

Contents

Part 1: NP Completeness	2
Part 2	4
Implementation Details	4
Selection and population	4
Crossover	4
Uniform Crossover	4
Order 1 Crossover	5
Mutation	5
Creating the Mating Pool	5
Results	5
Basic Runs	5
Configuration 1	5
Configuration 2	7
Configuration 3	10
Configuration 4	12
Configuration 5	13
Configuration 6	15
Comparison between Configurations	15
Varying Parameters.....	18
Varying Population Size.....	18
Varying Mutation Rate.....	21
Elitism.....	23

Part 1: NP Completeness

$$F = (\neg w_3 \vee w_4) \wedge (w_1 \vee \neg w_2 \vee w_3 \vee \neg w_4 \vee \neg w_5 \vee w_6)$$

$$F' =$$

1. $(\neg w_3 \vee w_4 \vee y_1)$
2. $\wedge (\neg w_3 \vee w_4 \vee \neg y_1)$
3. $\wedge (w_1 \vee \neg w_2 \vee y_2)$
4. $\wedge (\neg y_2 \vee w_3 \vee y_3)$
5. $\wedge (\neg y_3 \vee \neg w_4 \vee y_4)$
6. $\wedge (\neg y_4 \vee \neg w_5 \vee w_6)$.

One Solution for F' :-

$w_1 = F$	$y_1 = T$
$w_2 = T$	$y_2 = T$
$w_3 = F$	$y_3 = T$
$w_4 = T$	$y_4 = T$
$w_5 = T$	
$w_6 = T$	

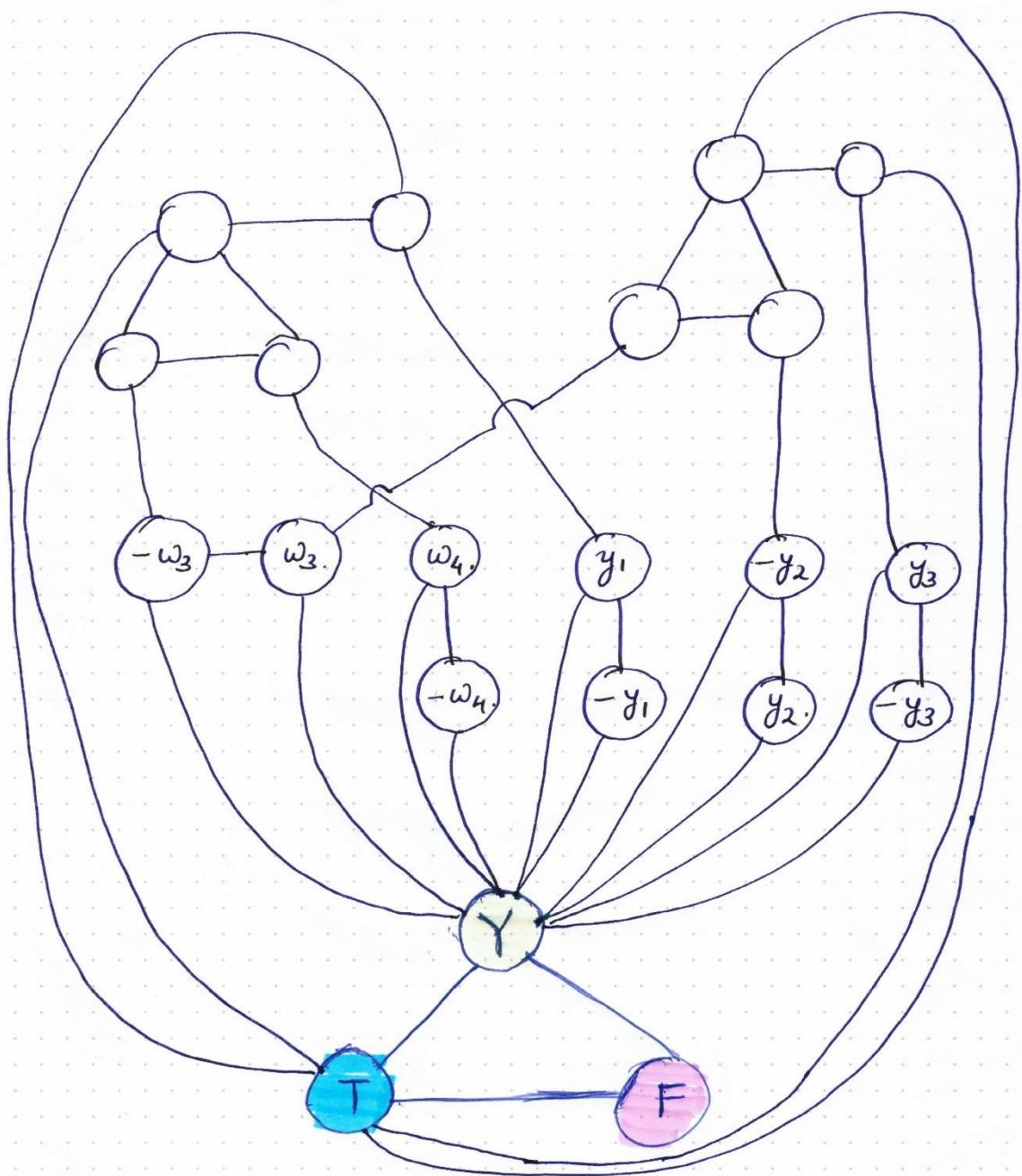
Clause 1 of F is satisfied because $w_4 = T$

Clause 2 of F is satisfied because $w_6 = T$

Therefore this solution for F' satisfies F

$3SAT \rightarrow 3COL$

- ① $(-\omega_3 \vee \omega_4 \vee y_1)$
- ② $(-\omega_2 \vee \omega_3 \vee y_3)$



Part 2

A genetic algorithm was created based on the skeleton code provided by Dr. Grimes. Tests were done with configurations 1-6.

The insertion heuristic used was the solution to the code provided by Dr. Diarmuid Grimes.

A multi-processing approach was used to run different runs in parallel. The default configuration is to do 7 runs in parallel. This can be changed by changing the global variable `g_n_processes`.

All graphs plotted were saved as `cpickle` in files. These graphs can be re-rendered using the script `plotagain.py`.

Note on debug statements and performance

For debugging, several asserts were added, and these asserts slow down the code by a factor between 2x and 10x. To turn these asserts off, the code can be run with the python option `-OO`, which turns off the `__debug__` built-in variable.

Implementation Details

Selection and population

Binary tournament selection was implemented.

Two methods for population replacement were used.

1. Complete replacement
2. A variant of Elitist replacement (This can be specified by specifying the ratio of parents with the option `-epr`)

Most of the detailed comparison was done using complete replacement. However, when the initial population was arrived at using insertion heuristic, a complete replacement didn't perform very well.

Elitist replacement performed much better when insertion heuristic was used. However, due to time constraints, only basic comparisons were made with Elitist replacement. Future work would involve varying the other parameters with Elitist replacement.

A heap was used to select k parents with the lowest distances. Since heapify runs in linear time, the overall complexity of this operation is $O(n + k \log(n))$.

Crossover

Uniform Crossover

Two variants of uniform crossover were implemented:

1. Random selection of number of elements to splice, with a low value set to 5
2. Number of elements to splice is between 50 and 75% of the number of genes

The second variant was coded later as an experiment when the first variant did not perform very well when using an insertion heuristic. Most of the comparisons, however, were run with the first variant. Further work would include running comparisons with the second variant as well.

Furthermore, a frozen set was used to avoid an $O(n^2)$ cost, to keep the cost down to $O(n)$.

Order 1 Crossover

Two variants of Order-1 crossover were implemented:

1. That selects a random number of fixed genes
2. That selects a random number of fixed genes with a minimum of 50% of total genes, and a maximum of 75% of total genes.

The second variant was coded later as an experiment when the first variant did not perform very well when using an insertion heuristic. Most of the comparisons, however, were run with the first variant. Further work would include running comparisons with the second variant as well.

A third variant of order-1 crossover was also implemented, in which two parents are crossed-over to form two children, with the same fixed-indices. However results are not included for the third variation.

Mutation

Scramble and inversion mutation were implemented

Creating the Mating Pool

Two versions were implemented:

1. A version that copies all individuals from the population to form the mating pool
2. A version that selects individuals based on a probability distribution that varies with the inverse of the total distance. However, all comparisons were performed with the former variant, because binary tournament selection also applies a probability and chooses better solutions with a higher probability, and if the second version was used, it would result in applying the probability twice. This version has not been included in the results.

Results

In the first are the results of the basic runs. In this section, I only discuss each configuration individually, and the characteristics of each configuration between each iteration.

In the second section, I will present a comparison of the various runs in the same place.

In the third section, I will present the effect of varying the different parameters (mutation rate and population size).

In the last section, I will present the results from the additional experiments with elitism and variants of different crossovers. This last section has been performed with fewer number of runs due to time constraints.

