Bilal Dinç - 150113008 Burkay Sabırsız - 150114034

CSE 482 – PROJECT 2

Assignment

Maximum Weighted Independent Set Problem(MWIPS) is an NP-Hard problem. We tried to find an approximate solution to the problem with a genetic algorithm. Solutions are represented with binary strings in which 1 represents vertex is being in the set and 0 represents vertex is not being in the set.

Implementation

First, file is read and graph is stored in an adjacency matrix. We used adjacency matrix representation because we need to check connectivity of two vertexes quite frequent and it should be constant time.

Second, we created individuals of initial population randomly by first deciding how many vertices should be in the solution set then selecting vertices by random. This way all solutions are given equal chance with the assumption that all representations are valid solutions.

After initial population is created generation loop is started. Inside the loop, first population is repaired then selection probabilities are calculated and mating pool is created with those probabilities after that crossover and mutation operations are performed.

Additionally, we implemented elaborative logging system for experiments and a script that performs experiments for given sets of parameters. Furthermore, we wrote a Matlab script that automatically creates figures for all csv files in the current directory.

Repair Functions

We considered three different type of repair function. First one is the most straightforward one. It simply checks whether all pairs of vertices in the solution are connected. If such pair is encountered, first one is removed from the set.

Second repair function is more complicated. First it searches for vertex that has most connection with other vertices in the set then it removes it. It does this until solution become valid.

For third repair function we defined a gain value G_i for every vertex i in the solution set S. Let V_i be a set of vertices that has edge with i and w_i is weight of i.

$$G_i = -w_i + \sum_{k \in (S \cap V_i)} w_k$$

According to this definition repair function calculates gain values for each vertex then removes the vertex that has most gain from the set. After every removal it recalculates gain values. This operation is done until the solution becomes valid.

Intuitively, this approach converts an invalid solution to a valid one by only remove operation with greedy manner.

Experiments

We used three different repair function in our experiments. These tables from the first repair function.

003					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	35.8	33.34	33.78	33.97
	200	34.17	34.84	35.13	36.13
200	100	37.49	36.65	35.08	38.71
	200	38.56	36.56	39.04	39.27
400	100	36.9	36.61	38.8	38.17
	200	38.04	37.44	37.25	38.29

This is the first graph file. 0.9 crossover seems to give better results in this one. Also as we can see as the generations proceed solutions become better.

015					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	13.57	12.84	13.43	11.93
	200	14.14	12.67	13.58	13.02
200	100	13.9	14.72	14.52	14.47
	200	14.94	15.06	14.73	14.97
400	100	14.08	14.73	14.04	14.73
	200	15.6	15.02	15.07	15.64

This is the second graph file. From the values we can say that this graph probably denser than previous one. In this example we can easily see that population size 200 showed significantly better results from the population size 100.

030					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	7.37	7.61	8.14	7.89
	200	8.76	8.35	7.86	8.28
200	100	9.42	8.47	8.7	9.66
	200	9.83	9.36	9.42	9.85
400	100	9.88	9.32	9.25	9.37
	200	9.41	9.92	10.11	9.72

Last graph seems even denser than second one. Again we can see population size 200 seems better here.

We repeated same experiments with other repair functions. Following tables corresponds third repair function

