Metaheuristic Optimisation

Assignment 1

Michael McAleer – R00143621

23rd October 2018

Part 1 – NP-Completeness

1. This problem concerns the proof of NP-Completeness of 3COL
   a. Convert the formula F into a 3COL graph

      $F = (Z_1 \lor \neg Z_2) \land (Z_1 \lor Z_2 \lor \neg Z_3 \lor Z_4)$

   b. Find a solution for the 3COL instance of F and verify that it is a solution for F

Solutions

A) Convert the formula F into a 3COL graph

Starting with the formula *'$F = (Z_1 \lor \neg Z_2) \land (Z_1 \lor Z_2 \lor \neg Z_3 \lor Z_4)$'* the first step is to break it down into its constituent parts for reduction from SAT to 3SAT. Breaking F into two parts for reduction results in the following clauses:

i. $(Z_1 \lor \neg Z_2)$
ii. $(Z_1 \lor Z_2 \lor \neg Z_3 \lor Z_4)$

With the first clause $(Z_1 \lor \neg Z_2)$ there are two variables present so only one additional variable will need to be added so the reduction finishes with exactly three literals. This addition of one variable will result in the first clause being replaced with the following two clauses, where X_1 is the newly added variable:

i. $C_1 = (Z_1 \lor \neg Z_2 \lor X_1)$
ii. $C_2 = (Z_1 \lor \neg Z_2 \lor \neg X_1)$

Adding the variable X_1 to the first clause results in $\neg X_1$ being the opposite additional variable in the second clause. These two clauses and one extra variable added to the first clause in the original formula F gives us the first part of the SAT to 3SAT reduction solution.

The second clause $(Z_1 \lor Z_2 \lor \neg Z_3 \lor Z_4)$ from the original formula F has four variables to start with, to convert from SAT to 3SAT only one additional variable is needed and the original clause split into two separate clauses. Below the original clause has been split into two separate clauses, with Y_1 being the new variable:

i. $C_1 = (Z_1 \lor Z_2 \lor Y_1)$
ii. $C_2 = (\neg Y_1 \lor \neg Z_3 \lor Z_4)$

Adding the variable Y_1 to the first clause results in being the opposite additional variable $\neg Y_1$ in the second clause in the reduction. With three literals in each clause no further reduction is required in the second clause in formula F.

With both clauses from the original formula now reduced to their 3SAT form with each clause containing exactly three literals no further reduction is required. Taking each of the clauses from the reduction process the original formula F can now be expressed as:

$F = (Z_1 \lor \neg Z_2 \lor X_1) \land (Z_1 \lor \neg Z_2 \lor \neg X_1) \land (Z_1 \lor Z_2 \lor Y_1) \land (\neg Y_1 \lor \neg Z_3 \lor Z_4)$

To prove the correctness of reduction and that the original formula F is satisfiable, the 3SAT reduction can be converted into a 3COL graph. Using the 3COL graph the aim is to prove each of the clauses produced from the reduction process can resolve as true with at least one true literal in it. The 3SAT reduction has four clauses in it which need to be represented in the 3COL graph:

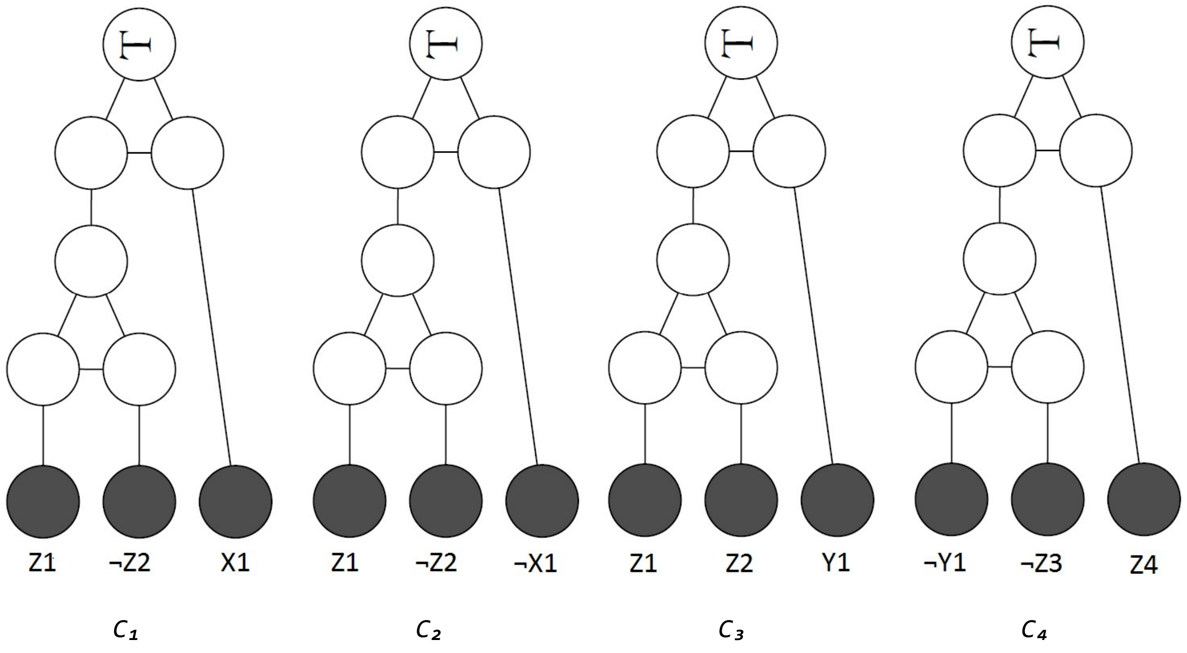
i. $C_1 = (Z_1 \lor \neg Z_2 \lor X_1)$

ii. $C_2 = (Z_1 \lor \neg Z_2 \lor \neg X_1)$

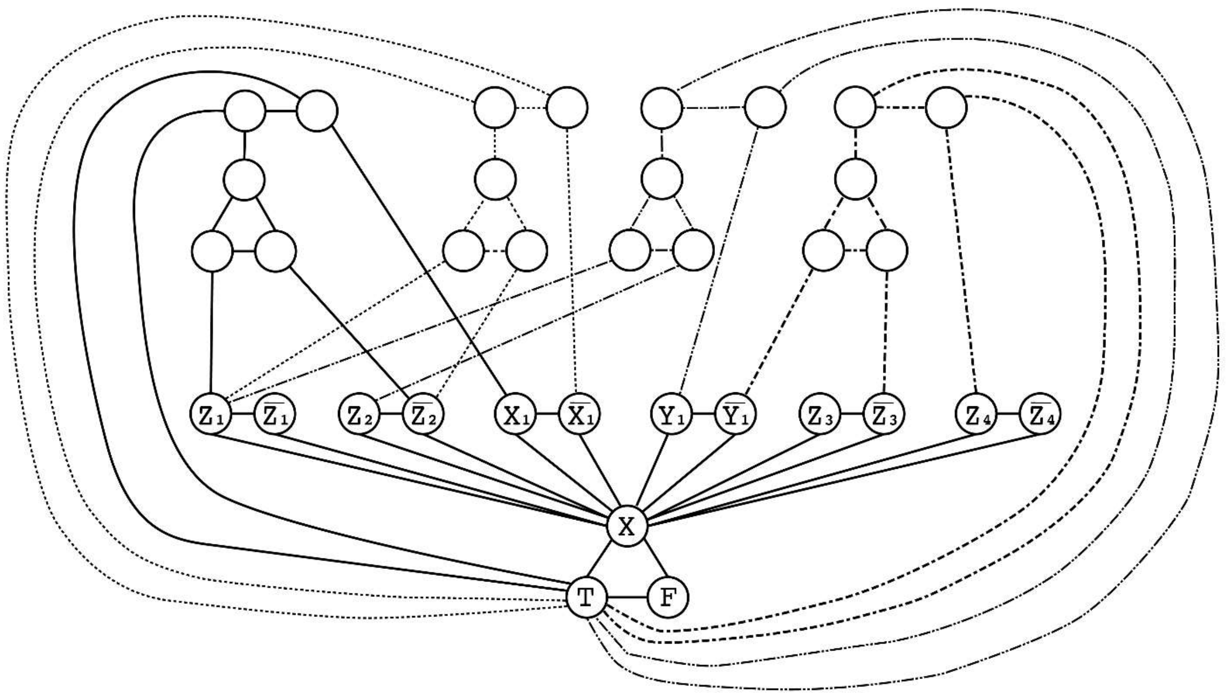
iii. $C_3 = (Z_1 \lor Z_2 \lor Y_1)$

iv. $C_4 = (\neg Y_1 \lor \neg Z_3 \lor Z_4)$

Using the standard template for converting the 3SAT clauses into constituent parts of the 3COL graph, each clause will resolve as follows:



With each part of the 3COL graph now constructed the sentences can be extracted to form the basis of the main 3COL graph; X_1, Y_1, Z_1, Z_2, Z_3, and Z_4. Taking each of the parts of the 3COL graph and putting them together so they all use the same graph components the following 3COL graph is produced, thus completing the SAT formula to 3COL graph conversion.



