MHO – ASSIGNMENT 1

CONTENTS

This report mostly contains analysis based on the tables and graphs generated at different runs of Genetic algorithms, looks bit lengthy, but mostly contains the graphs and values which makes it easy to analyse the outcome.

AUTHOR: SHANKAR PENDSE

STUDENT ID: R00195877

COURSE: MSc AI

MHO – ASSIGNMENT 1

PART 1

Proof of NP-Completeness of 3COL:

1. **Convert the formula F into a 3SAT formula F`, Find a solution to F` and verify that this is a solution to F:**

   My student ID is **R00195877**, I will be using the below formula:
   F = (-p1 ∨ -p3) ∧ (p1 ∨ -p2 ∨ p3 ∨ -p4 ∨ p5∨ -p6)

   Solution: The given formula F is having two clauses:
   C1 = (-p1 ∨ -p3)
   C2 = (p1 ∨ -p2 ∨ p3 ∨ -p4 ∨ p5∨ -p6)

   a. **Converting C1 = (-p1 ∨ -p3) into a 3SAT Clause:**
      This clause has 2 literals ~p1 and ~p3, so we need one extra literal let us call that extra literal as "y11" , we will also have two clauses as shown below:

      C`11 = (-p1 ∨ -p3 ∨ y11)
      C`12 = (-p1 ∨ -p3 ∨ -y11)

   b. **Converting C2 = (p1 ∨ -p2 ∨ p3 ∨ -p4 ∨ p5∨ -p6) into a 3SAT Clause:**
      This clause has 6 literals **(K = 6)**:
      i) We need **K-3 = 3** extra literals: "y21", "y22", "y23"
      ii) We will have **K-2 = 4** clauses in total as shown below:

      C`21 = (p1 ∨ -p2 ∨ y21)
      C`22 = (-y21 ∨ p3 ∨ y22)
      C`23 = (-y22 ∨ -p4 ∨ y23)
      C`24 = (-y23 ∨ p5 ∨ -p6)

   Now that we have converted each of the given 2 clauses in formula F into 3SAT clauses, our 3SAT formula F` will be Conjunctive Normal Form (CNF) of all the 3SAT clauses above.

   F` = (-p1 ∨ -p3 ∨ y11) ∧ (-p1 ∨ -p3 ∨ -y11) ∧ (p1 ∨ -p2 ∨ y21) ∧
   (-y21 ∨ p3 ∨ y22) ∧ (-y22 ∨ -p4 ∨ y23) ∧ (-y23 ∨ p5 ∨ -p6)

Now we have to find a solution (assignment of truth values to the F` literals) such that F` is satisfied (resolved to True) and the same truth value assignments will also satisfy the original F. And below is one such assignment which satisfies both:

Assignment of Truth values for F` is shown in below figure:

Literals	Truth Value Assignment	(-p1 V -p3 V y11)	(-p1 V -p3 V -y11)	(p1 V -p2 V y21)	(-y21 V p3 V y22)	(-y22 V -p4 V y23)	(-y23 V p5 V -p6)	CNF
p1	F	T	T	T	T	T	T	T
p2	T							
p3	F							
p4	T							
p5	T							
p6	T							
y11	T							
y21	T							
y22	T							
y23	T							
~p1	T							
~p2	F							
~p3	T							
~p4	F							
~p5	F							
~p6	F							
~y11	F							
~y21	F							
~y22	F							
~y23	F							

The same Truth values of literals are applied to original give formula F and its outcome is shown in below figure:

Literals	Truth Value Assignment	(-p1 V -p3)	(p1 V -p2 V p3 V -p4 V p5V -p6)	CNF
p1	F	T	T	T
p2	T			
p3	F			
p4	T			
p5	T			
p6	T			
~p1	T			
~p2	F			
~p3	T			
~p4	F			
~p5	F			
~p6	F			

We can see that the same truth value assignment as of F` satisfied the original given Formula F.

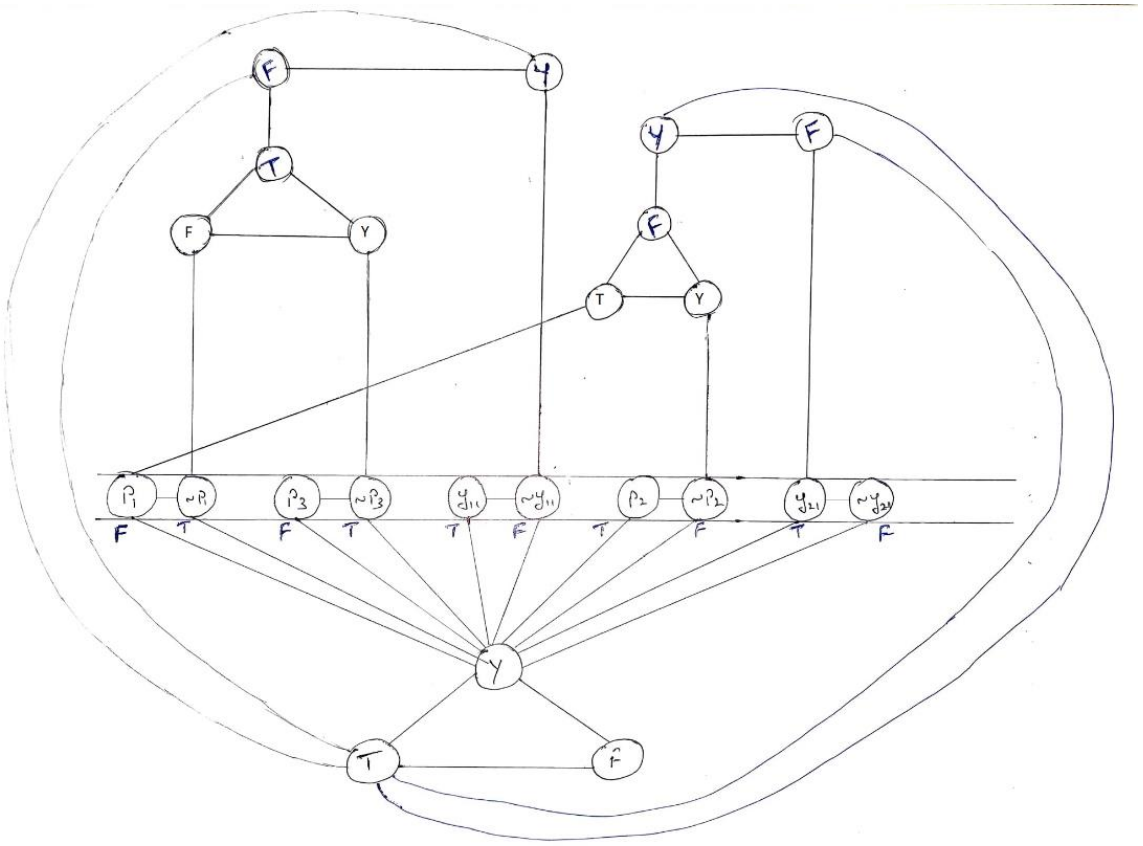
Thus, we have successfully converted SAT to 3SAT

MHO – ASSIGNMENT 1

2. Converting the subclauses in F` to a 3COL graph:

My First name is Shankar, I will be using 2nd and 3rd clauses of F`:
C`12 = (-p1 V -p3 V -y11)
C`21 = (p1 V -p2 V y21)

We need to use **OR Gadgets** to convert 3SAT to 3COL graph and the result of converting the above clauses to 3COL graph is as shown below:



CONCLUSION:

1. To prove that 3COL graph is in NP-COMPLETE:
    a. We have to show that 3COL is in NP (That is the solution is verifiable in polynomial time) which we can do so as we have fixed literals and assignment such that no two nodes connected by an edge can take the same colour
    b. Choose 3SAT which is in NP-Complete and reduce/convert the 3SAT problem into 3COL graph in polynomial time. (we did this in the second step)

MHO – ASSIGNMENT 1

PART 2

Genetic Algorithms:

We will be applying Simple genetic algorithm to our Travelling Salesman Problem (TSP) where our aim is to find the better solution out of all initial solutions provided.

Let's go through the steps in detail which are implemented as part of the solution provided in "**TSP_R00195877.py**"

1. **Population initialization:** we are using two methods to initialize the population:
    a. **Random initialization** which is provided by default in the code skeleton
    b. **Heuristic initialization** where I will be using Nearest Neighbor insertion method. (I am using my own code for this and a separate .py file is provided with this report just for reference, I have modified Individual.py to accommodate the use of Heuristic insertion)

2. **Fitness evaluation:** Initially for each of the individual/chromosome/tour generated in our population, we will calculate the fitness by computing the total distance travelled in the current tour using **Euclidean distance.**

3. **Selection: Binary Tournament selection** method is used, where randomly 4 tours/individuals will be chosen, from that random selection, we will make 2 pairs at random and return the fittest tour/individual from each pair as parent. (parent1 and parent2)

    Note: we will be referring to tour/individual as parent/child and city ids in the tour as genes, just to get the impression of **Genetic algorithm** and easier to explain

4. **Crossover:** It is a process where set of genes chosen from selected parents are combined to form one or more children. Below two crossover techniques are implemented:
    a. **Uniform order-based crossover:** Where certain indices are selected at random and the genes at those indices will not be changed. The remaining indices will be filled with missing genes from the other parent in order they appear. Since multiple indices can be selected, we are selecting $1/4^{th}$ of the geneSize indices where the gene values won't change

    b. **Order 1 crossover:** Select sequence of genes between two randomly selected indices from one parent1, remove the genes in that sequence from parent2, copy the remaining genes from parent2 into a child chromosome and then append initially selected sequence to the child chromosome.

    Note: only 1 child is created and returned from these methods

5. **Mutation:** It is a process where a new gene sequence/s can be introduced in the child obtained after crossover. While crossover selects the existing sequence, mutation has the ability to introduce a completely different gene sequence.

