Pauric Dawson

R00169689

MSc. Software Arch. And Design

Metaheuristic Optimization

Assignment 1

Part 1: NP-completeness

As the first letter of my name is P (Pauric) I am using the third formula:

$$F = (-z_1 \vee z_2) \wedge (z_1 \vee z_2 \vee z_3 \vee -z_4)$$

1st step, convert from SAT to 3SAT

Clause 1 = $(-z_1 \vee z_2)$

Clause 2 = $(z_1 \vee z_2 \vee z_3 \vee -z_4)$

Clause 1:

$(-z_1 \vee z_2 \vee u_1) \wedge$

$(-z_1 \vee z_2 \vee -u_1)$

Clause 2:

$(z_1 \vee z_2 \vee u_2) \wedge$

$(-u_2 \vee z_3 \vee -z_4 \vee)$

So, $F = (-z_1 \vee z_2) \wedge (z_1 \vee z_2 \vee z_3 \vee -z_4)$ expressed in 3SAT is:

$F= (-z_1 \vee z_2 \vee u_1) \wedge (-z_1 \vee z_2 \vee -u_1) \wedge (z_1 \vee z_2 \vee u_2) \wedge (-u_2 \vee z_3 \vee -z_4 \vee)$

Expressed in a 3COL graph:

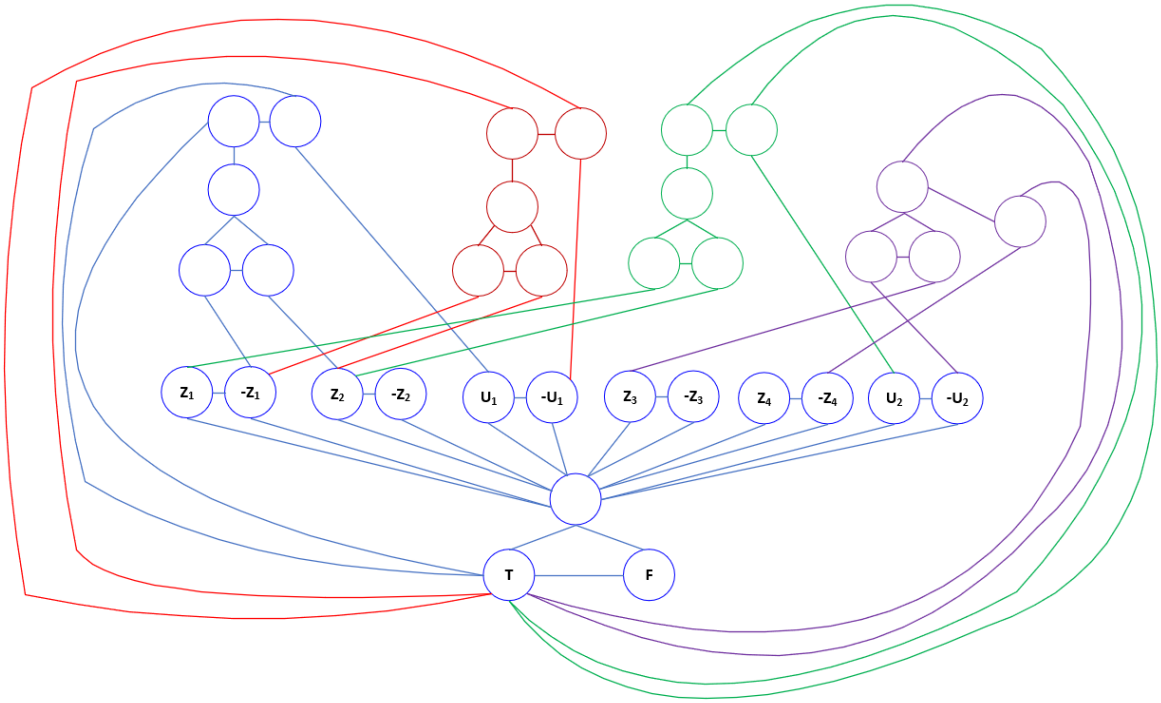


Figure 1 3COL graph (file: As1_3col.pdf)

Proposed solution for F is

$F = (-z_1 \vee z_2 \vee u_1) \wedge (-z_1 \vee z_2 \vee -u_1) \wedge (z_1 \vee z_2 \vee u_2) \wedge (-u_2 \vee z_3 \vee -z_4 \vee)$

z_1 = False, z_2 = True, z_3 = False, z_4 = False, u_1 = False, u_2 = False

Which equate to

$F = (T \vee T \vee F) \wedge (T \vee T \vee T) \wedge (F \vee T \vee F) \wedge (T \vee F \vee T)$

The solution inserted in the graph:

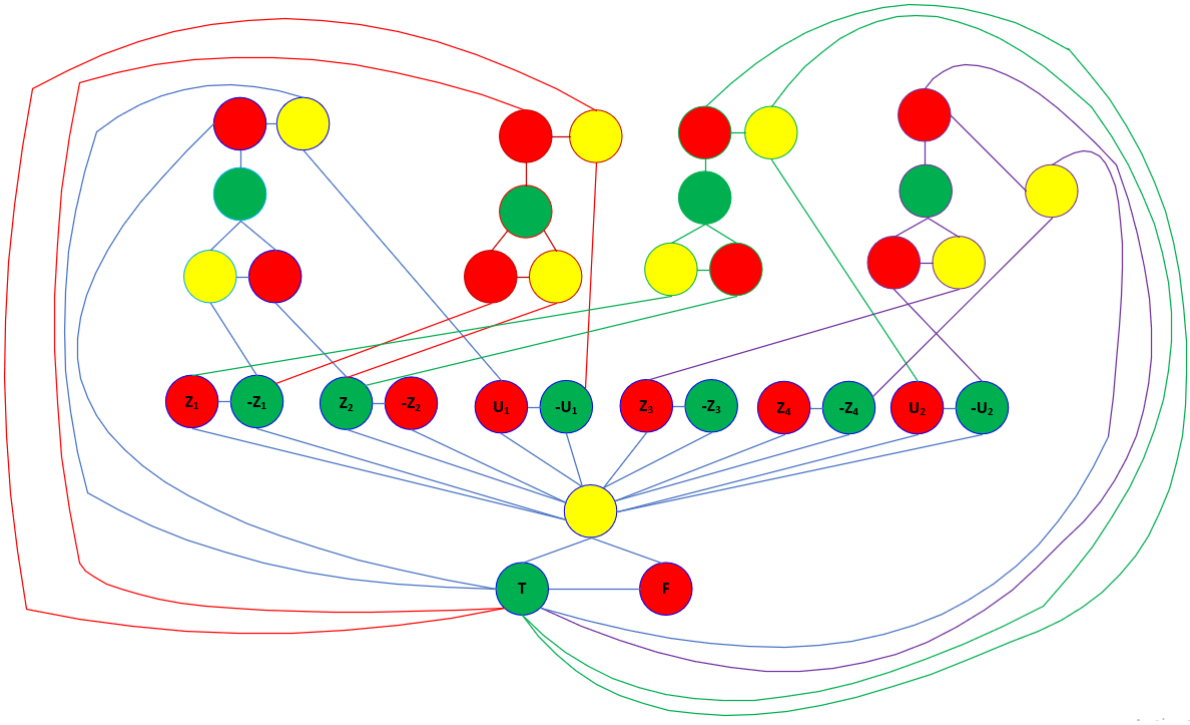


Figure 2 3COL graph with solution F = (T v T v F) ∧ (T v T v T) ∧ (F v T v F) ∧ (T v F v T) (file: As1_3col_Sol.pdf)

And to verify here is a truth table to show the solution, in fact all solutions.

z1	z2	z3	z4	u1	u2	-z1	-z2	-z3	-z4	-u1	-u2	(-z1 or z2 or u1)	(-z1 or z2 or -u1)	(z1 or z2 or u2)	(-u2 or z3 or -z4	Result	
0	1	0	0	0	0	1	0	1	1	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	3COL Solution Graph