B) Find a solution for the 3COL instance of F and verify that it is a solution for F

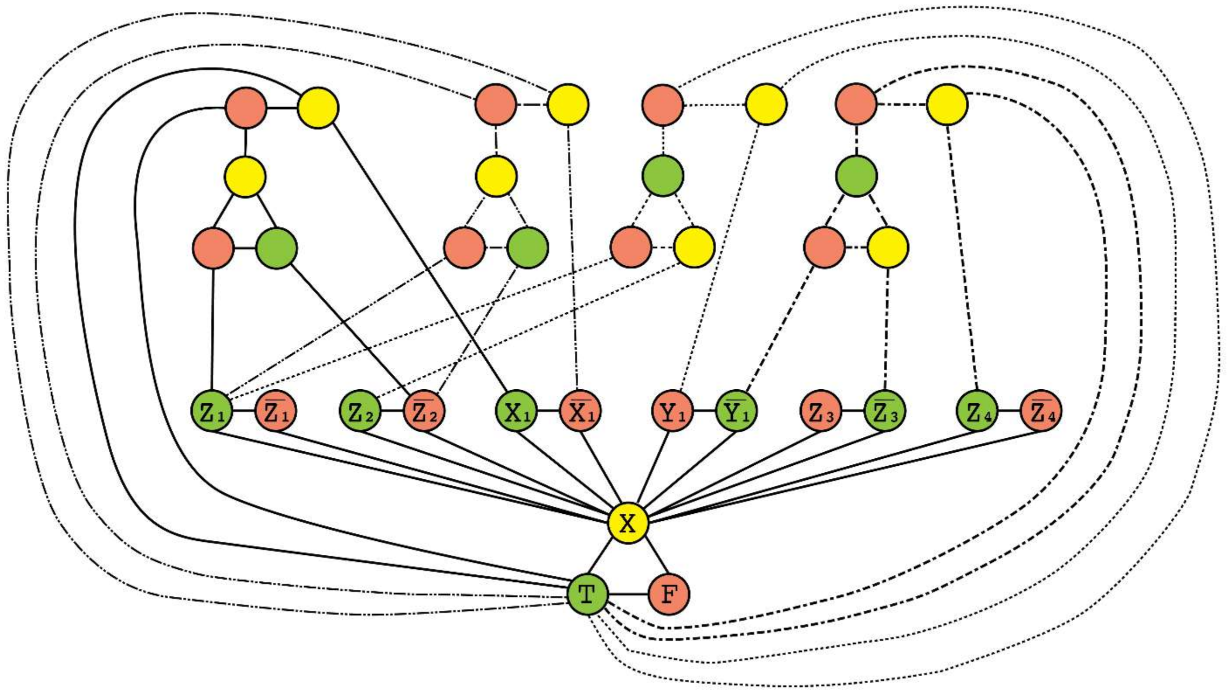
Using the 3SAT reduction formula a basis for colouring the 3COL graph can be found;

$$F = (Z_1 \lor \neg Z_2 \lor X_1) \land (Z_1 \lor \neg Z_2 \lor \neg X_1) \land (Z_1 \lor Z_2 \lor Y_1) \land (\neg Y_1 \lor \neg Z_3 \lor Z_4)$$

- 'Z_1 = True' makes C_1 & C_2 & C_3 resolve as True
- 'Y_1 = False' makes C_4 resolve as True

If it is assumed that in the colouring of the 3COL graph that both Z_1 and Y_1 are true, then the remainder of the variables can be coloured accordingly.

`



This verifies that the above 3COL graph is a valid solution for satisfying F. The graph has a valid 3-colouring, each literal gets a colour. Each clause gadget contains a true literal, so it can be coloured. Since each clause gadget is coloured properly, each must contain a true literal. There are no colours throughout the entire graph which has a neighbour of the same colour, and all can clauses resolve as True without touching same colour neighbour.

Following True/False designation from the 3COL graph for all variables results in 3SAT reduction formula resolving as True, so we can say that the 3SAT formula is satisfiable:

- X_1 = True
- Y_1 = False
- Z_1 = True
- Z_2 = True
- Z_3 = False
- Z_4 = True

`

$$F = (Z_1 \lor \neg Z_2 \lor X_1) \land (Z_1 \lor \neg Z_2 \lor \neg X_1) \land (Z_1 \lor Z_2 \lor Y_1) \land (\neg Y_1 \lor \neg Z_3 \lor Z_4)$$

$$F = (T \lor F \lor T) \land (T \lor F \lor F) \land (T \lor T \lor F) \land (T \lor T \lor T)$$

$$F = T \land T \land T \land T$$

$$F = T$$

Going backwards from our 3SAT formula to the original SAT formula F, using the variables from the 3SAT formula it is also possible to prove that the original SAT formula is satisfiable. If we say that is Z_1 true, then both clauses will resolve to true.

$$F = (Z_1 \lor \neg Z_2) \land (Z_1 \lor Z_2 \lor \neg Z_3 \lor Z_4)$$

$$F = (T \lor F) \land (T \lor T \lor T \lor T)$$

$$F = T \land T$$

$$F = T$$

Part 2 – Genetic Algorithms Solutions

1. Implementation of Genetic algorithms with the crossover and mutation operations (*note: for full code annotations and commenting please see source file, code here has been reduced to functional code only*):

a. Uniform order-based crossover

```
def uniformCrossover(chromosome_a, chromosome_b):
    """
    Uniform Order-Based Crossover implementation.
    """
    child_a, child_b = ([None] * chromosome_a.genSize,
                        [None] * chromosome_b.genSize)
    random_index_list = random.sample(range(0, chromosome_a.genSize - 1),
                                      int(round(chromosome_a.genSize) / 2))

    for i in random_index_list:
        child_a[i], child_b[i] = (chromosome_a.genes[i],
                                  chromosome_b.genes[i])

    def _ordered_crossover(child, target_chromosome):
        """
        Nested function which handles the assignment of remaining genes
        from opposite parent.
        """
        genes_for_child = [gene for gene in target_chromosome.genes if
                           gene not in child]
        for k in range(0, len(child)):
            if child[k] is None and genes_for_child:
                child[k] = genes_for_child[0]
                genes_for_child.pop(0)
        new_individual = target_chromosome.copy()
        new_individual.genes = child
        return new_individual

    return _ordered_crossover(child_a, chromosome_b), _ordered_crossover(
        child_b, chromosome_a)
```

b. Cycle crossover

```
def cycleCrossover(chromosome_a, chromosome_b):
    """
    Cycle Crossover Implementation.
    """
    child_a, child_b = ([None] * chromosome_a.genSize,
                        [None] * chromosome_b.genSize)

    def _run_cycle(chrom_a, chrom_b, alt_cycle):
        """
        Nested funtion to control the cycle selection and assignment
        process.
        """
        index = 0
        start_value = chrom_a[0]
        next_value = None
```