   With respect to TSP problem, crossover would select better sequence of city IDs from two selected parents and combine to form one or more children without altering the selected sequence, whereas mutation will introduce new sequence in the child obtained from crossover, if at all the induced mutation gives us better fitness, then we can consider the mutated child as the best solution (the child/children obtained just after crossover would also have better fitness)

   We will be using mutation probability per chromosome/individual instead of for each gene in a chromosome

   Below two mutation techniques/operators are implemented:

   a. **Inversion Mutation:** Select two indices at random, invert the sequence of genes between those two indices while other genes remain unaffected in their position/index. **With respect to TSP problem, after inversion, only two distances will be altered (one each at the selected indices)**

   b. **Scramble Mutation:** Select two indices at random, shuffle all the genes between those two indices which introduces a completely a new order of genes between the selected indices. Compared to Inversion mutation, **this introduces more variation** in the resulting child

6. **Updating the old population:** Once we are done till step 5, we will replace the old population with new population, after calculating the fitness of each individual/chromosome, create a copy of this in a mating pool and start with the selection process. Repeat the cycle Selection → Crossover → Mutation until a termination criterion is reached.

MHO – ASSIGNMENT 1

Evaluation of the implementation on Basic configurations:

Configuration	Initial Population	Crossover	Mutation	Selection
1	Random	Order -1 Crossover	Inverse Mutation	Binary Tournament Selection
2	Random	Uniform Crossover	Scramble Mutation	Binary Tournament Selection
3	Random	Order -1 Crossover	Scramble Mutation	Binary Tournament Selection
4	Random	Uniform Crossover	Inverse Mutation	Binary Tournament Selection
5	Heuristic	Order -1 Crossover	Scramble Mutation	Binary Tournament Selection
6	Heuristic	Uniform Crossover	Inverse Mutation	Binary Tournament Selection

For basic evaluation, all of the above configurations have been run once with 500 iteration.

As my surname is starting with **"P"**, I will be making use of **inst-4.tsp, inst-16.tsp and inst-6.tsp** files

SECTION 1:

Analysis on outcome from basic configurations:

Introduction: For TSP, we do not really know what is the good solution (least distance travelled while covering all the cities in the list exactly once), There could be multiple good solutions and each one of them could be different (In the sense, the order of the cities visited) and this cannot be solved in polynomial time. This comes under the category of **NP-Hard** problems.

We are making use of simple Genetic algorithm to find better solution for TSP, which may or may not produce the best solution. The steps of Genetic algorithm that is implemented as part of this project is already described in the initial section of this report. We will tag any solution as better solution with least fitness value. (least distance travelled) **As this is a minimization problem** fittest solution is the one with least fitness value.

Implementation: We always perform our selection from mating pool which gets updated after entire population is replaced, that will be used in our next iteration. We are not removing or replacing the selected parents from **mating pool**. This implementation is called "**with repetition"**, so during next selection process (till the entire length of population/entire list of tours that is generated is covered), the probability of the same parents being selected is same as it was for the previous selection. So, for one **new generation** step which spans the entire population of solutions, each solution will be replaced in the population with the newly generated solution (child) after performing crossover and mutation.

MHO – ASSIGNMENT 1

Analysis: Let's first look at the max iterations that we are using in our basic initial configuration which is a set at 500. As we do not know what the best solution is or what is the expected solution to our TSP, we cannot set any other condition as termination criteria for our Simple Genetic Algorithm. It is important to set a termination criterion as we run the steps of selection→crossover→mutation→updation repeatedly in a loop.

Setting the termination criteria to max of 500 iterations, ensures that we will definitely come out of the loop and hope that we get the better solution to our problem, which is again not guaranteed.

Let's look at each of the configuration and the result after using it on inst-4, inst-6 and inst-16 instances of TSP dataset. **Our main aim here is to try to generate different possible solutions, but in a faster way.**

Configuration1: (Random Initialization, order-1 crossover and inverse mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
1	inst-4.tsp	21968746.4	18498268.62	19
	inst-16.tsp	111439618.5	91722572.16	33.29
	inst-6.tsp	347962884.3	316425864.6	141.42

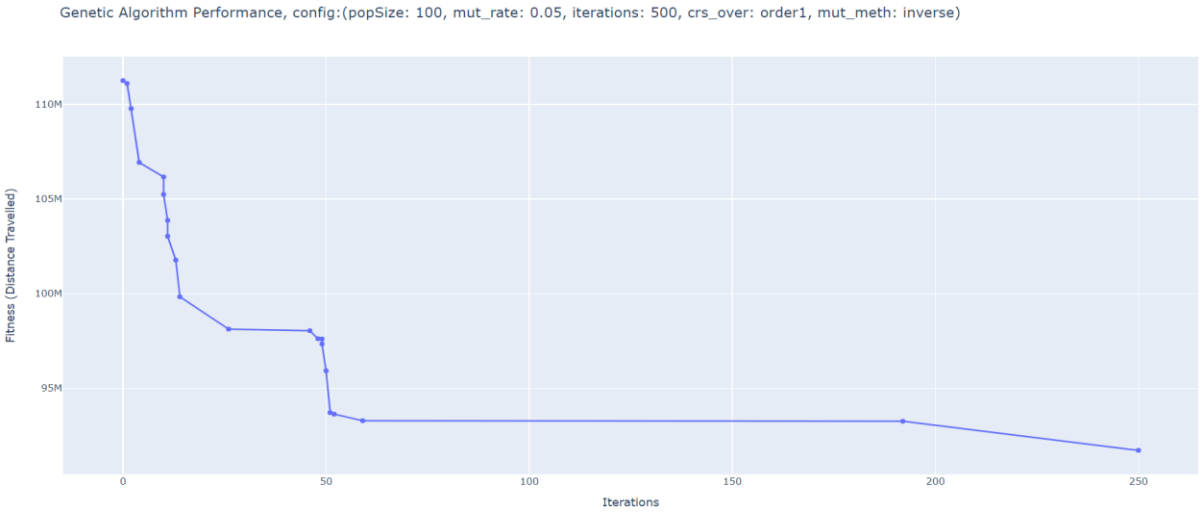
From above numbers, we can see that as the gene size (no of cities to be visited) increases, the run time of our genetic algorithm also increases, but we are getting better fitness score after 500 iterations when compared to the **initial Best solution**.

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

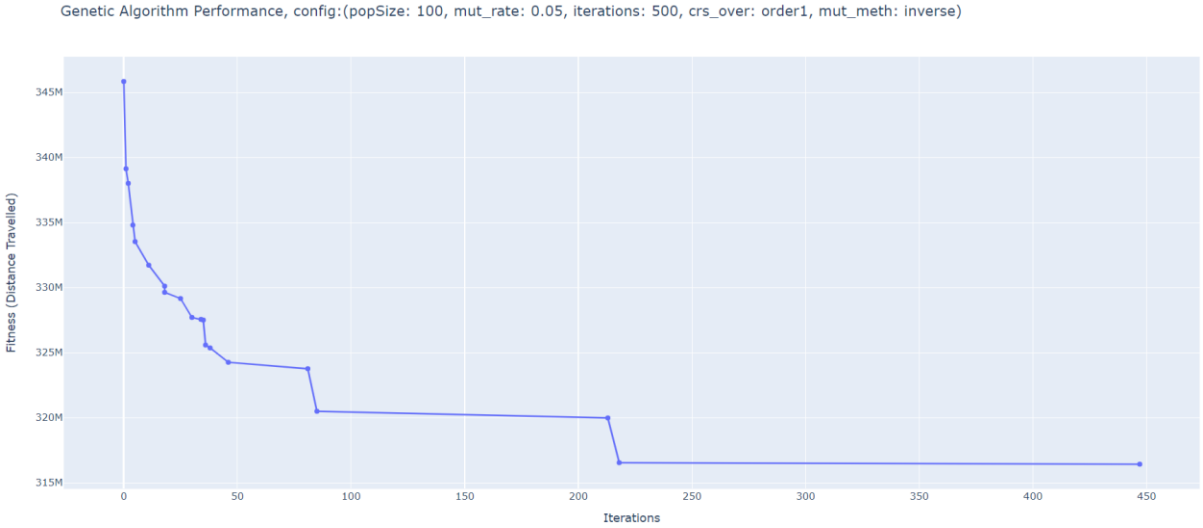
File: inst-4.tsp

MHO – ASSIGNMENT 1

File: inst-16.tsp



File: inst-6.tsp



From above graphs we can see that, initially for few iterations, more number of better solutions are generated, but as we progress, it gets bit hard to find the better solution than the previous ones, this is basically because, we are implementing **order-1 crossover** which **retains large part of the sequence of genes as it is** and later applying **inverse mutation** if at all it happens (depending on the mutation probability that is set) which just introduces two new combination of city order. Basically, we are not playing much with the result of crossover, which itself is retaining a larger part of the sequence unaltered.