Figure 3 Solution in a truth table

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	z1	z2	z3	z4	u1	u2	-z1	-z2	-z3	-z4	-u1	-u2	(-z1 or z2 or u1)	(-z1 or z2 or -u1)	(z1 or z2 or u2)	(-u2 or z3 or -z4)	Result	
2	0	0	0	0	0	0	1	1	1	1	1	1	TRUE	TRUE	FALSE	TRUE	FALSE	
3	0	0	0	0	0	1	1	1	1	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
4	0	0	0	0	1	0	1	1	1	1	0	1	TRUE	TRUE	FALSE	TRUE	FALSE	
5	0	0	0	0	1	1	1	1	1	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
6	0	0	0	1	0	0	1	1	1	0	1	1	TRUE	TRUE	FALSE	TRUE	FALSE	
7	0	0	0	1	0	1	1	1	1	0	1	0	TRUE	TRUE	TRUE	FALSE	FALSE	
8	0	0	0	1	1	0	1	1	1	0	0	1	TRUE	TRUE	FALSE	TRUE	FALSE	
9	0	0	0	1	1	1	1	1	1	0	0	0	TRUE	TRUE	TRUE	FALSE	FALSE	
10	0	0	1	0	0	0	1	1	0	1	1	1	TRUE	TRUE	FALSE	TRUE	FALSE	
11	0	0	1	0	0	1	1	1	0	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
12	0	0	1	0	1	0	1	1	0	1	0	1	TRUE	TRUE	FALSE	TRUE	FALSE	
13	0	0	1	0	1	1	1	1	0	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
14	0	0	1	1	0	0	1	1	0	0	1	1	TRUE	TRUE	FALSE	TRUE	FALSE	
15	0	0	1	1	0	1	1	1	0	0	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
16	0	0	1	1	1	0	1	1	0	0	0	1	TRUE	TRUE	FALSE	TRUE	FALSE	
17	0	0	1	1	1	1	1	1	0	0	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
18	0	1	0	0	0	0	1	0	1	1	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	3COL Solution Graph
19	0	1	0	0	0	1	1	0	1	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
20	0	1	0	0	1	0	1	0	1	1	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
21	0	1	0	0	1	1	1	0	1	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
22	0	1	0	1	0	0	1	0	1	0	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
23	0	1	0	1	0	1	1	0	1	0	1	0	TRUE	TRUE	TRUE	FALSE	FALSE	
24	0	1	0	1	1	0	1	0	1	0	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
25	0	1	0	1	1	1	1	0	1	0	0	0	TRUE	TRUE	TRUE	FALSE	FALSE	
26	0	1	1	0	0	0	1	0	0	1	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
27	0	1	1	0	0	1	1	0	0	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
28	0	1	1	0	1	0	1	0	0	1	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
29	0	1	1	0	1	1	1	0	0	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
30	0	1	1	1	0	0	1	0	0	0	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
31	0	1	1	1	0	1	1	0	0	0	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
32	0	1	1	1	1	0	1	0	0	0	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
33	0	1	1	1	1	1	1	0	0	0	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
34	1	0	0	0	0	0	0	1	1	1	1	1	FALSE	TRUE	TRUE	TRUE	FALSE	
35	1	0	0	0	0	1	0	1	1	1	1	0	FALSE	TRUE	TRUE	TRUE	FALSE	
36	1	0	0	0	1	0	0	1	1	1	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
37	1	0	0	0	1	1	0	1	1	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
38	1	0	0	1	0	0	0	1	1	0	1	1	FALSE	TRUE	TRUE	TRUE	FALSE	
39	1	0	0	1	0	1	0	1	1	0	1	0	FALSE	TRUE	TRUE	FALSE	FALSE	
40	1	0	0	1	1	0	0	1	1	0	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
41	1	0	0	1	1	1	0	1	1	0	0	0	TRUE	TRUE	TRUE	FALSE	FALSE	
42	1	0	1	0	0	0	0	1	0	1	1	1	FALSE	TRUE	TRUE	TRUE	FALSE	
43	1	0	1	0	0	1	0	1	0	1	1	0	FALSE	TRUE	TRUE	TRUE	FALSE	
43	1	0	1	0	0	1	0	1	0	1	1	0	FALSE	TRUE	TRUE	TRUE	FALSE	
44	1	0	1	0	1	0	0	1	0	1	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
45	1	0	1	0	1	1	0	1	0	1	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
46	1	0	1	1	0	0	0	1	0	0	1	1	FALSE	TRUE	TRUE	TRUE	FALSE	
47	1	0	1	1	0	1	0	1	0	0	1	0	FALSE	TRUE	TRUE	TRUE	FALSE	
48	1	0	1	1	1	0	0	1	0	0	0	1	TRUE	TRUE	TRUE	TRUE	TRUE	
49	1	0	1	1	1	1	0	1	0	0	0	0	TRUE	TRUE	TRUE	TRUE	TRUE	
50	1	1	0	0	0	0	0	0	1	1	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
51	1	1	0	0	0	1	0	0	1	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
52	1	1	0	0	1	0	0	0	1	1	0	1	TRUE	FALSE	TRUE	TRUE	FALSE	
53	1	1	0	0	1	1	0	0	1	1	0	0	TRUE	FALSE	TRUE	TRUE	FALSE	
54	1	1	0	1	0	0	0	0	1	0	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
55	1	1	0	1	0	1	0	0	1	0	1	0	TRUE	TRUE	TRUE	FALSE	FALSE	
56	1	1	0	1	1	0	0	0	1	0	0	1	TRUE	FALSE	TRUE	TRUE	FALSE	
57	1	1	0	1	1	1	0	0	1	0	0	0	TRUE	FALSE	TRUE	FALSE	FALSE	
58	1	1	1	0	0	0	0	0	0	1	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
59	1	1	1	0	0	1	0	0	0	1	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
60	1	1	1	0	1	0	0	0	0	1	0	1	TRUE	FALSE	TRUE	TRUE	FALSE	
61	1	1	1	0	1	1	0	0	0	1	0	0	TRUE	FALSE	TRUE	TRUE	FALSE	
62	1	1	1	1	0	0	0	0	0	0	1	1	TRUE	TRUE	TRUE	TRUE	TRUE	
63	1	1	1	1	0	1	0	0	0	0	1	0	TRUE	TRUE	TRUE	TRUE	TRUE	
64	1	1	1	1	1	0	0	0	0	0	0	1	TRUE	FALSE	TRUE	TRUE	FALSE	
65	1	1	1	1	1	1	0	0	0	0	0	0	TRUE	FALSE	TRUE	TRUE	FALSE	

Figure 4 Full solution Truth Table (file: TruthTable_ASSIG1.xlsx)

Part 2 Genetic Algorithms

A Genetic Algorithms was created to try and solve the TSP (Traveling Salesman Problem). The TSP is simply a problem where we want to know the shortest distance a salesman can travel between cities he/she needs to visit only once. The shortest distance is considered the fitness, and the minimum fitness value is the best solution for the salesman.

To try and solve this problem, there are several different approaches we can try to test various combinations to get to a solution to the problem in an efficient and timely manner.

The following are the different configuration combinations that I have used in the Genetic Algorithm to solve this problem.

Configuration	Initial Solution	Crossover	Mutation	Selection
1	Random	Uniform Crossover	Reciprocal Exchange	Random Selection
2	Random	Cycle Crossover	Scramble Mutation	Random Selection
3	Random	Uniform Crossover	Reciprocal Exchange	Roulette Wheel
4	Random	Cycle Crossover	Reciprocal Exchange	Roulette Wheel
5	Random	Cycle Crossover	Scramble Mutation	Roulette Wheel
6	Random	Uniform Crossover	Scramble Mutation	Best and Second best candidates

Configurations 1-2

Random Selection

As the name suggests it quite simply where two parents are select totally at random from the population. These two-parent's fitness is calculated, and initially this is the best solution, but only by default as it the first solution. These are then passed into crossover process and Mutation process.

Crossover

Crossover is an operator in Genetic Algorithms, it is also known as recombination. Crossover is comparable to biology, in the fact that we have two parents that contain a list of genes. These two parents will create two off-spring, the offprint will have some genes from one parent and some from the other parent. What genes each offspring get all depends on the type of Crossover used.

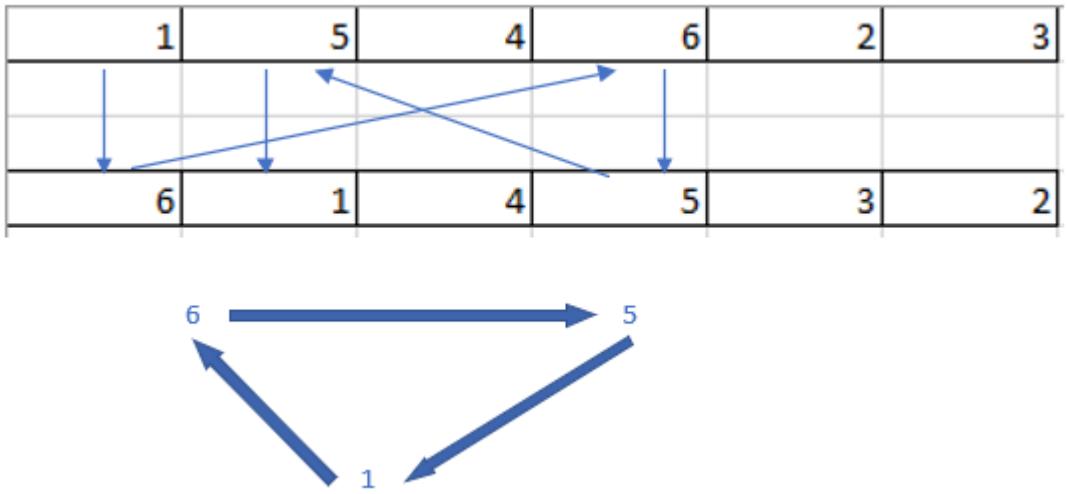
Uniform Crossover

Uniform Crossover process is where we take a uniform number of genes from parent A and add them to Child A and Child B. Take the same number of genes from Parent B and add them to Child A and Child B. In my solution this is done randomly. In my solution I only create one child, so in effect the child contains half the genes form Parent A and half from Parent B. It is worth noting that the position a gene copied from the parent it goes into the same position in the child. In my code I first create two temporary dictionaries that I use to identify genes that will be copied from the parents, while at the same time populating the Child with parent A genes. Then I loop back over to populate with the genes from Parent B.