Basic Runs

Configuration 1

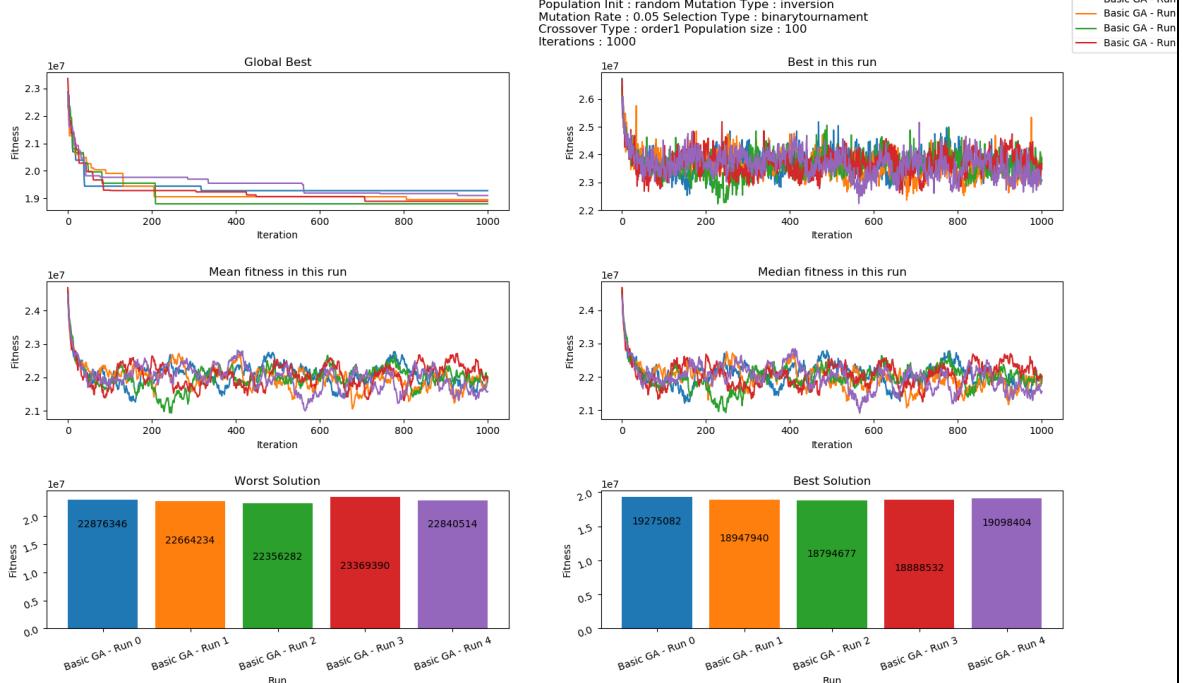
Initial Population: Random

Crossover: Order-1

Mutation: Inverse

Selection: Binary Tournament

File: inst-0.tsp, 184 Cities

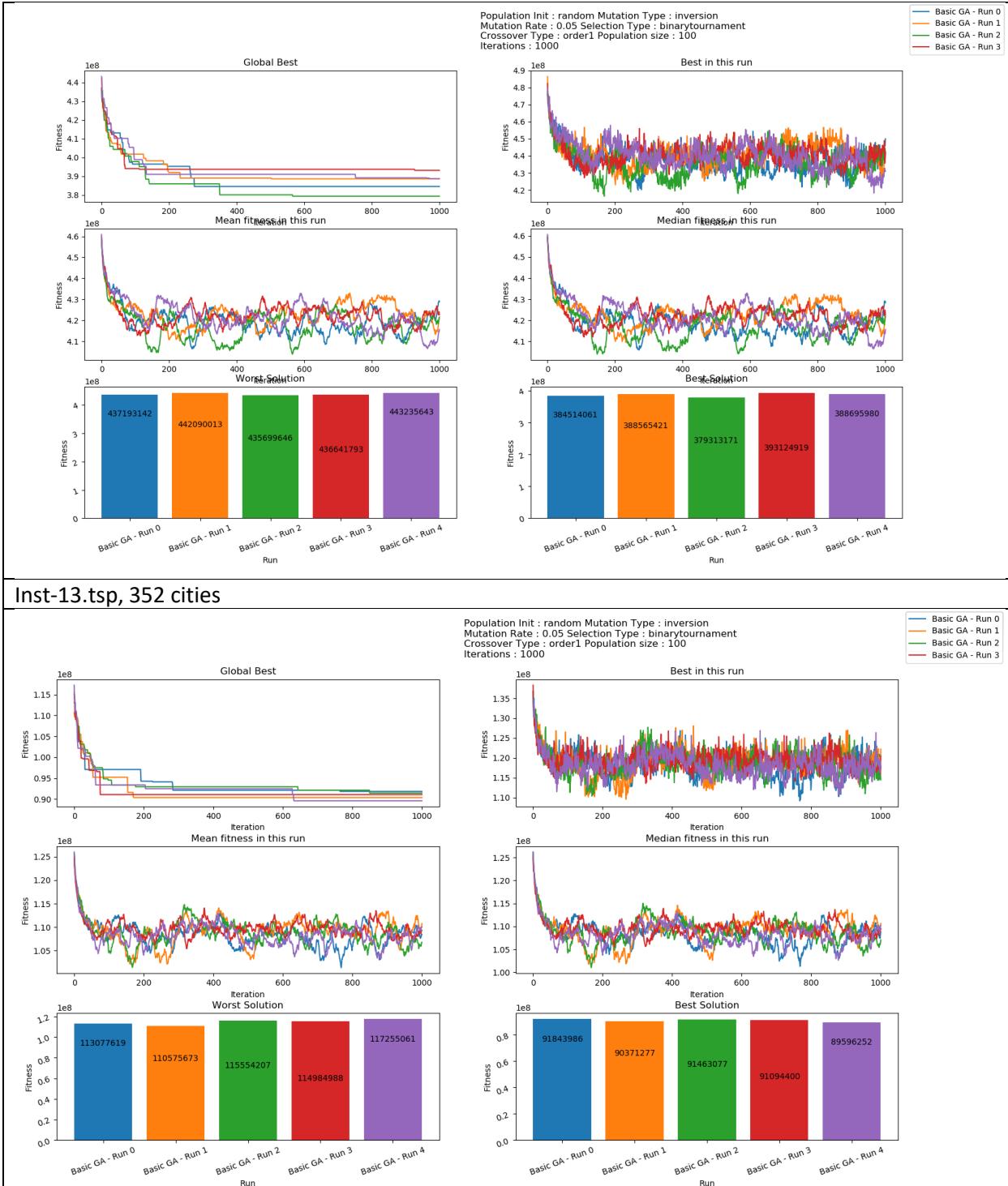


The legend is incomplete, there are 5 runs, but the legend only shows four.

The above chart shows the fitness in every run, and the global best fitness. The shows a sharp decrease till it flattens out after around 100 iterations. This is consistent with theory as GA's tend to improve the solution very quickly in the beginning, till flattening out, or becoming very slow to find any improvements.

Similar trends were also observed with files inst-5 and inst 13, but the solution flattened after a different number of iterations.

Inst-5.tsp, 819 cities



Configuration 2

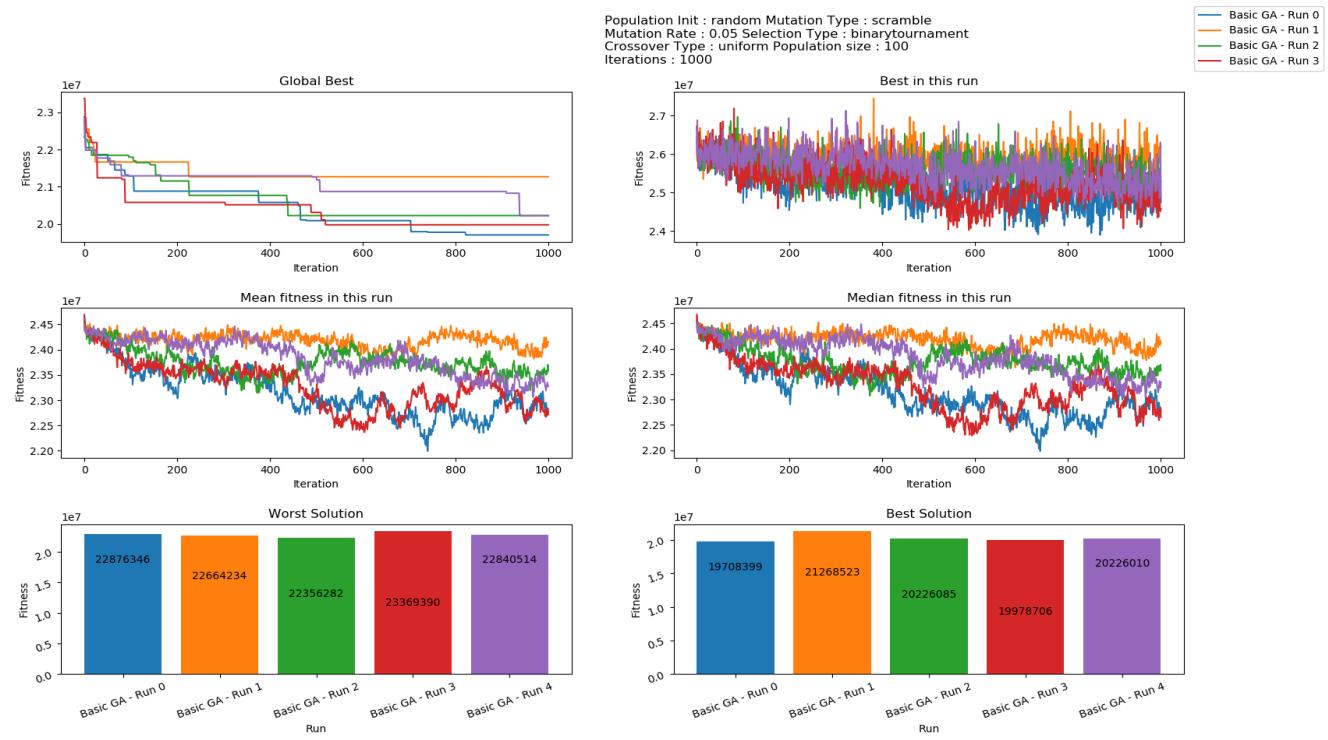
Scramble Mutation, Binary Tournament Selection, Random Initial Population Selection, Uniform Crossover.

Again, as before, a sharp fall in fitness was seen in the beginning till the graph levelled off. This is consistent with theory.

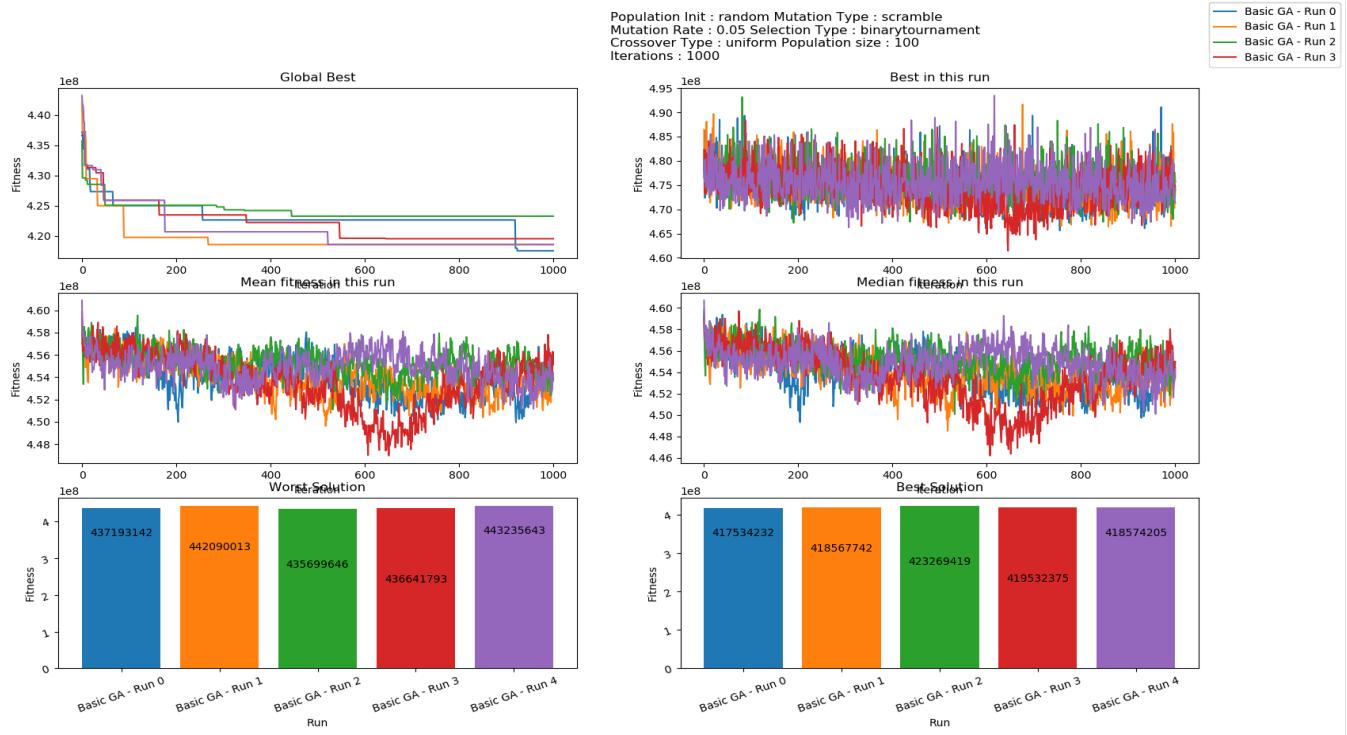
However, at the phase of the graph that levelled off, the median fitness in each run showed more variability than configuration 1. This effect could be due to the scramble mutation, which tends to produce more variance in the paths.

The graph for inst-13.tsp was somewhat different than the other two. In the beginning the median variance in each run was quite flat, after which it dropped and kept dropping consistently.

