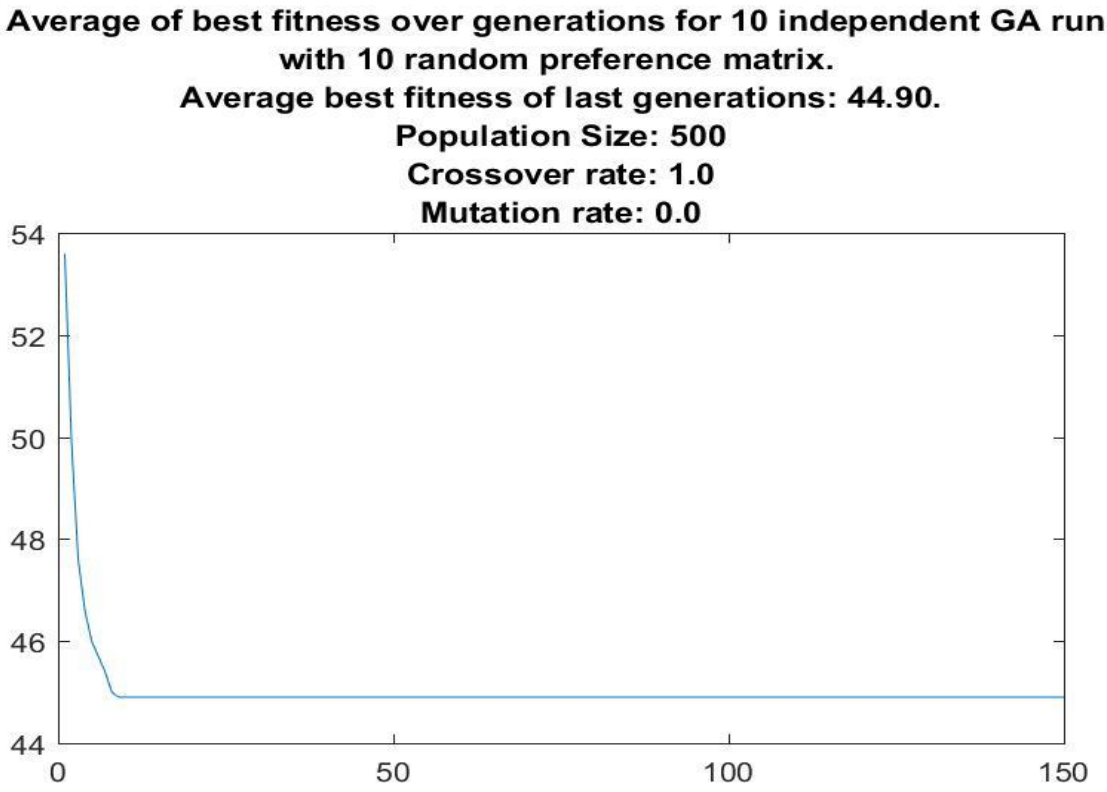


Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 39.50.
Population Size: 10
Crossover rate: 0.2
Mutation rate: 0.8



Average of best fitness over generations for 10 independent GA run with 10 random preference matrix.
Average best fitness of last generations: 41.50.
Population Size: 10
Crossover rate: 0.0
Mutation rate: 1.0





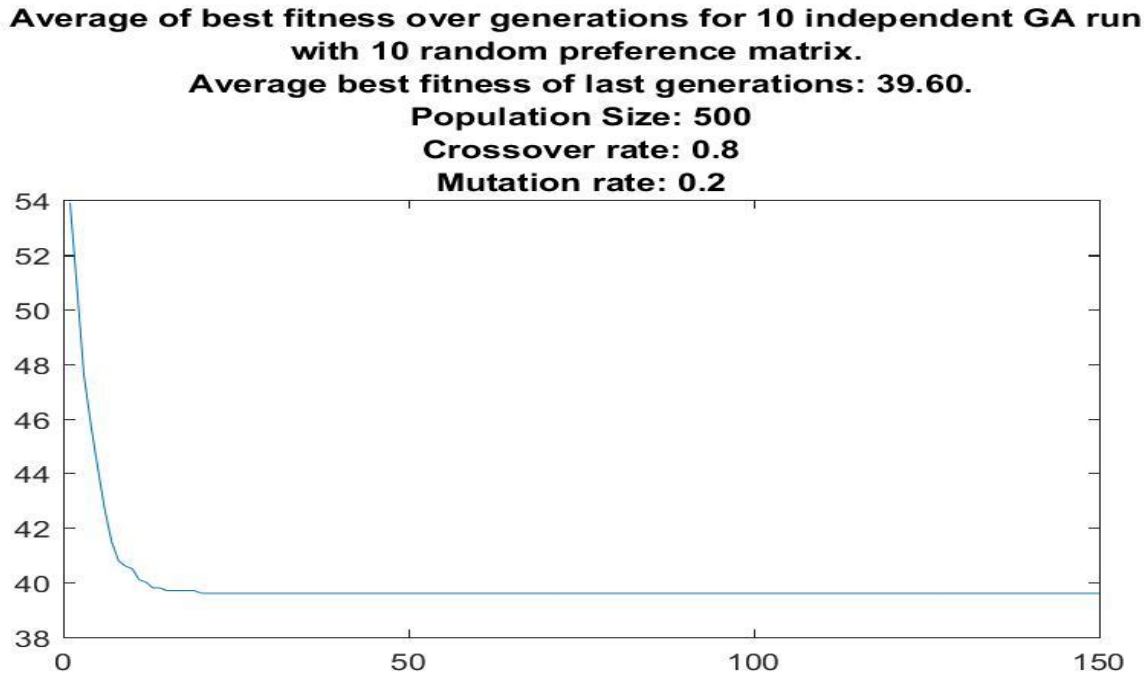


Figure 4(a-u): Plots of average of 10 iterations for each parameter pair.