Cycle Crossover

This a type is cross over is a bit more extensive. In Cycle crossover we establish relationships between a gene in parents A with that of the gene in parent B, then we follow on with the gene in B and find a relationship in parent A until we create a cycle of relationships.

This is an example of a cycle:



The cycle here is 1 to 6 to 5 and back to 1 again to complete the cycle. The remaining cycles in this example are be 4->4 and the final cycle is 2 ->3.

So now we have the cycles, we can start to copy genes from parent to child.

On the first cycle we copy the genes from Parent A to Child A and Parent B to Child B. In the same positions they are found. So then first cycle here is 1->6->5, we take these genes and copy them into the Child in the same position that they are found in the parents:

1	5		6			Child A

6	1		5			Child B

Then on the next cycle we take the genes in Parent A and put them in Child B, and vice versa Parent B into Child A. Once that cycle is done we revert to Parent A to Child A and Parent B to Child B. And we alternate like this until all cycle are done.

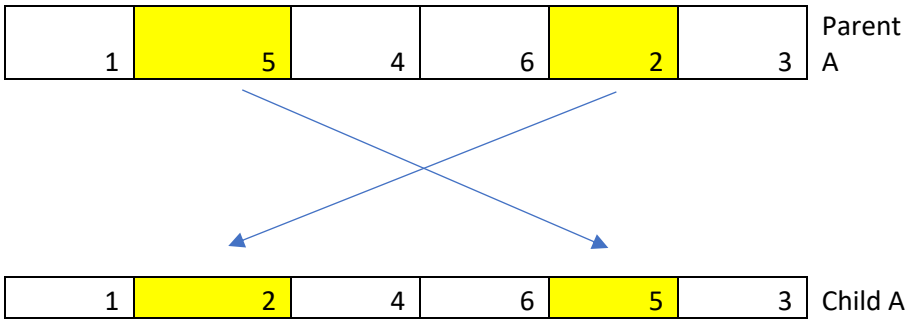
In my code I am creating just one child so only Child A. My first step is to identify all the cycle. I do this by first setting up to temporary dictionaries for parent A and B. I use these to mark the genes that have been included in a cycle. I use two while loops to create the cycles. The inner While loop creates a cycle, while the outer While loop control a new cycle loop. Once the cycles have been identified, I then loop though the different cycles copying the genes from Parent to child, alternating in the coping from Parent A to Child A and Parent B to Child A, until all genes are processed, and I have a new Child.

Mutations

It is a Genetic Algorithm operator used to maintain genetic diversity from one generation of the population to the next. A mutation will alter one or more gene. When a parent mutates its solution can changed completely and so we can come to a better solution. Mutation occurs according to mutation probability or mutation rate. It is best to have a low rate as if it is set to high the search would be comparable to a random search. Its worth noting mutations operate on just one parent, as opposed to crossover the needs two parents.

Reciprocal Exchange

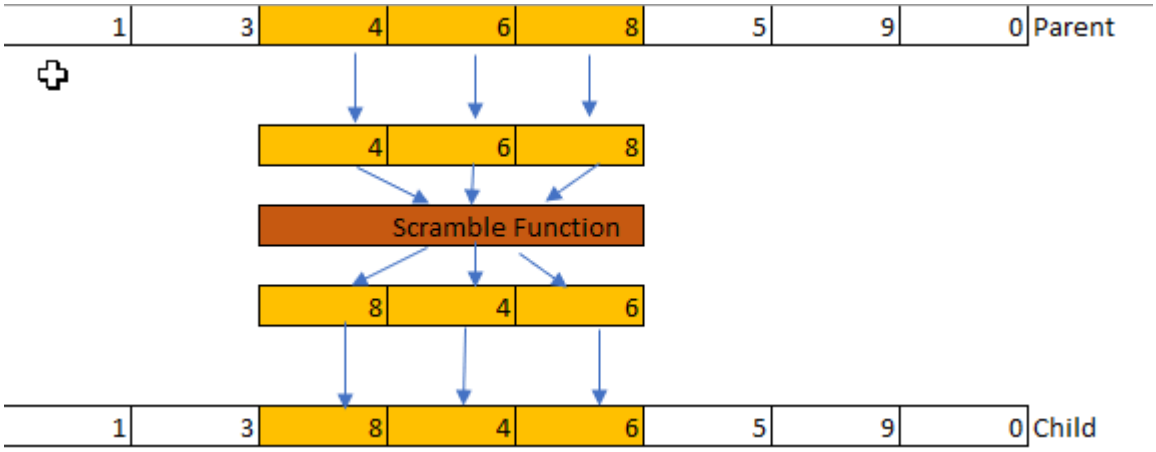
Is straight forward, in that we have a parent and two genes are identified at random, these two genes are that swapped with one another to for a new child. Here is an example:



5 and 2 are swapped over, and so child A is formed and its only difference to its parent is that genes 5 and 2 are in different positions.

Scrambled Mutation

Again, this is carried out on a single parent, this time a sequential range of genes is identified in the parent. This range of gens is then extracted and scrambles up and placed back in the parent to form a new Child. This diagram shows how the scramble mutation works:



Analysis of Config 1-2

Some analysis of Config 1-2. (See appendix for enviroment and reporting structure of results)

The results data I will present is based on the time for each run to execute (in seconds) and on the Best Fitness it produced.

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	39.3025895	300	22044050.912971500
dataset/inst-0.tsp	2	1	uniform	reciprocal	random	38.67903794	300	21822716.338371800
dataset/inst-0.tsp	3	1	uniform	reciprocal	random	38.81067083	300	21865067.363594900
					Average	38.93076609		21910611.54
					Median	38.81067083		21865067.36

Figure 5 Config 1 inst-0 Results

So for Configuration 1 with inst-0, where it is random selection, with Uniform Crossover and Reciprocal Mutation, there was not much difference between the 3 runs. The execution time averaged 38.9 seconds which it quite quick but that is due to the fact that the Selection, crossover and Mutation were on the simpler side of the scale and so did not take much processing power to execute.

The best fitness solution found was 21822716.33 .

dataset/inst-0.tsp	1	2	cycle	scramble	random	52.9773491	300	21862707.355678000
dataset/inst-0.tsp	2	2	cycle	scramble	random	52.40569443	300	21385587.684202100
dataset/inst-0.tsp	3	2	cycle	scramble	random	54.19527844	300	22023141.275475500
					Average	53.19277399		21757145.44
					Median	52.9773491		21862707.355678000

Figure 6 Config 2 inst-0 Results

For Configuration 2, where we have Random selection with Cycle Crossover and Scramble Mutation. Here we can see the execution time has now increased, to an average of 53 seconds, this would account for the more extensive processing power and computations needed to carry out Cycle crossover and Scramble mutations. On the plus side we do have a better solution in our results.