`

```
        while start_value != next_value:
            if None in child_a:
                if child_a[index]:
                    index += 1
                if not child_a[index]:
                    if alt_cycle:
                        child_a[index] = chrom_b[index]
                        child_b[index] = chrom_a[index]
                    else:
                        child_a[index] = chrom_a[index]
                        child_b[index] = chrom_b[index]
                    if chrom_b[index] == start_value:
                        index = 0
                        next_value = start_value
                    else:
                        index = chrom_a.index(chrom_b[index])
            else:
                break

    alternate_cycle = False
    while None in child_a:
        _run_cycle(chromosome_a.genes, chromosome_b.genes,
                   alternate_cycle)
        alternate_cycle = True if not alternate_cycle else False

    new_individual_a = chromosome_a.copy()
    new_individual_b = chromosome_a.copy()
    new_individual_a.genes = child_a
    new_individual_b.genes = child_b

    return new_individual_a, new_individual_b
```

c. Reciprocal exchange mutation

```
def reciprocalExchangeMutation(individual, mutationRate):
    """
    Reciprocal Exchange Mutation implementation
    """
    if random.random() > mutationRate:
        return

    mutate_index = random.sample(range(0, len(individual.genes) - 1), 2)
    gene_a, gene_b = (individual.genes[mutate_index[0]],
                      individual.genes[mutate_index[1]])
    individual.genes[mutate_index[0]] = gene_b
    individual.genes[mutate_index[1]] = gene_a
```

d. Scramble mutation

```
def scrambleMutation(individual, mutationRate):
    """
    Scramble Mutation implementation
    """
    if random.random() > mutationRate:
        return
    r_list = random.sample(range(0, individual.genSize), 2)
    r_list.sort()
```

`

```
    chromosome_genes = individual.genes
    list_pt1 = chromosome_genes[:r_list[0]]
    scramble_section = chromosome_genes[r_list[0]:r_list[1] + 1]
    list_pt2 = chromosome_genes[r_list[1] + 1:]

    random.shuffle(scramble_section)
    list_pt1 += scramble_section + list_pt2
    individual.genes = list_pt1
```

e. Additional (Roulette selection)

```
def rouletteWheel():
    """
    Roulette Wheel Selection with minimisaton focus.
    """
    max_fitness = self.worst.getFitness()
    fitness_sum = sum([((max_fitness + 1) - ind.fitness) for ind in
                       self.matingPool])
    probability_offset = 0
    roulette_pick = random.uniform(0, 1)

    for ind in self.matingPool:
        ind.probability = (
                probability_offset + (
                (max_fitness + 1) - ind.fitness) / fitness_sum)
        if ind.probability > roulette_pick:
            return ind
        else:
            probability_offset += (((max_fitness + 1) - ind.fitness) /
                                   fitness_sum)
```

2. A basic evaluation describing the two basic configurations:

Configuration	***Initial Solution***	***Crossover***	***Mutation***	***Selection***
1	Random	Uniform	Reciprocal Exchange	Random
2	Random	Cycle	Scramble	Random

When evaluating the performance of the various configurations of the implemented Genetic Algorithm a number of performance metrics were used for both high and low level results. Details of each iteration were saved along with details of each run, each configuration was run three times, each run with 300 iterations. At a high level, the following metrics were captured per configuration run:

- Total run time (seconds)
- Average run time per iteration (seconds)
- Lowest (best) fitness
- Highest (worst) fitness

`

- Fitness range
- Fitness mean
- Fitness median
- Fitness improvement likelihood % ((number of improvements % amount of iterations) x 100)

The last metric captured, the fitness improvement likelihood percentage, along with the lowest (best) fitness and mean fitness are considered as the key performance indicators for this genetic algorithm (GA) implementation. At a low level, the following metrics were captured per iteration:

- Lowest (best) fitness
- Highest (worst) fitness
- Fitness mean
- Fitness median

Both configurations in this basic evaluation work off the same initial population solution and selection method, whereby the initial population is constructed through randomisation of their genes to produce a population size of 100 individuals, each with a chromosome consisting of its genes (TSP solution) and respective fitness value. Using the initial population, two parents are randomly selected from the mating pool and used for crossover and mutation operations resulting in children whose genes are constructed from the operations performed on the parents.

Whilst these configurations are very similar in their function, both yielded very different results. Firstly the time taken per run and per iteration (*note: these runs were performed on the same computer on a single core/single thread program*).

Run #	Total Run Time (s)	Avg. run time per iteration (s)
1	77.60989809	0.25869966
2	77.08276677	0.256942556
3	77.27846527	0.257594884

Table 1 – Configuration 1 & inst-0.tsp

`

Run #	Total Run Time (s)	Avg. run time per iteration (s)
1	61.33458495	0.204448617
2	61.81899858	0.206063329
3	62.11594081	0.207053136

Table 2 – Configuration 2 & inst-0.tsp

Tables 1 & 2 both show the runtimes involved for running the GA against the problem file inst-0.tsp for both configurations 1 & 2. Both configurations have the same population size, chromosome size, and operation workflow (selection/crossover/mutation/repeat), so the time difference comes in to effect with the operators in use. Both configurations were consistent almost to the second in their performance across 300 iterations and 3 runs. It is assumed that the additional time taken to run configuration 1 is due to the uniform order based crossover operation, namely the two *for* loops within. Although efficiently used, having a *for* loop within any operator will decrease overall performance as all items in a list must be iteratively processed and some kind of processing performed. The second configuration whilst it makes use of a loop, it uses a *while* loop which can be used more effectively in terms of steps required to complete the crossover operation.

When determining if a GA mates effective use of the time taken, it is best to judge the time taken to run the program to completion against the best fitness value achieved and the likelihood of a fitness improvement being made. A GA, no matter how fast it may run, if it does not provide a satisfactory fitness solution for then it can be deemed a waste of time and/or computing power.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	23145261.31	27636064.39	4490803.083	25266078.45	25254747.55	0
2	23072590.39	27658595.12	4586004.73	25262423.42	25257299.6	0.333333333
3	22874648.95	27644625.89	4769976.938	25274754.6	25262524.38	1.333333333

Table 3 –Configuration 1 high level performance results against inst-0.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	23961365.16	31409517.06	7448151.9	30282919.86	31048622.01	0
2	23639061.45	31579298.51	7940237.061	30546196.17	31176947.77	0
3	23522333.06	31229009.57	7706676.505	30139491.99	30853384.45	0

Table 4 – Configuration 2 high level performance results against inst-0.tsp

From the high level results of configuration 1 demonstrated in table 3 and the comparison with the high level results of configuration 2 in table 4, it can be deduced that configuration 1 is significantly better at finding an optimal solution for the TSP. Although both configuration has a very low fitness improvement likelihood, the mean fitness along with the fitness range is a lot better than what is seen in configuration 2. It should also be pointed out that both configurations started with similar minimum fitness values, so the resulting fitness metrics are entirely as a result of the crossover and mutation operator in use. If we look at the results from another test file with double the amount of genes an individual's chromosome, we can see the behaviour is the same as what is seen when running the GA against inst-0.tsp. Configuration 1 runs result in much better fitness mean, media, range, and max value. As the fitness improvement likelihood is effectively 0% across both configurations the fitness minimum cannot be attributed to the performance of the GA, but instead random luck in the initial population randomisation solutions.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	115777133.7	144459382.2	28682248.49	130032378.8	129936441.2	0.333333333
2	115475646.5	144364712.1	28889065.64	129968734.7	129898052.3	0
3	116717924.6	144418039.3	27700114.66	130127232.1	130022191.2	0

Table 5 – Configuration 1 high level performance results against inst-13.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	120485804.2	176151353.1	55665548.95	167353258.9	175258639.3	0
2	118845182.8	174578062.1	55732879.35	167409730	173148771.1	0
3	120213021.8	168909454.6	48696432.84	161370988.1	165254509.2	0

Table 6 – Configuration 2 high level performance results against inst-13.tsp

`

In addition to comparing GA performance results at a high level, it is also possible to compare configuration performance from iteration to iteration and determine if any patterns can be detected across runs. From looking at the line graph results of both configuration 1 and 2 in Figures 1 and 2, there are some very surprising results displayed. Configuration 1 has a uniform line graph, whereby the lowest/best fitness values fluctuates along with the mean fitness value, but they remain consistent across the entire run of iterations and across runs. Configuration 2 appears to get worse as the iterations progress through a run, with each run exhibiting the same trend whereby both the best fitness value and mean fitness value get further away from an optimal TSP solution around iteration 70 in each run. After iteration 70 is reached the best fitness value achieved and mean fitness value fluctuate quite significantly.

