EE496 Spring 2016-2017
Homework 2

Assigning Students to Courses using Genetic Algorithm

May 2017

Middle East Technical University

Electrical and Electronics Engineering Department

JeanPiere Demir

Contents

Contents **2**

1 Introduction **3**

2 Proposed Genetic Algorithm **3**

2.0.1 Definition of The Chromosome 3

2.0.2 Fitness Algorithm 3

2.0.3 Pseudo Code of the General Algorithm 4

3 Results **6**

4 Conclusion **24**

References **25**

1 Introduction

In this project , our purpose was to assign each on N students to one of K courses with limited capacity. Students indicates their preferences. Our main goal is to maximize the satisfaction of preferences and we have some restrictions while achieving this goal.

- Each student should be assigned to exactly one course
- Course capacities should not be exceeded

The purpose of the genetic algorithm is to find the best candidate for the solution,and after pass the new childs to next generation.

2 Proposed Genetic Algorithm

I tried to explain my general system in this part. The general system consists of ,

- The main algorithm
- Crossing over
- Mutation
- Roulette Wheel
- Tournament Selection

2.0.1 Definition of The Chromosome

To achieve a good and correct genetic algorithm we have to define the chromosome properly. Our chromosome is constructed with regards to uniform or nonuniform distribution ,and nonuniform constructed according to preference popularity array. So we can check our code for four different situations based populations.

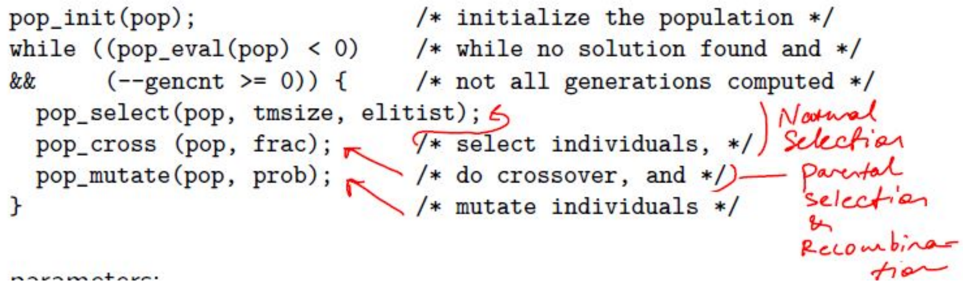
2.0.2 Fitness Algorithm

Before calculating fitness function , we have to check some constraints to decide about fitness function results.These constraints are ,

- If course capacity exceeded F = -2000
- Else it is $-1 * sum(abs(PreferenceMatrix - Chromosome))$

After checking the constraints , if we did not violate the conditions we calculate the fitness with regards to this equation $-1 * sum(abs(PreferenceMatrix - Chromosome))$. As we see fitness is a piece-wise function according to course capacity it will take different values.

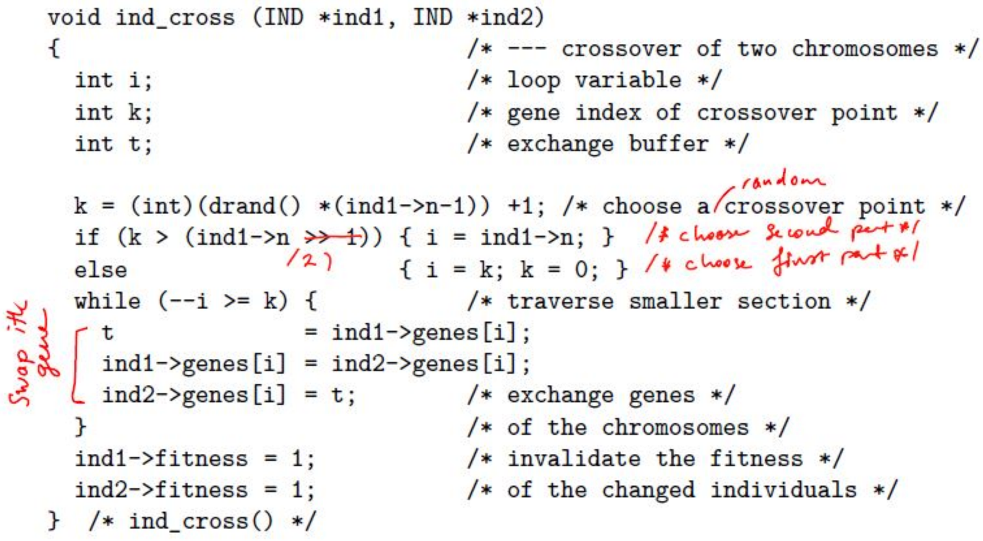
2.0.3 Pseudo Code of the General Algorithm



```
pop_init(pop);                  /* initialize the population */
while ((pop_eval(pop) < 0)      /* while no solution found and */
&&     (--gencnt >= 0)) {       /* not all generations computed */
  pop_select(pop, tmsize, elitist);
  pop_cross (pop, frac);        /* select individuals, */  ) Natural Selection
  pop_mutate(pop, prob);        /* do crossover, and */ ) —— Parental selection & Recombination
}                               /* mutate individuals */

parameters:
```

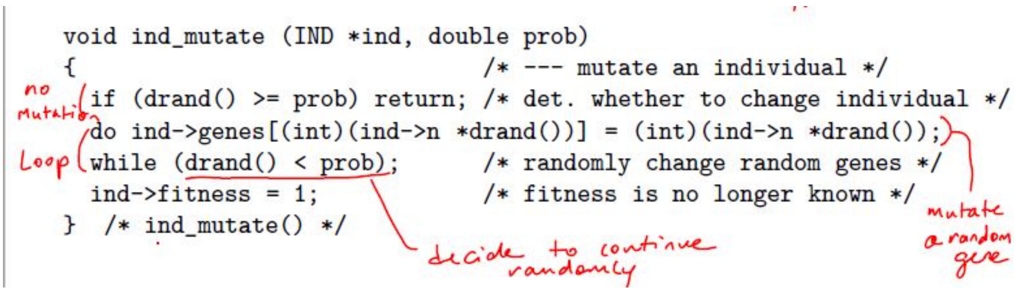
Figure 1: Pseudo Code of the General Algorithm



```
void ind_cross (IND *ind1, IND *ind2)
{                               /* --- crossover of two chromosomes */
  int i;                        /* loop variable */
  int k;                        /* gene index of crossover point */
  int t;                        /* exchange buffer */

  k = (int)(drand() *(ind1->n-1)) +1; /* choose a random crossover point */
  if (k > (ind1->n ~~>> 1~~)) { i = ind1->n; }  /* choose second part */
  else            /2)    { i = k; k = 0; }  /* choose first part */
  while (--i >= k) {            /* traverse smaller section */
Swap    t              = ind1->genes[i];
ith     ind1->genes[i] = ind2->genes[i];
gene    ind2->genes[i] = t;          /* exchange genes */
  }                             /* of the chromosomes */
  ind1->fitness = 1;            /* invalidate the fitness */
  ind2->fitness = 1;            /* of the changed individuals */
}  /* ind_cross() */
```

Figure 2: Pseudo Code of the Crossing Over



```
void ind_mutate (IND *ind, double prob)
{                               /* --- mutate an individual */
  if (drand() >= prob) return; /* det. whether to change individual */
  do ind->genes[(int)(ind->n *drand())] = (int)(ind->n *drand());
  while (drand() < prob);       /* randomly change random genes */
  ind->fitness = 1;             /* fitness is no longer known */
}  /* ind_mutate() */
```

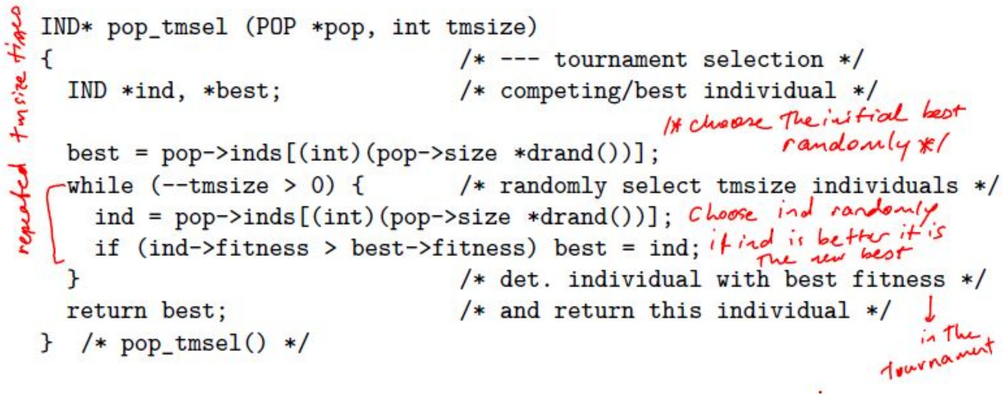
no mutation

Loop

decide to continue randomly

mutate a random gene

Figure 3: Pseudo Code of the Mutation



```
IND* pop_tmsel (POP *pop, int tmsize)
{                                /* --- tournament selection */
  IND *ind, *best;               /* competing/best individual */

  best = pop->inds[(int)(pop->size *drand())];
  while (--tmsize > 0) {         /* randomly select tmsize individuals */
    ind = pop->inds[(int)(pop->size *drand())];
    if (ind->fitness > best->fitness) best = ind;
  }                              /* det. individual with best fitness */
  return best;                   /* and return this individual */
}  /* pop_tmsel() */
```

repeated tmsize times

/* choose The initial best randomly */

Choose ind randomly

if ind is better it is The new best

in The Tournament

Figure 4: Pseudo Code of the Tournament Selection

3 Results

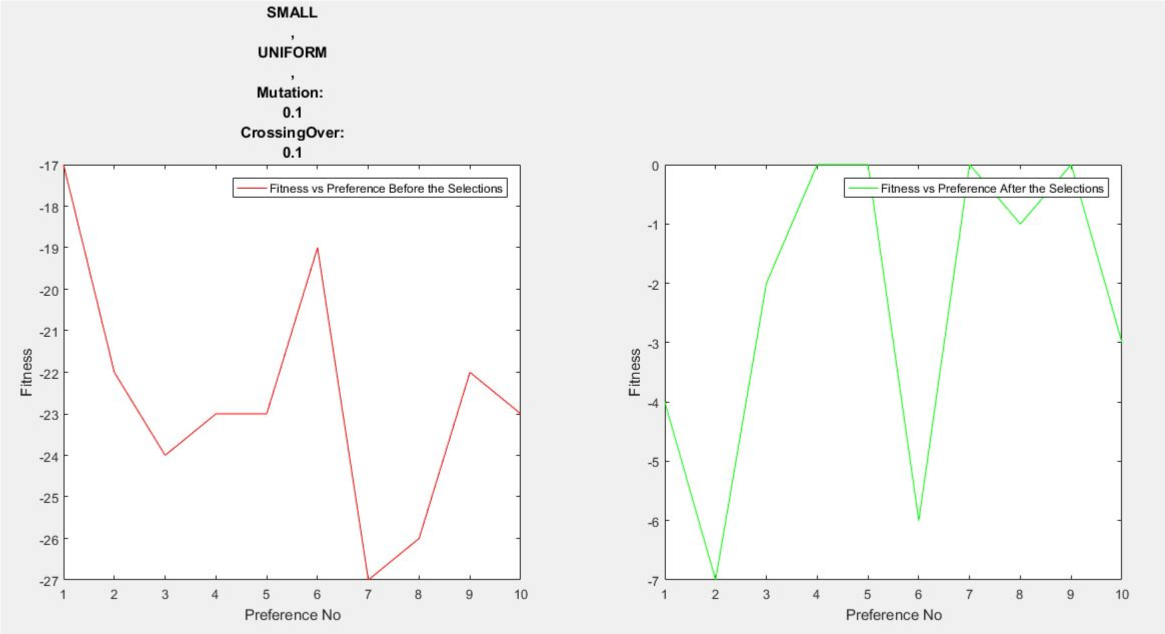


Figure 5: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.1, MutationRate:0.1