The other two instances inst-13 and inst-16 show a similar form:

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	107.7325255	300	109325129.122718000
dataset/inst-13.tsp	2	1	uniform	reciprocal	random	105.2751906	300	107990234.510067000
dataset/inst-13.tsp	3	1	uniform	reciprocal	random	104.7821771	300	110169683.718816000
					Average	105.9299644		109161682.450534000
					Median	105.2751906		109325129.122718000
dataset/inst-13.tsp	1	2	cycle	scramble	random	135.4631927	300	106195519.179238000
dataset/inst-13.tsp	2	2	cycle	scramble	random	134.8855782	300	107005375.966243000
dataset/inst-13.tsp	3	2	cycle	scramble	random	135.1846114	300	108778487.859615000
					Average	135.1777941		107326461.001699000
					Median	135.1846114		107005375.966243000

Figure 7 inst-13 Results for Config 1 and Config 2

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	85.45267956	300	107049850.188819000
dataset/inst-16.tsp	2	1	uniform	reciprocal	random	86.93666515	300	103102861.868869000
dataset/inst-16.tsp	3	1	uniform	reciprocal	random	85.36236661	300	108019968.084626000
					Average	85.91723711		106057560.047438000
					Median	85.45267956		107049850.188819000
dataset/inst-16.tsp	1	2	cycle	scramble	random	107.9030449	300	104248297.134213000
dataset/inst-16.tsp	2	2	cycle	scramble	random	106.8813837	300	101297639.135309000
dataset/inst-16.tsp	3	2	cycle	scramble	random	104.7216725	300	103990991.362149000
					Average	106.5020337		103178975.877224000
					Median	106.8813837		103990991.362149000

Figure 8 inst-16 Results for Config 1 and Config 2

It is worth point out at this point the volume of data for these instances, it can be seen in the table below:

Instance	Data
Inst-0	184
Inst-13	352
inst-16	302

Inst-13 has almost double the data of inst-0 which would explain the increase in times for processing that instance.

Configurations 3-6

In configurations 3 -6 we introduce two new types of selection:

- Roulette Wheel
- Best and Second Best

Roulette Wheel Selection

Is also known as Fitness proportionate selection and is another genetic operator that is used in selecting possible solutions that can then be passed to crossover and /or mutation for analysis as the best solution. It does this by associating the fitness level of each individual with the probability of selection. The reason it gets called Roulette selection as it can be seen as similar to that of a Roulette wheel in a casino. A portion of the wheel is assigned each possible selection based on their fitness. With a higher fitness an individual will get a larger section on the wheel and so there is a higher probability that it will get selected. The random selection is made by the "spin of the wheel". This gives a high preference to the best individuals, these best individuals are usually better candidates to use for crossover and mutation.

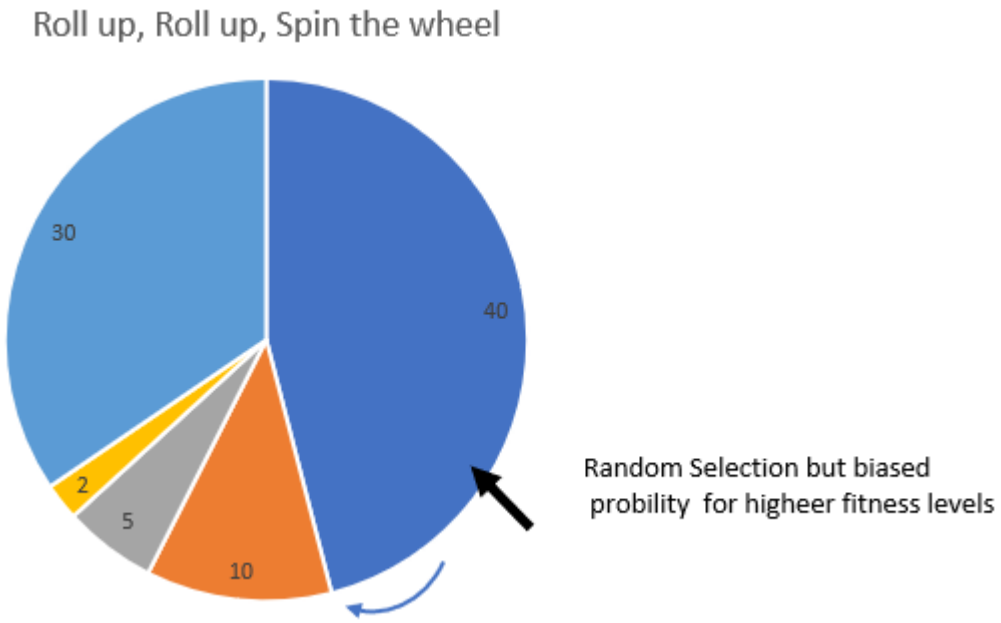


Figure 9 Roulette Selection

So to do Roulette Wheel Selection we must first work out the fitness of all individuals, then apply the percentage of their fitness against the total fitness (formula individual fitness / Total Fitness), this give us the probability. An example can be seen in the table below:

Individual	1	2	3	4	5	Totals
Fitness	40	30	10	5	2	87
Probability (fitness/total Fitness)	0.45977	0.344827586	0.114943	0.057471	0.022989	1

As we can see Individual 1 is almost 46% of the total fitness and so will take up more space on the roulette wheel and so have a higher chance of getting selected. To do the actual selection in code we select a random number between 0 and 1 and sum the fitness values and select the first individual whose fitness value added to the sum of the fitness is greater than or equal to the random number.

This process works great for when you want to find the maximum fitness individuals, but sometime a lower fitness number indicates a better fitness, this is particularly true in the Travel Salesman Problem, as the fitness is expressed as distance so the whole point is to get a lowest number of miles. To do this we make a slight change to the working of the probability so as to give a high probability rating to individuals with a low fitness.

We can do this by using one of two different formulas to work out the probability:

- 1/fitness_i
- MaxFitness +1 – Fitness

For my implementation for the Roulette Wheel I used the 1/fitness_i formula. Fitness_i is the fitness of the individual. What we are doing is to transform small values into big values to give them better probability.

Individual	1	2	3	4	5	Totals
Fitness	40	30	10	5	2	87
1/Fitness_i	0.025	0.033333333	0.1	0.2	0.5	0.858333
Probability	0.029126	0.038834951	0.116505	0.23301	0.582524	1

So now we can see 5 has the smallest fitness, and so would have been least likely to be selected pre-transformation, but with the transformation with the 1/fitness_i formula the re-working probability now shows 5 to have the highest probability. The individual 5 probability was arrived at by taking 1/5 = 0.5. Then the new total fitness for all individuals is 0.85833. So 0.5 / 0.85833 equals a probability for 0.582524.

Best and Second Best

Quite simply as the name suggests, the selection criteria is based on the best fitness. So to select the two best individuals we find the fitness for all individuals and sort the population of tier fitness score. Then the two best individuals with the best fitness score are used to pass though crossover and mutation. In my implementation I start by doing a sort of the population based on fitness, because this is a TSP problem the lowest fitness score is the

best and so if were to do a reverse sort and get the highest fitness values this would be the worst solutions to the TSP.

Analysis of config 3-6

As before I will focus on inst-0 run results. So here are the results for the 3-6 configs for inst-0.

Config 3 where Selection is Roulette Wheel, Mutation is Reciprocal and Crossover is uniform:

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	42.23653857	300	21516578.777662600
dataset/inst-0.tsp	2	3	uniform	reciprocal	roulette	41.32930869	300	22035560.534809300
dataset/inst-0.tsp	3	3	uniform	reciprocal	roulette	40.52327175	300	21798865.937107900
					Average	41.36303967		21783668.416526600
					Median	41.32930869		21798865.937107900

Figure 10 inst-0 config3 results

Config 3 has quite a good execution time compare with config 1 and config 2, but interestingly it does not have the best Solution between config 1 and 3. The time is interesting here in that we are now using a Roulette Wheel for our selection. This does require a bit more work to achieve the selection candidates.

Config 4 now, where we use Cycle Crossover, with reciprocal mutation and roulette selection.

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	43.79692486	300	22172598.863859400
dataset/inst-0.tsp	2	4	cycle	reciprocal	roulette	43.76937459	300	22211314.730658900
dataset/inst-0.tsp	3	4	cycle	reciprocal	roulette	44.01076951	300	22050085.446134300
					Average	43.85902299		22144666.346884200
					Median	43.79692486		22172598.863859400

Figure 11 inst-0 Config 4 results

What can be noted here is that compared to config 3 only the cross over method is different, in this config we are using Cycle where as in Config 3 we are using Uniform. We can see a slight increase in time but the Best solution it provided is one of the worst yet.

Config 5 where the crossover method is cycle, the mutation is scramble and the selection is roulette.

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	55.58393355	300	21864218.404094400
dataset/inst-0.tsp	2	5	cycle	scramble	roulette	54.45637697	300	21683225.868725100
dataset/inst-0.tsp	3	5	cycle	scramble	roulette	55.42982554	300	21495678.865333800
					Average	55.15671202		21681041.046051100
					Median	55.42982554		21683225.868725100

Figure 12 inst-0 Config 5 Results

Here we see an increase in time compared to the previous config. Though we do have a better solution and on average one of the best solutions so far.

Finally, we have **config 6**, where the selection is Best and Second Best, Crossover is Uniform and the Mutation is Scramble.