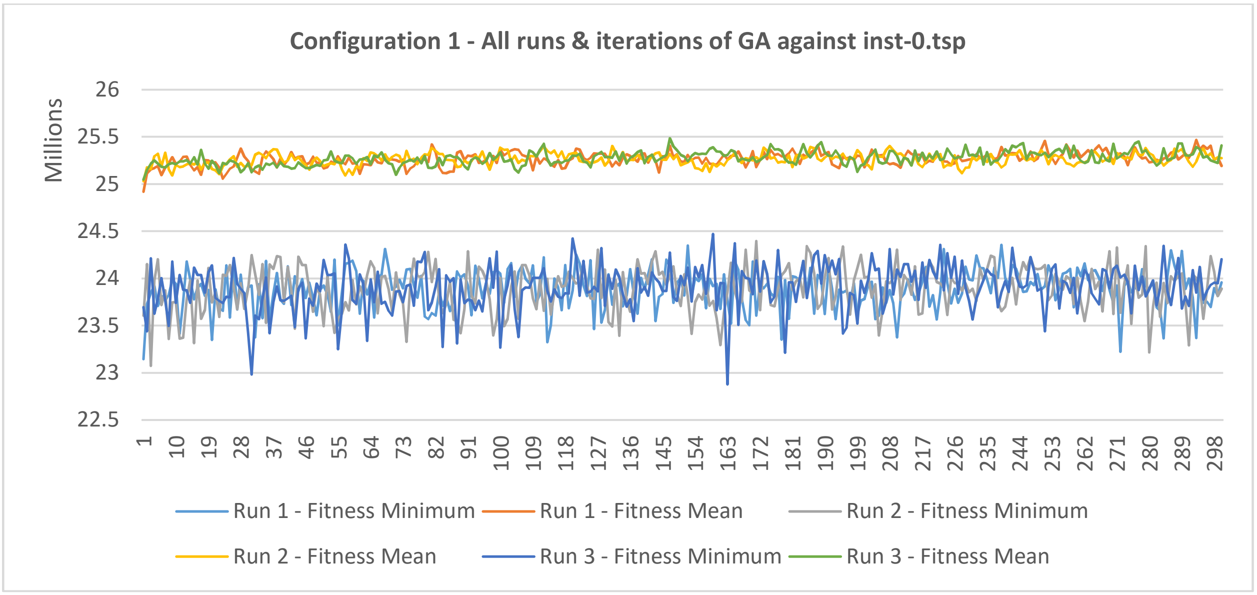


Figure 1 – Configuration 1 all runs & iterations: GA best & mean fitness value against inst-0.tsp

`

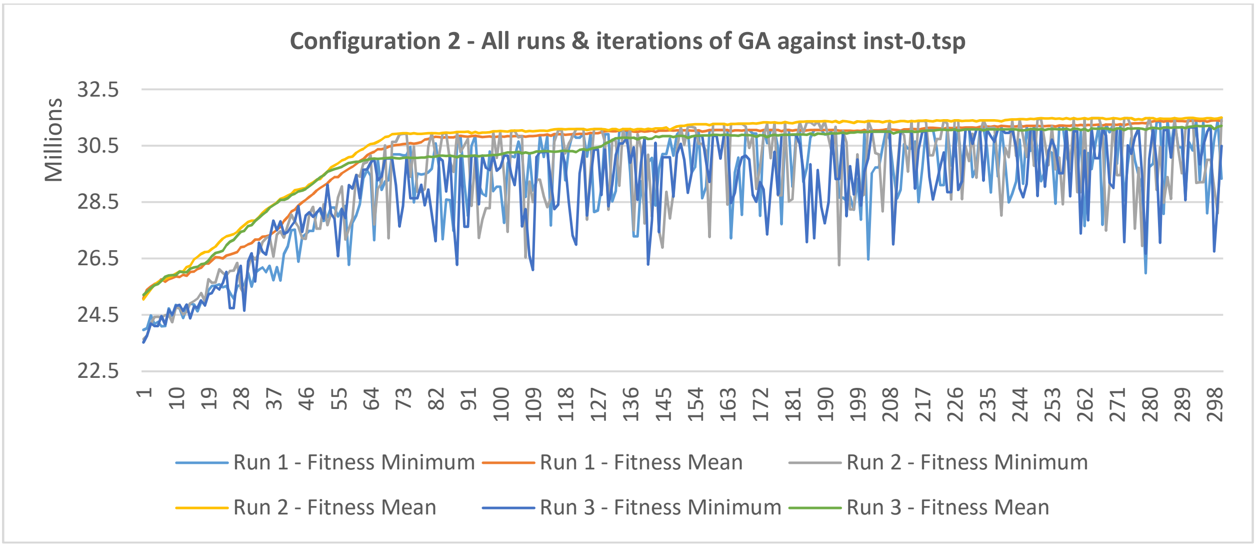


Figure 2 – Configuration 2 all runs & iterations: GA best & mean fitness value against inst-0.tsp

As the findings from configuration 2 were the opposite from what was expected, results were analysed from the other two datasets the configuration was run against (figures 3 & 4). The results of configuration 2 across all three data sets were consistent with each other, all exhibited behaviour which showed a steady increase in fitness value until approx. 70 iterations in to the run before levelling out and returning a wide range of fitness values poorer than what was started with.

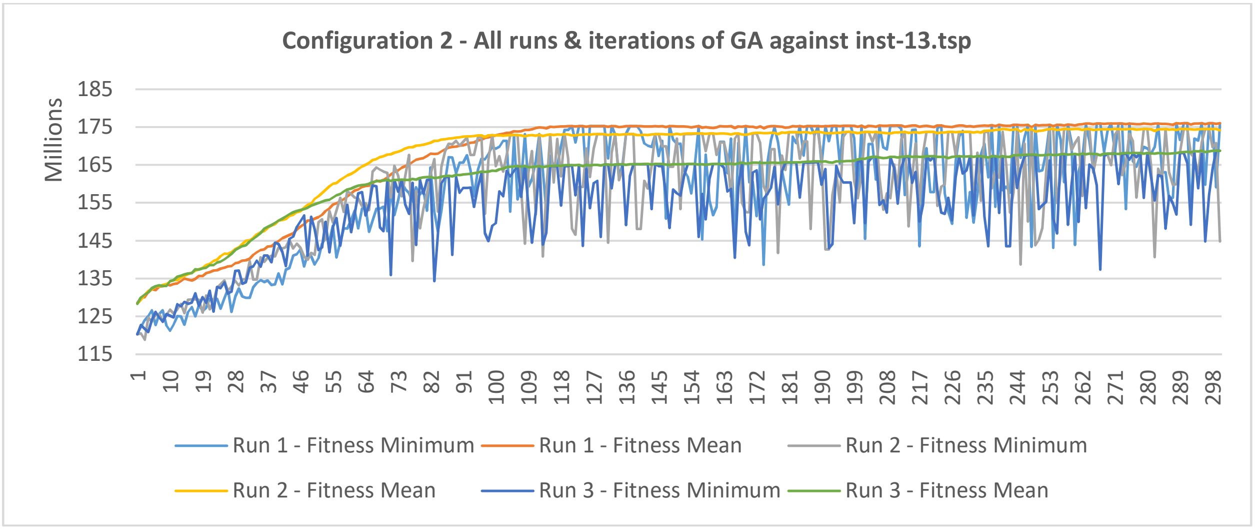


Figure 3 – Configuration 2 all runs & iterations: GA best & mean fitness value against inst-13.tsp

`

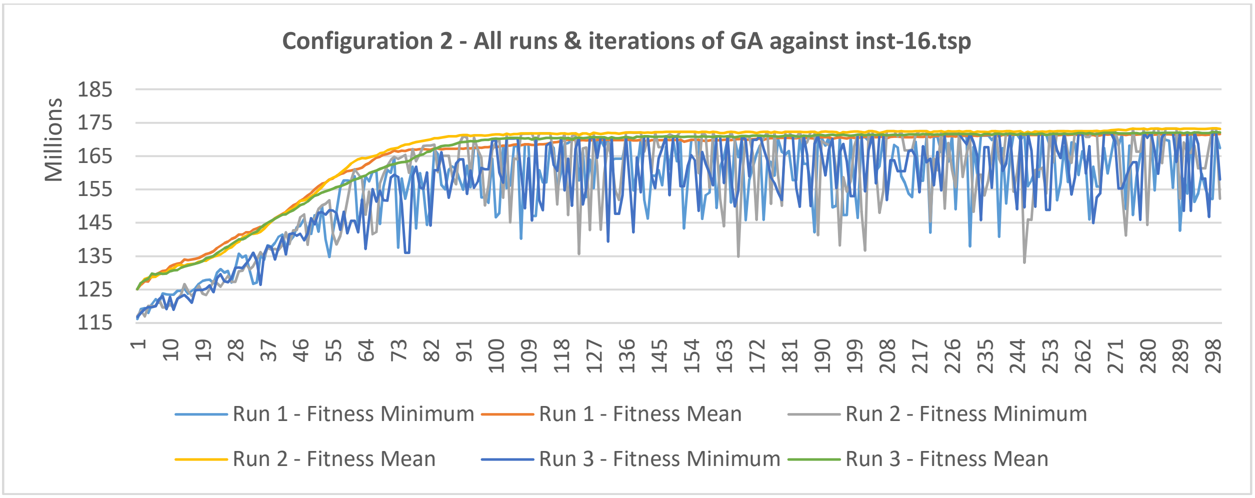


Figure 4 – Configuration 2 all runs & iterations: GA best & mean fitness value against inst-16.tsp