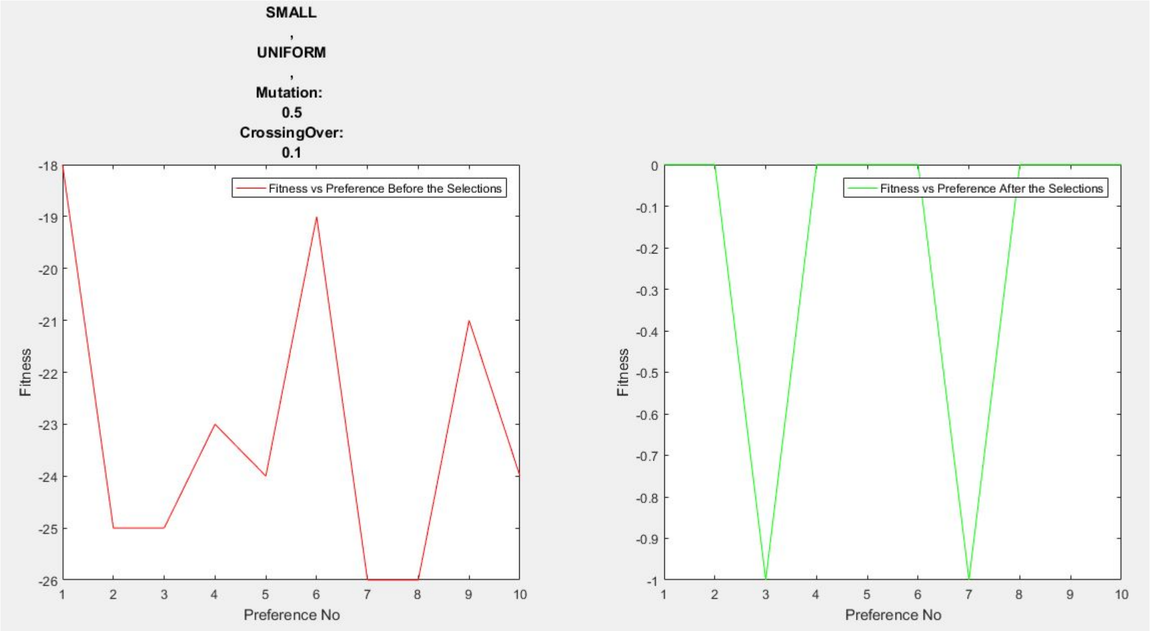


Figure 6: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.1, MutationRate:0.5

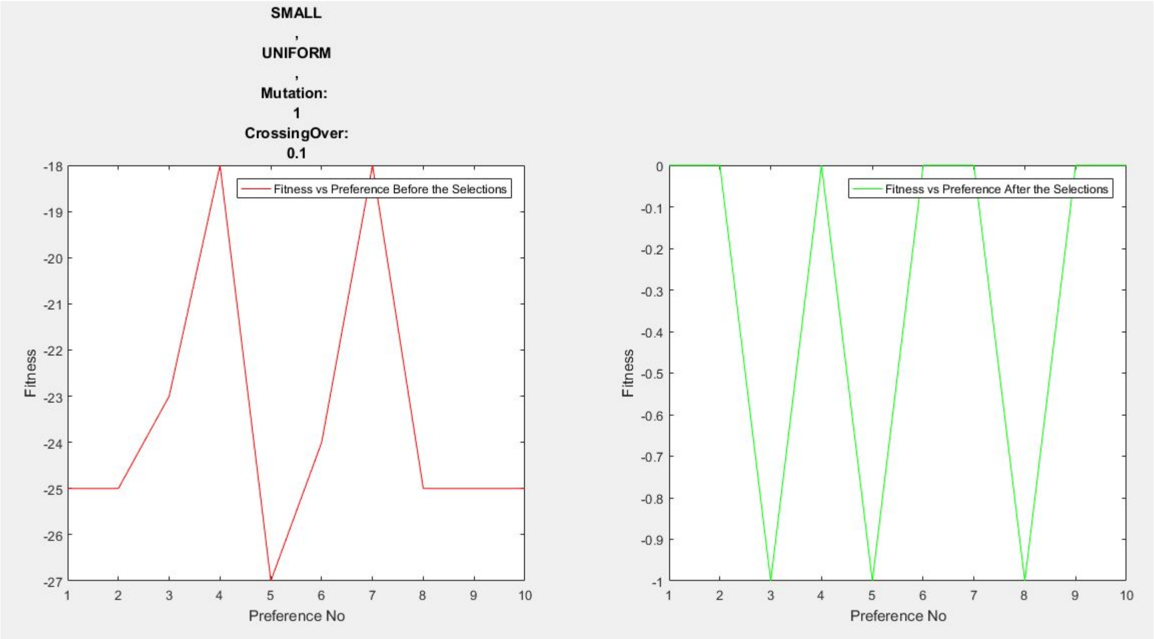


Figure 7: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.1, MutationRate:1

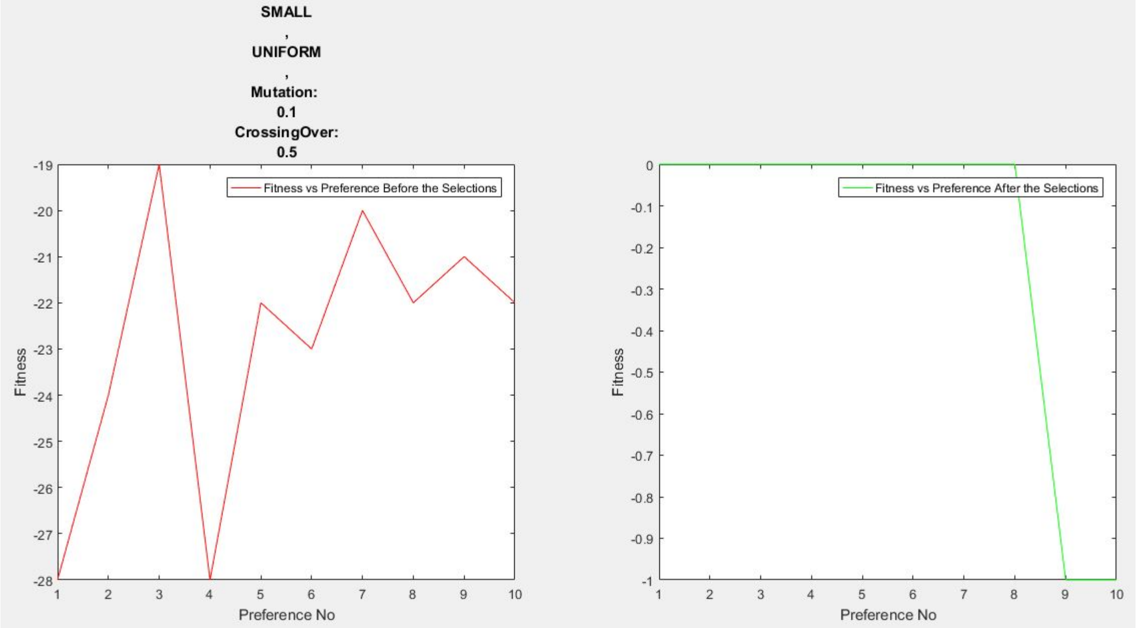


Figure 8: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.5, MutationRate:0.1

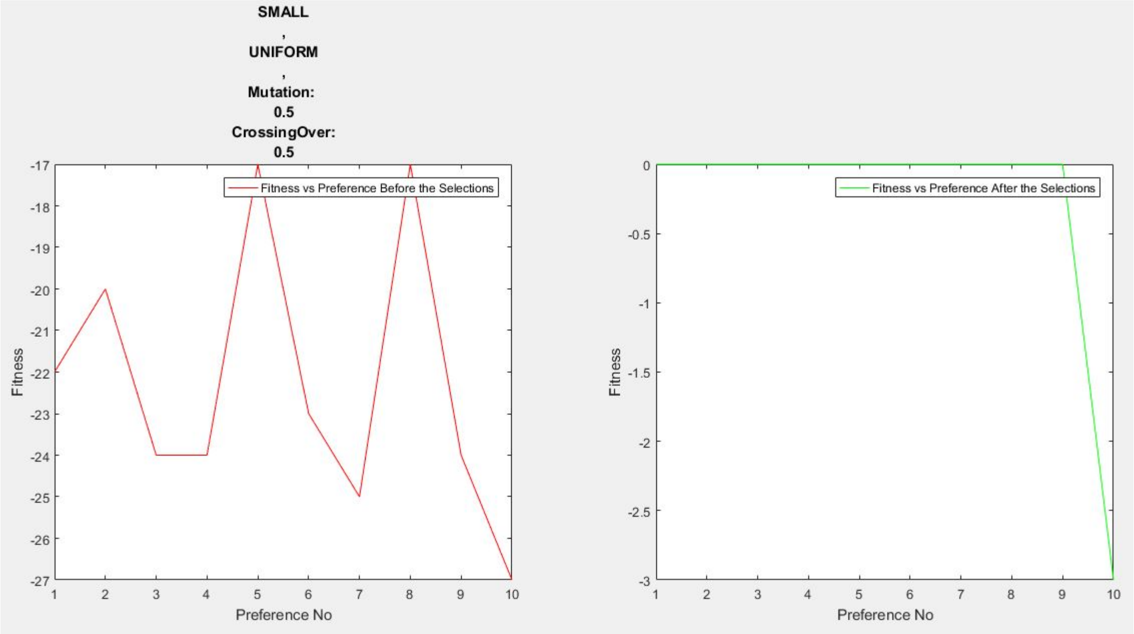


Figure 9: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.5, MutationRate:0.5

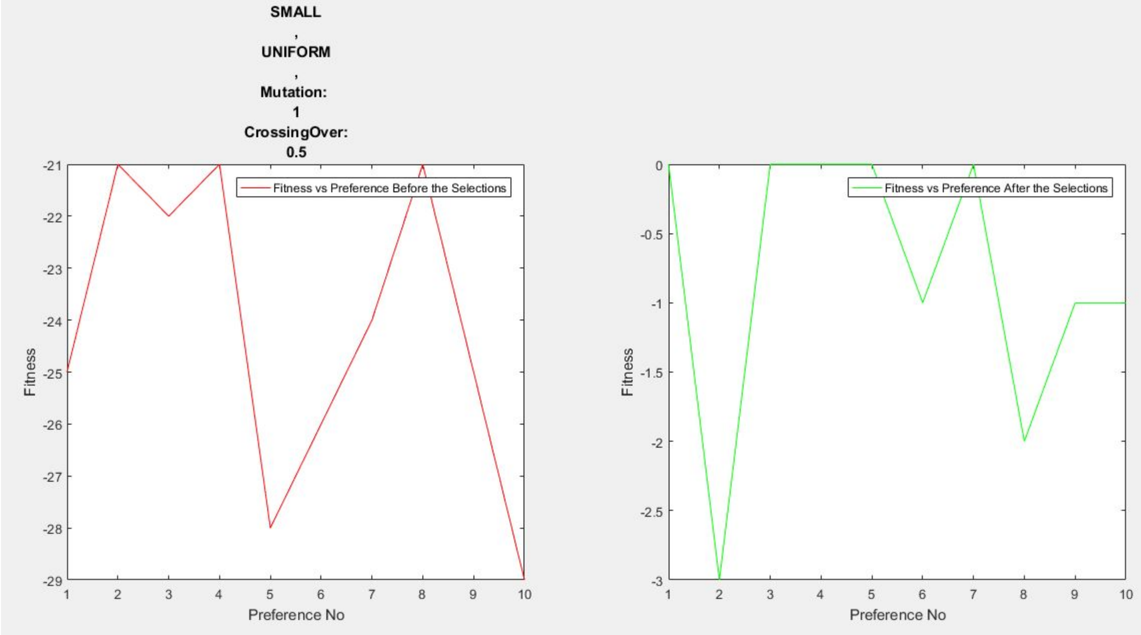


Figure 10: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:0.5, MutationRate:1