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	49.98967138	300	21449434.748263600
dataset/inst-0.tsp	2	6	uniform	scramble	bestandsecond	52.00932322	300	21574760.393878700
dataset/inst-0.tsp	3	6	uniform	scramble	bestandsecond	56.8605179	300	21464653.275131300
					Average	52.95317083		21496282.805757900
					Median	52.00932322		21464653.275131300

Figure 13 inst-0 Config-6 Results

This time we see a decrease in time taken and this can be put down to the fact that Best and Second Best is quite a simple selection process., though interesting it does show a very good solution.

For completeness here are the results for the other instance inst-13 and inst-16 for config 3-6

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	111.6226049	300	107797085.209857000
dataset/inst-13.tsp	2	3	uniform	reciprocal	roulette	109.0448139	300	111764523.300702000
dataset/inst-13.tsp	3	3	uniform	reciprocal	roulette	109.9034898	300	109846799.312239000
					Average	110.1903028		109802802.607599000
					Median	109.9034898		109846799.312239000
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	120.4736978	300	110460374.547647000
dataset/inst-13.tsp	2	4	cycle	reciprocal	roulette	113.2510179	300	109964491.691940000
dataset/inst-13.tsp	3	4	cycle	reciprocal	roulette	113.6928401	300	111398264.367502000
					Average	115.8058519		110607710.202363000
					Median	113.6928401		110460374.547647000
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	137.3940267	300	106557853.837938000
dataset/inst-13.tsp	2	5	cycle	scramble	roulette	137.944968	300	107483520.761226000
dataset/inst-13.tsp	3	5	cycle	scramble	roulette	136.7078347	300	107521023.262645000
					Average	137.3489431		107187465.953936000
					Median	137.3940267		107483520.761226000
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	126.0033324	300	107432659.904695000
dataset/inst-13.tsp	2	6	uniform	scramble	bestandsecond	125.948963	300	105575895.174845000
dataset/inst-13.tsp	3	6	uniform	scramble	bestandsecond	125.2976391	300	107884110.432212000
					Average	125.7499781		106964221.837251000
					Median	125.948963		107432659.904695000

Figure 14 inst-13 Config 3-6 Results

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	84.64592167	300	108265786.232663000
dataset/inst-16.tsp	2	3	uniform	reciprocal	roulette	84.50137791	300	105267327.387274000
dataset/inst-16.tsp	3	3	uniform	reciprocal	roulette	83.53805083	300	106655476.328515000
					Average	84.22845014		106729529.982817000
					Median	84.50137791		106655476.328515000
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	95.98677479	300	107981417.573159000
dataset/inst-16.tsp	2	4	cycle	reciprocal	roulette	96.4776753	300	104304440.098434000
dataset/inst-16.tsp	3	4	cycle	reciprocal	roulette	93.14620739	300	106627960.841421000
					Average	95.20355249		106304606.171005000
					Median	95.98677479		106627960.841421000
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	109.9239384	300	104258418.635852000
dataset/inst-16.tsp	2	5	cycle	scramble	roulette	109.9140566	300	103385446.283616000
dataset/inst-16.tsp	3	5	cycle	scramble	roulette	109.5532011	300	103865086.952630000
					Average	109.7970654		103836317.290699000
					Median	109.9140566		103865086.952630000
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	101.0974839	300	103587486.876600000
dataset/inst-16.tsp	2	6	uniform	scramble	bestandsecond	100.1975463	300	101847155.196828000
dataset/inst-16.tsp	3	6	uniform	scramble	bestandsecond	103.5671794	300	102958922.022047000
					Average	101.6207365		102797854.698492000
					Median	101.0974839		102958922.022047000

Figure 15 inst-16 Config 3-6 Results

Closer examination

Looking that the config it is worth while looking a bit closer and doing a compare at two sets of configs.

- Config 1 v Config 3
- Config 4 V Config 5

Config 1 v Config 3

I selected these two config as the only difference is the selection method. We use random selection in config 1 and Roulette Wheel selection in config 3.

Lets have a look at the results again:

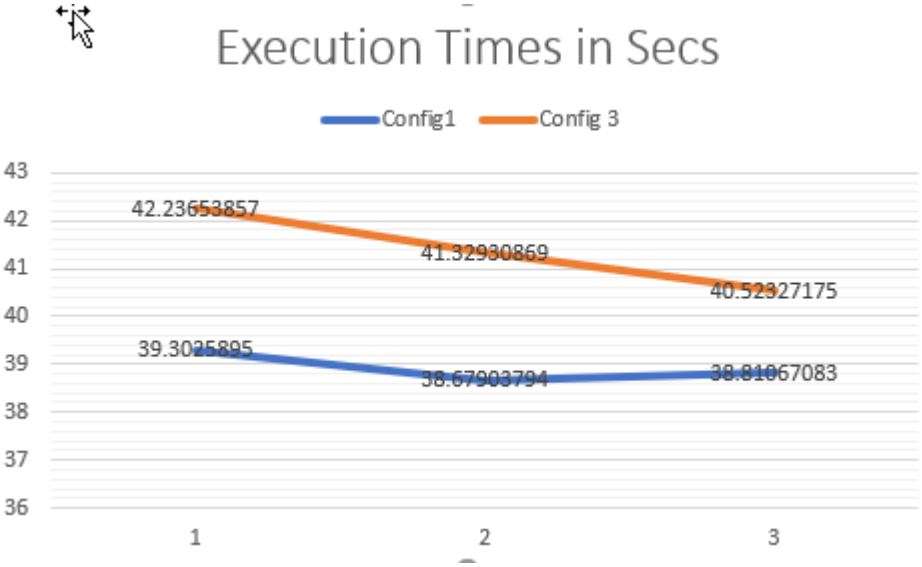
Config 1

Instance:	Run numb	Configura	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	39.3025895	300	22044050.912971500
dataset/inst-0.tsp	2	1	uniform	reciprocal	random	38.67903794	300	21822716.338371800
dataset/inst-0.tsp	3	1	uniform	reciprocal	random	38.81067083	300	21865067.363594900
					Average	38.93076609		21910611.538312700
					Median	38.81067083		21865067.363594900

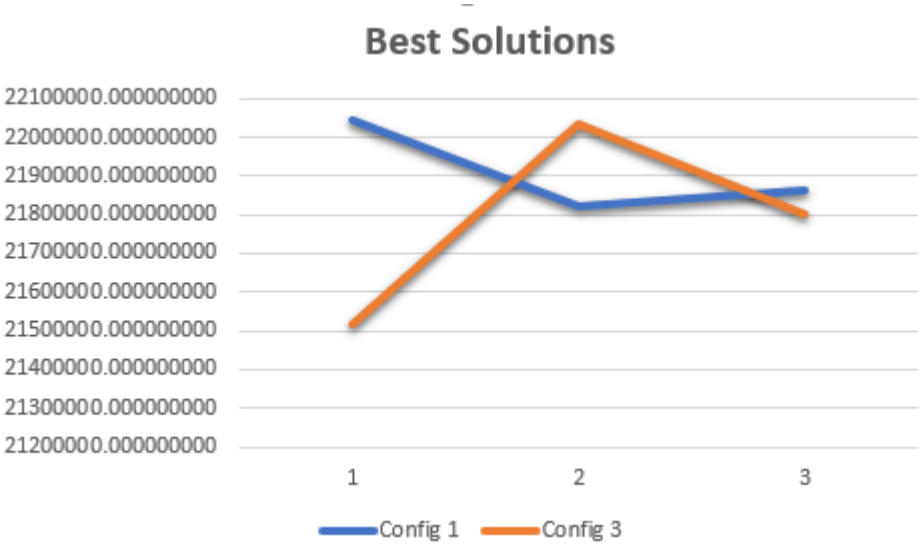
Config 3

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	42.23653857	300	21516578.777662600
dataset/inst-0.tsp	2	3	uniform	reciprocal	roulette	41.32930869	300	22035560.534809300
dataset/inst-0.tsp	3	3	uniform	reciprocal	roulette	40.52327175	300	21798865.937107900
					Average	41.36303967		21783668.416526600
					Median	41.32930869		21798865.937107900

Let's look at the execution times:



We can see over the 3 run there is similar downward curve but that only represents a small time difference. Though what we can see is the that Config 3 does take longer to run over all, so for that we can say that the Roulette Wheel does increase the time execution and we can put that down to more work to do in the selection process.



Here we see that the overall best solution was provided by config 3, so the Roulette Wheel did seem to help in providing a better solution. Even though we do see a peeking config 3, the Average still comes out the best between the two of them.

Config 4 V Config 5

The reason I select these two config was that the only difference between them is the Mutation method. Config 4 uses Reciprocal Exchange and Config 5 uses Scramble Method.