Inst-0.tsp, 184 cities

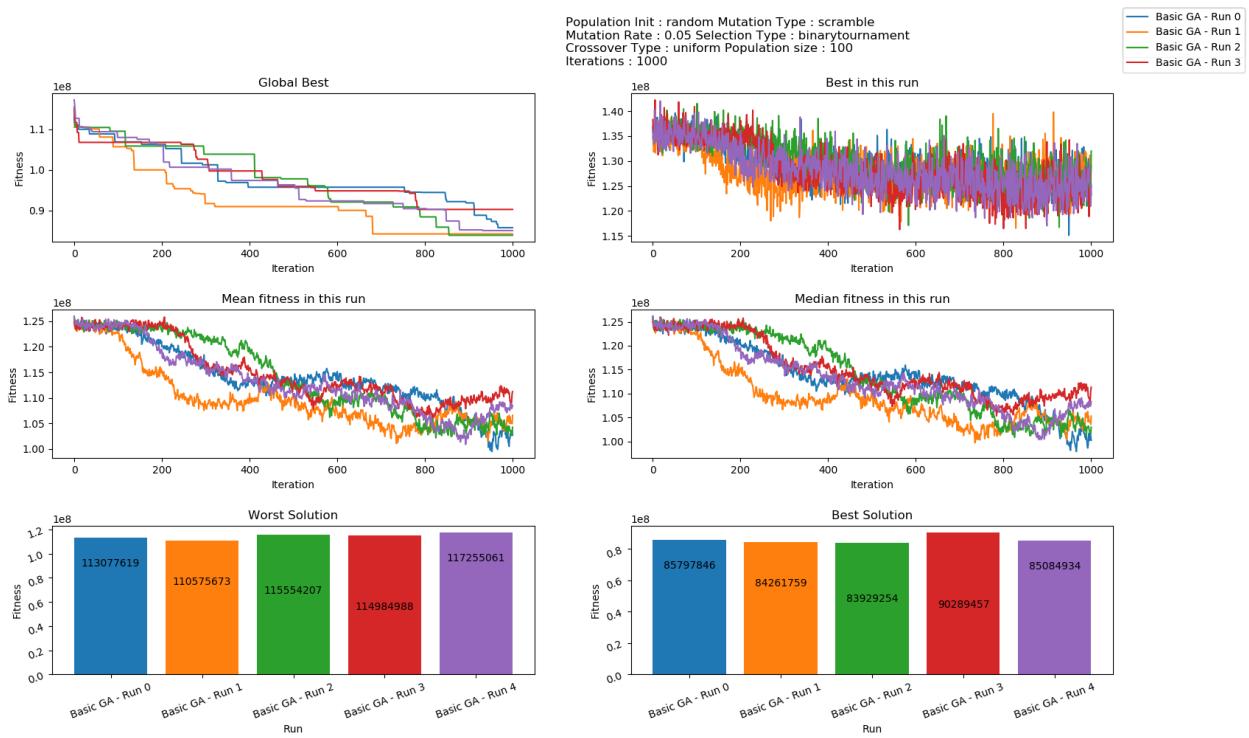


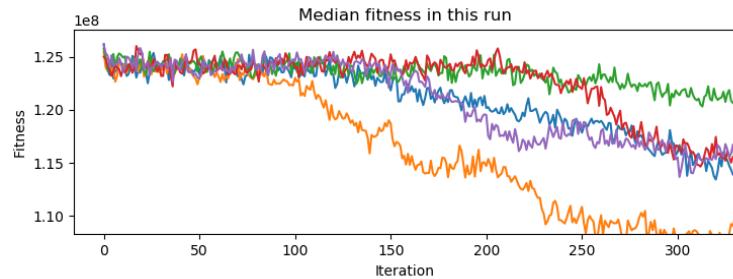
Inst-5.tsp, 819 cities



Inst-13.tsp 352 cities

Again, as noted earlier, this is somewhat different as the GA was initially unable to reduce the median fitness for the first 100 iterations, after which it kept dropping.



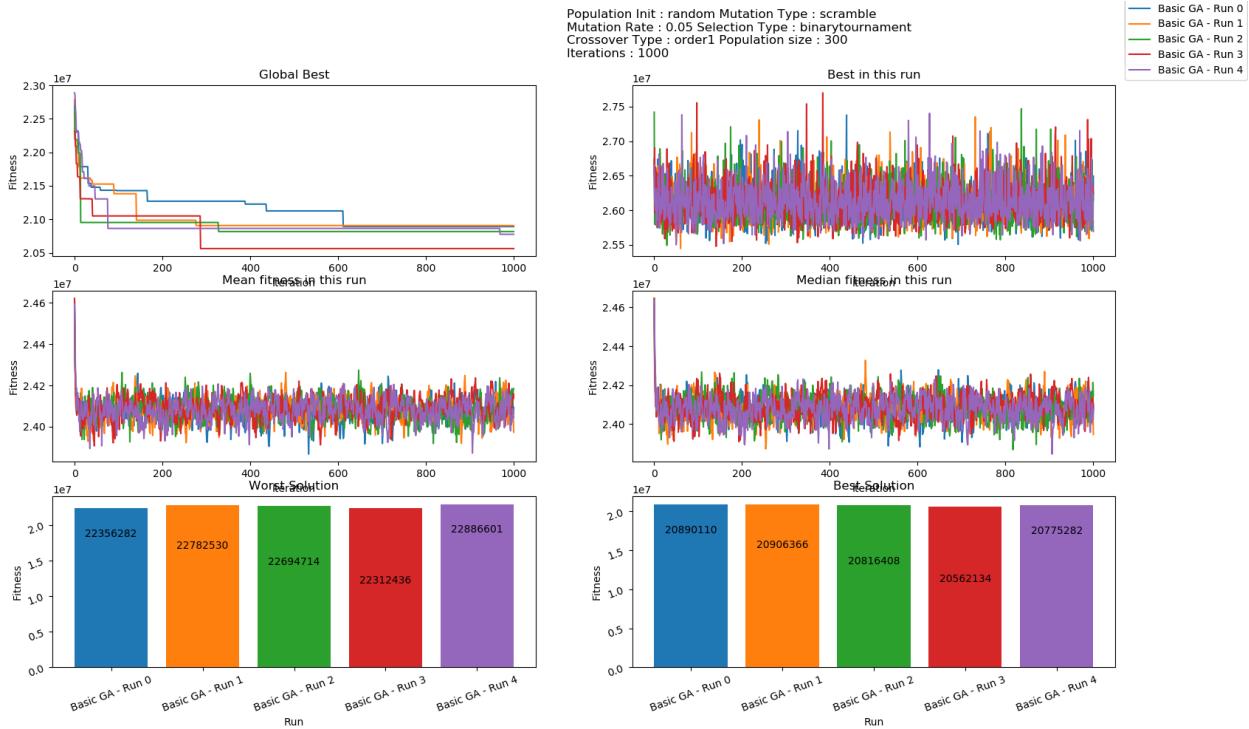


Configuration 3

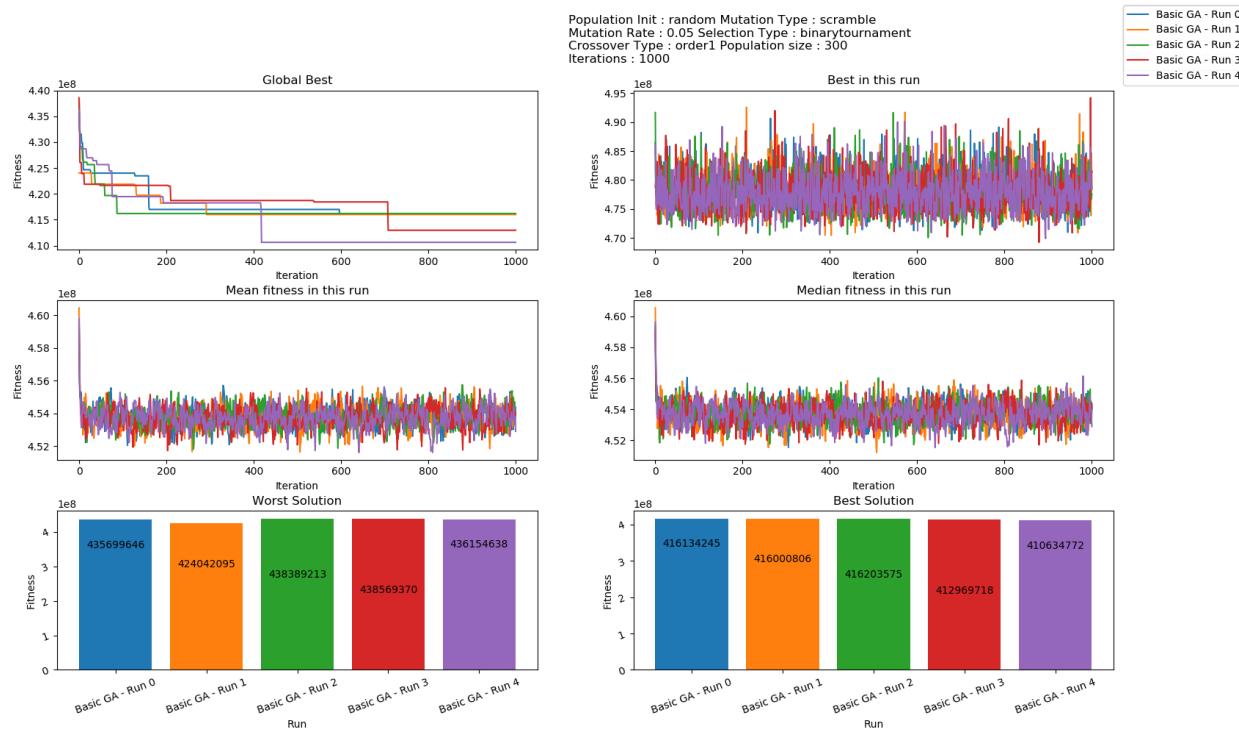
Random Initial Selection, Order-1 Crossover, Scramble Mutation, Binary Tournament Selection

The results in this run was also similar to configuration 2 in terms of trends. However the best fitness in every run showed much more variance than the previous configurations.

