CSE 4082

ARTIFICIAL INTELLIGENCE

PROJECT #2

REPORT

In this project, we are asked to implement Genetic Algorithm for the **Maximum Weighted Independent Set Problem**. This program returns a binary string as output which is indicating that each zero in the graph represents independent vertices of the given graph. In addition, each generation's average fitness value can be seen above the returned binary string.

In the genetic algorithm, certain steps were clear and what we've done were simply applying these algorithms in our code such as Roulette Selection Method, One-Point Crossover or Ten-Point Mutation, etc.

The unique part of our program is "**Repair Function**" which provides us feasible solutions after crossover and mutation operations carry out. We would like to give the details of our repair function now:

There are several steps to achieve valid strings after crossover and mutation operations in the repair function.

Firstly, for each individual of the population, we count how many zero(s) it has within its binary string(**checkOnes()**). And then, we search the whole graph and create a linked list to store indexes of the individuals which has the same amount of zeros(**checkAll()**).

Subsequently, we generate a random integer value in the range of linked list size and by starting the first, we go forward until counter is equal to that random integer. Following this, the function repair calls **fixIt()** and return the index of the individual. And **destroy()** eliminates the linked list for a decent memory usage of the program. Following these steps, **copy()** function is called to copy the new population from 2D array of edges to the 2D population array.

Due to the enourmous length of the string, we put references into the table.

003.txt					
	Crossover Prob.	0.6		0.9	
# Generations	**Population Size / Mutation Prob.**	0.02	0.2	0.02	0.2
50	**100**	460.960	460.000	460.960	460.000
	200	463.080	460.210	460.960	460.210
200	**100**	460.440	461.740	465.850	458.540
	200	457.280	459.200	459.590	**466.330**
400	**100**	465.850	463.460	460.210	457.550
	200	461.740	461.740	461.740	460.000

There was a significant execution time difference between crossover probability of 0.6 and 0.9. Chance of 0.9 finished calculation with greatly reduced time. As we can see, there were no big difference at the weights of the independent graphs for 50 generation iteration. The biggest weight has been achieved for the greatest crossover and mutation probability. This show us the huge impact of randomization of the population. The more diversity, the higher chance of getting higher sum of weight of the independent set. However, this algorithm needs to run as many as possible to obtain better results because it also can be affected by chance. This was what we have observed so far after 003.txt.

015.txt					
	Crossover Prob.	0.6		0.9	
# Generations	**Population Size / Mutation Prob.**	0.02	0.2	0.02	0.2
50	**100**	387.630	387.630	387.630	386.090
	200	387.140	387.140	387.630	387.140
200	**100**	387.630	387.630	387.630	387.630
	200	387.630	387.630	**397.550**	**397.550**
400	**100**	387.630	387.630	387.630	387.630
	200	387.630	387.140	387.630	387.630

As expected, but not fairly, we obtained greater weights due to high number of iterations and population size. However, in this file, we didn’t see the effects of crossover or mutation over the greatest results. The values is shown above.

So far, program executes faster when the chance of mutation or the chance of crossover are low.