So a quick recap on the results of config 4 and 5:

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	43.79692486	300	22172598.863859400
dataset/inst-0.tsp	2	4	cycle	reciprocal	roulette	43.76937459	300	22211314.730658900
dataset/inst-0.tsp	3	4	cycle	reciprocal	roulette	44.01076951	300	22050085.446134300
					Average	43.85902299		22144666.346884200
					Median	43.79692486		22172598.863859400

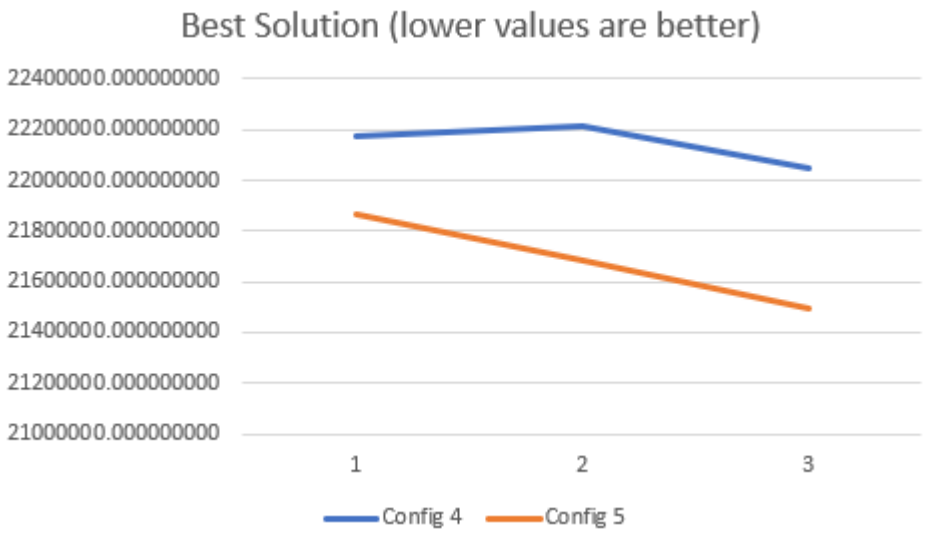
Figure 16 inst-0 Config 4 Results

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	55.58393355	300	21864218.404094400
dataset/inst-0.tsp	2	5	cycle	scramble	roulette	54.45637697	300	21683225.868725100
dataset/inst-0.tsp	3	5	cycle	scramble	roulette	55.42982554	300	21495678.865333800
					Average	55.15671202		21681041.046051100
					Median	55.42982554		21683225.868725100

Figure 17 inst-0 Config 5 results

Let's look at their execution times, which is usually a good indication of the work needed to be done:

We can see here that it takes longer to run Config 5, but that is expected as Scramble Mutation does take more work and effort to complete mutation, so on average for my executions it took 11 seconds more. But is the solution better? If it is then maybe the time is well spent, let see.



Good news that extra time was well spent as we can see config 5 consistently returned the best result over the 3 runs.

Further Testing

I decided to take the following parameters and adjust them to see what impact it had on my results.

- Iteration Rates
- Mutation Rate
- Population size.

In the next section I just do the analysis on inst-0 but I have run them against all the instances and have included all the results in the appendix.

Iteration Rates

I ran two different rates. I used the above execution results as my baseline. For Iterations in my Baseline it was set to 300. First, I ran with increase iterations, I set it to 600 and then I ran it for iterations set to 100.

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	77.34925518	600	21993373.28738470
dataset/inst-0.tsp	1	2	cycle	scramble	random	103.5104218	600	21304914.98697670
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	80.51163105	600	21966940.83488010
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	90.95904765	600	22176361.96542710
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	111.655467	600	20633123.47485910
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	99.95700184	600	21587319.74898330

Figure 18 Iteration results for Iterations set to 600

Instance:	Run number:	Configuratio	Crossove	Mutation	Selection	Executio	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	15.28926	100	21882274.61461820
dataset/inst-0.tsp	1	2	cycle	scramble	random	20.41276	100	22003134.59879310
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	15.74552	100	22604043.85166340
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	17.43477	100	22511696.91219540
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	20.94301	100	21864073.81239370
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	19.10419	100	21683678.14553560

Figure 19 Iteration results for Iterations set to 100

Figure 20 Iterations test Execution Times 100 V's Baseline V's 600

As we can see, and what we would expect is that the less Iterations there is the less time it will take, as these figures are across all configurations 1-6.

Let's see now if increasing the iteration improves our solution.

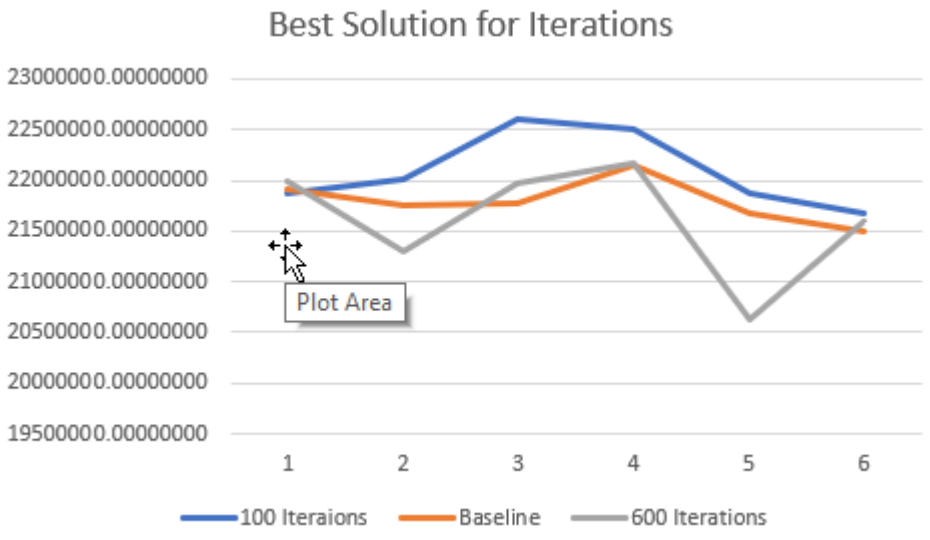


Figure 21 Iterations test Best Solutions 100 V's Baseline V's 600

First, over all we can see the 100 iterations gave the worst solutions. There really is not much between the Baseline and 600 Iterations (double the baseline) It is better in config 2 and hugely better in config 5. Though looking at the time comparison chart these are the two configs that took 600 iterations the lonest to complete. What we can see is that config 2 and config 5 both had Cycle crossover and Scramble mutation which we have seen can take the most time but looks like they can improve the results the more iterations you use. Also config 5 used Roulette Wheel selection would explain again why it took the longest.

Mutation Rate

For my testing on Mutation rate I again did two executions once with a mutation rate of .4 and then a Mutation rate on .001. If you recall the mutation rates for the initial run was set to .1 and I will use that run for my baseline.

So let's look a t a comparison of execution time between a mutation rate of .4 v' Baseline (.1) v' .001.

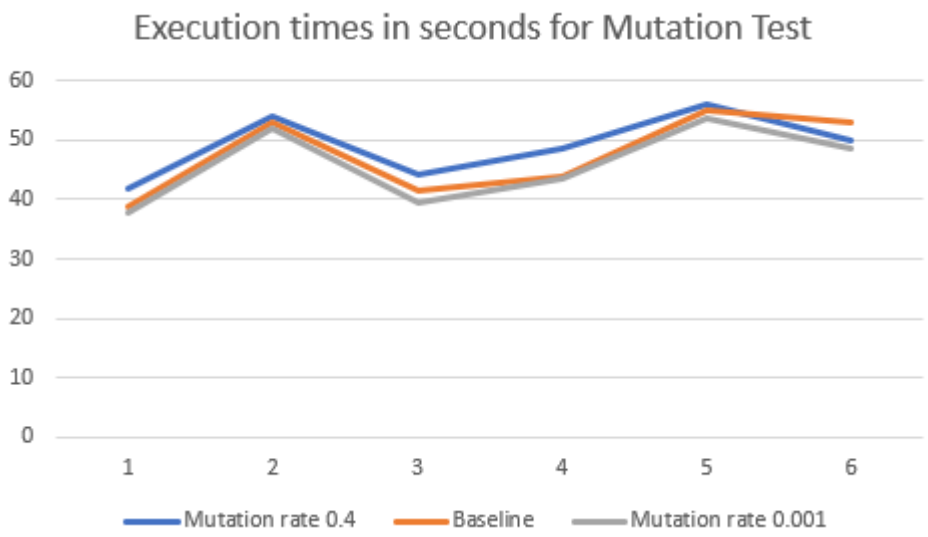


Figure 22 Mutation Rate test Execution Times 0.4 V's Baseline V's 0.001

As we can clearly see the increase or the decrease in the mutation rate did not impact the execution time at all, with the a mutation rate of 0.4 taking just a slight bit longer.