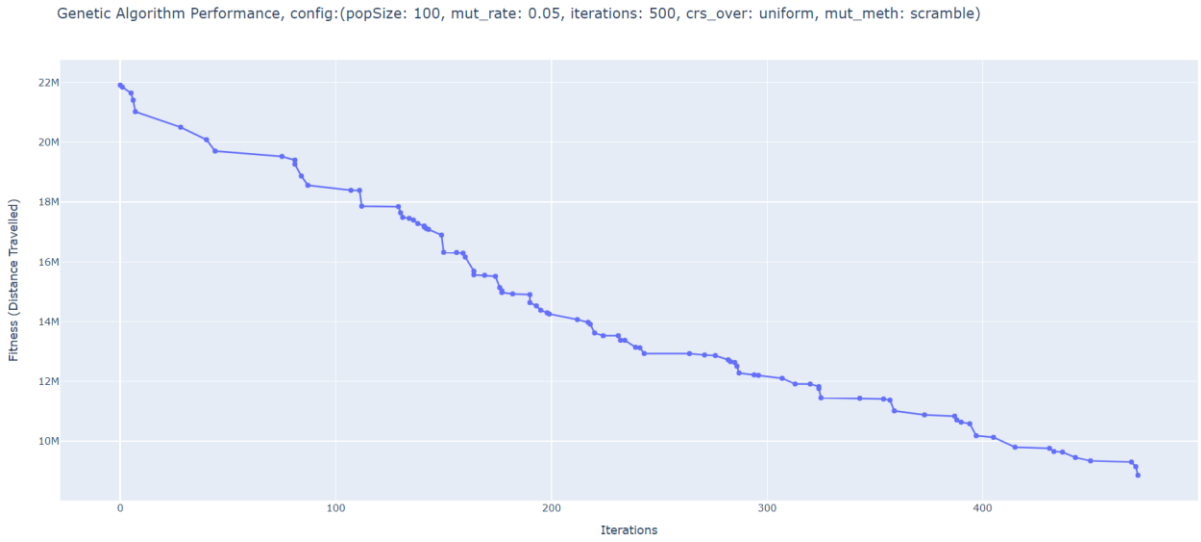
Configuration2: (Random Initialization, uniform order-based crossover and scramble mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
2	inst-4.tsp	21968746.4	8857354.41	33.95
	inst-16.tsp	111439618.5	30745475.09	70.64
	inst-6.tsp	347962884.3	300218820.9	416.12

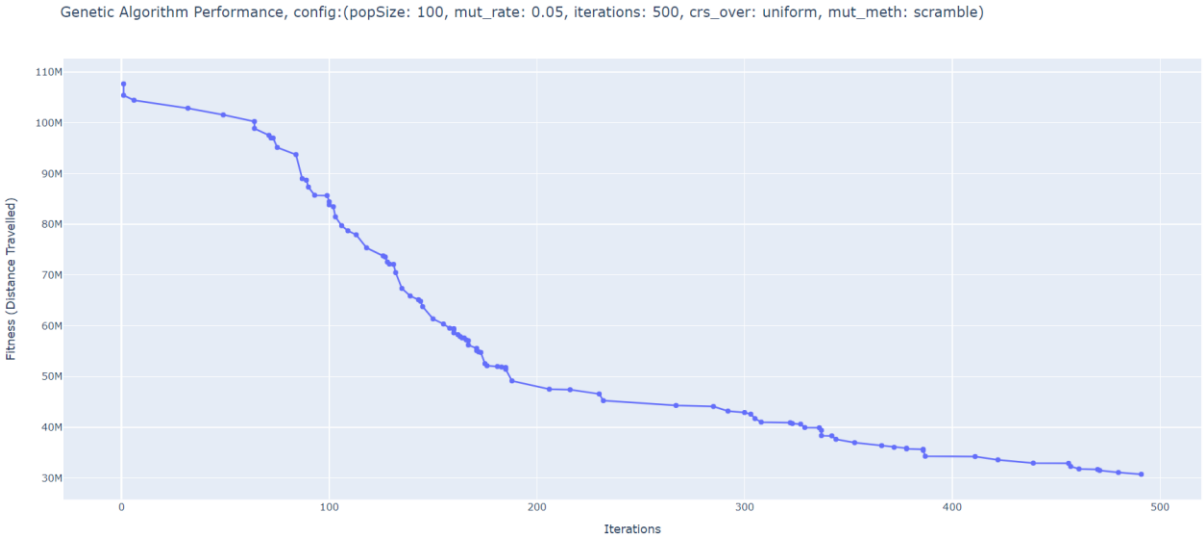
We can see that the run time has increased drastically with this configuration , because in uniform order based crossover we need to search for the empty index in our child, to insert the non-repeating element(gene) in the order it appears in the other parent, this takes higher toll on run time if there are more number of genes in the individual.

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

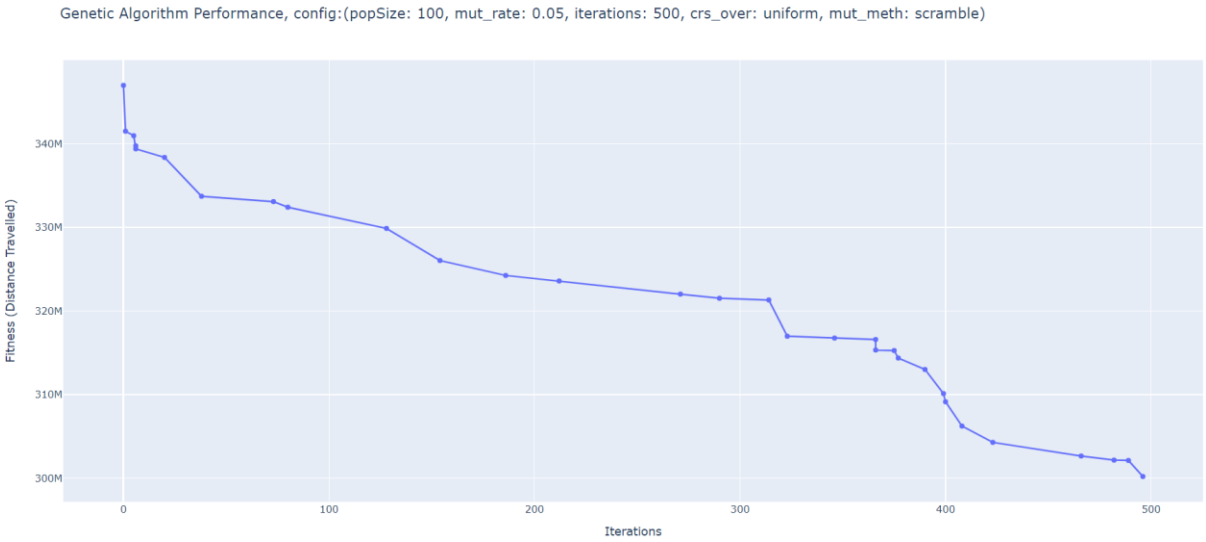
File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp



From above graphs we can see the improvement in fitness more frequently and sometimes, multiple better solution on the same iteration (as we are replacing the entire population in each iteration), this is because of the **combination of crossover and mutation operator that we have chosen**. **Uniform order based crossover**, basically does not maintain the order from any of the parent, it just inserts them in the order that they appear into the empty index of the child, thus not retaining larger part of the sequence as it is, which introduces completely new tour. Also, important to note that we are making use of **Scramble mutation**, which shuffles the genes between the selected indices, thus again introducing a new set of sequence in the child resulted from crossover. Scramble mutation introduces more variation when compared to Inverse mutation.

Configuration3: (Random Initialization, order-1 crossover and scramble mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
3	inst-4.tsp	21968746.4	19224805.89	19.28
	inst-16.tsp	111439618.5	92092435.7	34.48
	inst-6.tsp	347962884.3	318417168.8	146.25

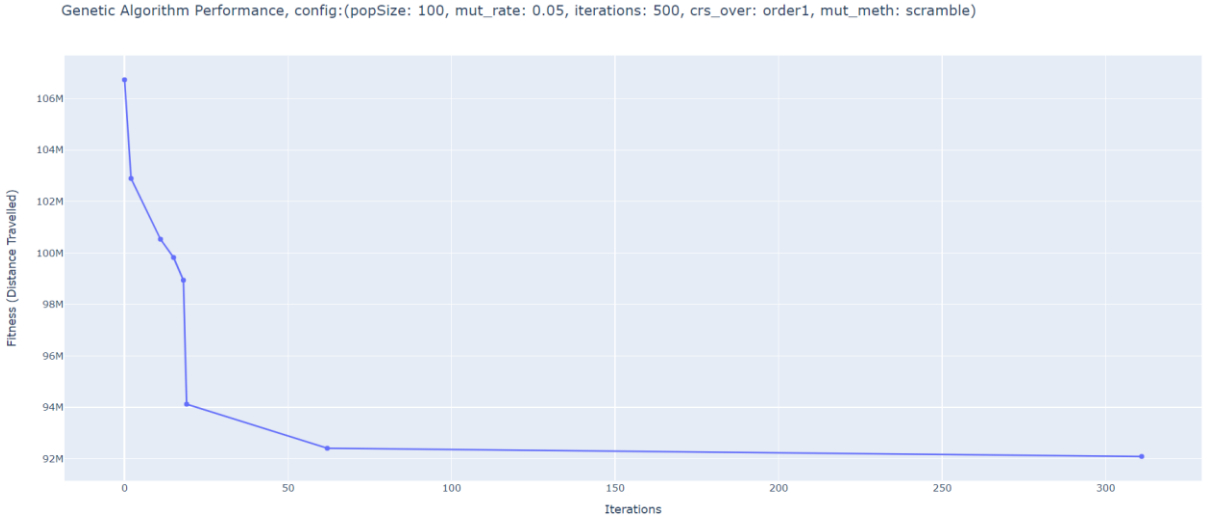
We can see that; the run time is almost similar to the one we had for our **Configuration1**,