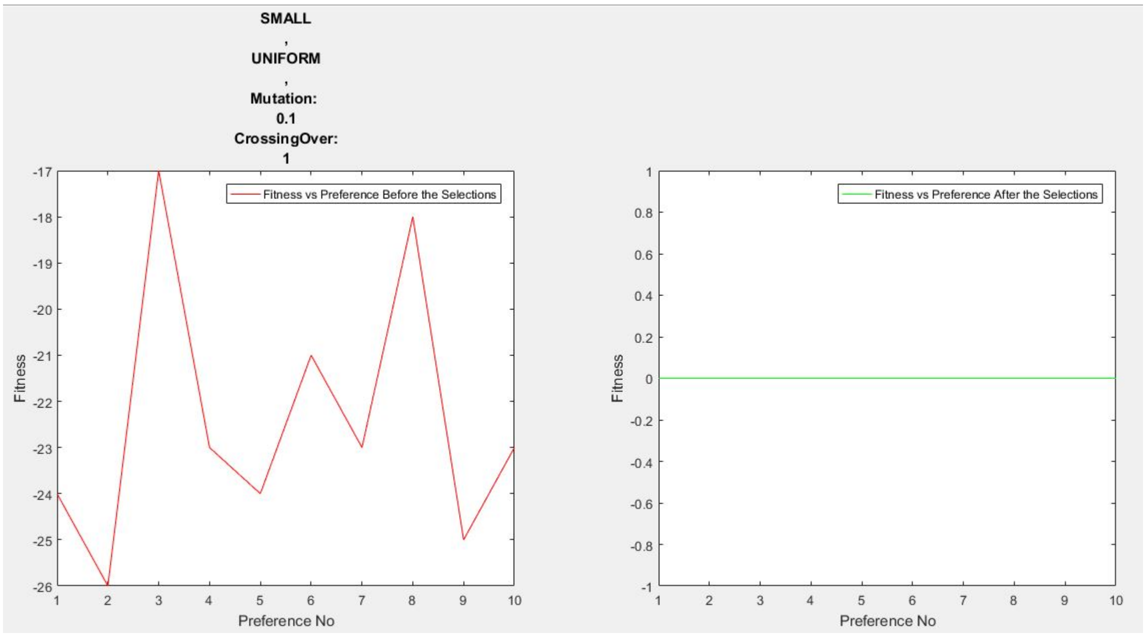


Figure 11: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:1, MutationRate:0.1

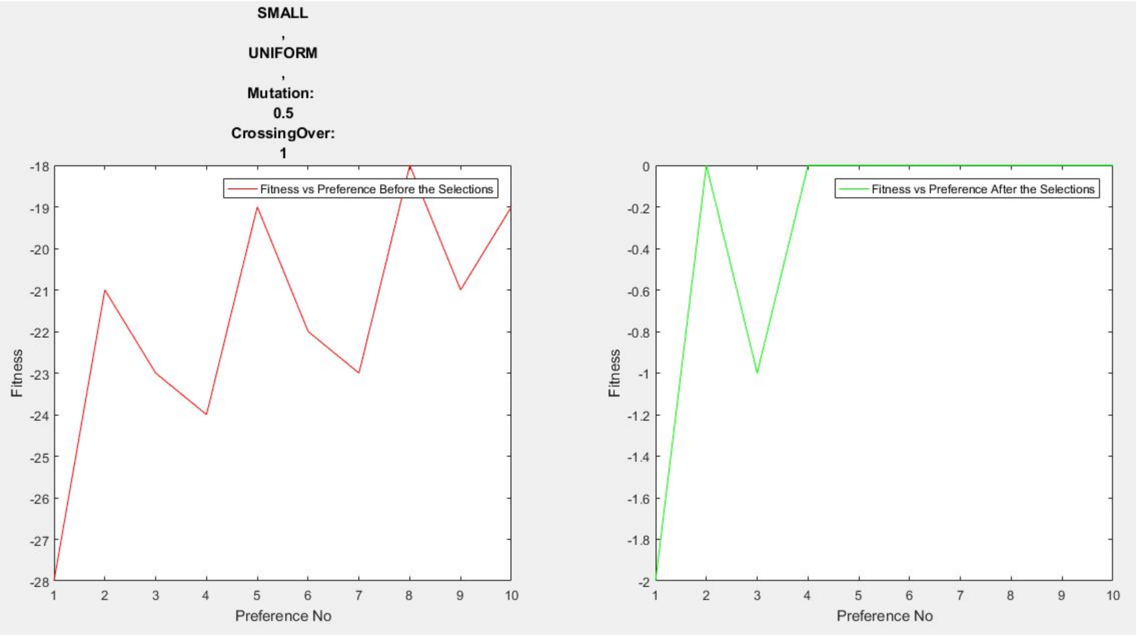


Figure 12: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:1, MutationRate:0.5

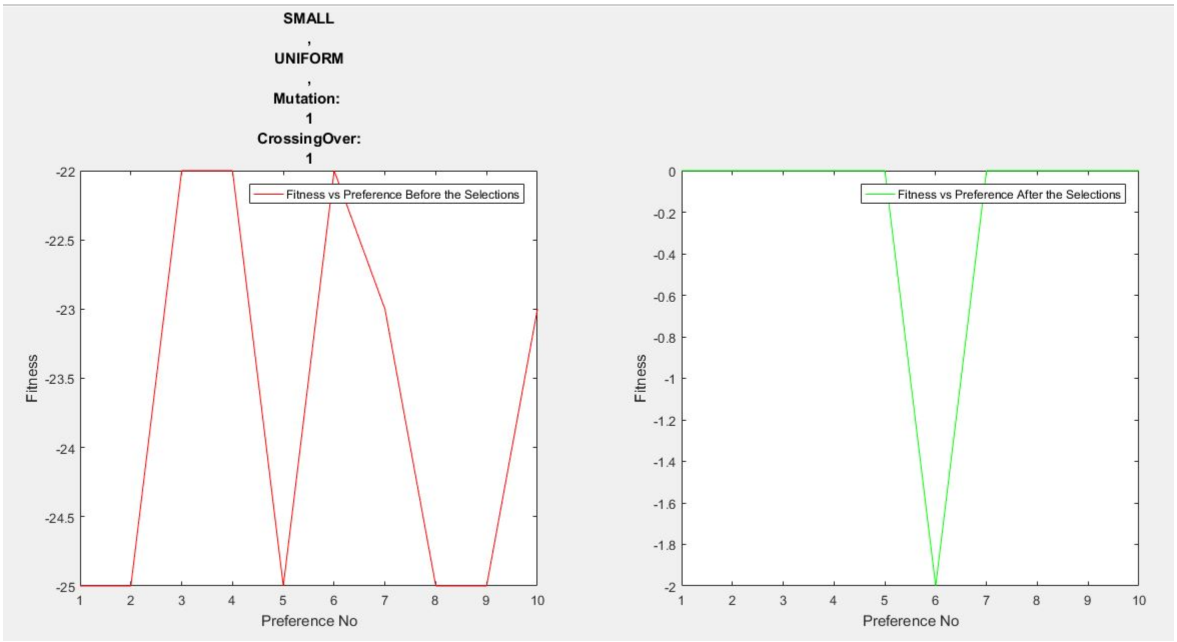


Figure 13: Fitness vs Preference Waveform Uniform and Crowd:20 , CrossRate:1, MutationRate:1

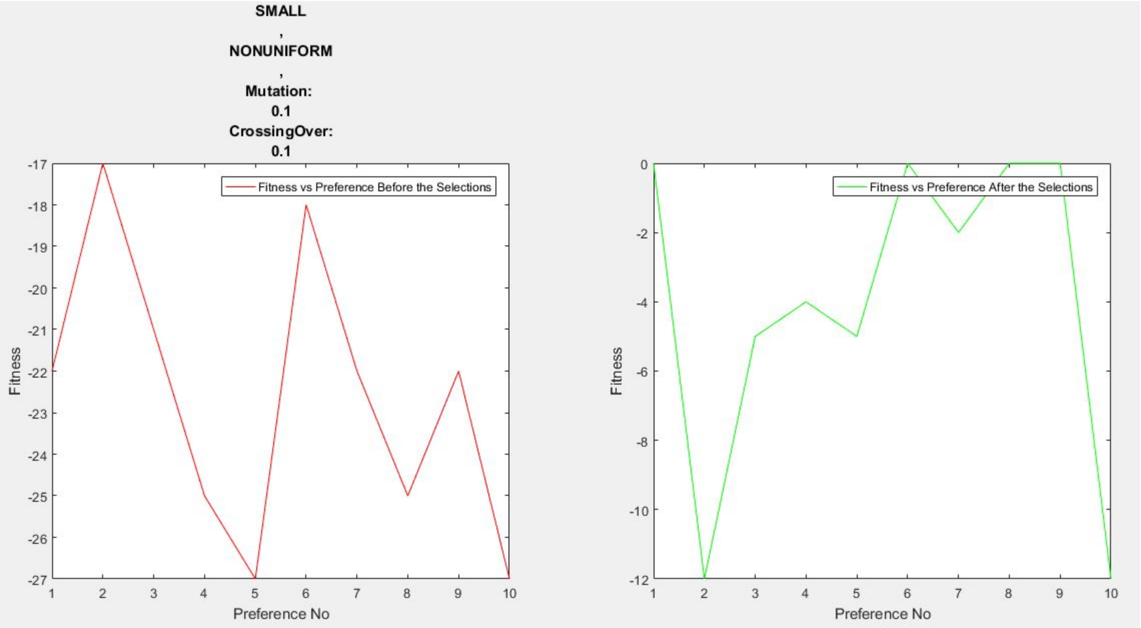


Figure 14: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.1, MutationRate:0.1

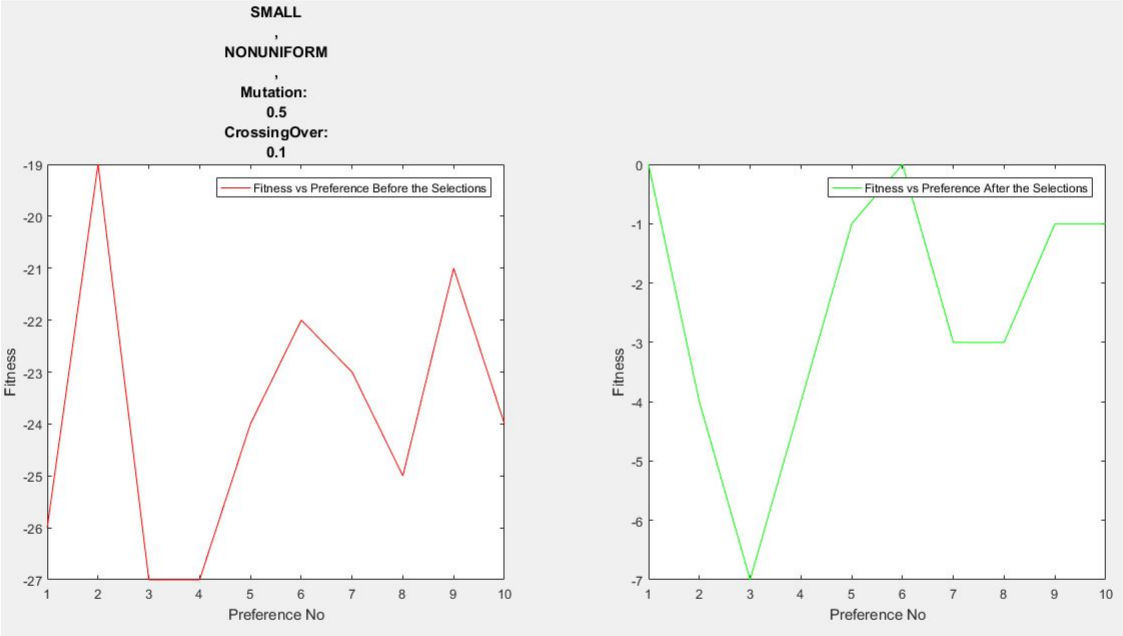


Figure 15: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.1, MutationRate:0.5