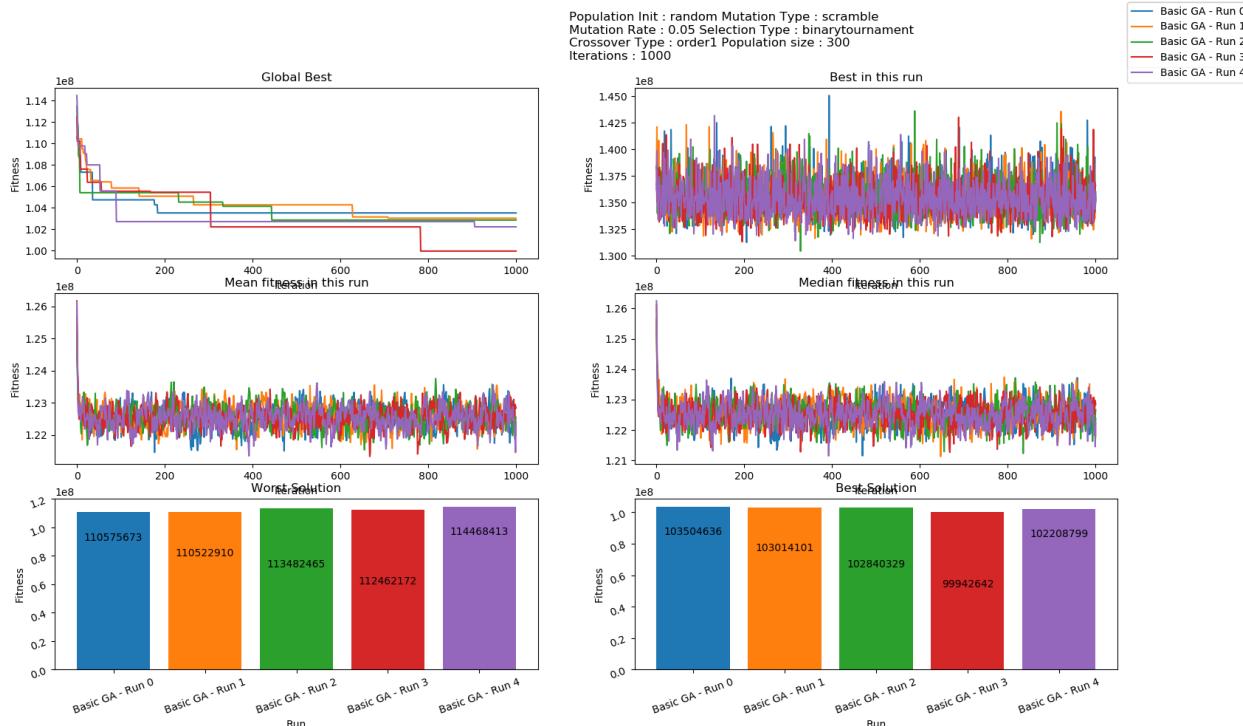
Inst-0.tsp, 184 cities



Inst-5.tsp



Inst-13.tsp

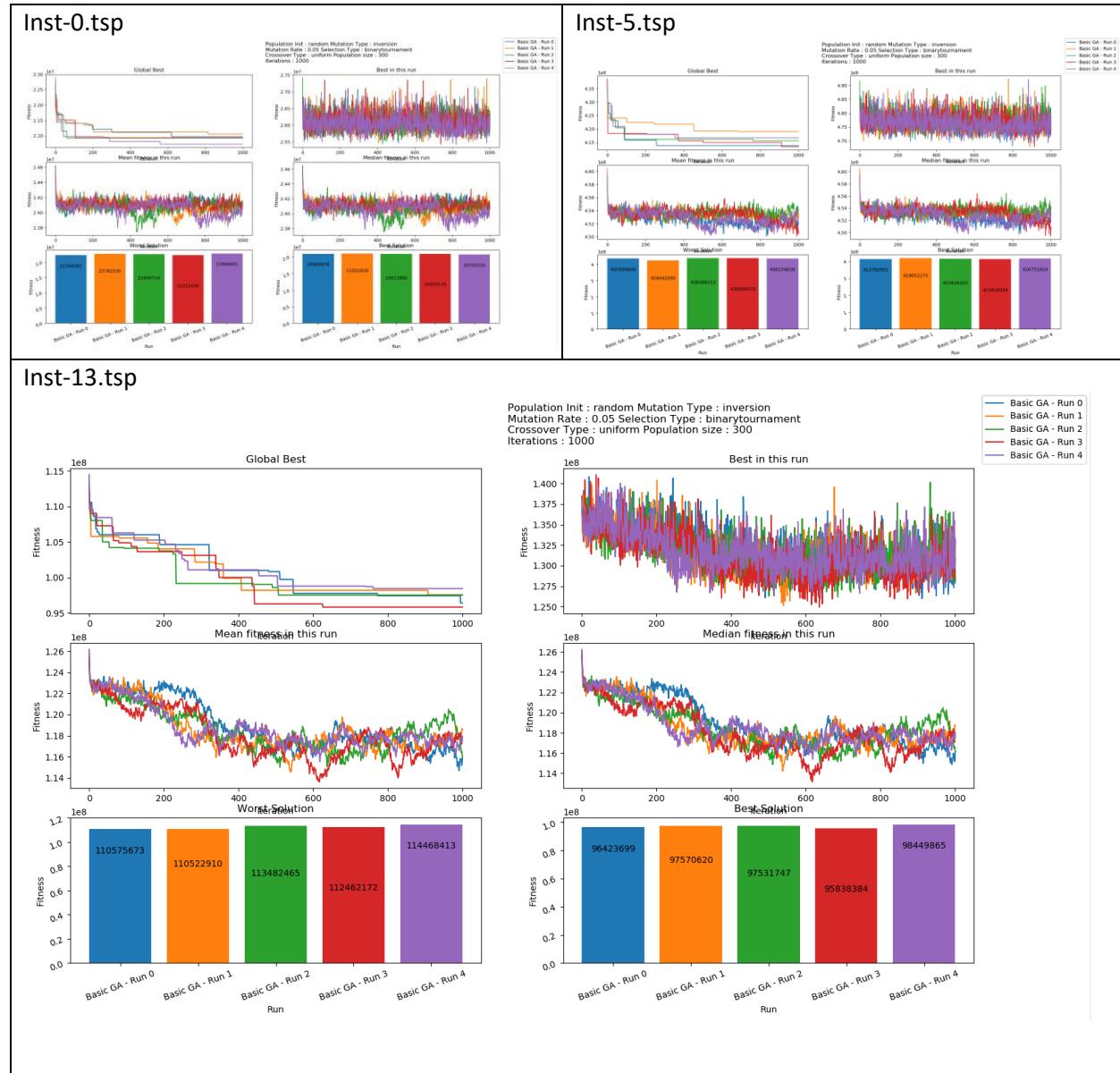


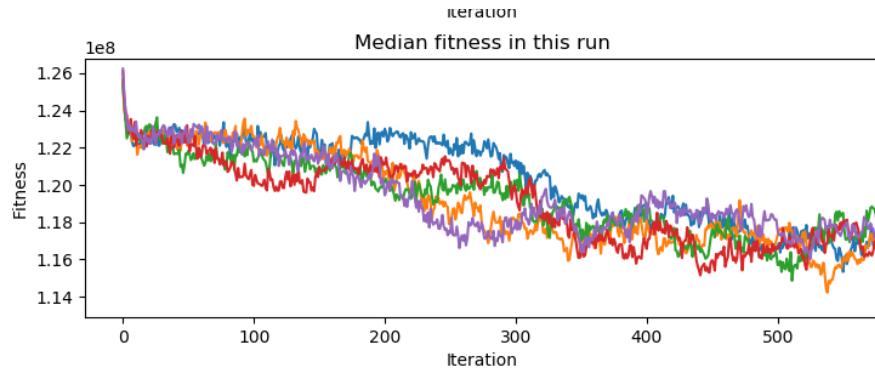
Configuration 4

Random Population Selection, Inversion Mutation, Uniform Crossover, Binary Tournament

Again, this configuration had similar characteristics to configuration 4. The graphs are presented below, but there is nothing noteworthy here, hence the graphs have been made smaller in size.

Isn't-13.tsp showed a trend similar to configuration2. Initially the GA was not able to find good solutions for some time, and then it kept dropping continuously and consistently.





The above graph shows a sharp drop in the first few iterations, then a flattening till iteration 100, then a gradual drop between iteration 150 and iteration 350, and then a flattening.

Configuration 5

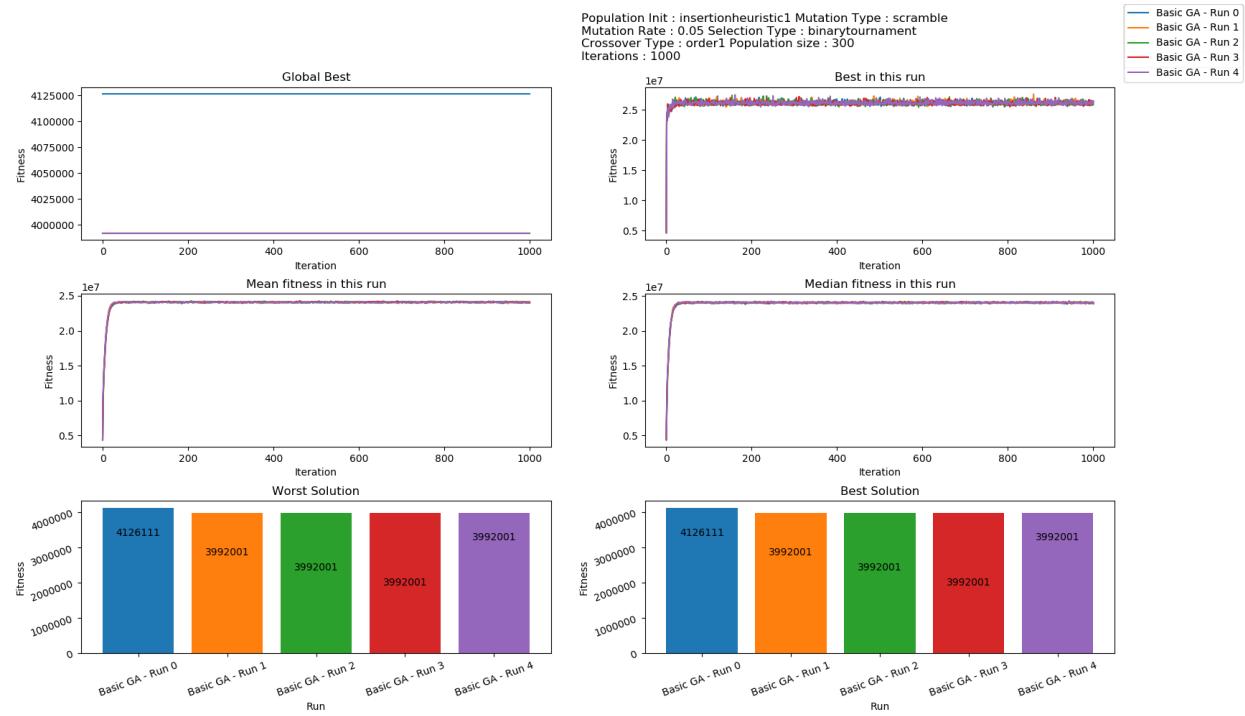
Insertion Heuristic population initialization, scramble mutation, order-1 crossover, binary tournament selection.

In this configuration, the initial population selection produces a solution that is an order of magnitude better than random population selection.

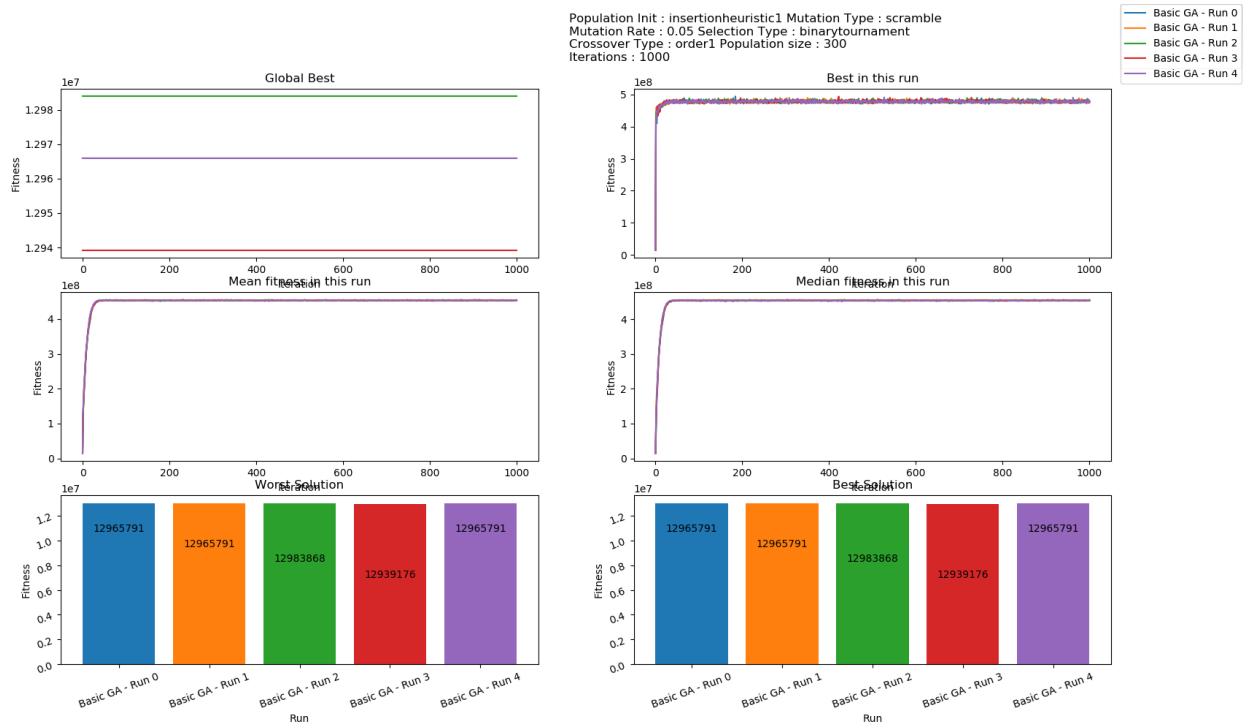
The GA, however, is not able to improve upon this solution, and it consistently found worse solutions. The fitness rose sharply, till stabilizing at a value about 5x the initial solution.

We will see later that some of this was mitigated using elitist selection.