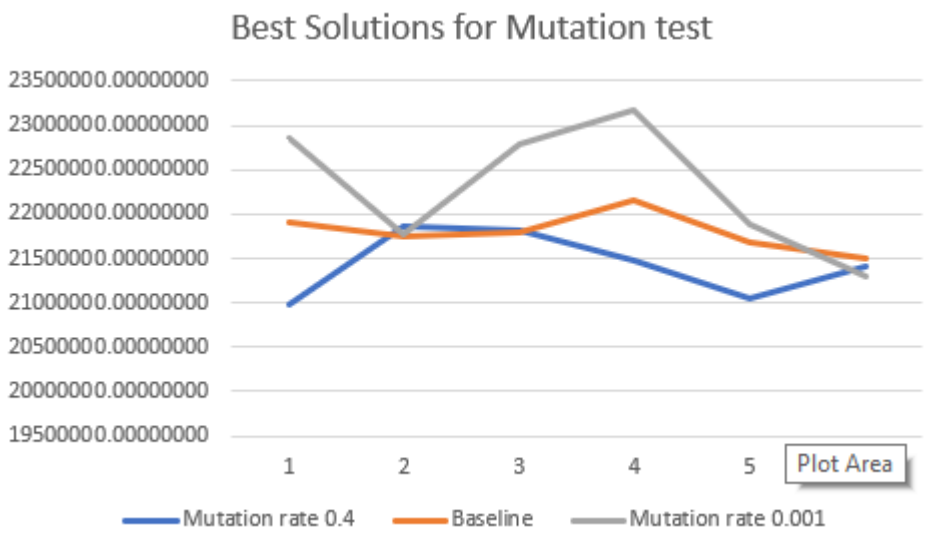


Figure 23 Mutation Rate test Best Solution 0.4 V's Baseline V's 0.001

This time it appears that the best solution is returned with a Mutation Rate on .001 (ie a smaller mutation rate). Interestingly, Config 2 and config 6 Best Solution appears to be the same for all Mutation rates.

Population Size

For my final test I have altered the Population Size. Again I did two executions once with a Population size of 400 and then a Population size of 30. If you recall the Population size for the initial run was set to 100 and I will use that run for my baseline.

As before lets first look at the comparison in terms of execution time.

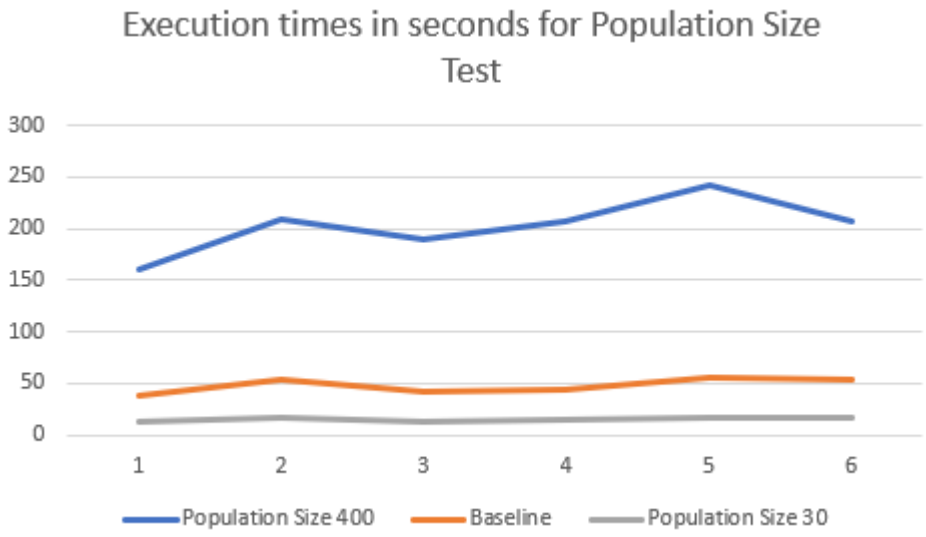


Figure 24 Population Size Test Execution Times 400 V's Baseline V's 30

And again, what we would expect to see is that with the increase in the Population size the execution times increase as well. This is down to the simple fact that there is more work to do over each iteration, there is a bigger population for the GA to work it way thorough. But did this have an impact on the Solutions? have we used that time wisely in our GA?

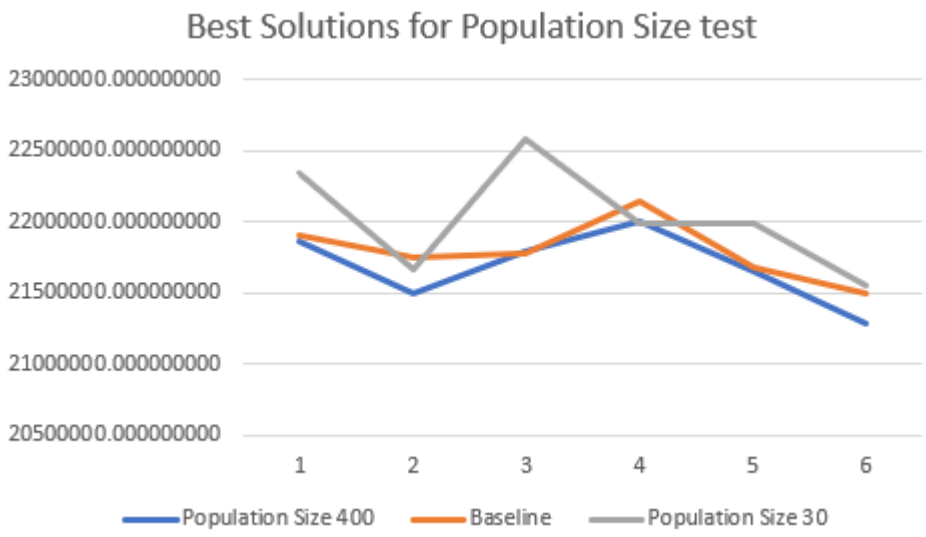


Figure 25 Population Size Best Solution 400 V's Baseline V's 30

The answer is but not as much as you would think in compared to the time to took to get a result. The larger Population size helped to get a better solution in Config 2 and Config 6, but in comparison to the baseline results there is not a massive increase in the best solution. This could be down to the fact that the large population size could come to a point where it plateaus and has found it best solution but still have a lot of individuals in the population still to process.

Appendix

Tests were carried out this the following enviroment unless otherwise stated)

Enviroment

Windows 10, 32GB of Ram, i7 processor @2.80GHz, 1TB SSD

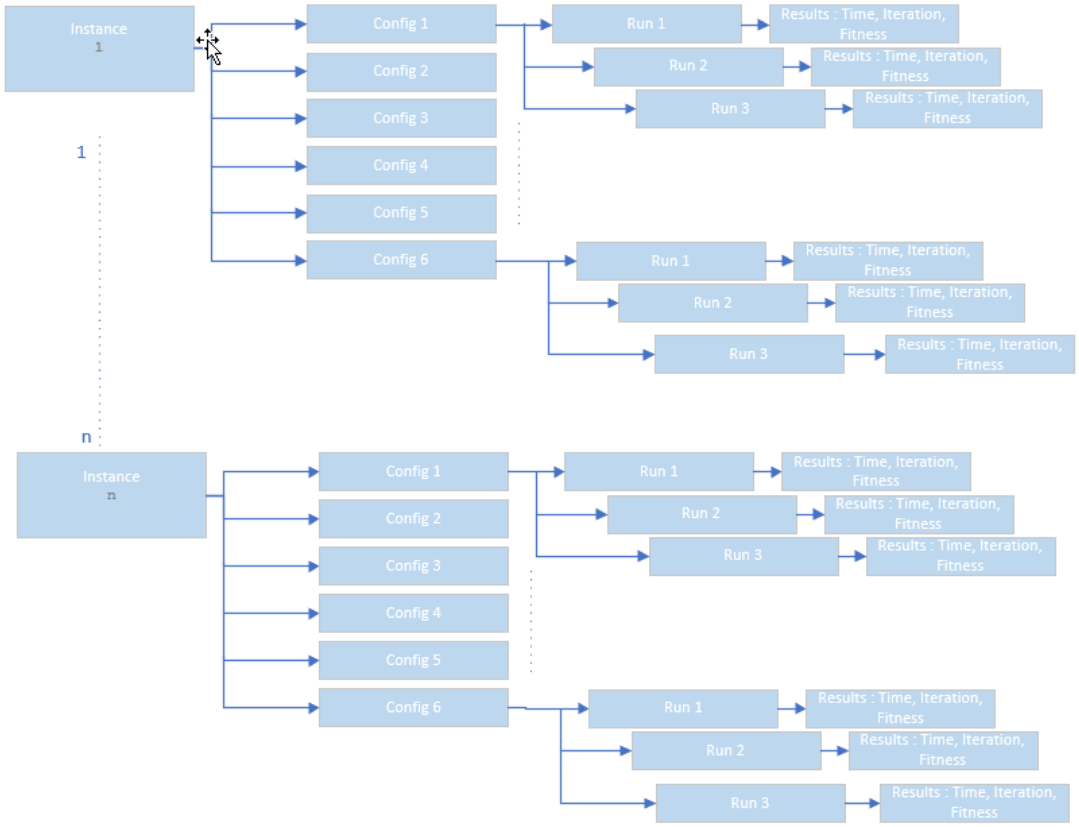
Python

Python 3.7 using Pycharm IDE

Population and Mutations Rate

Population size was 100 and Mutation rate was 0.1

Results Run Order and Structure



Further Analysis full results

Iterations Full Test Results

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	77.34925518	600	21993373.28738470
dataset/inst-0.tsp	1	2	cycle	scramble	random	103.5104218	600	21304914.98697670
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	80.51163105	600	21966940.83488010
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	90.95904765	600	22176361.96542710
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	111.655467	600	20633123.47485910
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	99.95700184	600	21587319.74898330