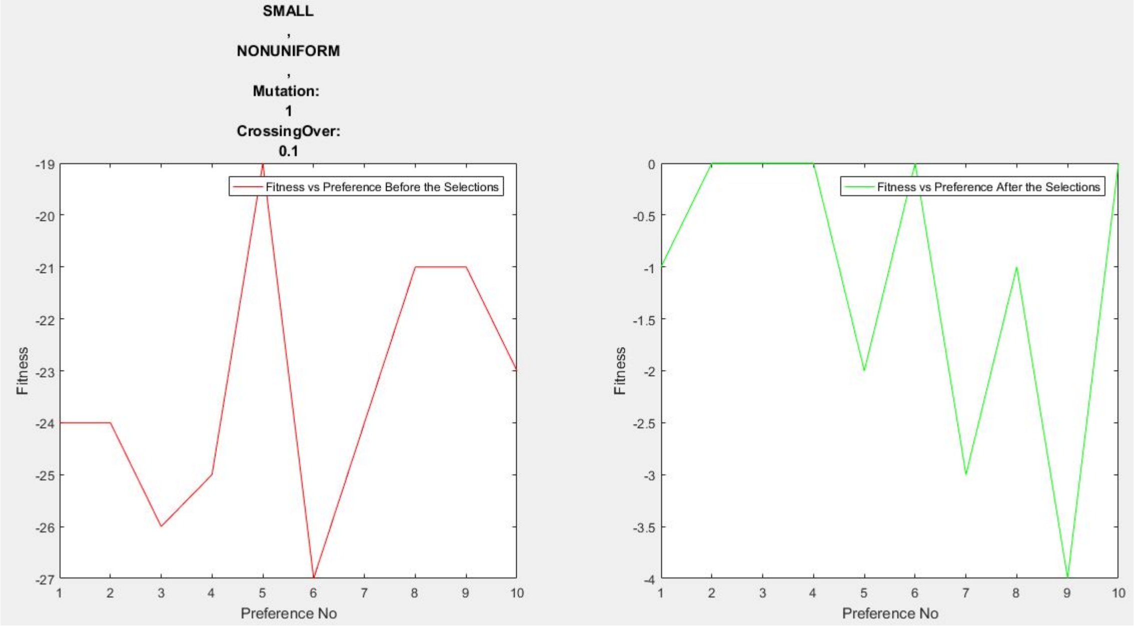


Figure 16: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.1, MutationRate:1

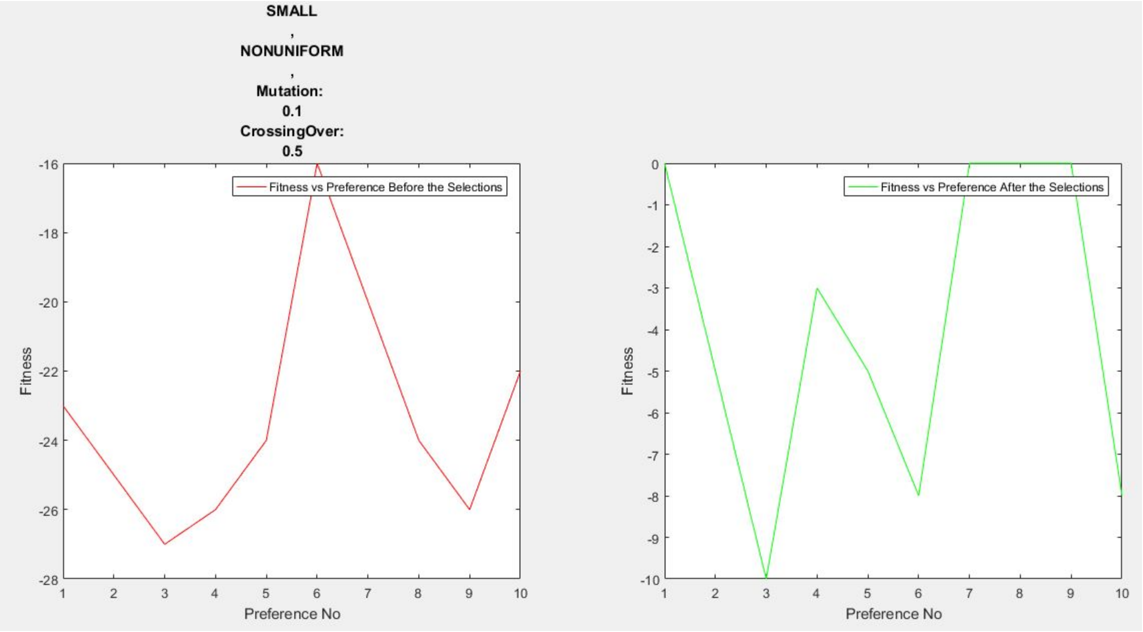


Figure 17: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.5, MutationRate:0.1

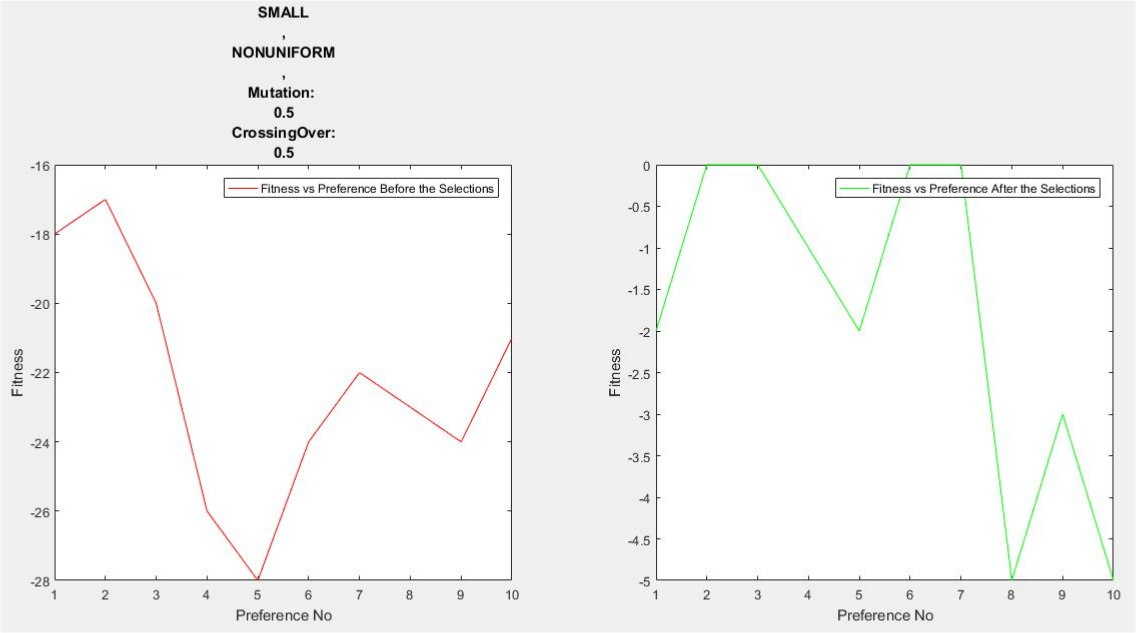


Figure 18: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.5, MutationRate:0.5

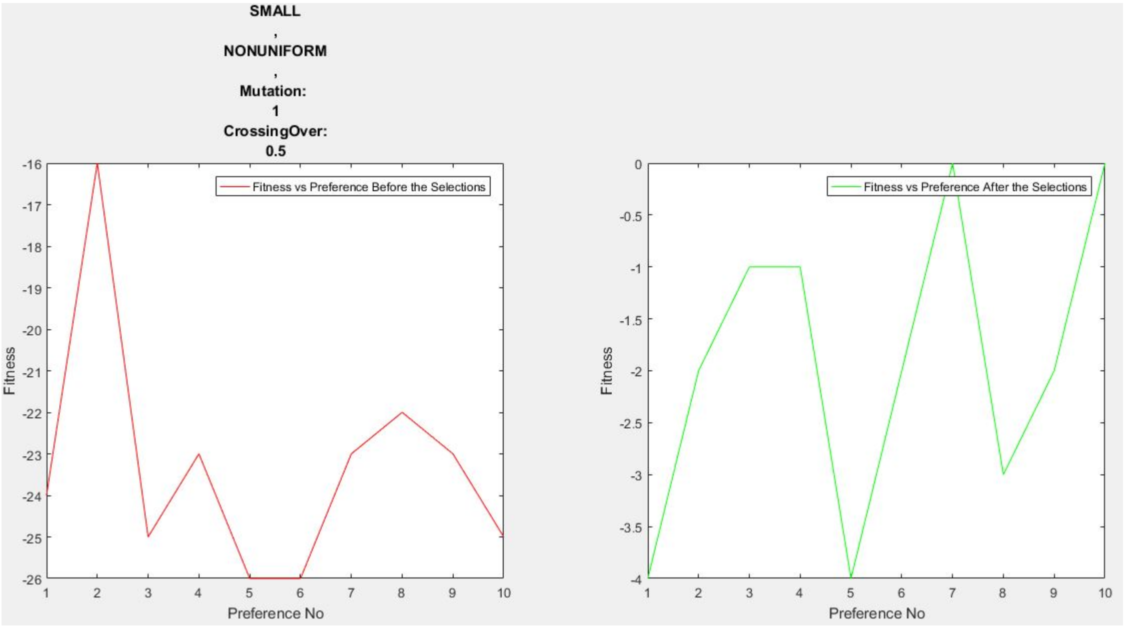


Figure 19: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:0.5, MutationRate:1

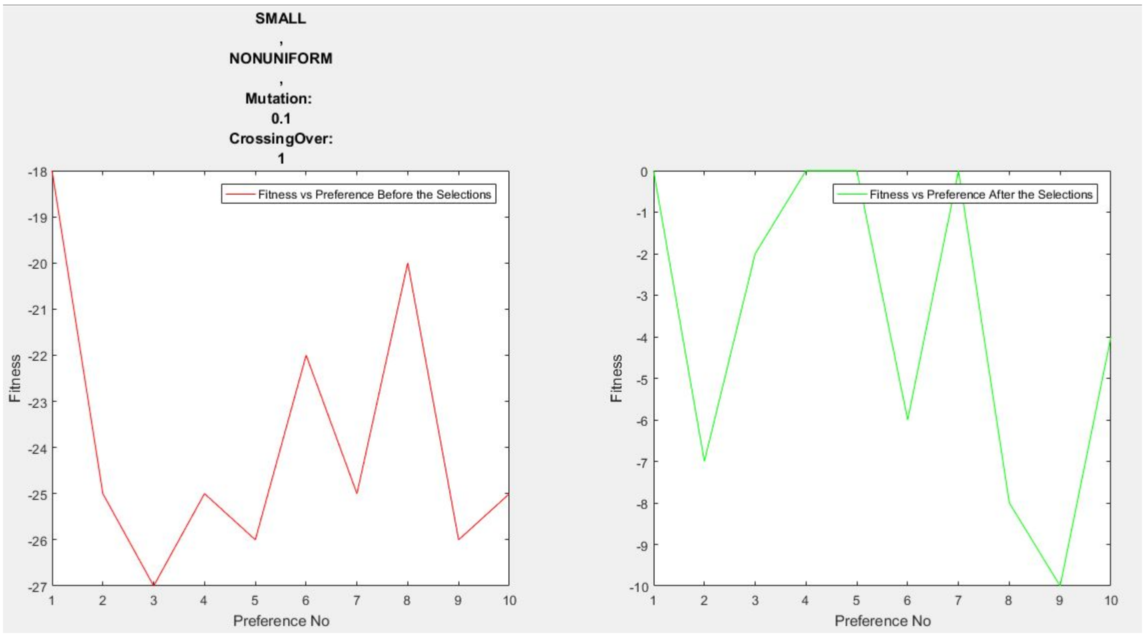


Figure 20: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:1, MutationRate:0.1