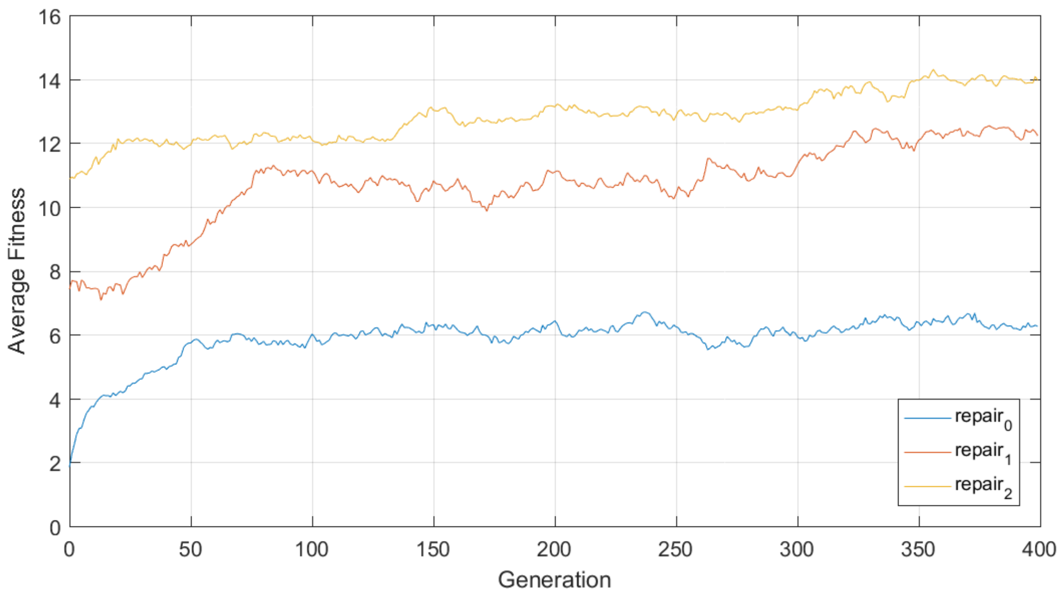
003					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	73.92	74.58	75.76	75.42
	200	75.72	75.96	77.54	76.61
200	100	74.22	74.21	78.11	76.53
	200	75.2	76.82	76.29	76.02
400	100	76.29	76.02	76.16	75.77
	200	77.36	75.87	75.07	75.92

015					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	24.42	26.02	24.6	25.71
	200	25.74	25.8	25.93	24.6
200	100	23.83	24.51	24.82	25.03
	200	25.73	25.3	25.79	25.19
400	100	25.85	25.95	26.11	24.74
	200	25.89	27.34	26.14	24.86

030					
	Crossover Pr.	0.6		0.9	
# Generations	Pop.Size \ Mut Pr.	0.02	0.2	0.02	0.2
50	100	14.02	14.92	14.3	15.08
	200	14.46	14.74	15.61	14.02
200	100	15.24	15.5	15.03	14.48
	200	15.59	15.17	14.55	15.1
400	100	14.89	15.15	15.07	15.18
	200	14.72	16.08	15.11	15.55

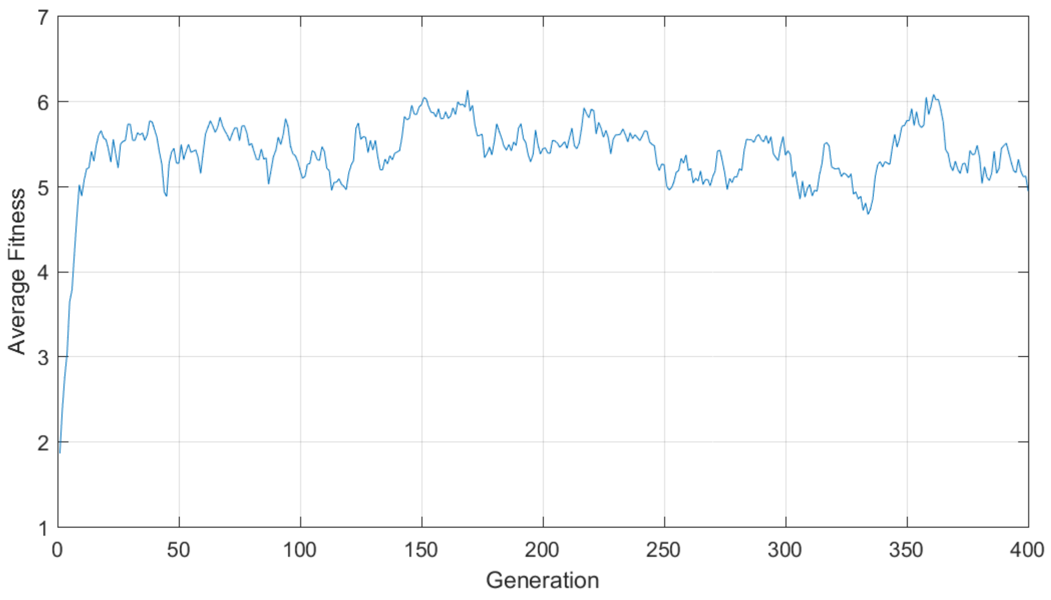
First thing that we can see is results are almost doubled. Secondly, improvements over generations seems little less in this one. We think reason for this improvement is that using this repair function after initializing population acted like greedy construction heuristic and algorithm started with more valuable solutions.