From the results obtained from the two initial configurations, it is apparent that configuration 1 is the stronger of the two configurations in terms of maintaining a consistent fitness value across all runs and iterations. Whilst this suits our minimisation focus on fitness values, it does fall short in that it does not provide a satisfactory amount of fitness improvements across multiple iterations and runs. If configuration 1 was to be implemented against a more computationally demanding dataset with more cities, and the runs contained more iterations, it would have to be determined if configuration 1 is worthwhile. The best (lowest) fitness values were found when randomisation techniques were used to initialise the starting population, rather than through crossover and mutation operations. As there was no facility built in to the configuration to keep the best X% of individuals, each run could be seen as a fresh run with all new individuals. If the top 10% of individuals by fitness value were kept from the previous run, these best individuals could be better used to influence a new run of individuals to getting even better fitness values. With initial selection being completely random, if the initial population contains vast amounts of poor fitness individuals, it does not create a great point from which to start to find optimal fitness values.

`

3. **Extensive evaluation of your GA, i.e., initial population, crossover & mutation operators. You are expected to extensively evaluate the following combination of operations:**

Configuration	*Initial Solution*	*Crossover*	*Mutation*	*Selection*
3	Random	Uniform	Reciprocal Exchange	Roulette Wheel
4	Random	Cycle	Reciprocal Exchange	Roulette Wheel
5	Random	Cycle	Scramble	Roulette Wheel
6	Random	Uniform	Scramble	Best/2nd Best

Starting with configuration 3, a combination of roulette wheel selection, uniform order-based crossover, and reciprocal exchange mutation were carried out over 3 runs of 300 iterations each. The high-level results from each dataset run through configuration 3 can be seen in tables 7, 8, and 9. Overall there does is not a lot of improvement from the initial configurations where random selection was in effect, but this could be due to initial population and no retention of best solutions between runs. In effect every run is a blank slate with no improvements kept for the next run to incorporate. The fitness minimum and maximum values are consistent across runs with each dataset, this indicates a relatively good attempt at obtaining optimal fitness values but with no retention of best solutions there is no chance of evolution of solutions between runs to further improve on results.

Run #	**Fitness Minimum**	**Fitness Max**	**Fitness Range**	**Fitness Mean**	**Fitness Median**	**Fitness Improvement Likelihood %**
1	22417625.84	27081759.31	4664133.47	24587684.93	24527981.27	0.666666667
2	22923653.19	27272518.34	4348865.147	25049434.39	25049562.45	0
3	22874793.71	27459901.04	4585107.337	25062850.66	25049449.52	0.333333333

Table 7 – Configuration 3 high level performance results against inst-0.tsp

Run #	**Fitness Minimum**	**Fitness Max**	**Fitness Range**	**Fitness Mean**	**Fitness Median**	**Fitness Improvement Likelihood %**
1	112864161.7	144146686.6	31282524.87	128173147.4	128107799.2	0
2	115064909.8	145943652.7	30878742.86	128970781.2	128997850.4	0.666666667
3	112810538.2	143501695.3	30691157.07	129144712.6	129062730.9	0

Table 8 – Configuration 3 high level performance results against inst-13.tsp

`

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	109537209.2	140709778.4	31172569.24	124909788.8	124873172.1	1
2	110579315.5	140905862.8	30326547.35	125551738.7	125508091.7	0
3	107488392.6	138469223.6	30980830.95	122726127.3	122348476.4	1.333333333

Table 9 – Configuration 3 high level performance results against inst-16.tsp

If we look at the high-level results of configuration 4, a direct comparison with configuration 3 can be made to gauge the impact of using uniform order-based crossover or cycle crossover. From the findings of configurations 1 and 2, cycle crossover was deemed to be better suited to maximisation problems, whereby a higher fitness value is desirable. This behaviour can be seen again in configuration 4 in tables 10, 11, and 12. In contrast with configuration 3, configuration 4 returns significantly higher max fitness values, larger fitness ranges, and higher fitness mean and media values. The fitness minimum values present in configuration 4 cannot be attributed to the selection/crossover/mutation process as the likelihood of improvements is sitting mostly at 0%, with one run against inst-0.tsp returning one fitness improvement and a likelihood of 0.33% for improving the fitness. At a high level it does not look like either configuration 3 or 4 are efficient at solving the TSP problem with the GAs implemented up until this point. Roulette wheel selection has improved the likelihood of improvements but only by the smallest margins for uniform order-based crossover, it still cannot be seen as a major improvement over random selection.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	23212056.06	27623890.69	4411834.633	26481757.71	26629257.9	0.333333333
2	23336290.71	31250306.07	7914015.358	28002129.23	27659868.91	0
3	23509948.39	27658659.43	4148711.031	26415286.98	26526005.96	0

Table 10 – Configuration 4 high level performance results against inst-0.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	114992899.1	152917446.8	37924547.71	136155975	134498259.4	0
2	119949515.1	149503070.7	29553555.54	138884030.4	139251381.6	0
3	120025267.1	167968076.5	47942809.39	148382311.6	148830802	0

Table 11 – Configuration 4 high level performance results against inst-13.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	115553594.8	141787255.2	26233660.34	133883493.9	134525606.2	0
2	115452749.3	163140857.6	47688108.27	142195511	134001667.7	0
3	114642163.2	145036355.6	30394192.43	133618415.8	133555292.5	0

Table 12 – Configuration 4 high level performance results against inst-16.tsp

Going down a level and looking at the GA performance from iteration to iteration in figure 5, a lot of similarities can be draw between the trend patterns of configuration 1 and 3. Each iteration across all three runs of the GA configuration maintain a trend of achieving an optimal range of fitness values, but not improving across multiple runs. This can be attributed to no elite survival across runs again, each new run results in an entirely new set of chromosomes and no solutions from previous runs kept in the population from which to use as parents for new children.

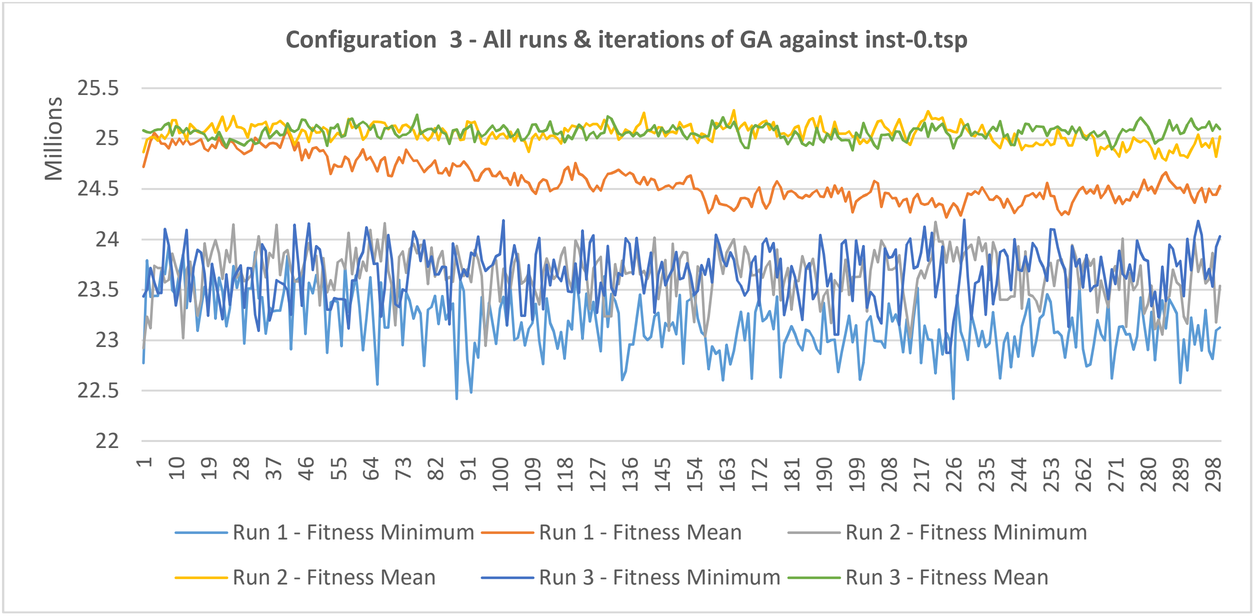


Figure 5 – Configuration 3 all runs & iterations: GA best & mean fitness value against inst-0.tsp

Comparing the fitness minimum and mean values for configurations 1 and 3 allows for insights into the impact of using roulette wheel selection over random selection. Figure 6 shows the difference in GA performance of configuration 1 and configuration 3 in terms of lowest fitness value and mean fitness per

`

iteration. The impact is not huge over 3 runs of 300 iterations, but an improvement can be seen in terms of consistently lower fitness and lower fitness mean values overall for configuration 3. With more time and experimentation this could be analysed further to determine if the downward trend would continue and provide an optimal solution after X amount of iterations.