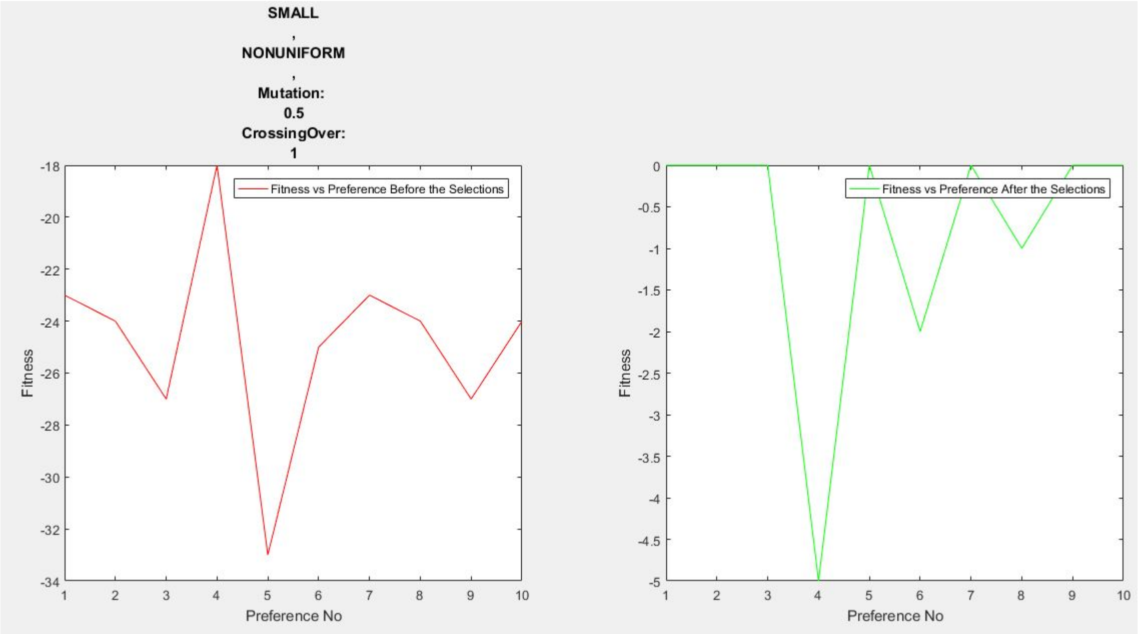


Figure 21: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:1, MutationRate:0.5

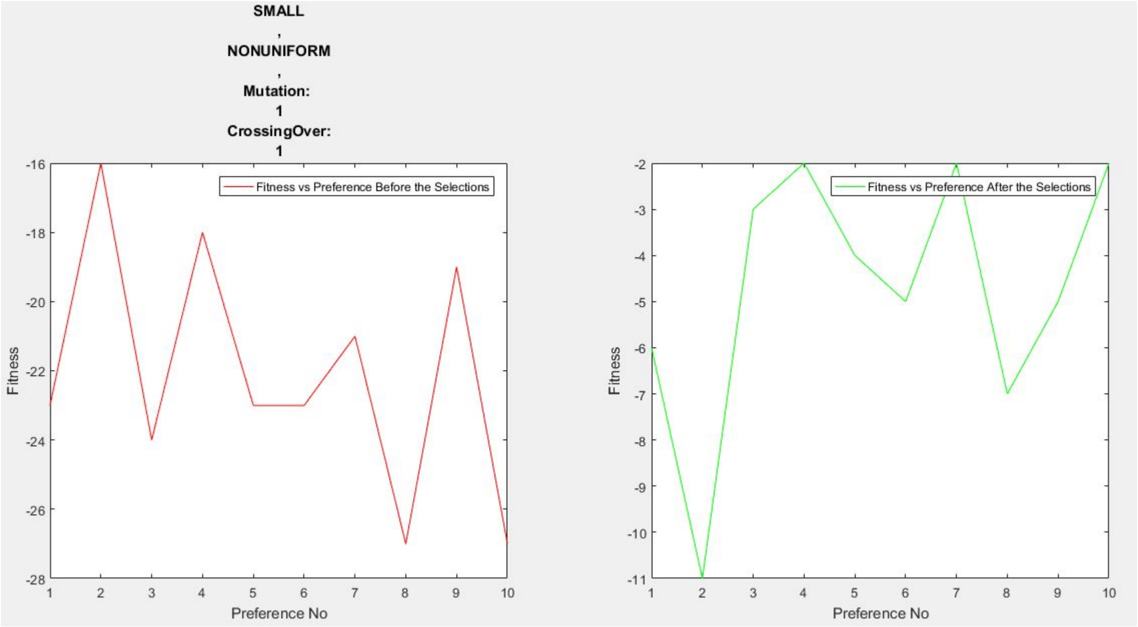


Figure 22: Fitness vs Preference Waveform NonUniform and Crowd:20 , CrossRate:1, MutationRate:1

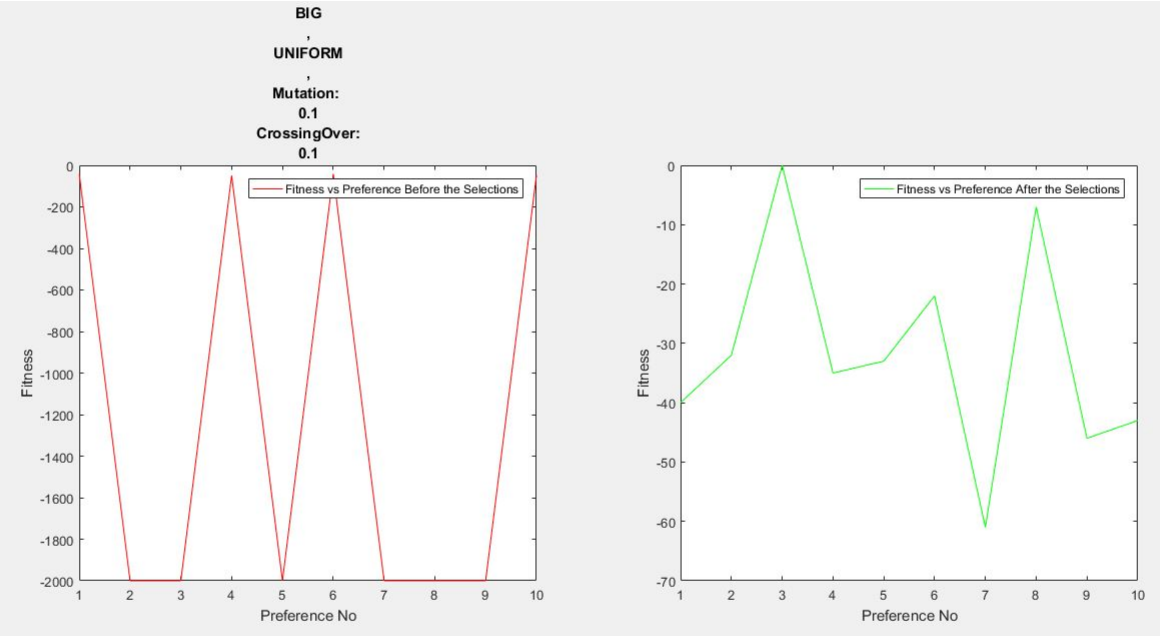


Figure 23: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.1, Mutation-Rate:0.1

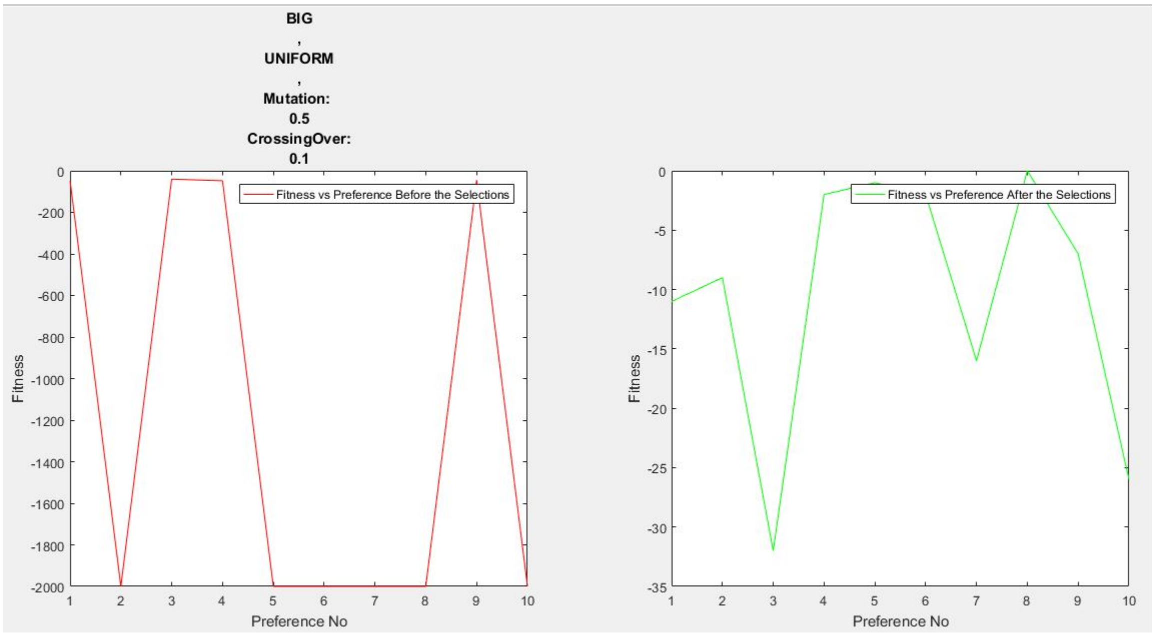


Figure 24: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.1, Mutation-Rate:0.5

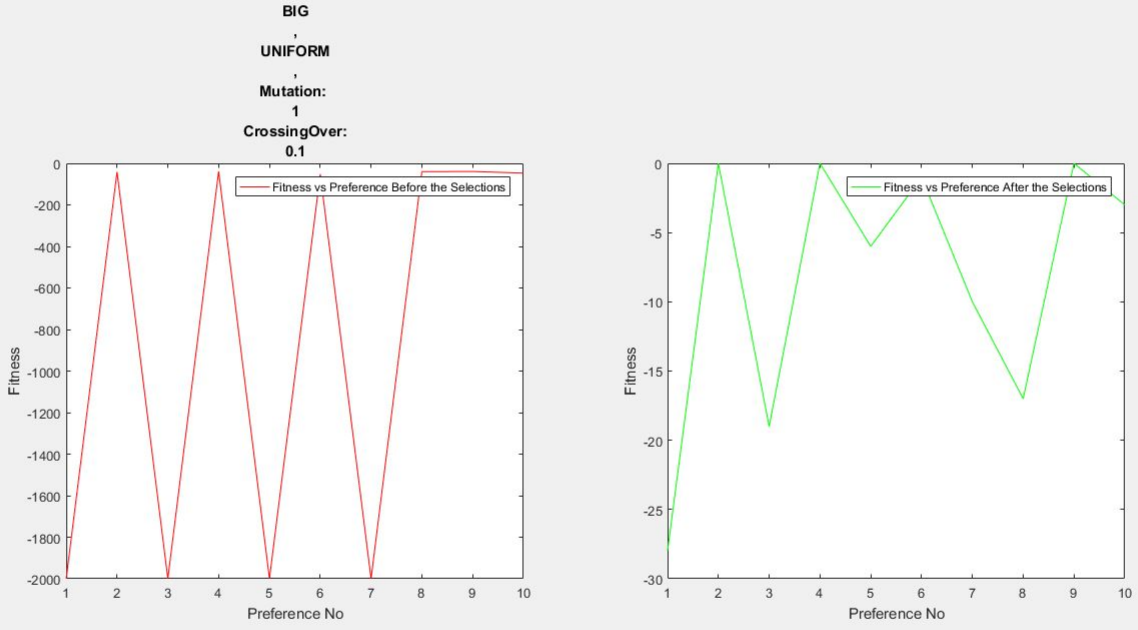


Figure 25: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.1, MutationRate:1