By inspection of the experiment result plots, the following inference can be made.

- By increasing mutation rate (and decreasing crossover rate), convergence speed (in terms of generations) is decreased. However, populations converge to better individuals if the mutation rate is high. This makes sense since with increasing mutation rate, the algorithm tends to explore the search space and with increasing crossover rate, the algorithm uses its good genetic information to make new individuals.
- A crossover rate of 0.8 and mutation rate of 0.2 seems to have the ideal balance between exploration and exploitation.
- Increasing population size seems to have a beneficial effect to both convergence speed and converged fitness values. However, this is not the case since with a higher population size, more time is spent to process each generation. So, extremely large populations does not have a beneficial effect for genetic algorithms. A population size of 50 seems to yield best results in the shortest amount of time.

After determining the optimal metaparameters for genetic algorithm; they are used to assign 30 students to 4 courses with unbalanced popularity. Figure 5, 6, 7 show the preference matrix, output matrix of genetic algorithm and plot of best and mean fitness of the population for 150 generations.

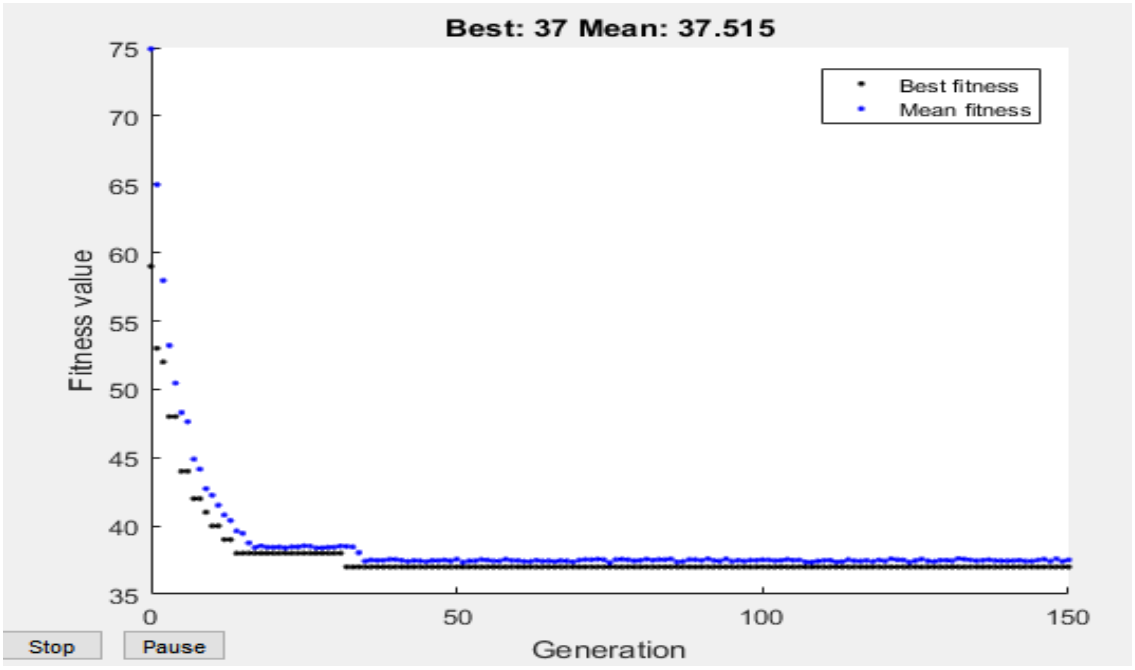


Figure 5: Best and mean fitness for 150 generations

Course1	Course2	Course3	Course4
4	1	2	3
2	4	1	3
2	4	1	3
4	1	3	2
1	2	3	4
1	3	4	2
1	3	4	2
2	4	1	3
2	3	1	4
2	3	4	1
2	4	1	3
2	3	1	4
1	2	4	3
1	3	4	2
1	2	3	4
1	3	2	4
1	3	4	2
2	3	1	4
1	3	2	4
3	2	1	4
1	2	4	3
1	3	2	4
3	2	1	4
3	2	4	1
2	4	3	1
1	2	4	3
2	1	3	4
4	3	1	2
2	1	3	4
1	2	3	4

Figure 6: Input preference matrix

Course1	Course2	Course3	Course4
1	0	0	0
0	2	0	0
0	0	0	1
0	1	0	0
0	0	0	2
1	0	0	0
0	0	0	2
1	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	2
0	0	0	1
0	1	0	0
1	0	0	0
0	0	1	0
0	0	1	0
0	1	0	0
1	0	0	0
0	0	1	0
0	0	0	2
0	1	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	0	0	2
0	0	1	0
0	0	1	0
1	0	0	0

Generation: 150
Cost: 6

Figure 7: Output matrix

IV. CONCLUSION

In this homework, a number of students are assigned to 4 limited-capacity courses considering their course preferences. For this task, genetic algorithm is used with MATLAB's Genetic Algorithm Toolbox.

Firstly; mutation, crossover, fitness and initial population generation functions are defined. Then, they are used to determine some reasonable metaparameters. Afterwards, crossover rate, mutation rate and population size metaparameters are experimented through iteration and their effects to convergence rate and converged fitness are observed.

Mutation rate is determined to be 0.2 and crossover rate is determined to be 0.8 for a balance between exploration of search space and exploitation of genetic information. Also, population size of 50 is determined to be optimal number (for 4 variables) for the quickest convergence to the optimal solutions.