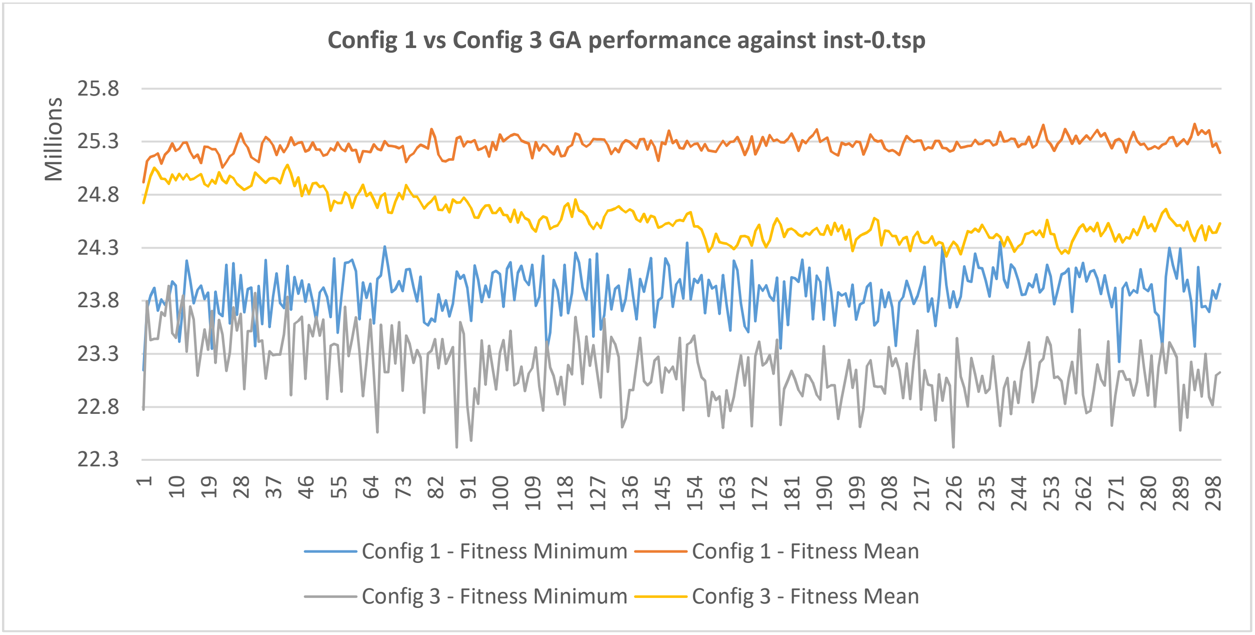


Figure 6 – Configuration 1 vs. configuration 3: GA best & mean fitness value against inst-0.tsp

The trend pattern displayed in figure 7 by configuration 4 is similar to what is seen from configuration 2 with cycle crossover in use. Whilst low fitness values are encountered at the start of a run, the behaviour of the GA results in an increasing lower fitness value and increasing fitness mean value across all iterations and runs. A significant increase in mean fitness value can be observed around the iteration 130 mark in figure 7, however this can be attributed to the increase in the maximum fitness value to 31250306.07 and range increasing to 7914015.358, so it may be another example of the impact of a poor initial population to start with and no retention of best solutions from previous runs.

`

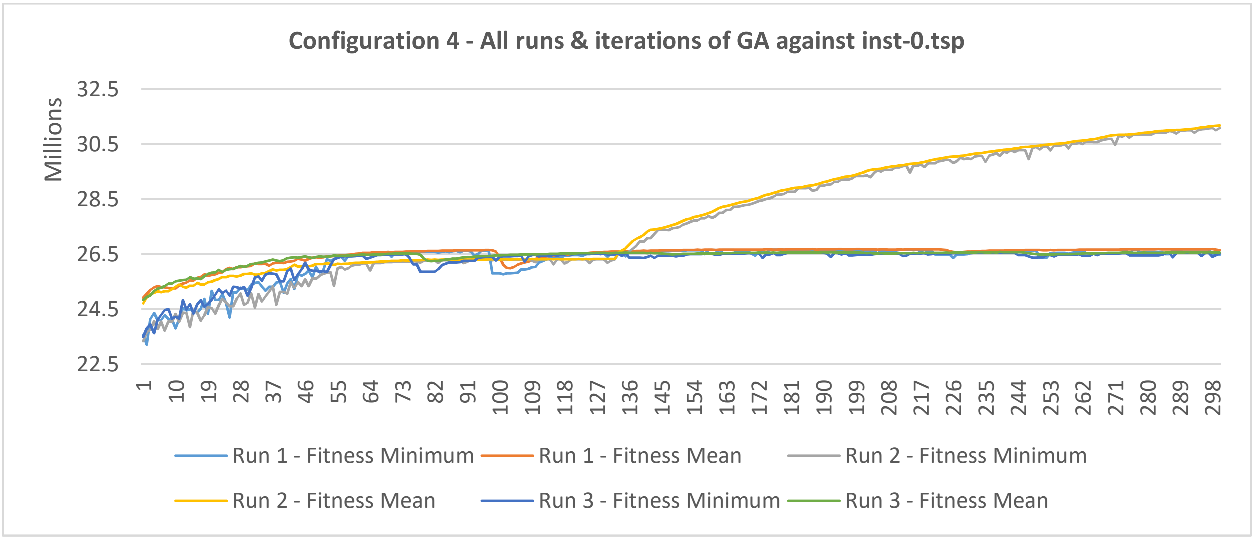


Figure 7 – Configuration 4 all runs & iterations: GA best & mean fitness value against inst-0.tsp

Looking at the trend pattern of configuration 4 against dataset inst-13.tsp (figure 8) it is evident that the behaviour is similar across both datasets, figure 8 shows a dramatic increase in lowest fitness value and mean fitness value. So whilst it can be deduced that this configuration does not lend itself well to GA minimisation algorithms, a large part of this can be attributed to the possibility of a poor initial population containing individuals with large fitness values. Even with roulette wheel selection implemented it does not negate the effect of poor individual selection.