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	208.3095547	600	110332116.06485000
dataset/inst-13.tsp	1	2	cycle	scramble	random	281.6029067	600	108238724.40747600
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	245.4479731	600	111034555.70705400
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	264.2338849	600	109972434.10818500
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	308.5373824	600	105573791.07347600
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	288.8711711	600	106351097.55608800

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	191.7013344	600	106473469.00809100
dataset/inst-16.tsp	1	2	cycle	scramble	random	246.3916932	600	104854336.14924200
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	196.5168896	600	105128505.33445300
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	215.640965	600	106737241.99136800
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	252.5223812	600	103166382.73415600
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	233.9348965	600	100801782.38265400

Figure 26 All instances with Iterations Set to 600

Instance:	Run number:	Configuratio	Crossove	Mutation	Selection	Executio	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	15.28926	100	21882274.61461820
dataset/inst-0.tsp	1	2	cycle	scramble	random	20.41276	100	22003134.59879310
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	15.74552	100	22604043.85166340
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	17.43477	100	22511696.91219540
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	20.94301	100	21864073.81239370
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	19.10419	100	21683678.14553560

Instance:	Run number:	Configuratio	Crossove	Mutation	Selection	Executio	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	40.19799	100	108951749.40182300
dataset/inst-13.tsp	1	2	cycle	scramble	random	51.18965	100	108807378.07546200
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	40.83795	100	111435206.87287800
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	44.55825	100	108870222.11187900
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	51.74689	100	108448897.28465900
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	48.23508	100	107906608.42199600

Instance:	Run number:	Configuratio	Crossove	Mutation	Selection	Executio	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	32.36168	100	108631529.16141500
dataset/inst-16.tsp	1	2	cycle	scramble	random	41.05401	100	104476264.15315800
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	32.75114	100	108239932.05688000
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	35.42188	100	107585463.90281100
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	41.68695	100	104080854.12712800
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	38.42702	100	104363657.85006700

Figure 27 All instances with Iterations sett to 100

Mutation Rate Test Results

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	41.90405906	300	20976593.86016800
dataset/inst-0.tsp	1	2	cycle	scramble	random	53.99986033	300	21854905.39813450
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	44.3046076	300	21815765.22486330
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	48.69418424	300	21478930.90828570
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	55.95621232	300	21051473.36890150
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	50.10880841	300	21413745.98619430

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	113.0559922	300	106404372.65442800
dataset/inst-13.tsp	1	2	cycle	scramble	random	130.632994	300	105744373.15498600
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	117.2222985	300	107658283.59716400
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	125.5816172	300	109674108.44474400
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	138.9556062	300	106127992.40136100
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	130.5013126	300	105831806.97825700

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	90.30181749	300	105573390.99342600
dataset/inst-16.tsp	1	2	cycle	scramble	random	106.6359008	300	104240810.10903500
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	92.54383527	300	105761500.29441900
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	100.1775973	300	104315114.29628700
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	110.7399725	300	104831852.08504100
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	101.704968	300	102048297.02427300

Figure 28 All Instances with Mutation Rate set to 0.4 Results

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iterations	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	37.75574861	300	22855378.07103450
dataset/inst-0.tsp	1	2	cycle	scramble	random	51.9011446	300	21763432.25155550
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	39.60911054	300	22792363.45461060
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	43.49546657	300	23160736.13951840
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	53.83633137	300	21887511.38081170
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	48.77320256	300	21305740.65345140

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iterations	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	104.7666662	300	113363578.50973500
dataset/inst-13.tsp	1	2	cycle	scramble	random	135.8634707	300	107826097.28594600
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	105.981572	300	113926731.49039500
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	116.8019377	300	112671612.84118300
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	135.2134709	300	104761114.90355200
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	126.1233191	300	105696727.75554900

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iterations	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	83.44117998	300	109935726.42411900
dataset/inst-16.tsp	1	2	cycle	scramble	random	107.6273573	300	102127637.04760900
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	84.75696844	300	110225999.82972400
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	93.35024307	300	110580804.28069700
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	110.7317845	300	103797978.73977600
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	100.0822956	300	103051678.61183500

Figure 29 All Instances with Mutation Rate set to 0.001 Results

Population Size Test Results

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	160.1493242	300	21857039.085754000
dataset/inst-0.tsp	1	2	cycle	scramble	random	209.9270878	300	21491512.123552300
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	189.6017857	300	21797380.920132100
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	206.5400897	300	22009131.271810300
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	241.2612036	300	21653348.553535600
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	206.3428814	300	21287147.232218800

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	430.6033276	300	110037272.709812000
dataset/inst-13.tsp	1	2	cycle	scramble	random	533.3988127	300	105261981.493569000
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	460.3714612	300	108298127.077754000
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	495.4303109	300	108230843.285538000
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	554.5138111	300	104637431.320572000
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	500.7482772	300	105208475.657901000

Instance:	Run numb	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	331.0169266	300	104259674.726314000
dataset/inst-16.tsp	1	2	cycle	scramble	random	417.8079092	300	102941034.598591000
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	368.7957214	300	104440065.017140000
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	403.7456811	300	104440902.323298000
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	467.6476258	300	101220588.735130000
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	422.4765758	300	101371465.667963000

Figure 30 All Instances with Population Size to set to 400 Results

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-0.tsp	1	1	uniform	reciprocal	random	12.54155534	300	22340684.46572140
dataset/inst-0.tsp	1	2	cycle	scramble	random	16.96275382	300	21669744.20339540
dataset/inst-0.tsp	1	3	uniform	reciprocal	roulette	12.20549036	300	22577666.60761920
dataset/inst-0.tsp	1	4	cycle	reciprocal	roulette	13.76118441	300	21990419.64187930
dataset/inst-0.tsp	1	5	cycle	scramble	roulette	16.88228374	300	21996594.44328790
dataset/inst-0.tsp	1	6	uniform	scramble	bestandsecond	15.78436007	300	21550071.90948060

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-13.tsp	1	1	uniform	reciprocal	random	33.7013114	300	112020894.24805300
dataset/inst-13.tsp	1	2	cycle	scramble	random	42.64735848	300	105854895.30439500
dataset/inst-13.tsp	1	3	uniform	reciprocal	roulette	32.92206622	300	113356867.11540300
dataset/inst-13.tsp	1	4	cycle	reciprocal	roulette	37.73498306	300	112592819.73193100
dataset/inst-13.tsp	1	5	cycle	scramble	roulette	41.1290532	300	108122087.64967800
dataset/inst-13.tsp	1	6	uniform	scramble	bestandsecond	39.04016502	300	110972564.16671400

Instance:	Run number:	Configuration	Crossover	Mutation	Selection	Execution time	Iteration	Best Solution
dataset/inst-16.tsp	1	1	uniform	reciprocal	random	25.70941187	300	106900871.49935600
dataset/inst-16.tsp	1	2	cycle	scramble	random	33.27995902	300	105279996.07507400
dataset/inst-16.tsp	1	3	uniform	reciprocal	roulette	25.35485764	300	107841249.18771200
dataset/inst-16.tsp	1	4	cycle	reciprocal	roulette	29.09927478	300	110331843.85526400
dataset/inst-16.tsp	1	5	cycle	scramble	roulette	33.81068942	300	106572108.79572100
dataset/inst-16.tsp	1	6	uniform	scramble	bestandsecond	30.46313326	300	102245003.37092800

Figure 31 All Instances with Population Size to set to 30 Results