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

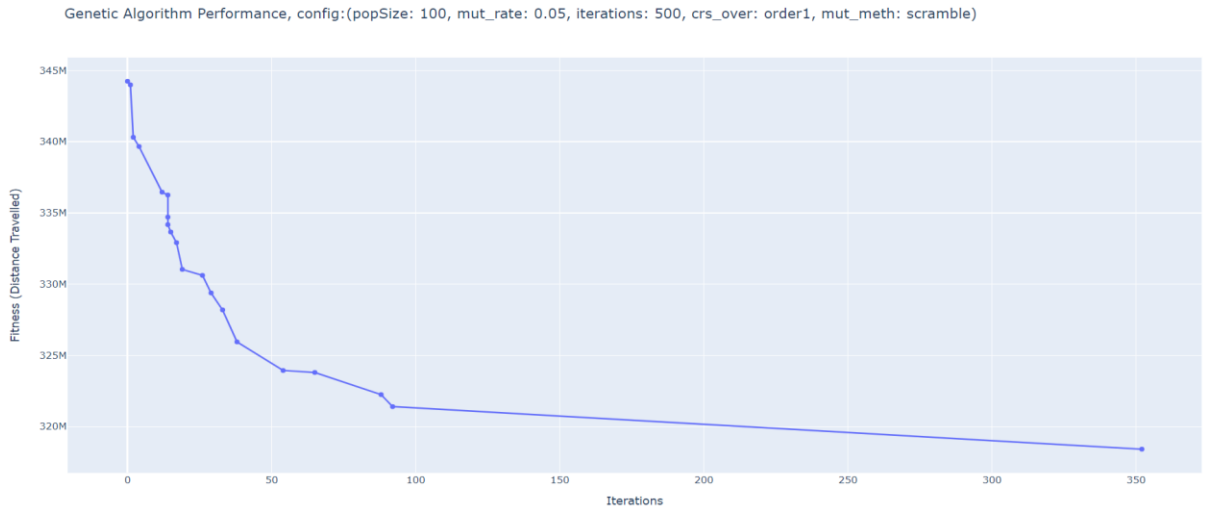
MHO – ASSIGNMENT 1

File: inst-4.tsp

File: inst-16.tsp



File: inst-6.tsp



If we look at the above graphs, they are somewhat similar to the ones we had for our **configuration1**. As we progress with higher iteration, it is bit hard to find the better solution than the previous ones. This is again because of the crossover method chosen which is order-1 crossover. But if we observe the graphs closely, we can see that there are more number of better solutions during initial iterations due to the nature of scramble mutation(more destructive in nature) compared to inverse mutation. If you look at the final best solution after 500 iterations and compare it to the ones obtained from **configuration1**, the results are not better. (**fitness values in this case is higher than the ones obtained from configuration1**). This is because order-1 based crossover retains larger part of the gene sequence as it is.

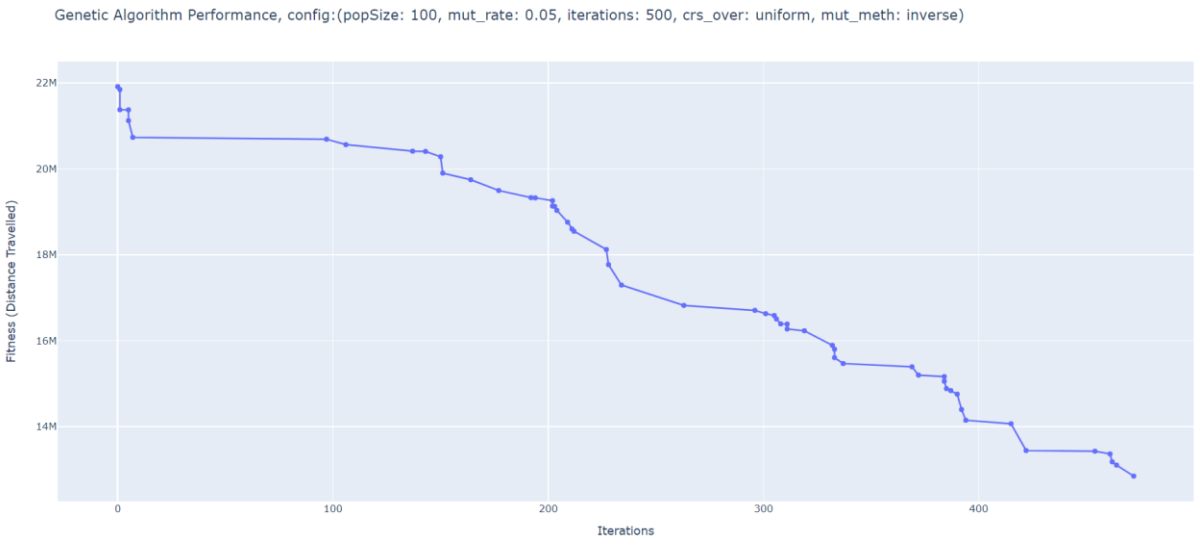
Configuration4: (Random Initialization, uniform order-based crossover and inverse mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
4	inst-4.tsp	21968746.4	12847982.03	33.84
	inst-16.tsp	111439618.5	43244970.06	70.12
	inst-6.tsp	347962884.3	302537940.6	412.87

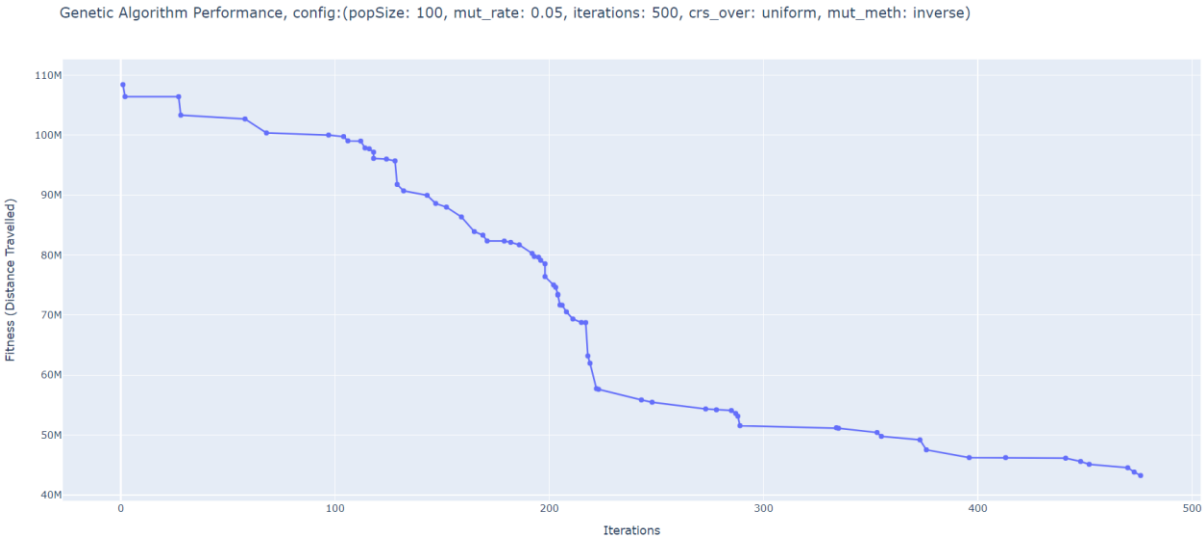
We can see that; the run time is almost similar to the one we had for our **Configuration2**,

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

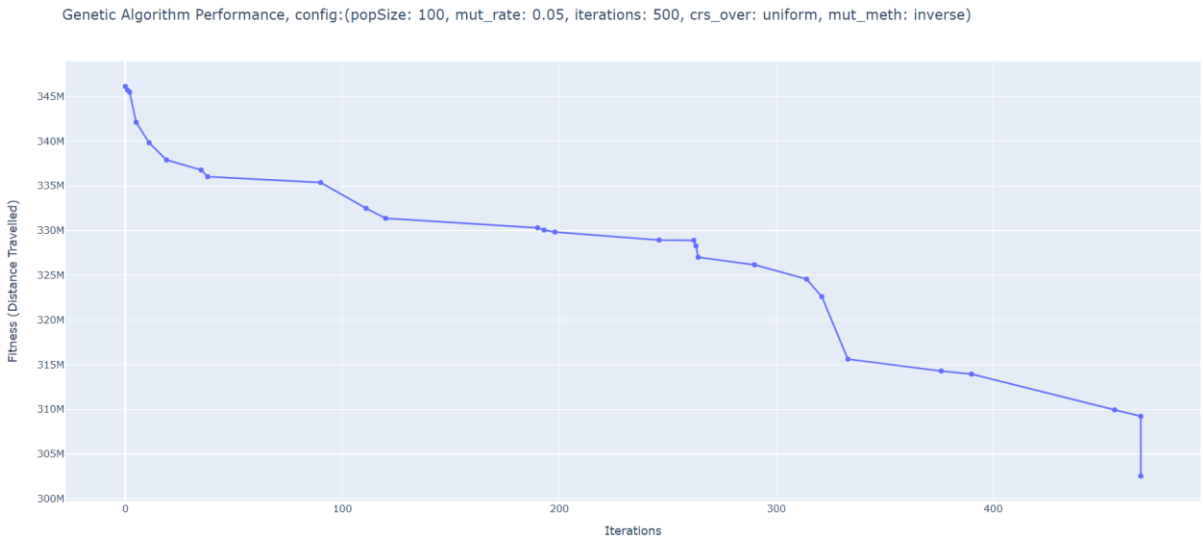
File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp



From above graphs we can see the improvement in fitness is more frequent, this is because of the **combination of crossover and mutation operator that we have chosen**.

Configuration5: (Heuristic Initialization, order-1 crossover and scramble mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
5	inst-4.tsp	3885429.23	3885429.23	24.35
	inst-16.tsp	7401694.91	7401694.91	42.65
	inst-6.tsp	12892859.59	12892859.59	187.17

We can see that; the run time is almost similar to the one we had for our **Configuration1/3**, but there is a slight increase by few seconds, as the initialization method is heuristics (nearest neighbor insertion).

Our GA is not able to improve the initial best solution produced from heuristic initialization for this configuration.

Configuration6: (Heuristic Initialization, Uniform order-based crossover and inverse mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
6	inst-4.tsp	3885429.23	3885429.23	69.89
	inst-16.tsp	7401694.91	7401694.91	162.81
	inst-6.tsp	12892859.59	12892859.59	1061.98

Same is the case with this configuration as of configuration5, our GA is not able to improve the best solution produced from heuristic initialization. we will further explore if it is possible to improve this in our experimentation section.