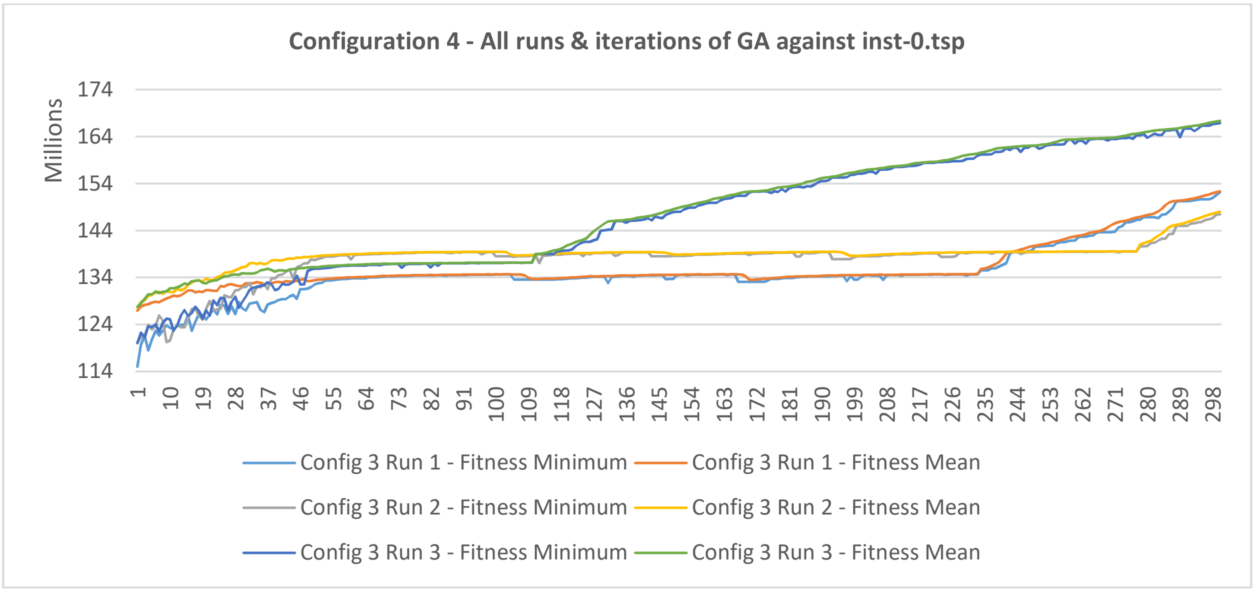


Figure 6 – Configuration 3 all runs & iterations: GA best & mean fitness value against inst-13.tsp

`

Moving on to configurations 5, the impact of scramble mutation in use with both cycle crossover can be analysed. With configuration 5 results can be seen which are in line with what is seen with other configurations with cycle crossover implemented, the chances of fitness value improvement are 0% across 3 runs of 300 iterations. A more max fitness value and fitness range can be observed across all three data sets so from a high level it can be said that the scramble mutation operator adds stability to fitness values.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	23394738.65	27441842.88	4047104.234	25560992.97	25562523.93	0
2	23331627.73	27228552.61	3896924.879	25899666.94	25964785.87	0
3	23538844.59	27337999.91	3799155.319	26108566.15	26214565.25	0

Table 13 – Configuration 5 high level performance results against inst-0.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	116501927	143013284	26511356.95	132189576.1	132344934.8	0
2	120076976.1	144904028.6	24827052.53	132576668.8	132631524.6	0
3	117908847.2	145131406.6	27222559.38	134180519	134372010.2	0

Table 14 – Configuration 5 high level performance results against inst-13.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	114358495.2	142661404.5	28302909.3	129579877.9	130132153.6	0
2	111445388.3	143142018.8	31696630.45	130511246.8	130661716.8	0
3	114884776	141346530.5	26461754.53	128562517.5	128590150.6	0

Table 15 – Configuration 5 high level performance results against inst-16.tsp

For a clearer picture of the performance of configuration 5 it is necessary to look at the lower level performance metrics. Figures 7 and 8 offer a different insight into the impact of alternating configuration make up, the previous trend patterns where cycle crossover is concerned all point towards a general increase in fitness values. Both figures show that when using cycle crossover with scramble mutation the fitness value has the opportunity to go down as well as up, this is evident in the peaks and troughs in the mean fitness values across all three runs. There are also some notable drops in lowest fitness value throughout each run, this would indicate better suitability of this configuration for minimisation focus on GA

`

fitness solutions, if these solutions could be maintained across runs then greater improvements in fitness solutions could be expected. It should also be noted that throughout each of the runs and iterations for all data sets using configuration 5, even though the trend pattern is closer to what is intended, as in improvements in the lowest fitness values seen from iteration to iteration, none of the runs using configuration 5 resulted in an improvement in the original lowest/best fitness value at the start of each run.

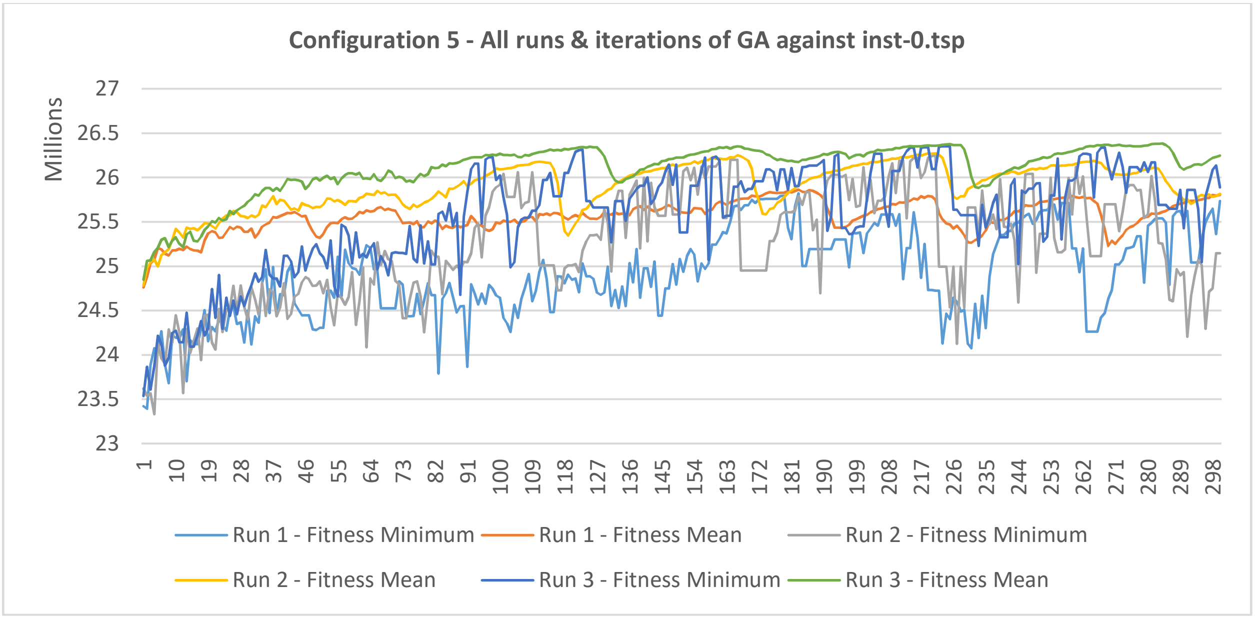


Figure 7 – Configuration 5 all runs & iterations: GA best & mean fitness value against inst-0.tsp

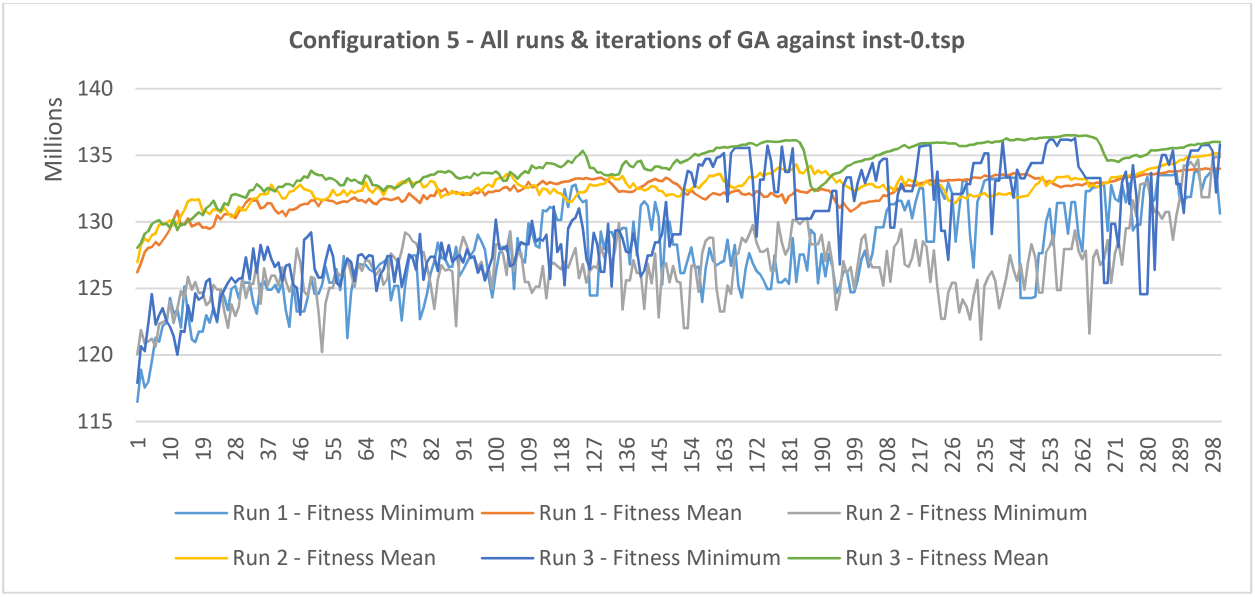


Figure 8 – Configuration 5 all runs & iterations: GA best & mean fitness value against inst-13.tsp