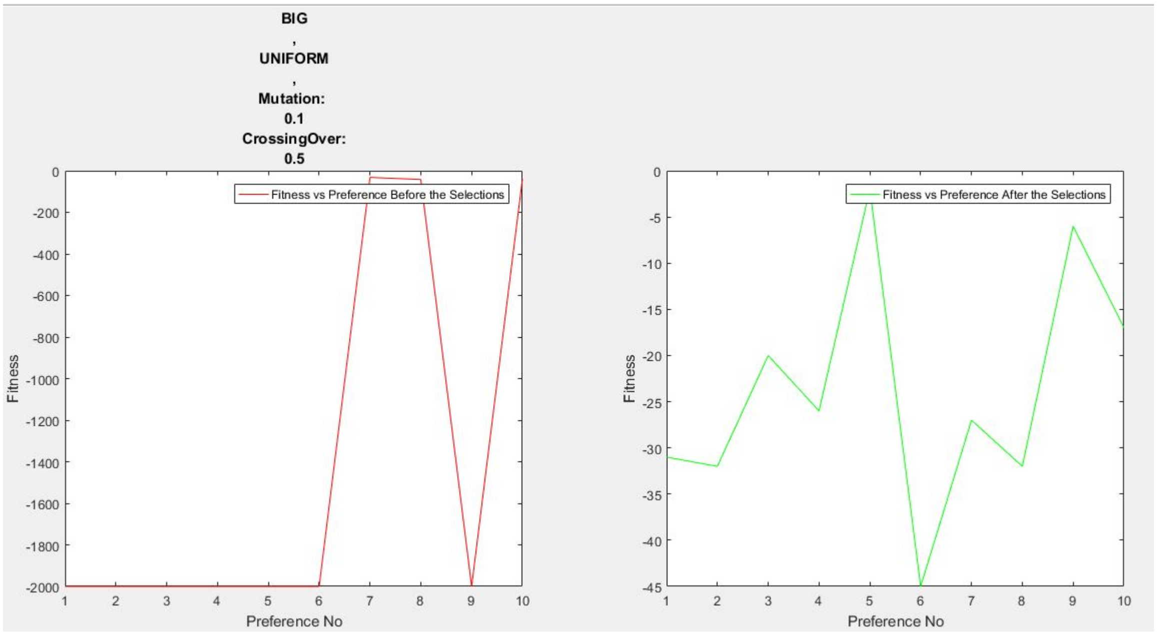


Figure 26: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.5, MutationRate:0.1

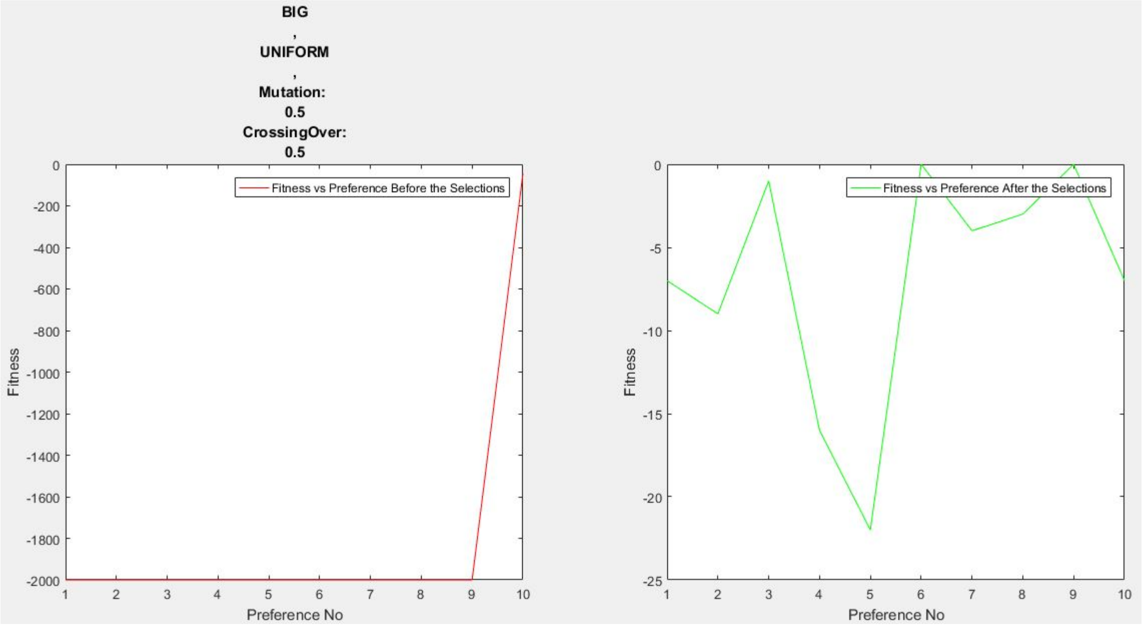


Figure 27: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.5, MutationRate:0.5

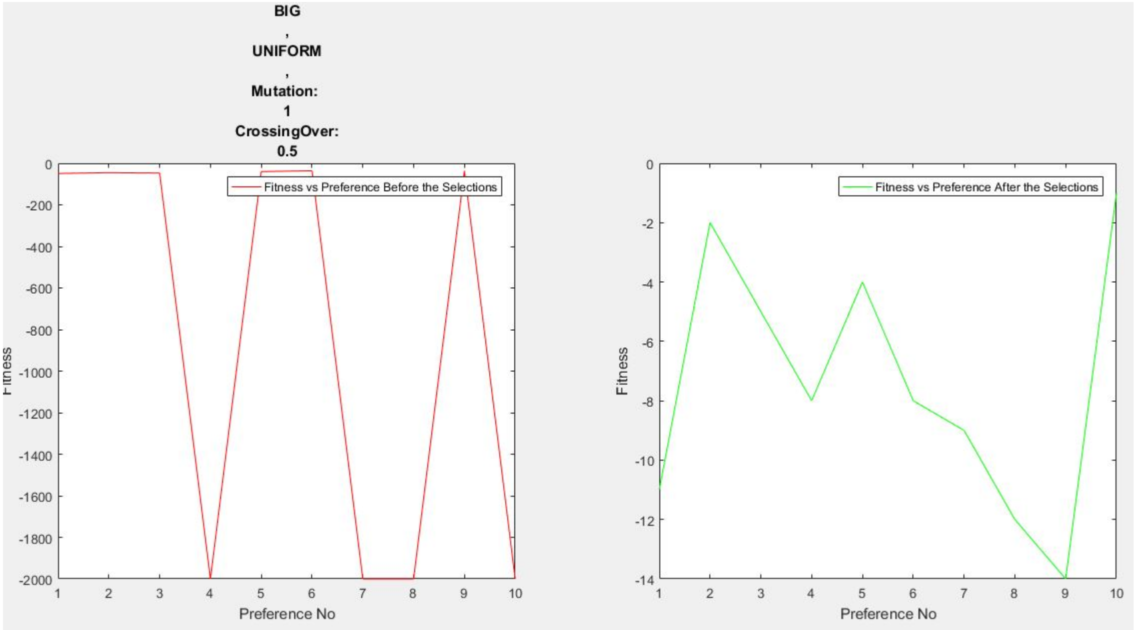


Figure 28: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:0.5, MutationRate:1

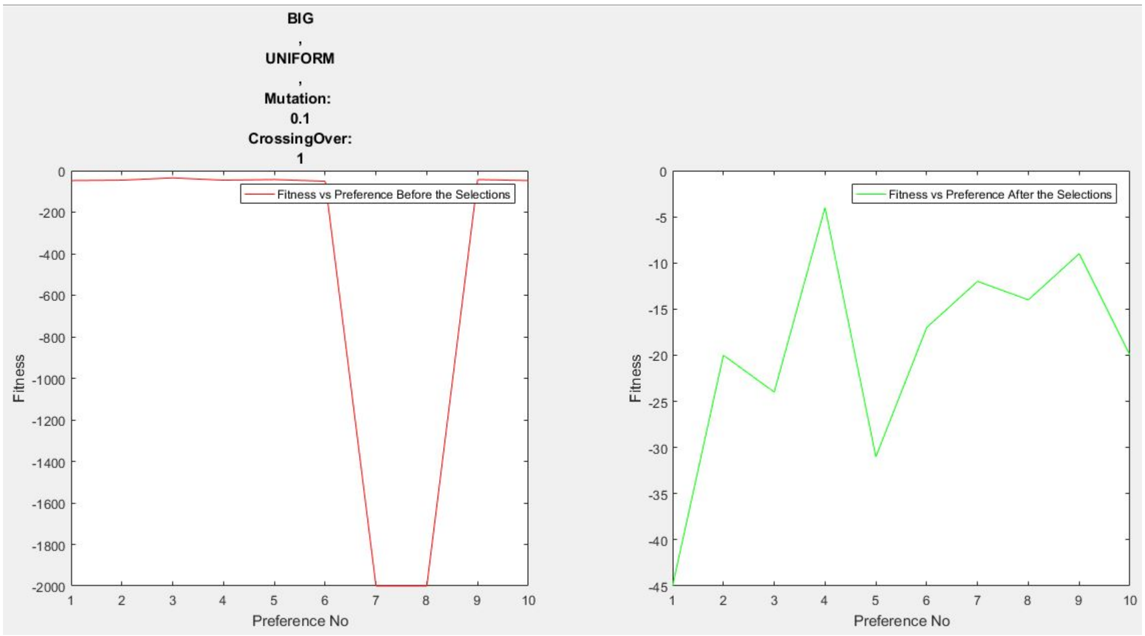


Figure 29: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:1, MutationRate:0.1

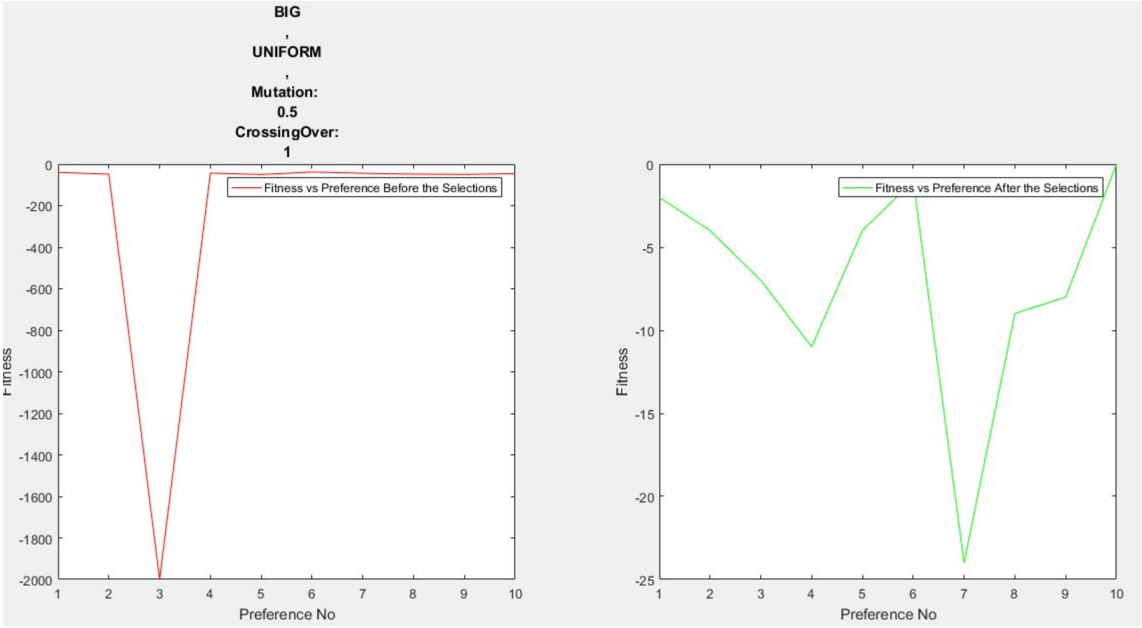


Figure 30: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:1, MutationRate:0.5

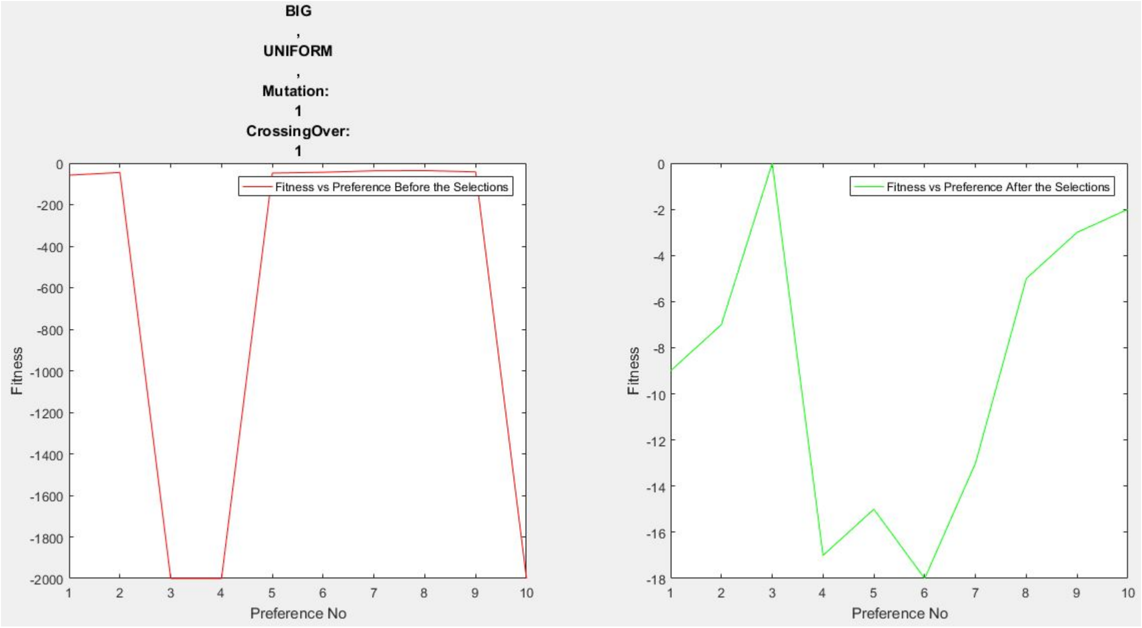


Figure 31: Fitness vs Preference Waveform Uniform and Crowd:30 , CrossRate:1, MutationRate:1