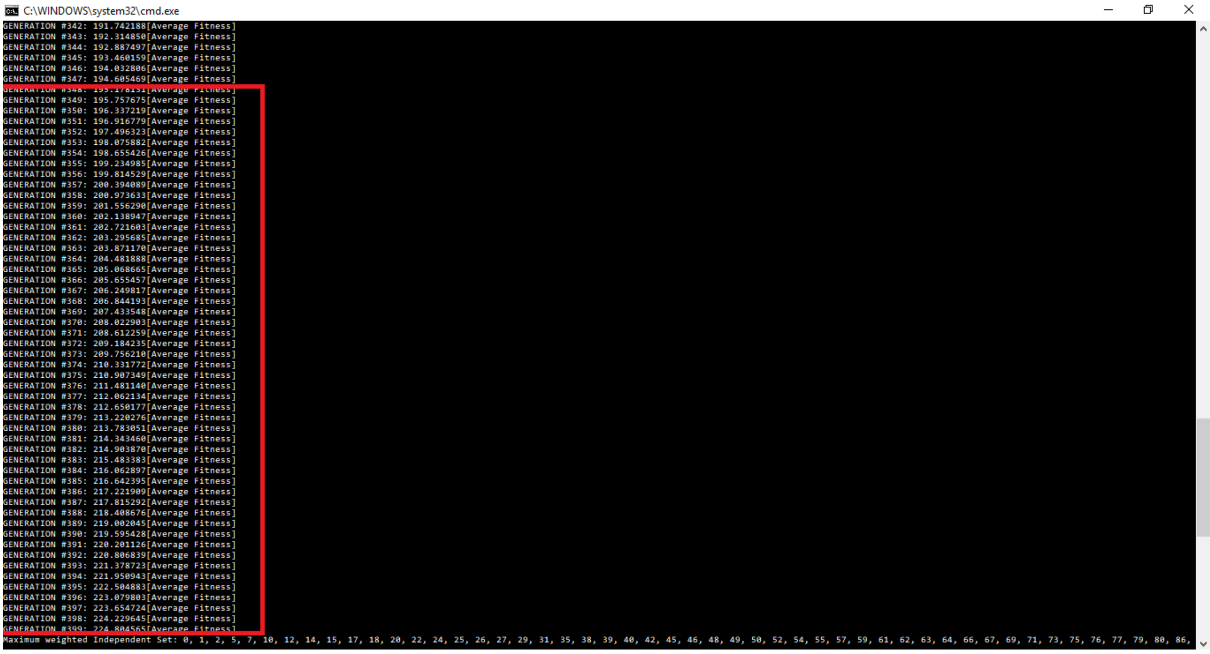
The remaining test file is going to be 030.txt…

030.txt					
	Crossover Prob.	0.6		0.9	
# Generations	**Population Size / Mutation Prob.**	0.02	0.2	0.02	0.2
50	**100**	293.219	287.469	291.179	293.219
	200	293.219	287.469	283.839	291.099
200	**100**	287.469	291.179	287.469	287.469
	200	**308.630**	298.309	293.219	298.309
400	**100**	291.179	291.179	300.229	291.179
	200	291.410	291.179	287.469	287.469

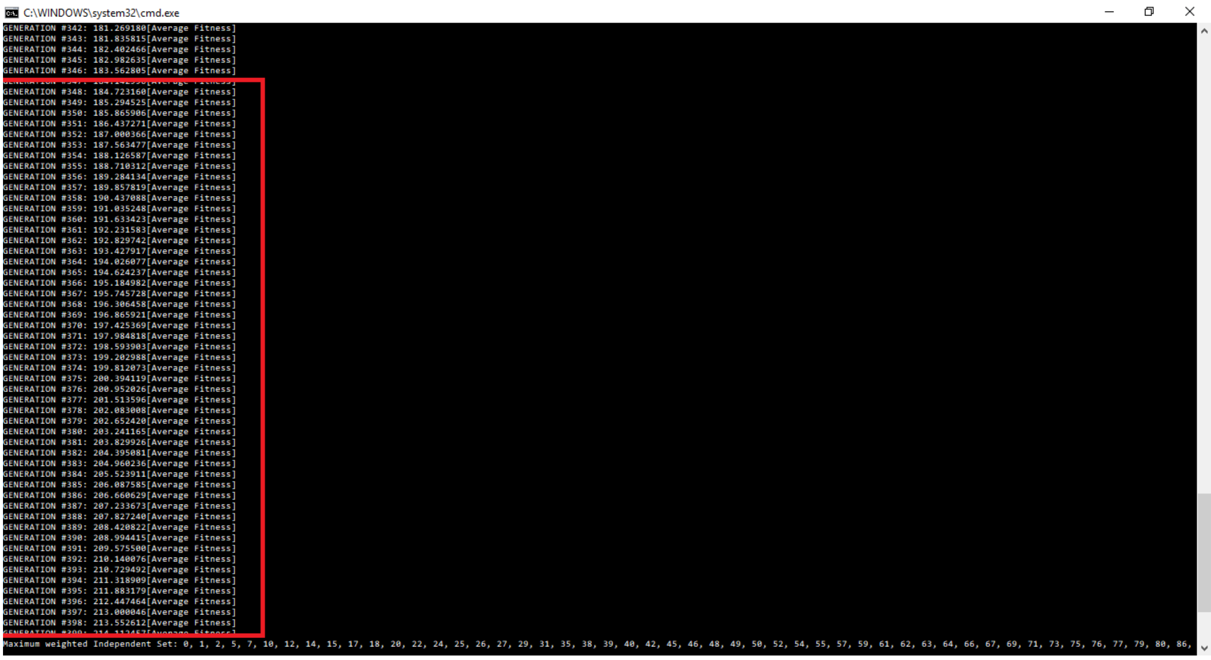
For 50 iteration of generations, same probability of crossover and mutation, the overall weights didn’t undergo any changes which are **293.219**. Number of individuals and higher chance of mutation leds the algorithm generate better results. For instance, same probabilities with different population sizes has contributed greater summation of weights which can be seen for **308.630**. Whenever our algorithm needs to carry out repair operation, the execution time increases.