We compared different repair function with graph file 030 and parameters initial population 200, crossing over probability 0.6, and mutation 0.2.

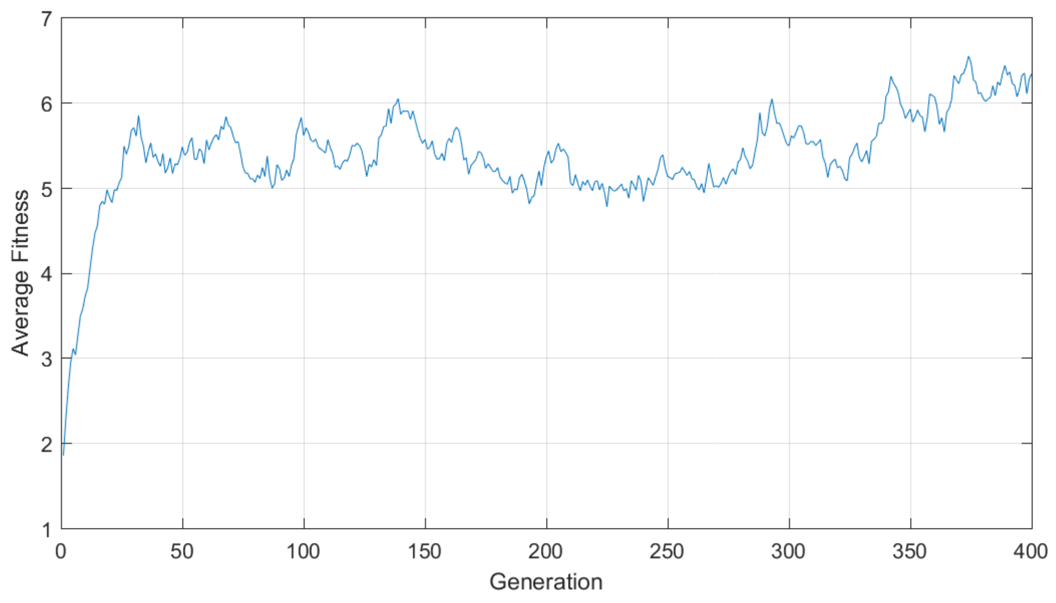


It is clear that third repair function performed better. We chose these parameters because for all repair functions, they showed better performance in terms of average population size.

Following graphs from file 030, with Crossover Probability 0.6 and first repair function is used.

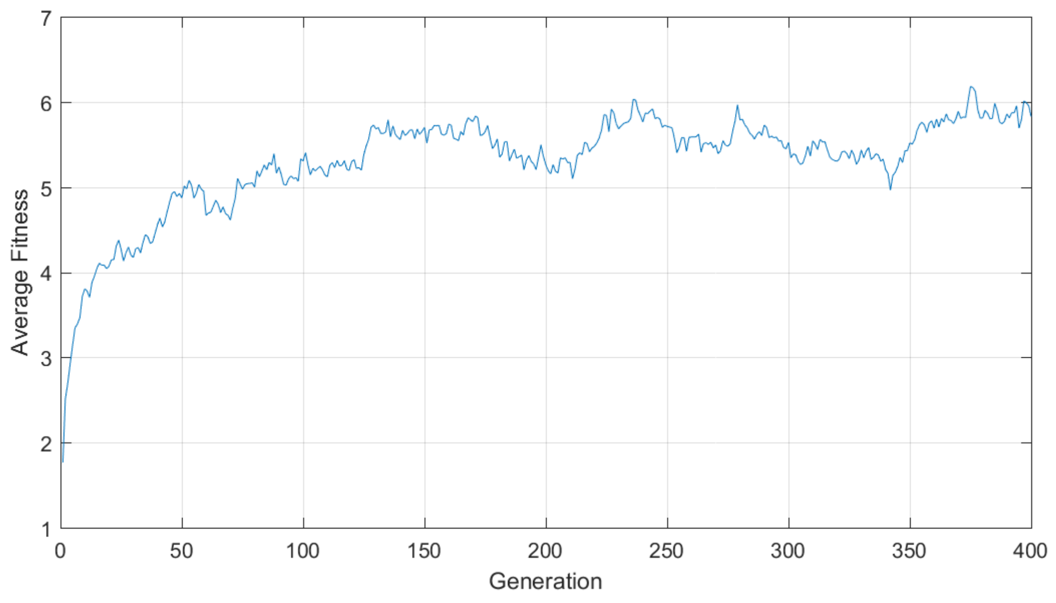


Pop. Size 100, Mutation Prob. 0.02.