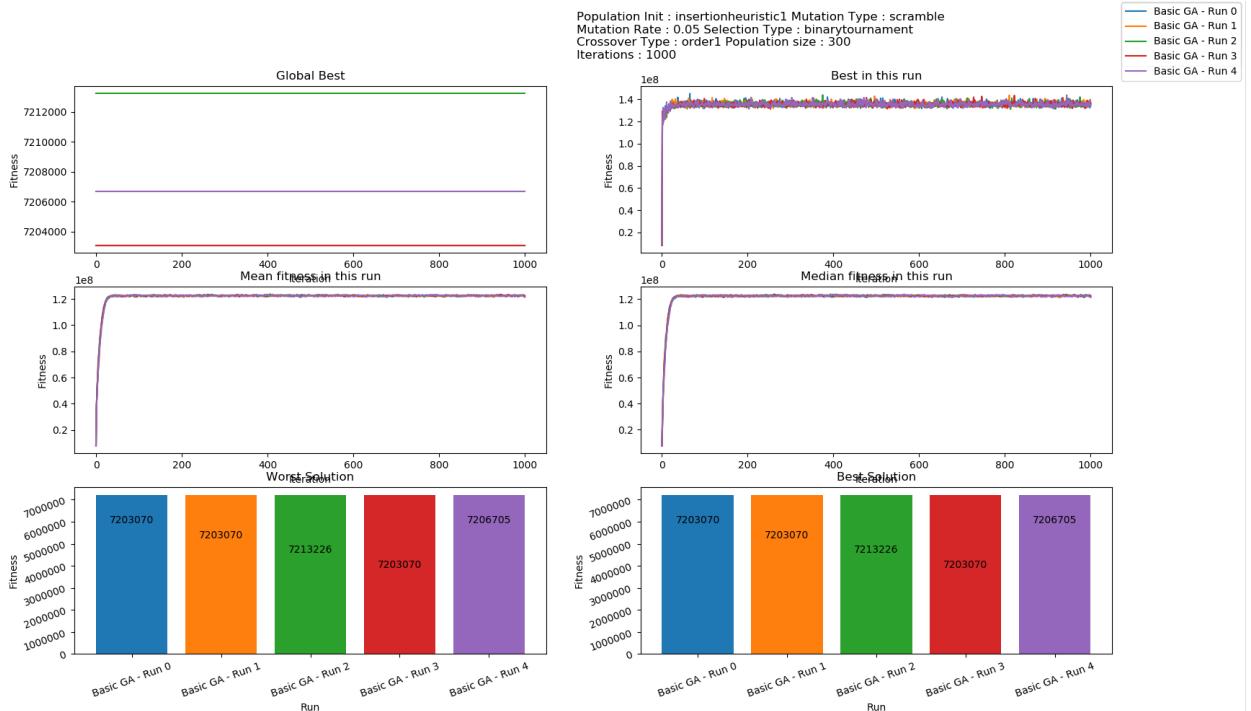
Inst-0.tsp



Inst-5.tsp

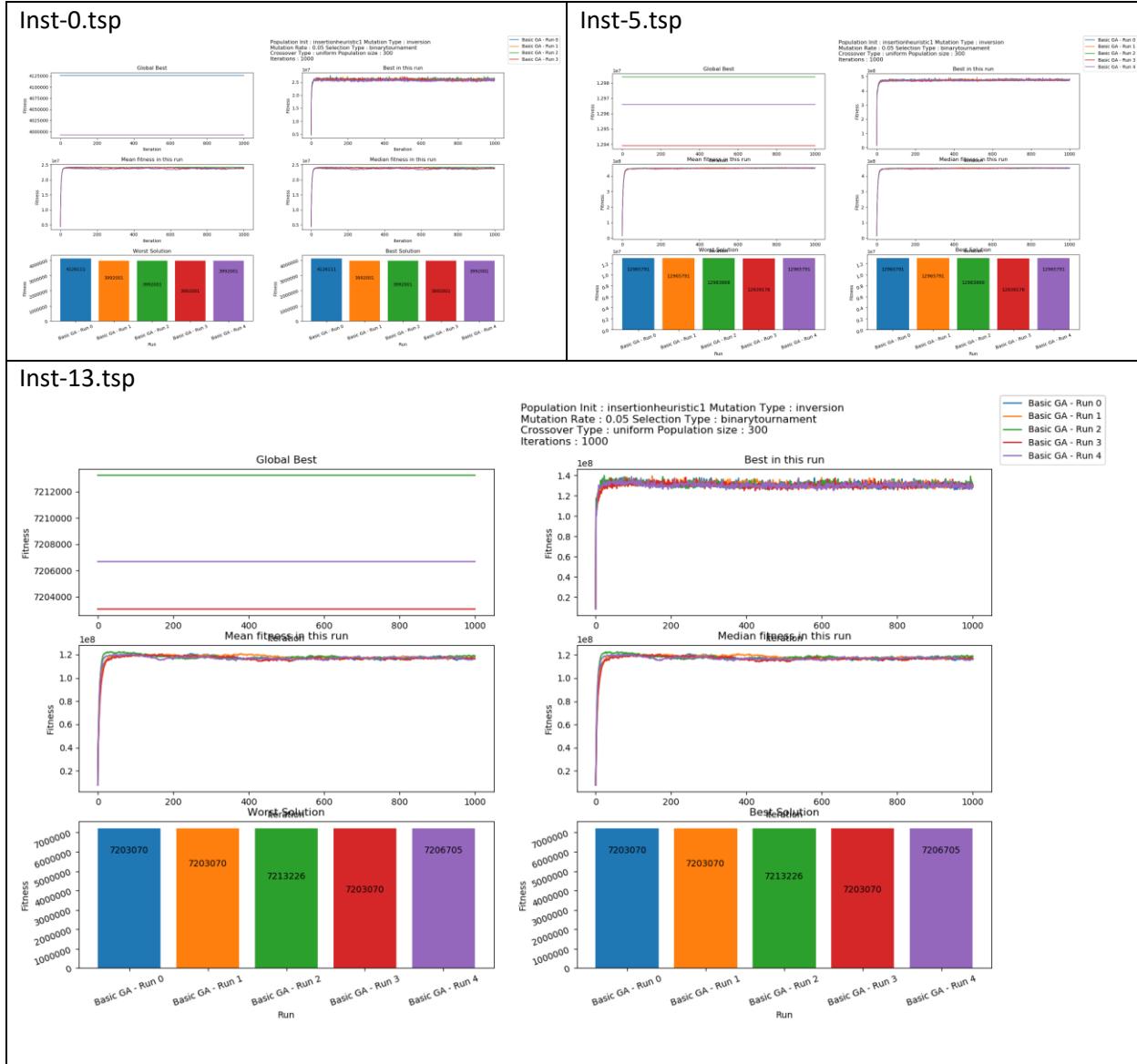


Inst-13.tsp



Configuration 6

Again, the results were similar to configuration 5. All the observations in configuration 5 are also valid here.



Comparison between Configurations

Speed: Configurations 2, 4 and 6 are much faster than 1, 3 and 5.

This would indicate that order-1 cross-over is faster than uniform crossover on an average.

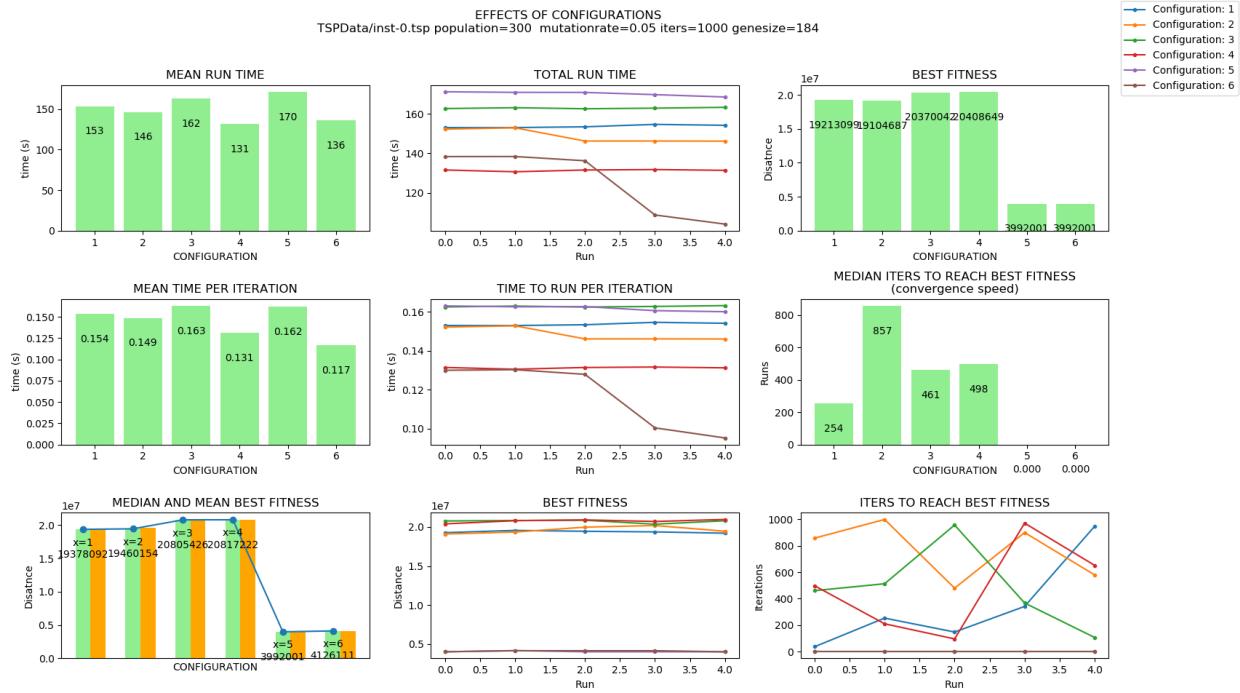
Configurations 5 and 6 produced results that were an order of magnitude better than the other configurations. But that is because of the initial population selection heuristic. The GA didn't perform much better in both of these configurations, and in the last chart, we show that all of the GA's ended up roughly in the same place after 1000 iterations. The difference in optimality was solely due to a better initial population selection.

Configurations 1 and 2 performed slightly better than configurations 3 and 4 in terms of giving the better solution.

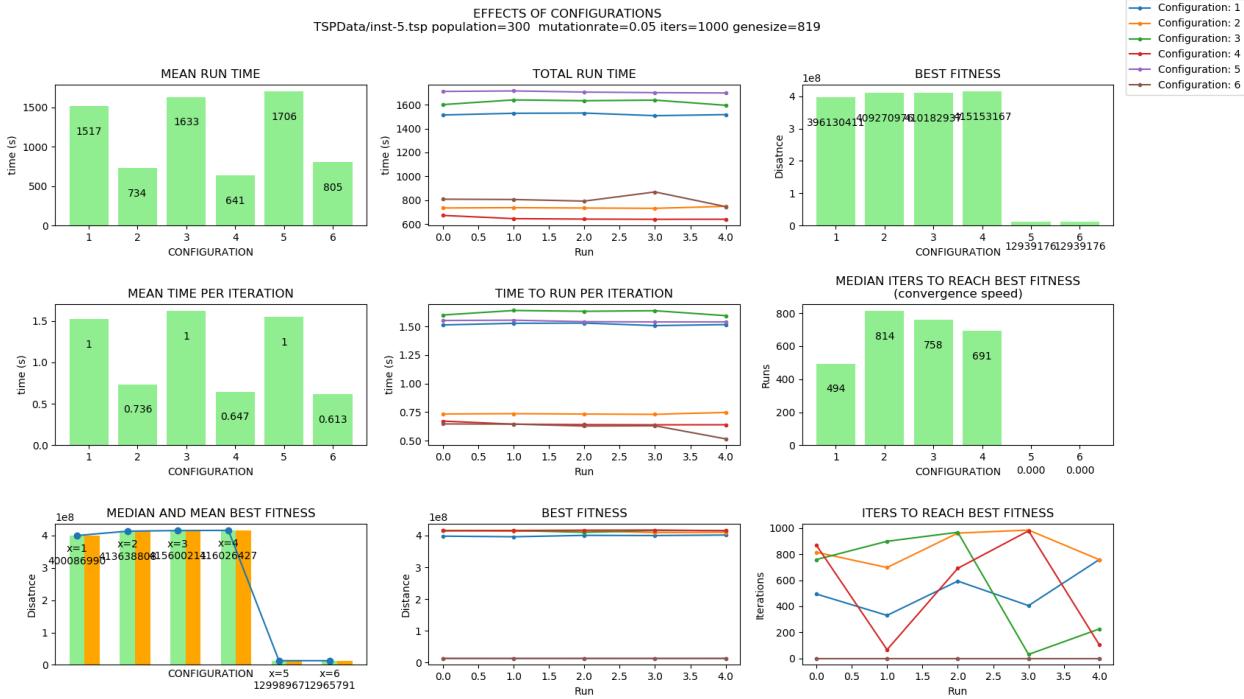
In terms of the number of iterations needed to converge to the best solution the GA could find, configuration 2 performed the worst. Configuration 1 performed better than configuration 3, which performed better than configuration 4.