SECTION 2:

Analysis of GA performance by increasing the population size, max_iterations and exploring different mutation rates:

We will run our analysis on smallest file "inst-4.tsp" for all 6 configurations.

We choose smallest available file to get the results quicker, as the number of cities increases, it gets computationally expensive.

	CONFIG 1				
Tries	**1**	**2**	**3**	**4**	**5**
NEW_POP_SIZE	300	300	600	1000	1000
NEW_MAX_ITERATIONS	800	800	800	800	800
MUTATION_RATE	0.05	0.005	0.005	0.005	0.001
INITIAL BEST FITNESS	21864037.71	21864037.71	21649080.11	21547951.51	21547951.51
FINAL BEST FITNESS	18793558.5	19298045.63	19102496.99	18699352.25	18983649.29
BASIC BEST FITNESS	18498268.62				
BASIC RUN TIME(IN SEC)	19				
NEW RUN TIME(IN SEC)	83.25	83.81	168.43	286.35	281.17

	CONFIG 2				
Tries	**1**	**2**	**3**	**4**	**5**
NEW_POP_SIZE	300	300	600	1000	1000
NEW_MAX_ITERATIONS	800	800	800	800	800
MUTATION_RATE	0.05	0.005	0.005	0.005	0.001
INITIAL BEST FITNESS	21864037.71	21864037.71	21649080.11	21547951.51	21547951.51
FINAL BEST FITNESS	6666513.98	5150814.95	4831887.23	5307565.89	5834235.44
BASIC BEST FITNESS	8857354.41				
BASIC RUN TIME(IN SEC)	33.95				
NEW RUN TIME(IN SEC)	158.37	156.45	316.35	529.64	519.76

MHO – ASSIGNMENT 1

	CONFIG 3				
Tries	**1**	**2**	**3**	**4**	**5**
NEW_POP_SIZE	300	300	600	1000	1000
NEW_MAX_ITERATIONS	800	800	800	800	800
MUTATION_RATE	0.05	0.005	0.005	0.005	0.001
INITIAL BEST FITNESS	21864037.71	21864037.71	21649080.11	21547951.51	21547951.51
FINAL BEST FITNESS	19150628.26	19130337.92	18878357.49	18482260.92	18778949.01
BASIC BEST FITNESS	19224805.89				
BASIC RUN TIME(IN SEC)	19.28				
NEW RUN TIME(IN SEC)	86.6	167.1	178.92	305.15	293.45

	CONFIG 4				
Tries	**1**	**2**	**3**	**4**	**5**
NEW_POP_SIZE	300	300	600	1000	1000
NEW_MAX_ITERATIONS	800	800	800	800	800
MUTATION_RATE	0.05	0.005	0.005	0.005	0.001
INITIAL BEST FITNESS	21864037.71	21864037.71	21649080.11	21547951.51	21547951.51
FINAL BEST FITNESS	6664966.51	5213237.5	5613375.01	4691948.92	5206175.49
BASIC BEST FITNESS	12847982.03				
BASIC RUN TIME(IN SEC)	34.6				
NEW RUN TIME(IN SEC)	160.7	165.15	314.79	526.81	543.39

	CONFIG 5				
Tries	**1**	**2**	**3**	**4**	**5**
NEW_POP_SIZE	300	300	600	1000	1000
NEW_MAX_ITERATIONS	800	800	800	800	800
MUTATION_RATE	0.05	0.005	0.005	0.005	0.001
INITIAL BEST FITNESS	3885429.23	3885429.23	3885429.23	3885429.23	3885429.23
FINAL BEST FITNESS	3885429.23	3885429.23	3885429.23	3885429.23	3885429.23
BASIC BEST FITNESS	3885429.23				
BASIC RUN TIME(IN SEC)	24.35				
NEW RUN TIME(IN SEC)	118.09	112.12	235.26	384.87	394.04

Observations:

1. Keeping Iterations same and increasing the population size, will generally improve the fitness score, but at the cost of run time.
2. Mutation probability must be chosen carefully as we do not have any standard rule as to what value should be set. We will just have to try out different mutation probabilities and compare the fitness score

For configuration 5 our GA is not able to find the better solution than the one with heuristic initialization found, and same holds good for configuration 6 as well. We will be checking the solutions again for all of the configurations with different approach for each of the implemented methods for crossover as part of our experimentation.

SECTION 3:

Analysis of overall performance of GA:

As it has been asked to run each configuration for 5 times, we will be capturing the best fitness during each run and record the mean and median fitness across the runs. Let's look at them one by one.

We will use the smallest instance file inst-4.tsp as it is expensive to run each configuration for 5 times with largest file inst-6.tsp.

Expectation: We are expecting results for each configuration to be different (as we are making use of different combinations of crossover and mutation operators) but across the 5 runs of each config, the results should not vary much (they should be consistent) as we are initializing the population at every run

Below are the results of running each configuration for 5 times:

Configuration1:

File	Config1				
inst-4.tsp	**Run1**	**Run2**	**Run3**	**Run4**	**Run5**
Best Initial Solution	21968746.4	21583990.47	22105120.56	21725598.48	21757957.06
Best Solution	18498268.62	**18389260.06**	18505660.12	18596136.74	18631385.09
Mean Fitness	18524142.13				
Median Fitness	18505660.12				
Run time	89.23				

Below are the observations:

1. Run time is in acceptable terms around 90 seconds in total for all the 5 runs
2. We can see very little difference across the runs for initial and final best solution, which is basically because of the randomness in place
3. Mean fitness and Median fitness are almost equal, which is another factor indicating there is very little variance with the values across all the runs
4. We can not say, there is an improvement at each run, as we are running the whole algorithm starting from population initialization, otherwise it would be just like running the once initialized population for 500*5 = 2500 iterations. Let's look at it by running it once for those many iterations

File	Config1
inst-4.tsp	**Run for 2500 iterations**
Best Initial Solution	21968746.4
Best Solution	18114379.4
Run time	89.42

5. There is no significant improvement with best solution when compared to 500 iterations, this is basically because of the choice of crossover and mutation operator (order-1 and inverse) which really does not change much in the already

present sequence. I tried running it with larger population to include more variety, but the final best solution has the fitness score of 18928301.92, which is actually bad than the one we got for lesser population which was 18114379.4.

Configuration2:

File	Config2				
inst-4.tsp	Run1	Run2	Run3	Run4	Run5
Best Initial Solution	21968746.4	21757957.06	22165107.9	21851963.72	21880609.82
Best Solution	**8857354.41**	10938638.72	9031776.53	11747925.77	10045354.03
Mean Fitness	10124209.89				
Median Fitness	10045354.03				
Run time	179.26				

Below are the observations:

1. Run time is greater than that of config1, basically because of uniform order-based crossover, which affects the ordering of the gene sequence much more than that of order-1 based configuration(used in config1).
2. Best solution fitness is better than that of config1, as the selected crossover and mutation operator does more alteration in the gene sequence and we are able to find better results with this one.
3. Comparing Mean and Median fitness scores, the results are meeting our expectation as we set out earlier. There is not much variation with the scores obtained
4. Let's check out the results with only one run but with 2500 iterations. Our expectation is that the best solution fitness improves by a considerable amount

File	Config2
inst-4.tsp	Run for 2500 iterations
Best Initial Solution	21968746.4
Best Solution	**5454328.5**
Run time	180.35

As expected the best solution fitness has improved by a considerable amount

Configuration3:

File	Config3				
inst-4.tsp	Run1	Run2	Run3	Run4	Run5
Best Initial Solution	21968746.4	21888871.46	21451654.28	22239657.6	22277042.35
Best Solution	19224805.89	18951170.61	19147951.82	19051457.86	18747402.83
Mean Fitness	19024557.8				
Median Fitness	19051457.86				
Run time	90.15				

Below are the observations:

1. Run time is almost same as that for Config1
2. Compared to config1, results are not better, instead we can slightly see the higher values for best fitness score at each run. This is because of the nature of scramble mutation, which introduces more variation. Again, to stress the point here, mutation does not guarantee to take us towards the better fitness score and also, it is not that mutation happens at each iteration over the entire population because of mutation probability that we are using here, and the randomness involved
3. Comparing Mean and Median fitness scores, the results are meeting our expectation as we set out earlier. There is not much variation with the scores obtained
4. This config will have similar effect on running it for 1 time for 2500 iterations, as of config1. That is, we cannot expect the fitness score to improve by considerable amount. Let's see the result

File	Config3
inst-4.tsp	**Run for 2500 iterations**
Best Initial Solution	21968746.4
Best Solution	**18485610.37**
Run time	88.85

The results are as expected, that is we did not see considerable amount of improvement in the fitness score

Configuration4:

File	Config4				
inst-4.tsp	**Run1**	**Run2**	**Run3**	**Run4**	**Run5**
Best Initial Solution	21968746.4	22060598.75	22116891.43	21694006.84	21958014.66
Best Solution	12847982.03	11164318.12	11300498.7	11387941.71	11200804.24
Mean Fitness	11580308.96				
Median Fitness	11300498.7				
Run time	176.68				

Below are the observations:

1. Compared to config2, we are getting better fitness scores here, though the crossover operator is same (uniform order-based), the mutation operator "**inverse**" is doing good job in providing us with the better fitness score.
2. Comparing Mean and Median fitness scores, the results are meeting our expectation as we set out earlier. There is not much variation with the scores obtained
3. From the above observation, this configuration should get us to better fitness score (considerable amount of improvement) if we run it for 1 time, but with 2500 iterations. Let's look at the result

File	Config4
inst-4.tsp	Run for 2500 iterations
Best Initial Solution	21968746.4
Best Solution	**5180874.97**
Run time	162.64