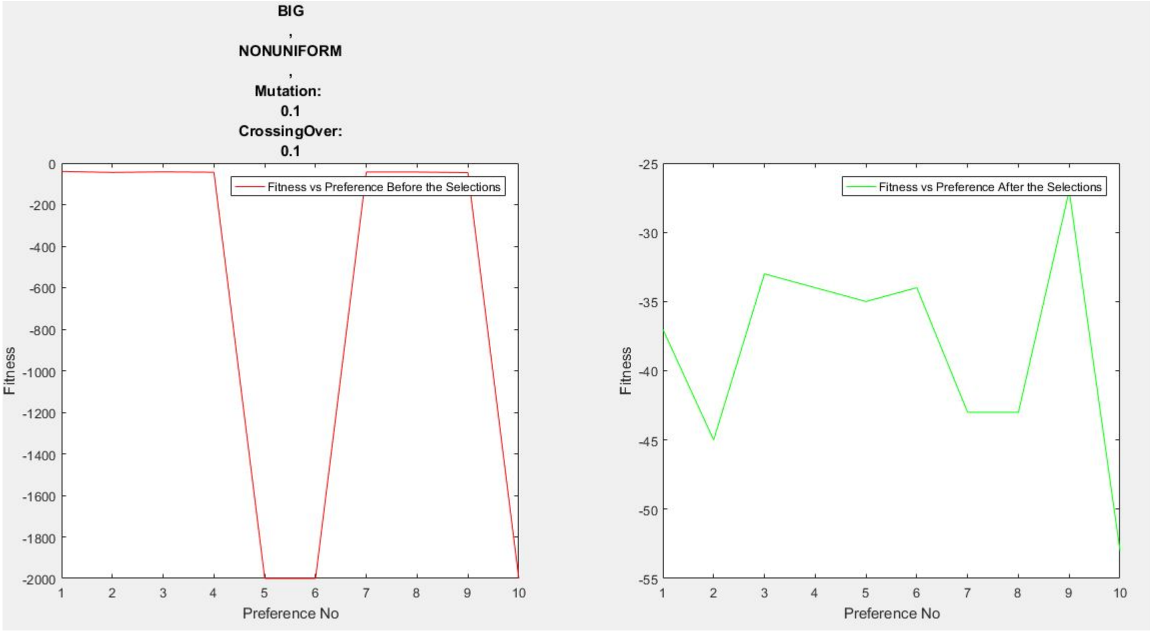


Figure 32: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.1, MutationRate:0.1

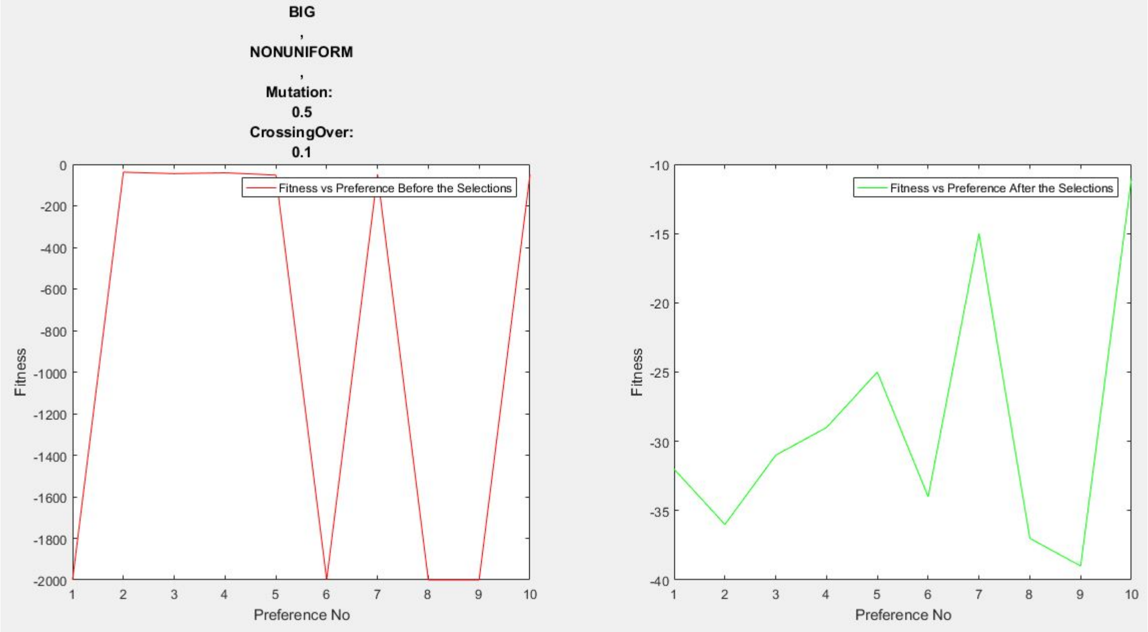


Figure 33: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.1, MutationRate:0.5

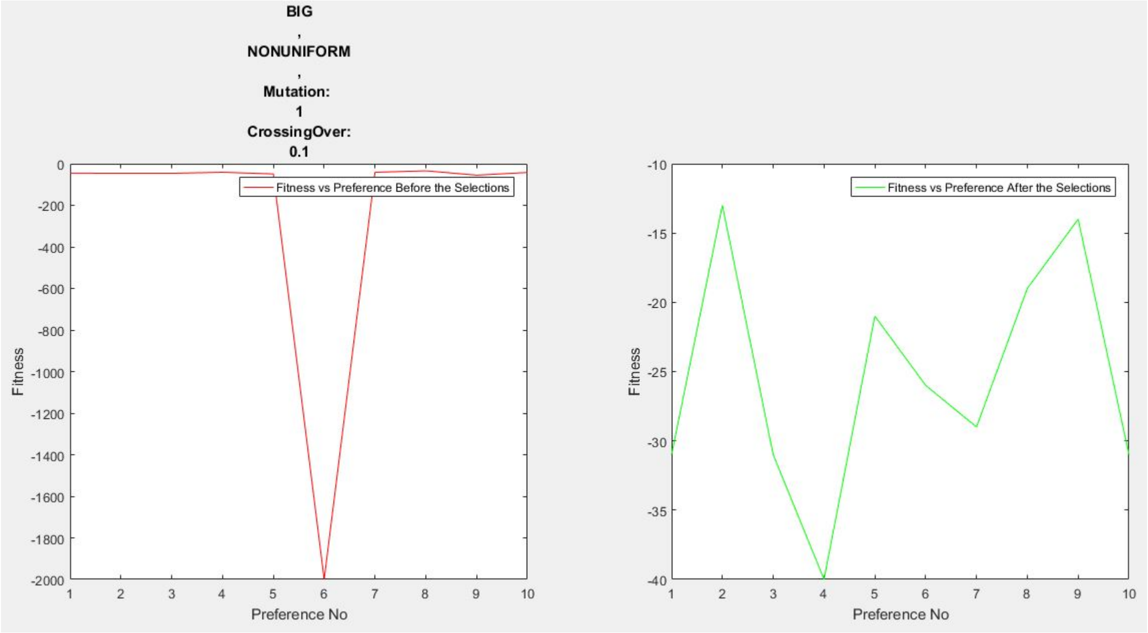


Figure 34: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.1, MutationRate:1

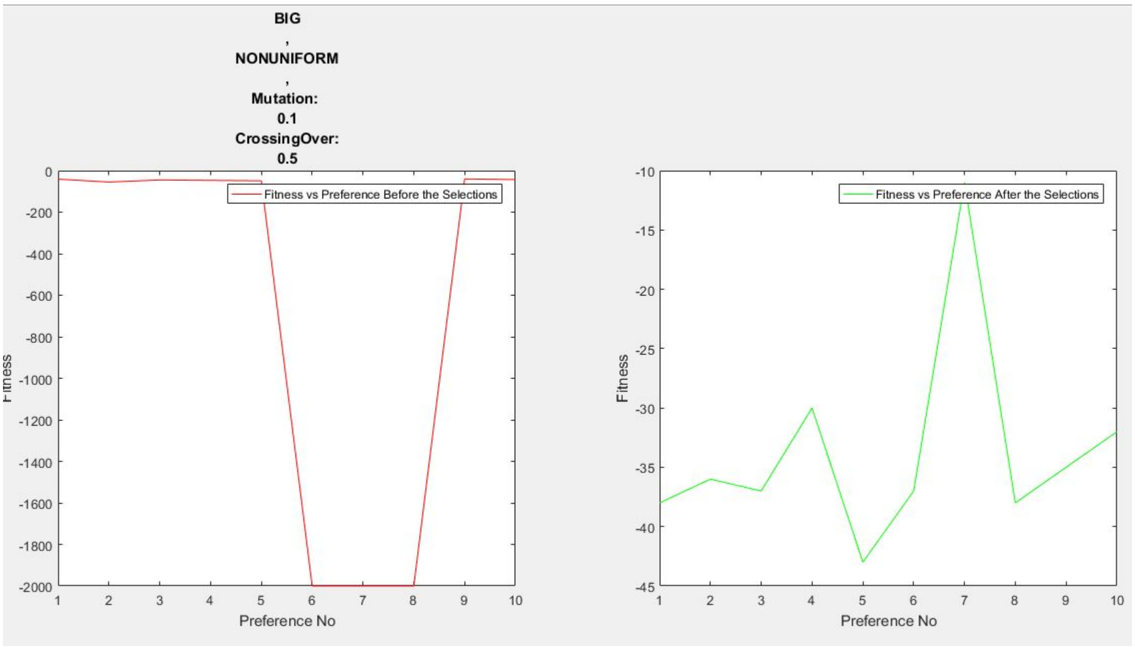


Figure 35: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.5, MutationRate:0.1

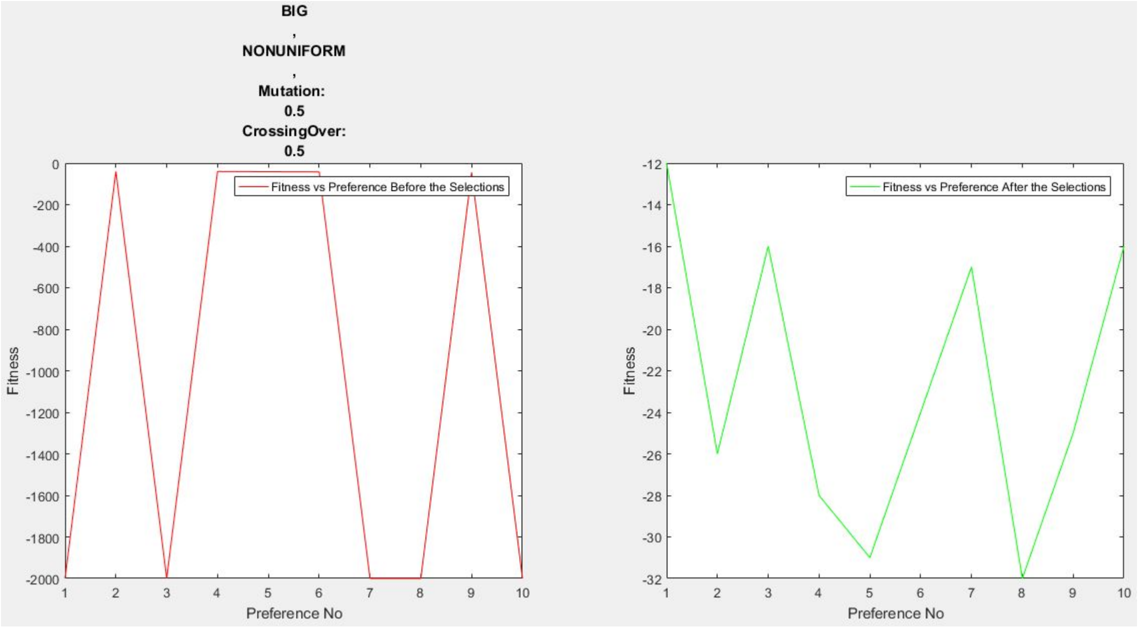


Figure 36: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.5, MutationRate:0.5