This could be down to the fact that both uniform crossover and scramble mutation produce more variation on each application, and with such variation, the GA takes time to converge. The combination of the two takes the greatest time to converge.

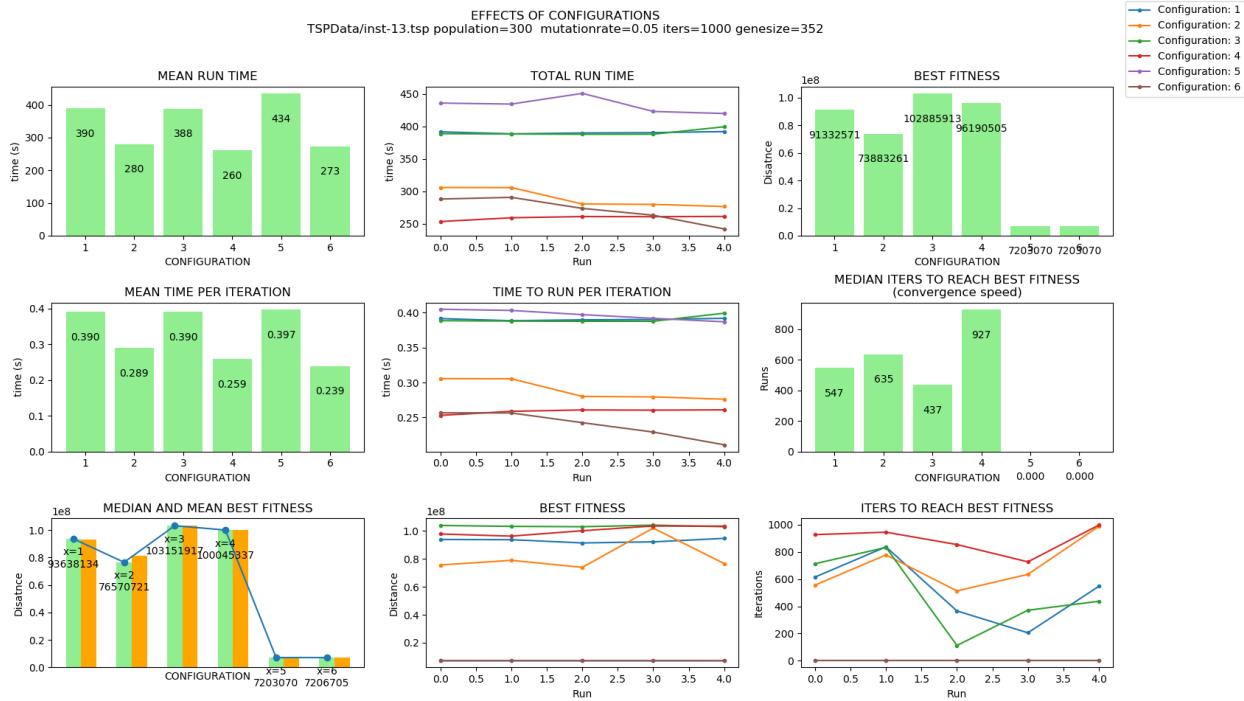
Inst-0.tsp



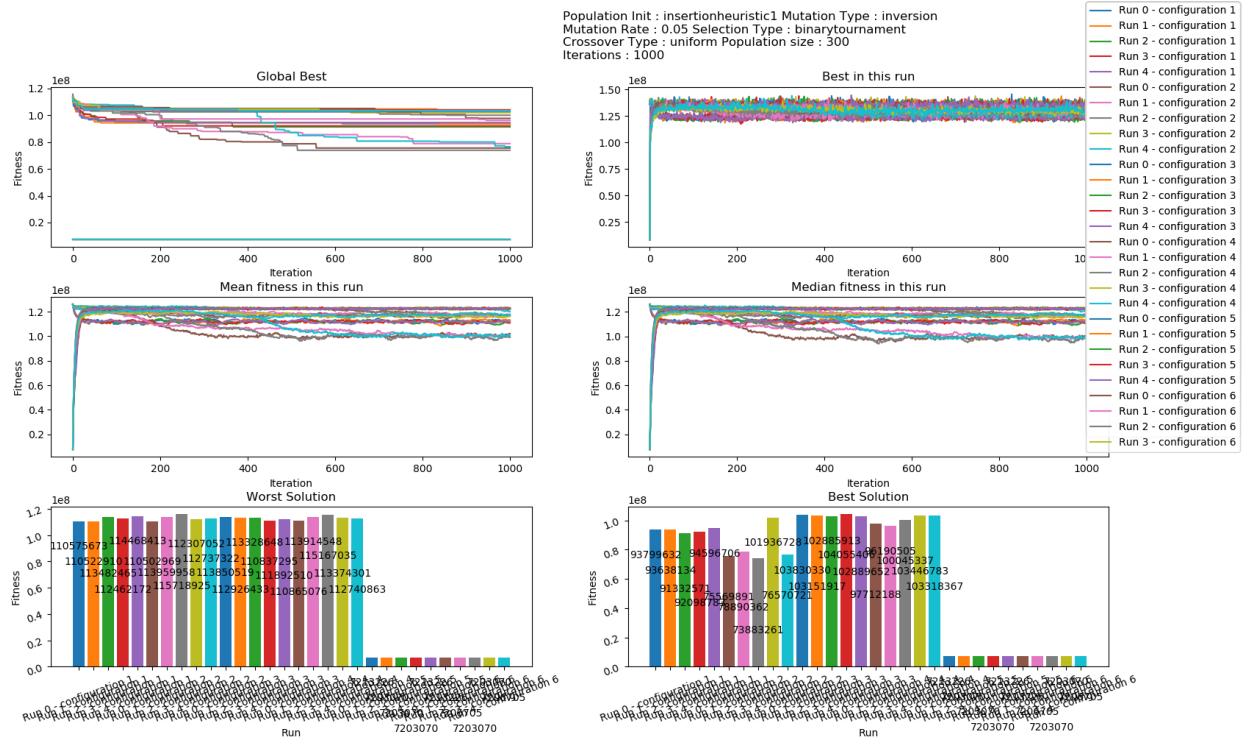
Inst-5.tsp



Inst-13.tsp



The following chart shows that no matter where the initial population started from, the GA algorithm made a bad solution better, and a good solution worse, and roughly converged to the same location.



Varying Parameters

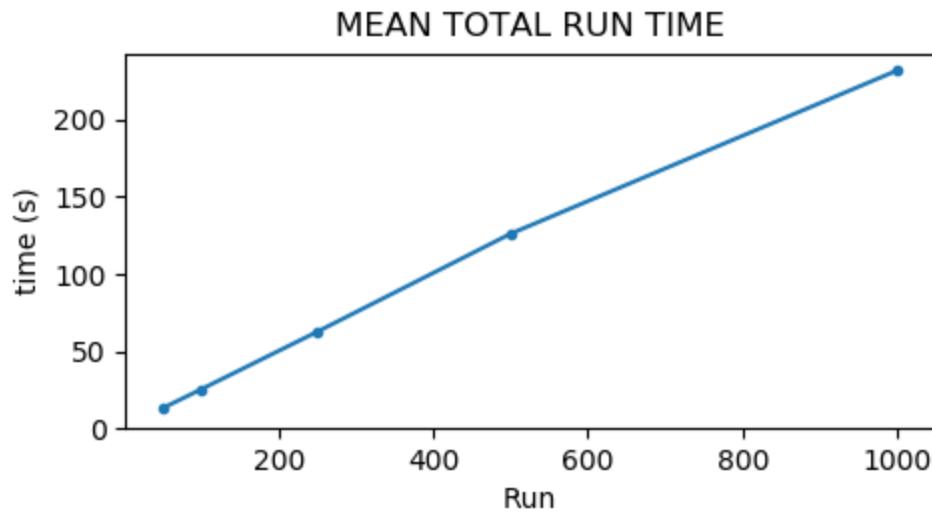
The effect of varying different parameters is discussed here.

Charts were plotted for inst-0.tsp, inst-5.tsp and inst-13.tsp, however, for the benefit of brevity, only the charts for inst-0.tsp and inst-5.tsp are presented here. The remaining charts are saved as pickle files, and can be viewed using the script plotagain.py.

Varying Population Size

Total Run Time

Throughout this experiment, the mean total run time showed a linear relationship with the population size. This trend was observed for all configurations, but is only presented below for config-1 and inst-0.tsp.



Convergence Speed (time to reach best fitness)

The general trend that was noted was that the higher the population, the more time it took to converge to a solution. The trend however was not strong, and there were many graphs which showed a different behavior.

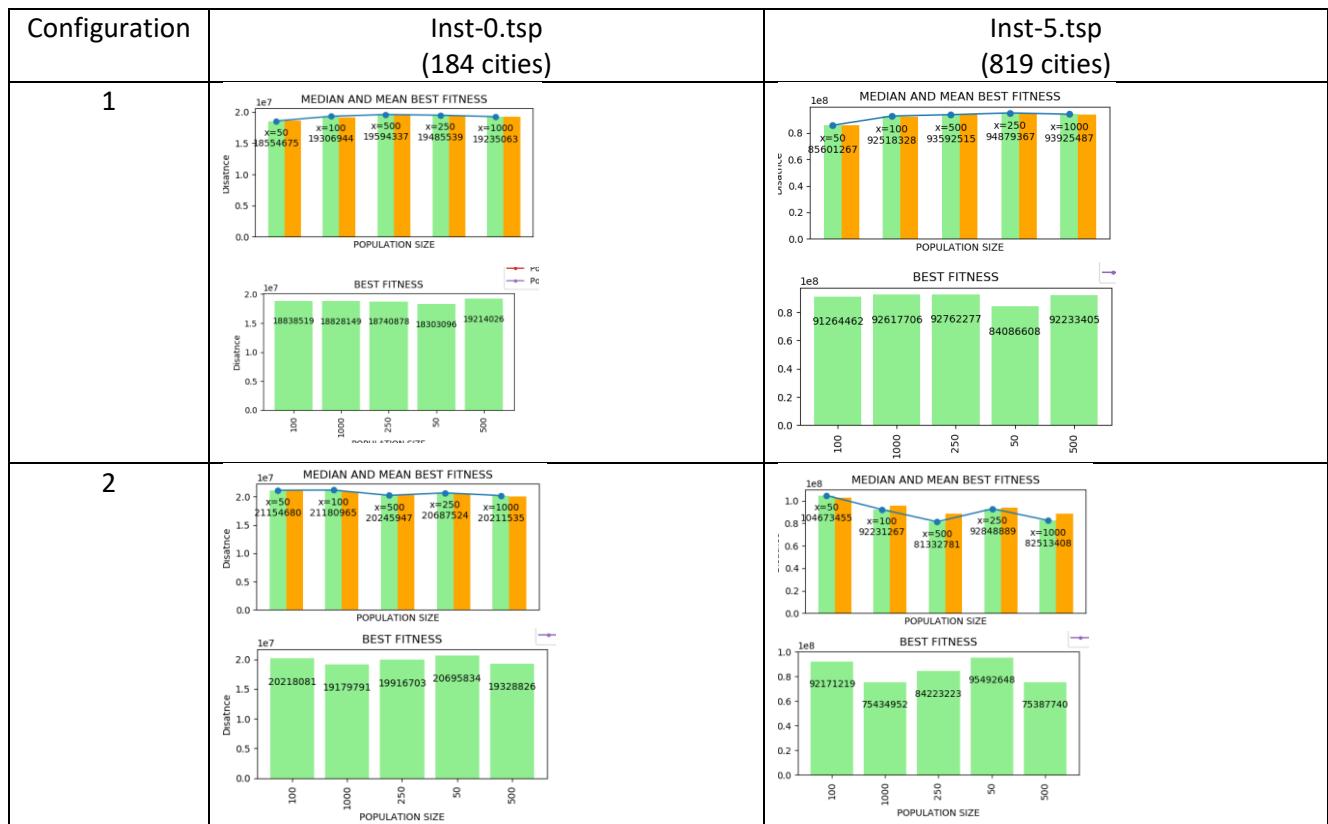
Configuration	Inst-0.tsp (184 cities)	Inst-5.tsp (819 cities)
1	<p>POPULATION SIZE MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>	<p>POPULATION SIZE MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>
2	<p>POPULATION SIZE MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>	<p>POPULATION SIZE MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>
3	<p>POPULATION SIZE MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>	<p>MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)</p>

4	<p>POPULATION SIZE MEDIAN ITERERS TO REACH BEST FITNESS (convergence speed)</p>	<p>MEDIAN ITERERS TO REACH BEST FITNESS (convergence speed)</p>
5	n/a since the initial population contains the best	n/a since the initial population contains the best
6	n/a since the initial population contains the best	n/a since the initial population contains the best

Fitness Achieved

The mean and median best fitness across 5 runs are presented, as the best fitness of all the runs.