The result is as expected, the fitness score improved significantly

Configuration5:

File	Config5				
inst-4.tsp	Run1	Run2	Run3	Run4	Run5
Best Initial Solution	3885429.23	3885429.23	3885429.23	3885429.23	3885429.23
Best Solution	3885429.23	3885429.23	3885429.23	3885429.23	3885429.23
Mean Fitness	3885429.23				
Median Fitness	3885429.23				
Run time	115.68				

Below are the observations:

1. There is no improvement in the fitness score on any of the runs as we can see from above number, this is mainly because in this configuration we are using nearest neighbour insertion heuristics for initialising the population, which is providing much better solution compared to any other configurations best fitness score.
2. We may or may not see any improvement if we run it for only one time with 2500. Let's see, if we are lucky enough, our crossover and mutation operators take us to the better solution

File	Config5
inst-4.tsp	Run for 2500 iterations
Best Initial Solution	**3885429.23**
Best Solution	**3885429.23**
Run time	108.7

We did not get any improvement after running it for 2500 iterations

Configuration6:

File	Config6				
inst-4.tsp	Run1	Run2	Run3	Run4	Run5
Best Initial Solution	3885429.23	3908620.31	3885429.23	3900881.29	3885429.23
Best Solution	3885429.23	3908620.31	3885429.23	3900881.29	3885429.23
Mean Fitness	3893157.86				
Median Fitness	3885429.23				
Run time	354.45				

MHO – ASSIGNMENT 1

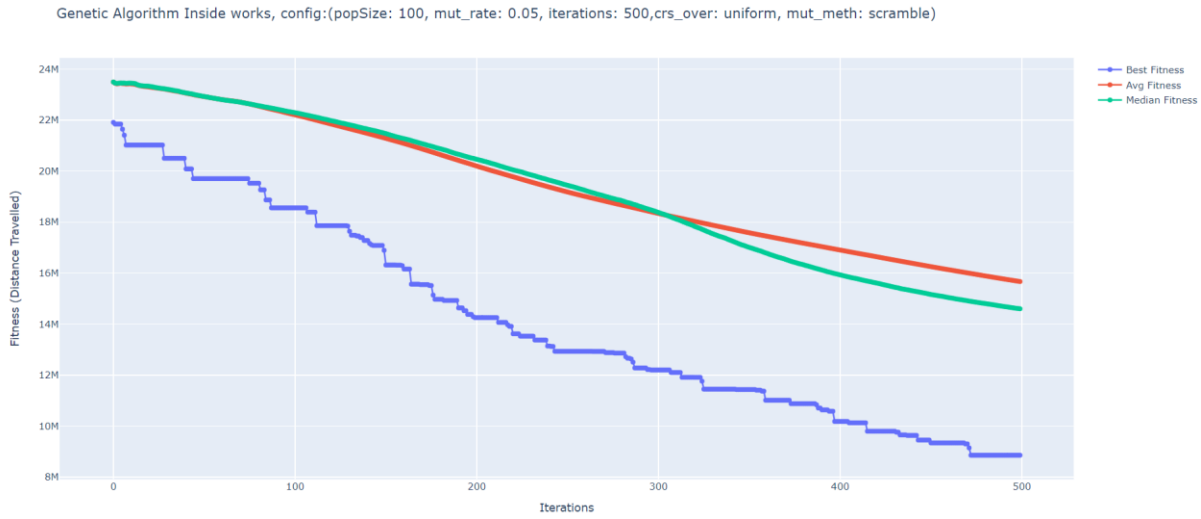
Below are the observations:

1. Observation is similar to that of Configuration5, there is no improvement, but only difference is on the best initial solution on Run2 and 4, which is no where related to the GA.
2. We may or may not see any improvement if we run it for only one time with 2500 iterations. Let's see if we get any

File	Config6
inst-4.tsp	Run for 2500 iterations
Best Initial Solution	3885429.23
Best Solution	3885429.23
Run time	345.37

With this configuration also we did not get any improvement from our GA

Now let us look at how the best fitness, average fitness and median fitness values vary across population for 500 iterations of configuration 2 on file inst-4.tsp:



From above graph, we can infer that, across the population over 500 iterations, mean and median values are not varying much, but the best individual varies largely, as we are applying crossover and mutation operators which brings in variety of gene sequence and different fitness values accordingly

SECTION 4:

EXPERIMENT:

Change in crossover logic of both operators (Uniform order-based and order-1):

1. In the first implementation, we generated and returned only one child after crossover
2. In this experimental analysis, we will create 2 children and return the fittest one

In this way, we are biasing more towards the fittest individual, and for next iteration, the mating pool will get updated with the fittest population from previous iterations.

We will see how the performance of our GA changes.

Expectation: As we are more biased towards the fittest individual during selection process at each iteration, our expectation here is to see much better results than the earlier run

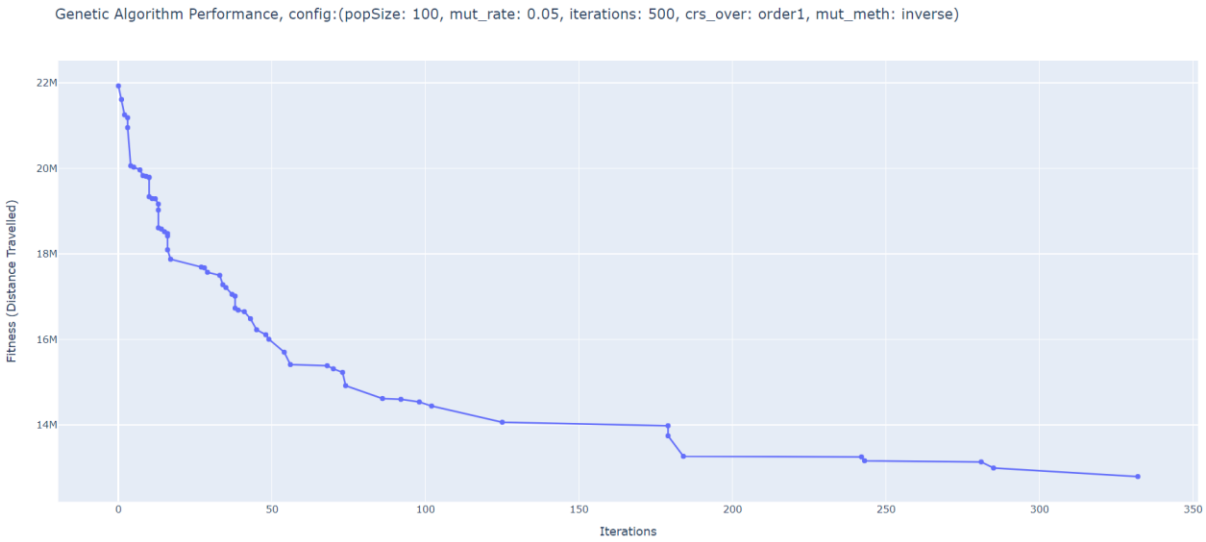
We will run each configuration for all the three instance files (inst-4, inst-16 and inst-6) and record the observation

Configuration1: (Random Initialization, order-1 crossover and inverse mutation)

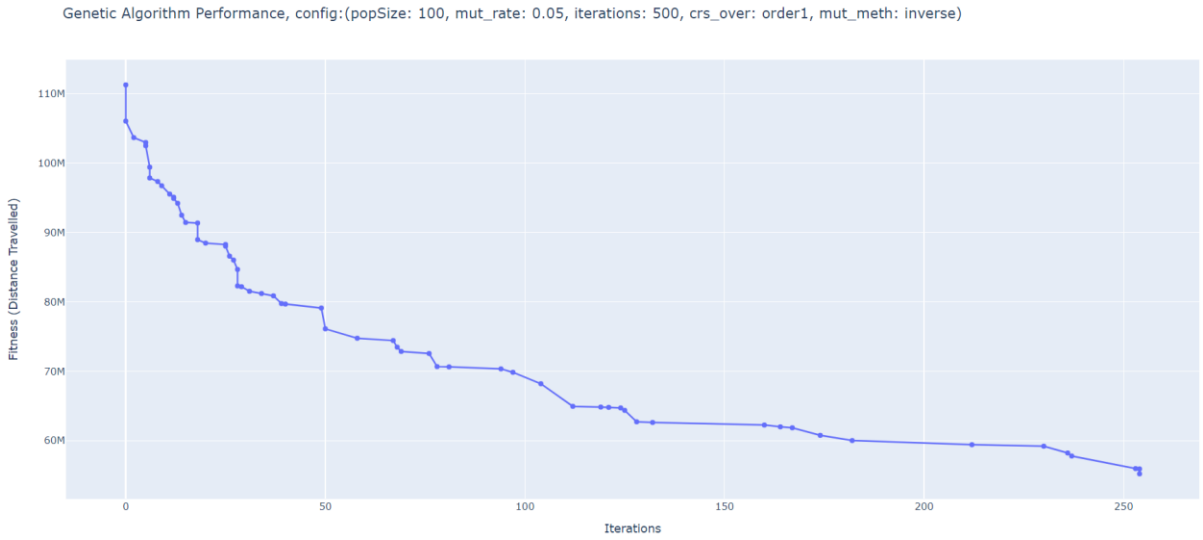
Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
1	inst-4.tsp	21968746.4	12793859.36	46.53
	inst-16.tsp	111439618.5	55211184.36	77.6
	inst-6.tsp	347962884.3	189129177.2	320.15