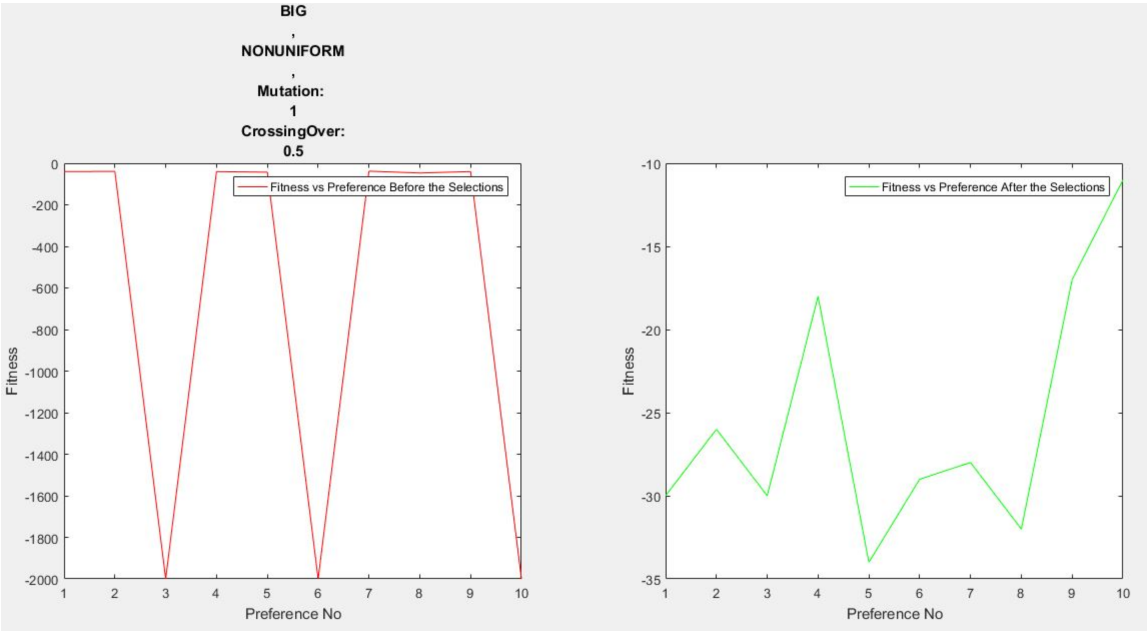


Figure 37: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:0.5, MutationRate:1

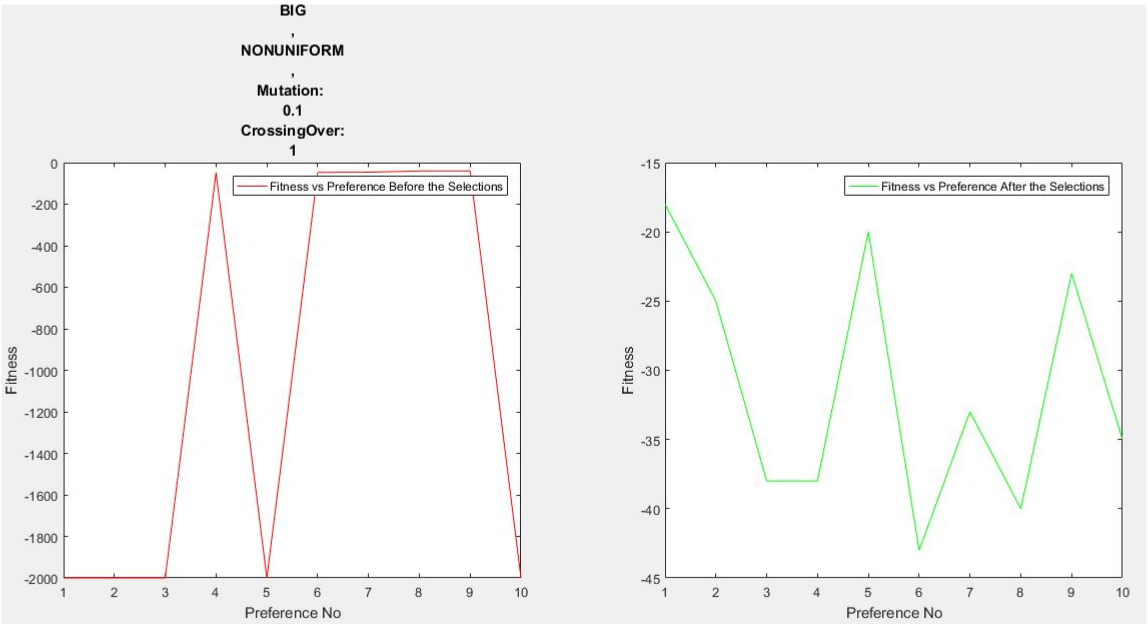


Figure 38: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:1, MutationRate:0.1

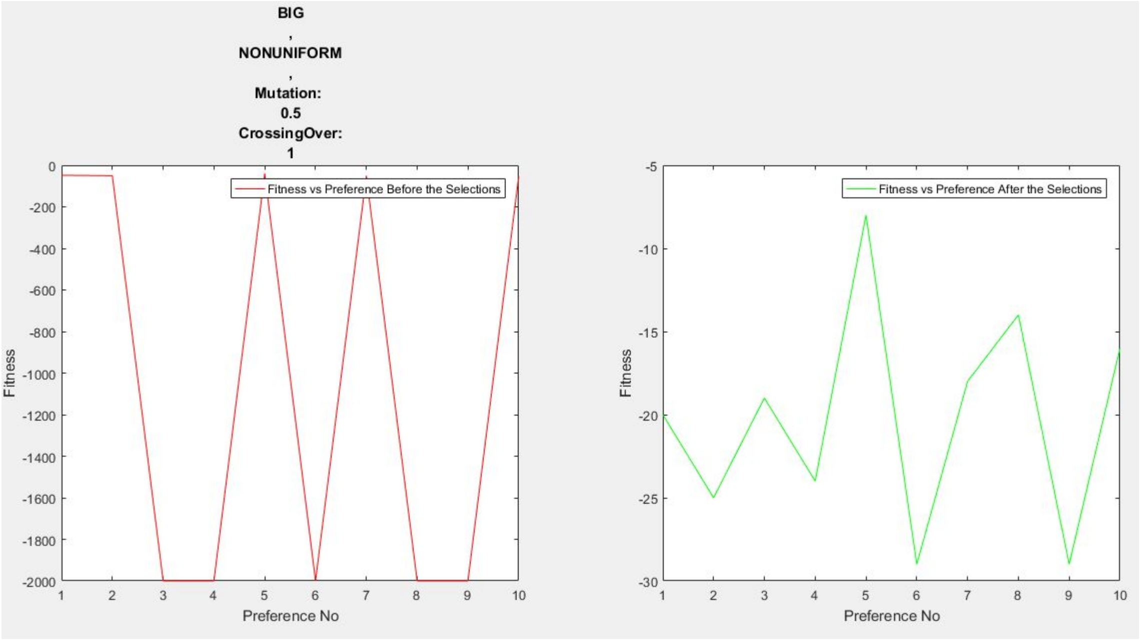


Figure 39: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:1, Mutation-Rate:0.5

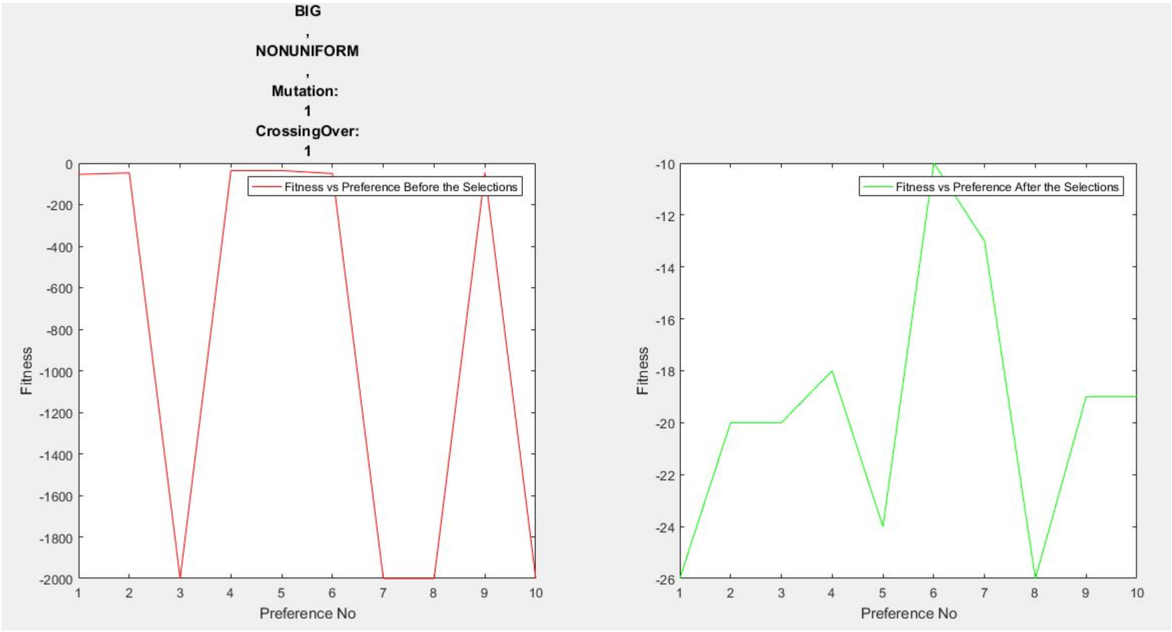


Figure 40: Fitness vs Preference Waveform NonUniform and Crowd:30 , CrossRate:1, Mutation-Rate:1

4 Conclusion

From the results, the effect of each parameter observed separately . In this part of the report, the effects of these parameters discussed. The first argument about the results are difference between difficulty level of four experimented cases. If we look when the crowd is 20 and students are uniformly distributed,due to same popularity there will be more possible solutions and this will make easier to find the best solutions.

If we look to crossing over rates ,their different values change the result magnificently. Thanks to cross over, new generation is provided to the population by swapping genes of the parents between the each other. One of the most important property of the genetic algorithm is variance of the solution ,so to increase it we can use cross over .However, if we increase the rate too much worse individuals will be created much more and they will dominate the population. As a result of this reason approximately the best cross over rate is 0.5 .

One of the other important is mutation, because it will also create variation inside the populations. However ,it will create a completely new gene instead of only swapping between them.So if a chromosome is not inside the population,it is possible to create it via mutation.As a result to create variety mutation is a necessary operation.However, this variance brings a huge risks such as it will corrupt the chromosomes with well fitness. To prevent this we have to protect the best candidate (Elitism), but other candidates with good fitness can be damaged due to high mutation rate. We can see that from the figures clearly.

To increase the variety increasing the population size is the other option. So larger populations increase the variety.To sum up , there is too much components that will affect the result. Tournament Selection,crossover rate, Mutation rate,Population size , etc. ,so to analyze them we can use superposition method . To clarify , we can test for different values while keeping other parameters constant.

References

[1] http://www.computational-intelligence.eu/?lang=en **Rudolf Krause**

[2] http://www.cs.cmu.edu/~tom/10701_sp11/ **Tom Mitchell**