The trend that was noticed was that the higher the population, the better solutions it tended to find.





Varying Mutation Rate

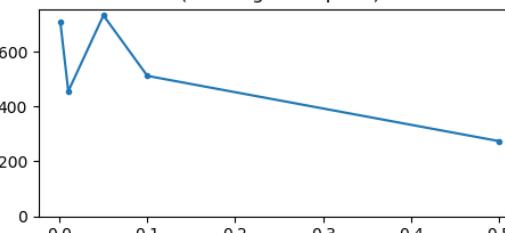
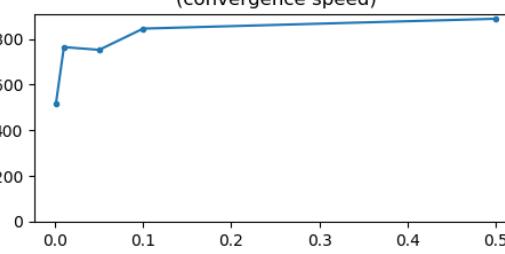
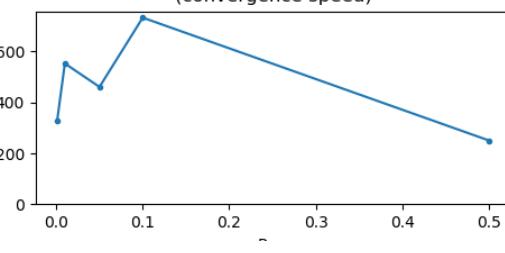
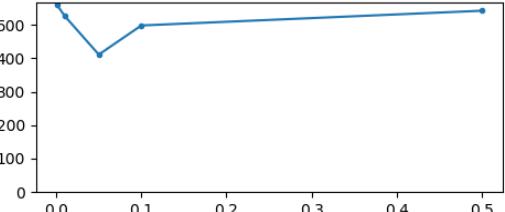
The same graphs were plotted for varying mutation rate as well. However, for the benefit of brevity, only the charts for inst-0.tsp are presented here. The remaining charts are saved as a pickle

Total Run Time

The total run time remained fairly constant as the mutation rate was increased.

Convergence Speed

No consistent trend was noticed in convergence speed as mutation rate was changed.

Configuration	Inst-0.tsp (184 cities)												
1	MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)  <table border="1"><caption>Data for Configuration 1</caption><thead><tr><th>Mutation Rate</th><th>Median Iterations</th></tr></thead><tbody><tr><td>0.00</td><td>650</td></tr><tr><td>0.02</td><td>450</td></tr><tr><td>0.05</td><td>700</td></tr><tr><td>0.10</td><td>500</td></tr><tr><td>0.50</td><td>300</td></tr></tbody></table>	Mutation Rate	Median Iterations	0.00	650	0.02	450	0.05	700	0.10	500	0.50	300
Mutation Rate	Median Iterations												
0.00	650												
0.02	450												
0.05	700												
0.10	500												
0.50	300												
2	MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)  <table border="1"><caption>Data for Configuration 2</caption><thead><tr><th>Mutation Rate</th><th>Median Iterations</th></tr></thead><tbody><tr><td>0.00</td><td>500</td></tr><tr><td>0.02</td><td>750</td></tr><tr><td>0.05</td><td>750</td></tr><tr><td>0.10</td><td>800</td></tr><tr><td>0.50</td><td>850</td></tr></tbody></table>	Mutation Rate	Median Iterations	0.00	500	0.02	750	0.05	750	0.10	800	0.50	850
Mutation Rate	Median Iterations												
0.00	500												
0.02	750												
0.05	750												
0.10	800												
0.50	850												
3	MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)  <table border="1"><caption>Data for Configuration 3</caption><thead><tr><th>Mutation Rate</th><th>Median Iterations</th></tr></thead><tbody><tr><td>0.00</td><td>350</td></tr><tr><td>0.02</td><td>550</td></tr><tr><td>0.05</td><td>450</td></tr><tr><td>0.10</td><td>700</td></tr><tr><td>0.50</td><td>300</td></tr></tbody></table>	Mutation Rate	Median Iterations	0.00	350	0.02	550	0.05	450	0.10	700	0.50	300
Mutation Rate	Median Iterations												
0.00	350												
0.02	550												
0.05	450												
0.10	700												
0.50	300												
4	MEDIAN ITERS TO REACH BEST FITNESS (convergence speed)  <table border="1"><caption>Data for Configuration 4</caption><thead><tr><th>Mutation Rate</th><th>Median Iterations</th></tr></thead><tbody><tr><td>0.00</td><td>550</td></tr><tr><td>0.02</td><td>450</td></tr><tr><td>0.05</td><td>400</td></tr><tr><td>0.10</td><td>500</td></tr><tr><td>0.50</td><td>550</td></tr></tbody></table>	Mutation Rate	Median Iterations	0.00	550	0.02	450	0.05	400	0.10	500	0.50	550
Mutation Rate	Median Iterations												
0.00	550												
0.02	450												
0.05	400												
0.10	500												
0.50	550												
5	n/a since the first population generated contains the best individual												
6	n/a since the first population generated contains the best individual												

Solution Optimality

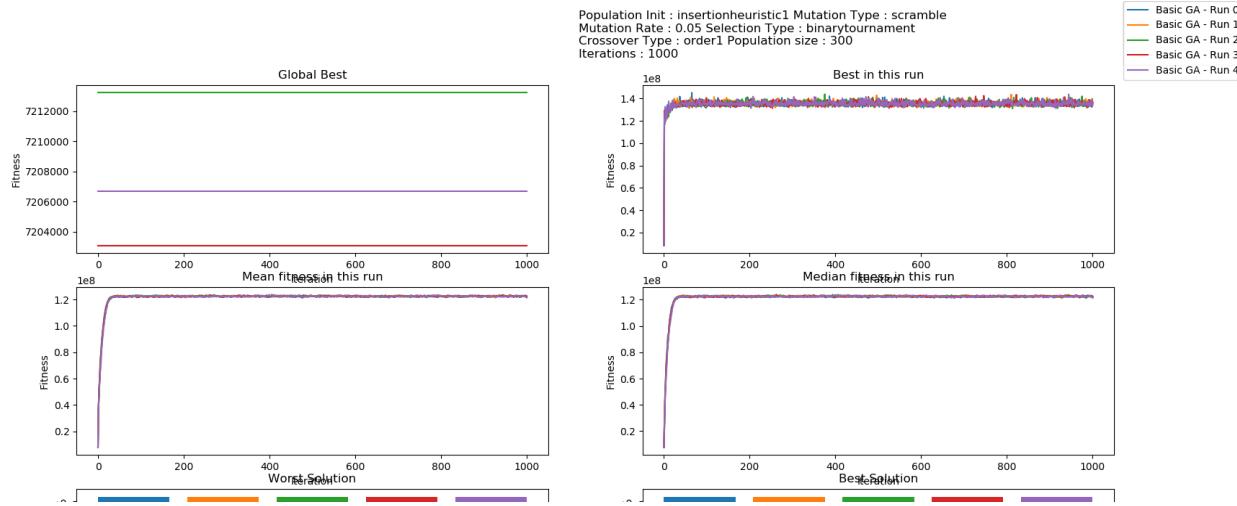
No strong trend was noticed between the quality of solutions and the mutation rate. However, there is a weak sweet spot between the mutation rates of 0.01 and 0.05.

Configuration	Inst-0.tsp (184 cities)					
1						
2						
3						
4						
5	Constant at 3992001 since the first population contains the best individual.					
6	Constant at 3992001 since the first population contains the best individual.					

Elitism

A variant of elitism was implemented, where a fixed ratio of the best solutions from the previous generation were added to the population. This approach was able to mitigate the problems with configuration 5 and 6 where the GA was not able to find any solution better than the initial solution, and the median solution actually became worse with every run till stabilizing at a point.

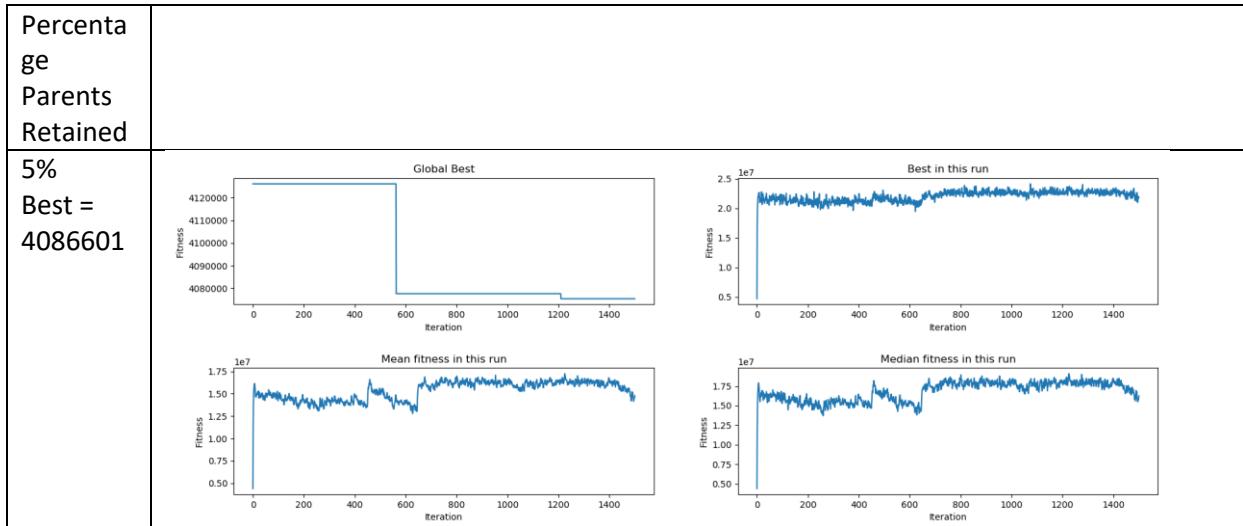
For ease of viewing, the graph from config 5 is presented below, a similar graph was seen for config 6.