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp

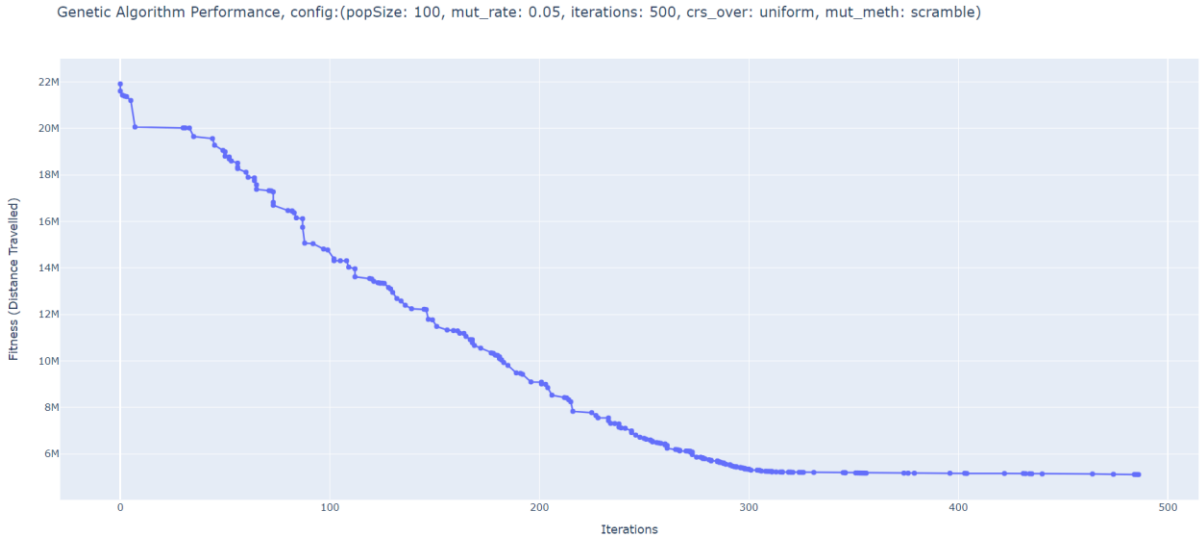
Configuration2: (Random Initialization, uniform order-based crossover and scramble mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
2	inst-4.tsp	21968746.4	5113368.12	73.34
	inst-16.tsp	111439618.5	14559650.65	147.23
	inst-6.tsp	347962884.3	132892127.3	855.17

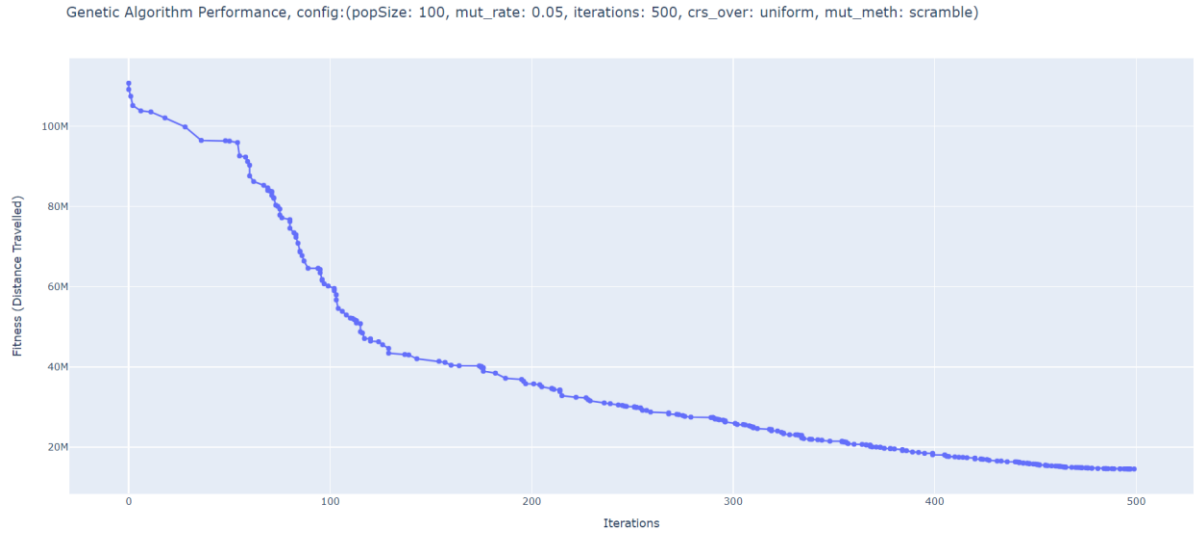
The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

MHO – ASSIGNMENT 1

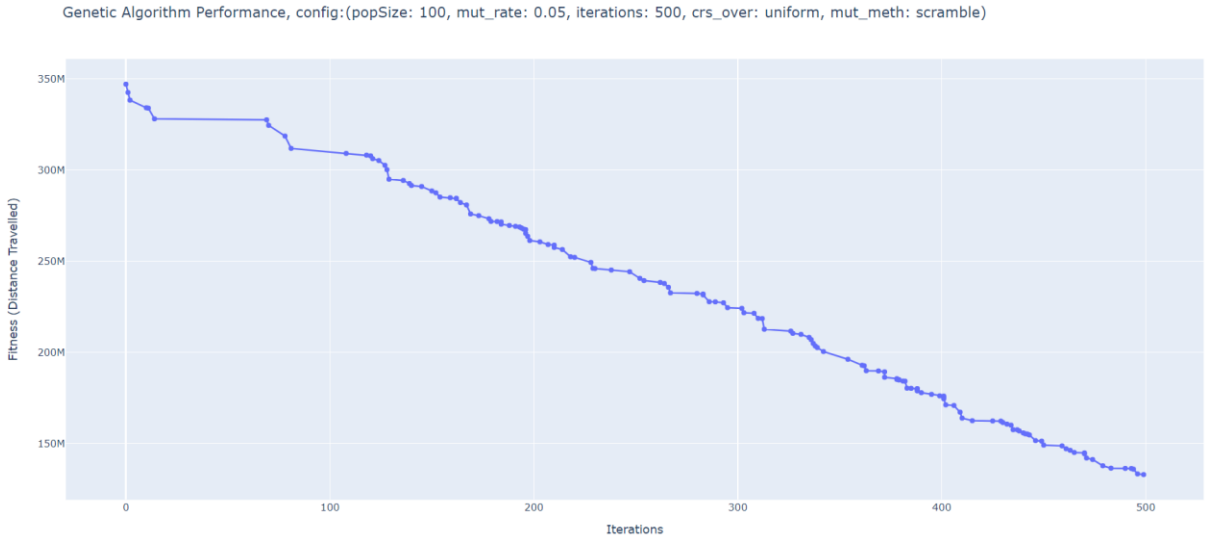
File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp



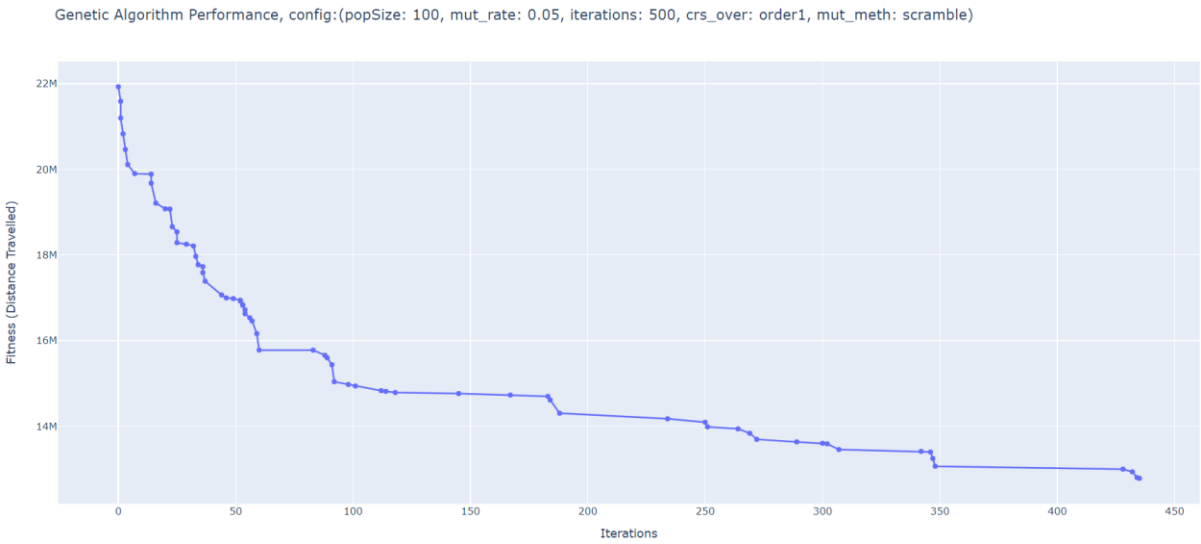
MHO – ASSIGNMENT 1

Configuration3: (Random Initialization, order-1 crossover and scramble mutation)

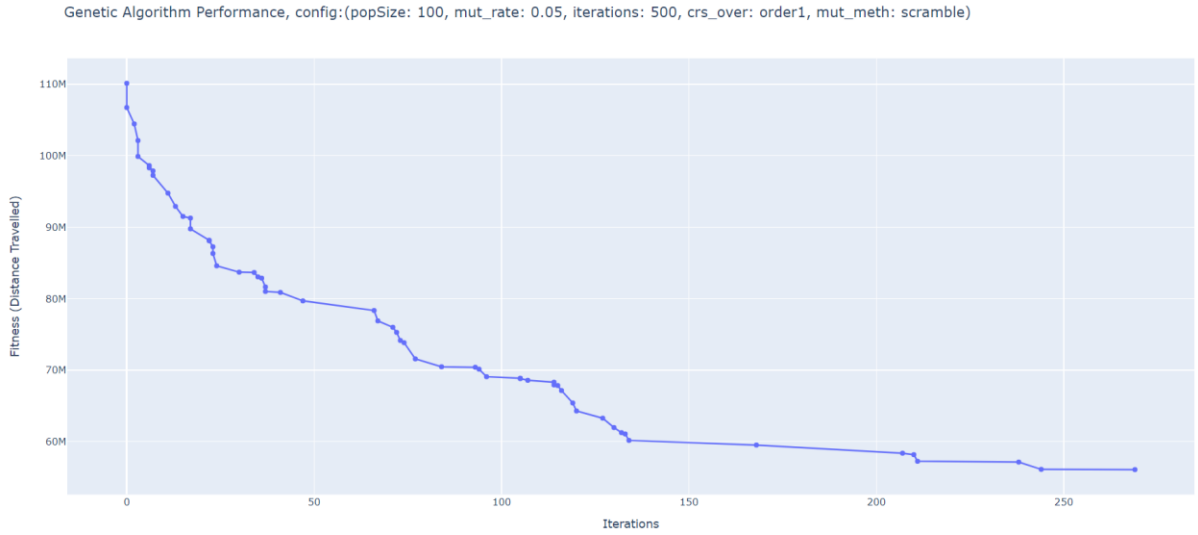
Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
3	inst-4.tsp	21968746.4	12779865.47	46.21
	inst-16.tsp	111439618.5	56093579.96	79.23
	inst-6.tsp	347962884.3	208626006.3	320.15