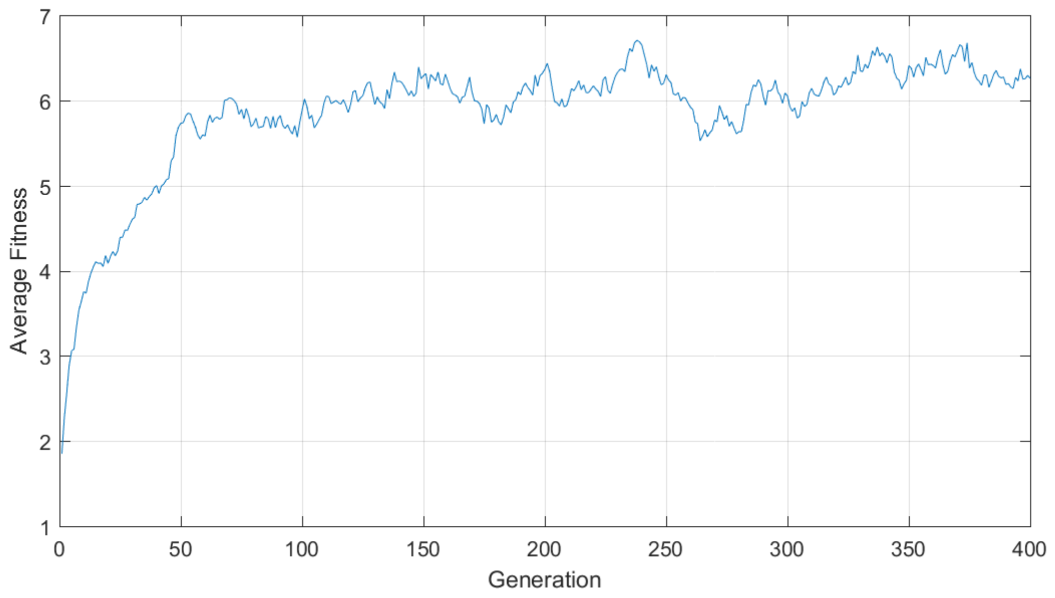
Pop. Size 100, Mutation Prob. 0.2.

From these first two graph we can see effects of mutation first graph with lower mutation probability seem more stable. However, in the second graph about generation 200 higher mutation probability makes average fitness decrease. Also about generation 340 average fitness increases significantly due to probably ten times higher mutation.



Pop. Size 200, Mutation Prob. 0.02.

From third graph we can see that as population size increases values changes more smoothly and slowly. We can observe this easily from greedification speed of initial population. It took almost 150 generations to reach the same fitness level with previous graphs.



Pop. Size 200, Mutation Prob. 0.2.

Higher population size gave more slow but stable increase in average. Combining this with higher mutation seems gave the best results in this specific problem. We observed almost same tendencies with other repair functions with higher average values.

Further Work

After some observation of results, we saw that cardinalities of solutions sets are around some value. For example, most of the solutions of graph file 030 includes about 20 vertices. We tried to exploit this information by initially favoring the individuals that has 20 vertices. After some experiments, we concluded that this idea leads to nowhere because results got worse.

Furthermore, we tried greedier method then roulette selection method. It creates probabilities by exponentially weighting. Probability of selection an individual i for mating pool is given as following formula.

$$p_i = \frac{e^{\frac{f_i}{\tau}}}{\sum_j^n e^{\frac{f_j}{\tau}}}$$

f_i is fitness of i and τ is temperature value. As τ goes infinity selection becomes uniform. As τ goes to zero selection becomes greedy respect to fitness values.

Experiment results with this selection gave better average values throughout generations however it did not affect that much best found solutions over generations. But it showed slight improvement.