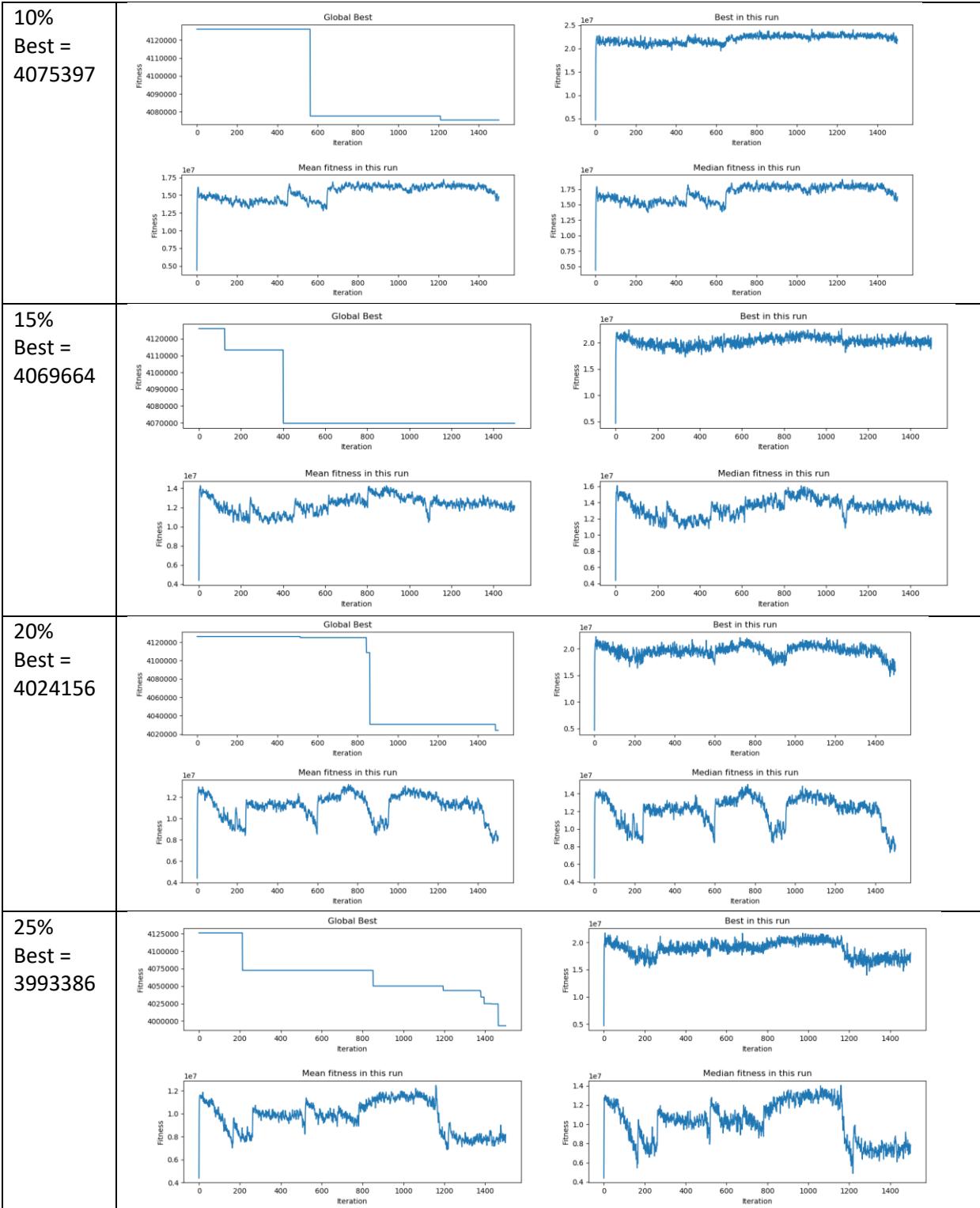


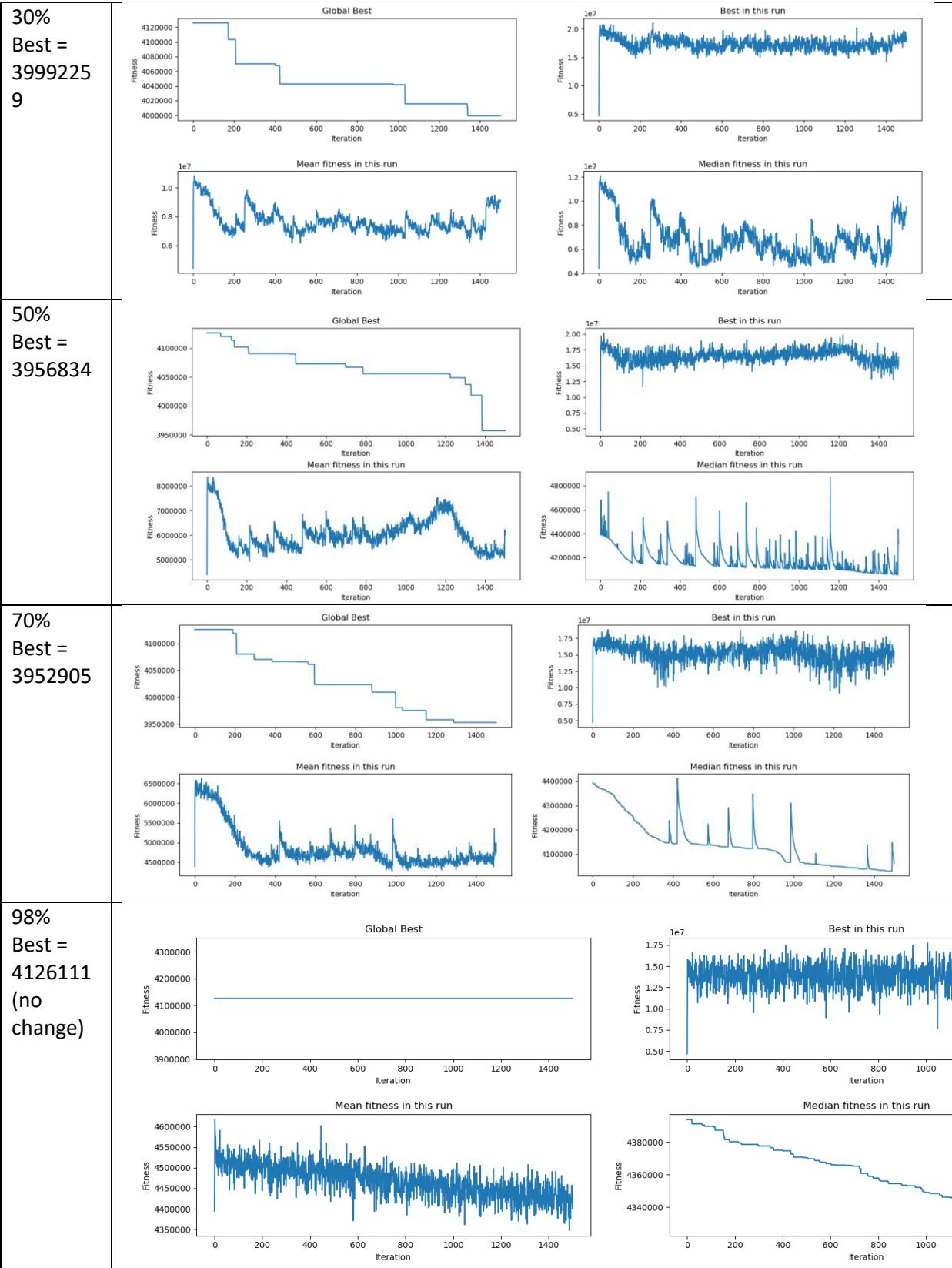
The following was tried with configuration 6 and inst-0.tsp, running each configuration once every invocation of the script.

The observations were as follows:

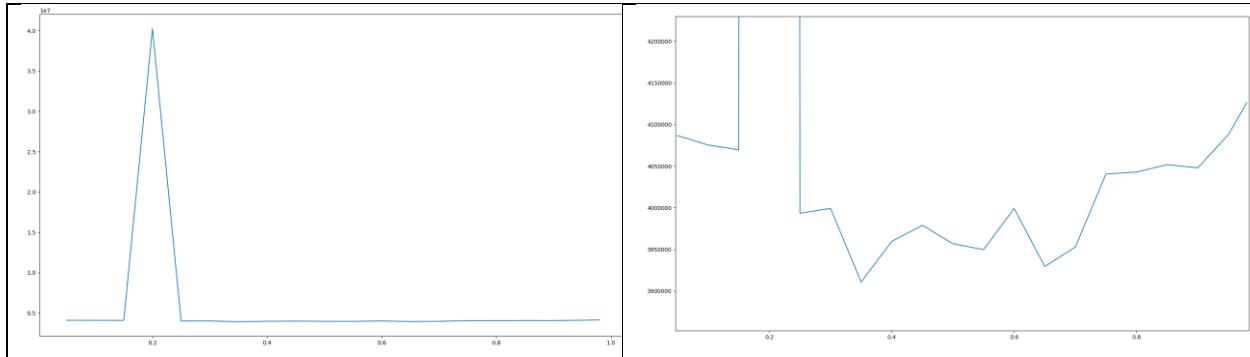
1. As the percentage of parents was increased, the value of the distance first improved till a point, and then started becoming worse
2. The mean and median fitness in every run became less variable as the percentage was increased, as expected, as there is no change in the parents, and all change comes from the children. So a higher number of parents means less variation.



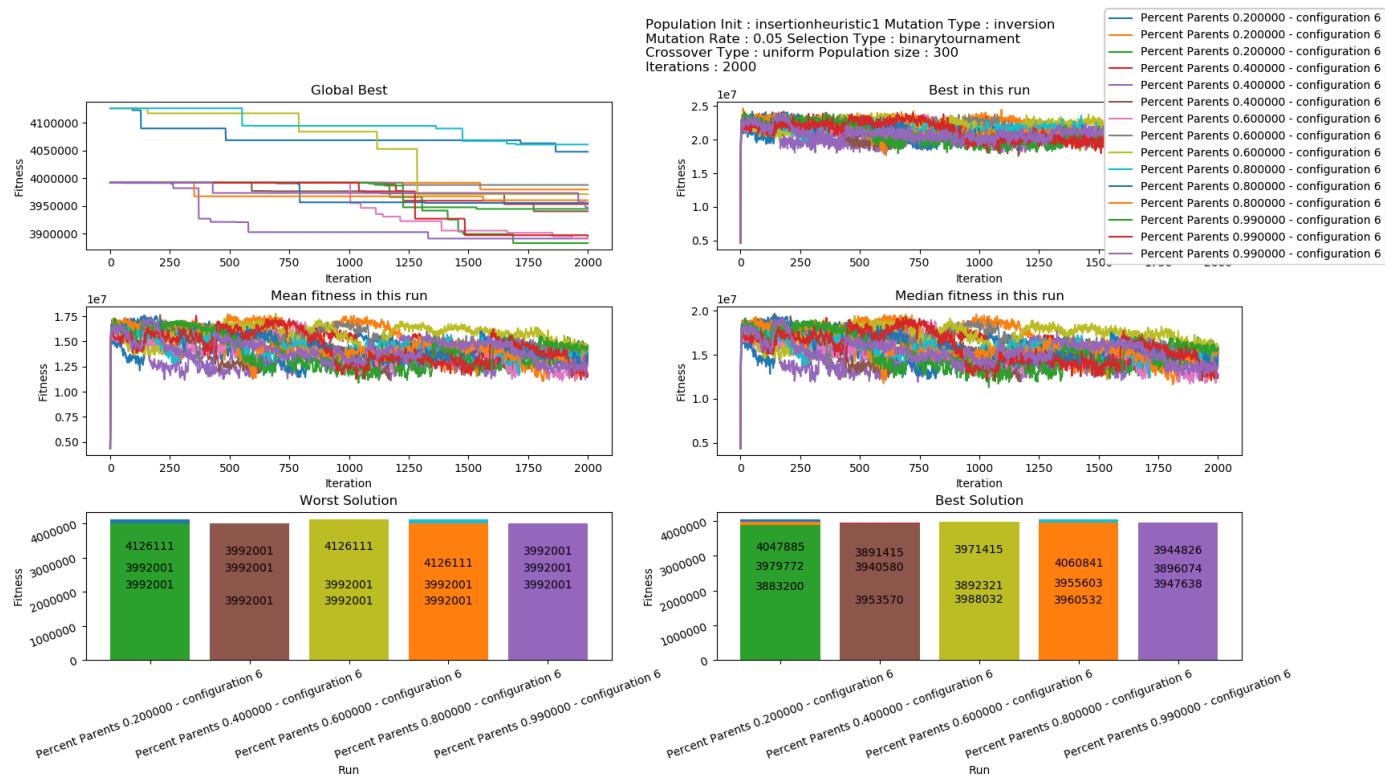




The above data plotted as the best fitness against the percentage of parents retained shows the trend. The chart on the right is a zoomed in view of the chart on the left:



In the following graph, 3 runs were run in parallel (using -vemr option), with config 6 and inst-0.tsp.
`(-f TSPdata/inst-0.tsp -vepr -mt -c 6 -i 2000 -nr 3 -ucl -ocl)`



From the graph below, it is clear that a low value of the percentage of parents in the new population produced better results. In terms of speed of convergence, a value of around 60% converged fastest.