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

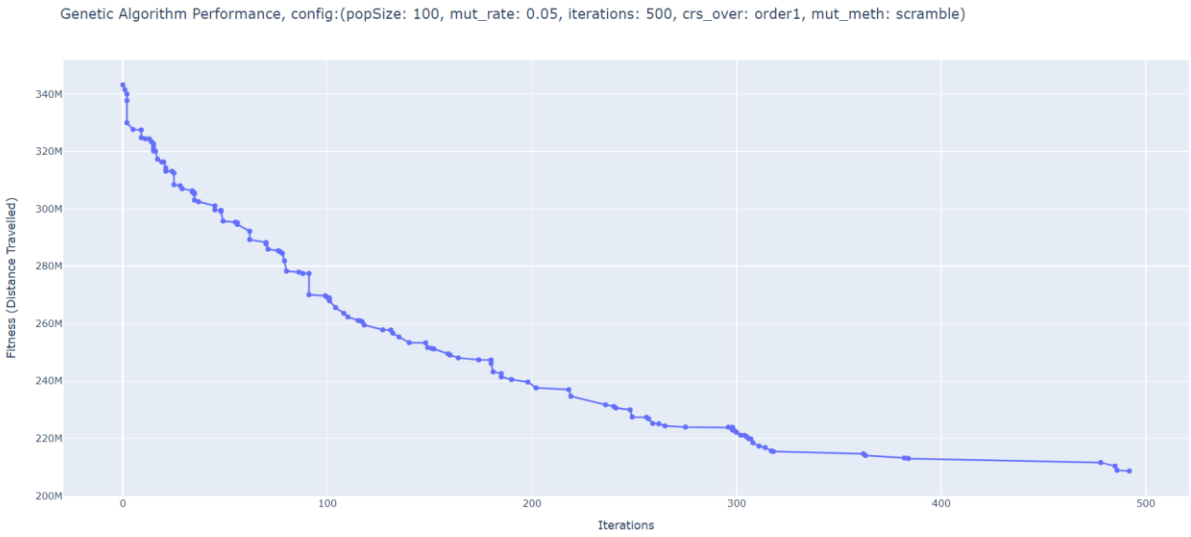
File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp

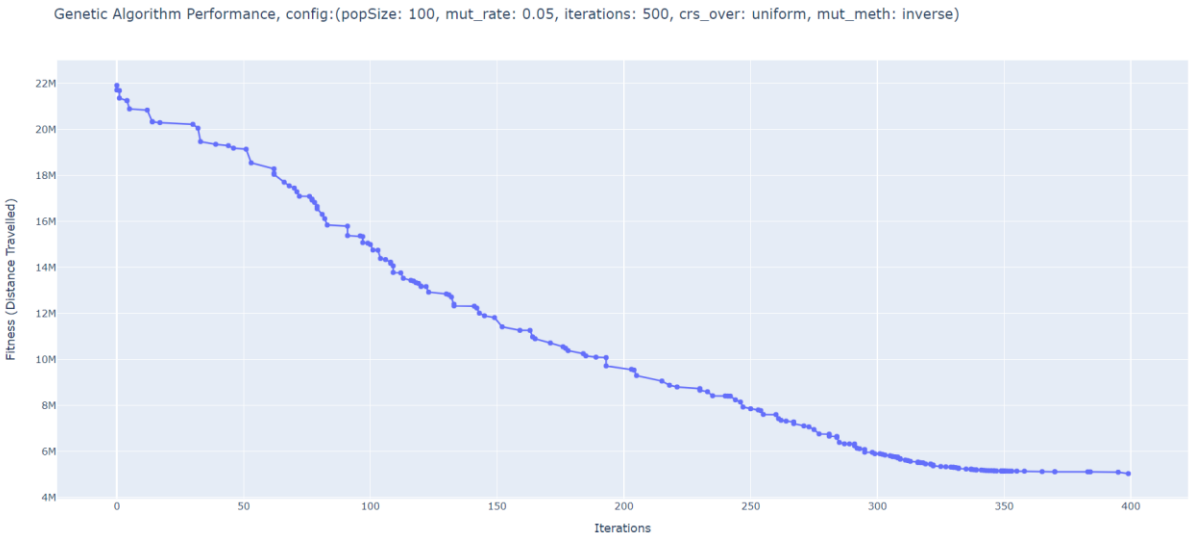


Configuration4: (Random Initialization, uniform order-based crossover and inverse mutation)

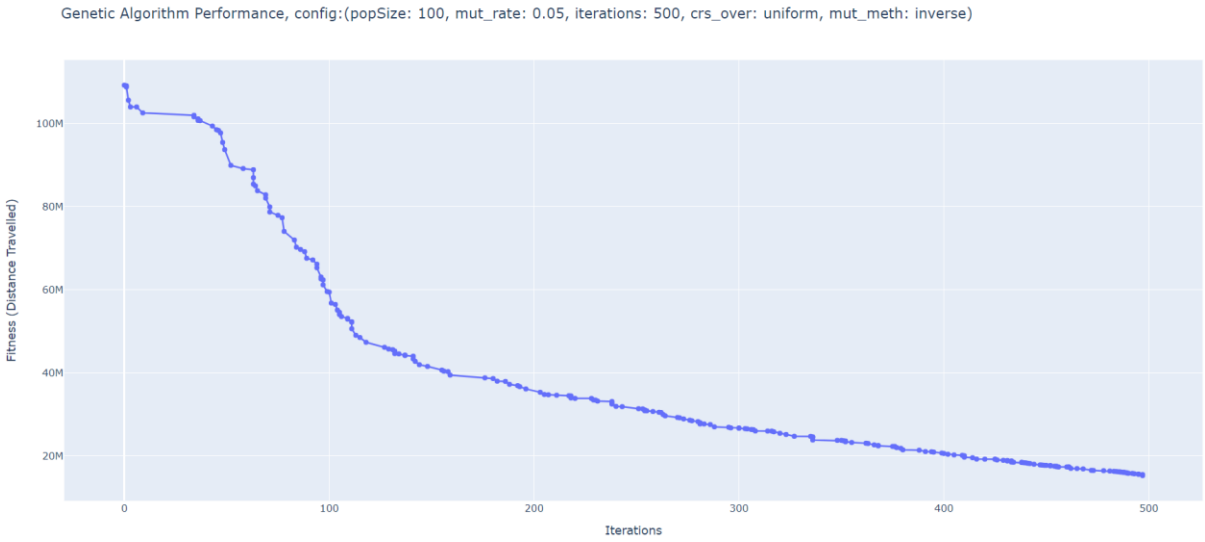
Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
4	inst-4.tsp	21968746.4	5037726.12	72.57
	inst-16.tsp	111439618.5	15284616.74	147.71
	inst-6.tsp	347962884.3	159948773.9	854.68

The convergence (reduction in the distance travelled) at different iterations (basically the better solution than the previous best one) is shown in below graphs for each of the instance files

File: inst-4.tsp



File: inst-16.tsp



File: inst-6.tsp

Configuration5: (Heuristic Initialization, order-1 crossover and scramble mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
5	inst-4.tsp	3885429.23	3885429.23	59.89
	inst-16.tsp	7401694.91	7401694.91	102.29
	inst-6.tsp	12892859.59	12782326.87	411.28

We can see that, our GA was not able to improve upon the score for the first two files, but for the larger file "**inst-6.tsp**" we got one improvement on top of our heuristic initial best solution.

Configuration6: (Heuristic Initialization, Uniform order-based crossover and inverse mutation)

Config	File	Initial Best solution	Final best solution	Runtime (in seconds)
6	inst-4.tsp	3885429.23	3885429.23	145.62
	inst-16.tsp	7401694.91	7401694.91	337.17
	inst-6.tsp	12892859.59	12892859.59	2046.62

Unfortunately, we did not get to see any improvement for this configuration.

As expected, the fitness scores are much better in this implementation

CONCLUSION:

1. Genetic algorithm tries to provide better solution at each iteration, but the results are not guaranteed. We have to play with population size and mutation probability to get to a better solution.
2. Generally, larger population should yield better result but at the cost of run time.
3. There is a kind of tradeoff between mutation probability and the resulting fitness of the individual. If the child from crossover is having better fitness, and we set high mutation probability, it is likely to get mutated and could result in undesired fitness of the mutated child, which we end up updating in the mating pool after completing one iteration. If child from crossover is having worse fitness score and we set very low mutation probability, it is likely that it won't get mutated and thus we end up updating the individual in population with worse fitness score.
4. As we have seen from all of the above analysis, higher population size (induces greater variation in each individual) starts to produce better results
5. We cannot directly set the mutation probability, but have to try out multiple values and check the result for each of them