Lastly, there are 4 screenshots we provide in this project report to demonstrate. We added partial of the generated average fitness values into the picture below:

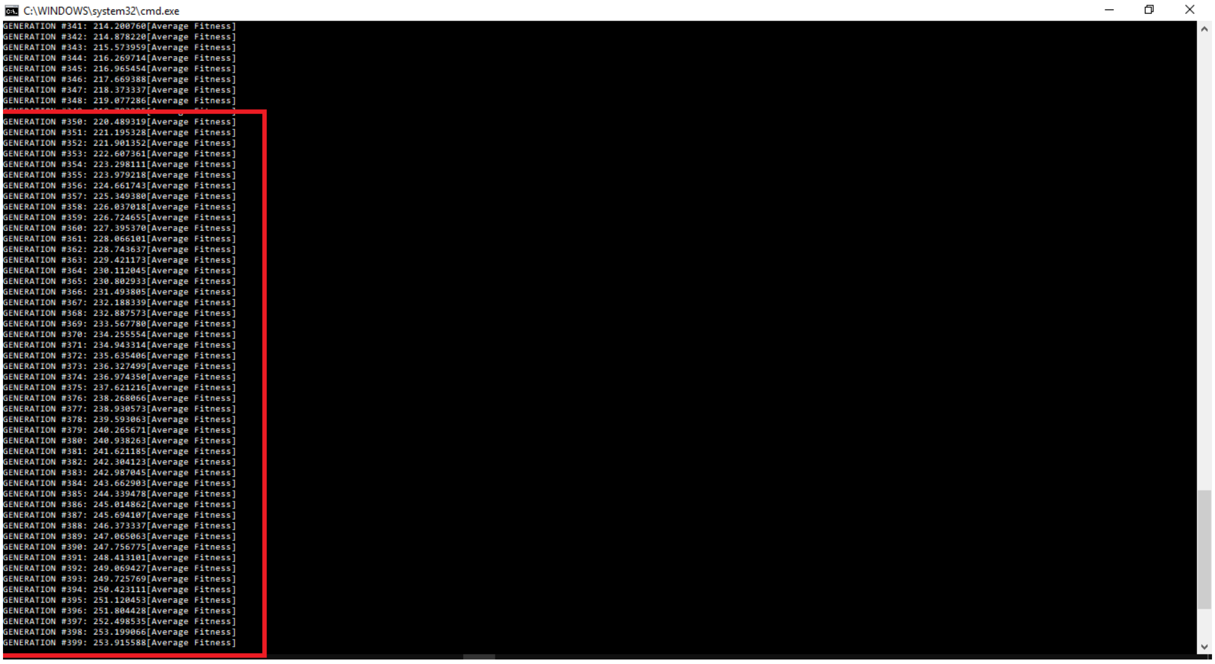
1. File: 030.txt, # Generations: 400, Pop. Size 100, Crossover Prob. 0.6, Mutation Prob. 0.02



2. File: 030.txt, # Generations: 400, Pop. Size 200, Crossover Prob. 0.6, Mutation Prob. 0.02



3. File: 030.txt, # Generations: 400, Pop. Size 100, Crossover Prob. 0.6, Mutation Prob. 0.



4. File: 030.txt, # Generations: 400, Pop. Size 200, Crossover Prob. 0.6, Mutation Prob. 0.2

ZAFER EMRE OCAK 150113075
FEYZA EKSEN 150113005