`

Configuration 6 can be seen to have the most impressive results in terms of fitness improvements. Looking at tables 16, 17 and 18, from the first run the improvement likelihood percentage is multiple times higher than any other configuration tested prior, with some runs getting into double digit percentages. Considering there is no elite survival between runs and all improvements are made on randomly constructed chromosomes of individuals, the improvements made are very satisfactory. However there is a major caveat with configuration 6, the selection process selects only the best and second best solutions from the mating pool for each run, this in effect renders the rest of the population useless as they are not used for any parent to child genetic processing.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	21297982.98	26569586.55	5271603.572	22517834.58	21483078.93	7
2	19819735.79	26760822.49	6941086.703	20592828.14	19950919.01	10.66666667
3	21158106.78	26862953.24	5704846.46	22066224.27	21175059.94	6.333333333

Table 16 – Configuration 6 high level performance results against inst-0.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	107587212.8	141266625.7	33679412.95	119543012.2	123212288	6
2	93527602.05	137429893.4	43902291.37	100326605	93535648.38	12.66666667
3	110604913.8	140176598.2	29571684.35	123426263.1	123338692.2	1.333333333

Table 17 – Configuration 6 high level performance results against inst-13.tsp

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	96435385.32	135400048	38964662.65	99075587.16	96438760.11	6.666666667
2	94875881.26	134628545.1	39752663.87	99702191.89	94887618.04	11
3	99067351.49	134883206.2	35815854.74	110695672.6	111203326.2	6

Table 18 – Configuration 6 high level performance results against inst-16.tsp

In addition, using the same parents for each and every iteration in a run increases the likelihood of hitting an limit on the amount of improvements that can be made without either dramatically increasing the mutation value. For example, all cycle crossover operations using this selection method would result in the exact same children for each iteration, the only variation would come from the mutation operator. This is evident in figure 9, it can be seen from the graph that across all runs, there reaches a point in the run where the improvement rate levels out and very few, or none at all, improvements are made to the fitness value. In order to increase the likelihood of further improvements it would be necessary to further mutate the individuals to return different variations of solutions. Using the same parents for each and every iteration also defeats the purpose of genetic algorithms, the intention of this genetic algorithm is to simulate biological procedures and ensure survival of the fittest. Whilst it could be argued that the best parents will produce the best children in most cases, this does not allow for variation or diversity in the population and so the GA ends up reaching a point where improvements are not as frequent.

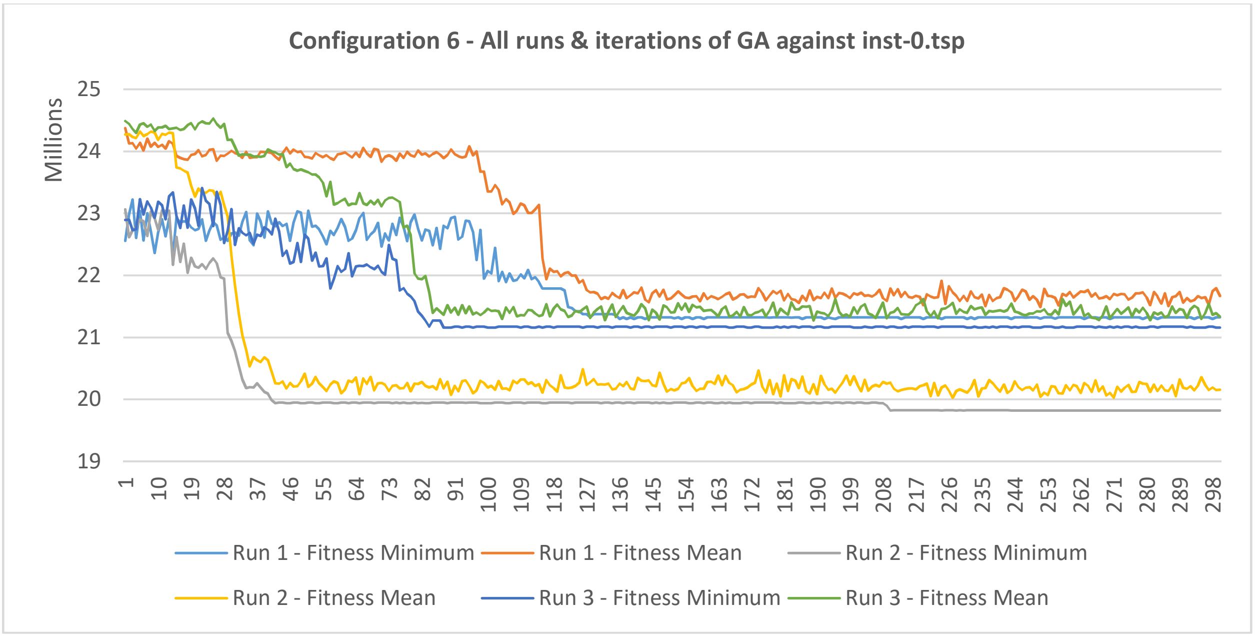


Figure 9 – Configuration 6 all runs & iterations: GA best & mean fitness value against inst-0.tsp

4. **Investigate the impact of varying the population size, explore different mutation rates, and evaluate the impact of your GA with and without elite survival.**

As a basis for investigating the impact of varying parameters and evaluating their impact on GA performance, configurations 3 and 6 are used as they display the most promising results in terms of fitness improvement likelihood and for improvements to the best fitness solution available. Ideally, a combination of variations across multiple parameters will yield higher fitness likelihood percentages and better fitness scores.

The first parameter under evaluation for impact on GA performance is mutation rate. Mutation rates are set by specifying a value between 0 and 1, this dictates how often a child will be mutated in the course of an iteration, if the mutation rate is set to 1.0 mutation will occur in every iteration, if it is set to 0.5 it should occur 50% of the time. As the mutation rate has been tested conclusively at 0.1 across all configurations, additional rates of 0.5 and 1.0 were tested. Tables 19, 20, and 21 show the impact of mutation rate variations on a successful GA configuration. The tables show that whilst mutation can have a positive impact on fitness improvements, mutating all children is counter-productive and results in lower fitness improvement likelihood percentage. This tells us that whilst it is good to have variation, it is important to balance inheritance from suitable parents with the variation mutation provides.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	21297982.98	26569586.55	5271603.572	22517834.58	21483078.93	7
2	19819735.79	26760822.49	6941086.703	20592828.14	19950919.01	10.66666667
3	21158106.78	26862953.24	5704846.46	22066224.27	21175059.94	6.333333333

Table 19 – Configuration 6 high level performance results against inst-0.tsp with mutation rate of 0.1

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	19741703.84	26677167.17	6935463.331	21787031.88	21109679.02	14.66666667
2	19149553.36	26643134.26	7493580.904	21696547.04	20868872.28	17.33333333
3	18975212.38	26371021.16	7395808.778	21538128.7	20971710.49	26.33333333

Table 20 – Configuration 6 high level performance results against inst-0.tsp with mutation rate of 0.5

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	21664873.98	27003237.5	5338363.516	24072917.53	24027127.07	3
2	21096405.21	26531543.66	5435138.448	23735031.4	23870424.81	5
3	22074182.5	26700585.77	4626403.272	24216633.38	24174853.63	3.666666667

Table 21 – Configuration 6 high level performance results against inst-0.tsp with mutation rate of 1.0

`

Looking at the impact of the mutation rate on each run and the iterations of each run in figure 10, there is a visible impact of the likelihood of fitness improvements later into the run. When comparing the trend patterns of figure 9 with a mutation rate of 0.1 against figure 10 with a mutation rate of 0.5, improvements to the fitness solution are seen more consistently across iterations instead of a flood of improvements at the start of a run and none in later iterations. With a mutation rate of 0.5 in use, having the same parents for each iteration becomes less of an issue as the mutation operation varies the children enough to add an acceptable level of variation against limited inheritance variation.

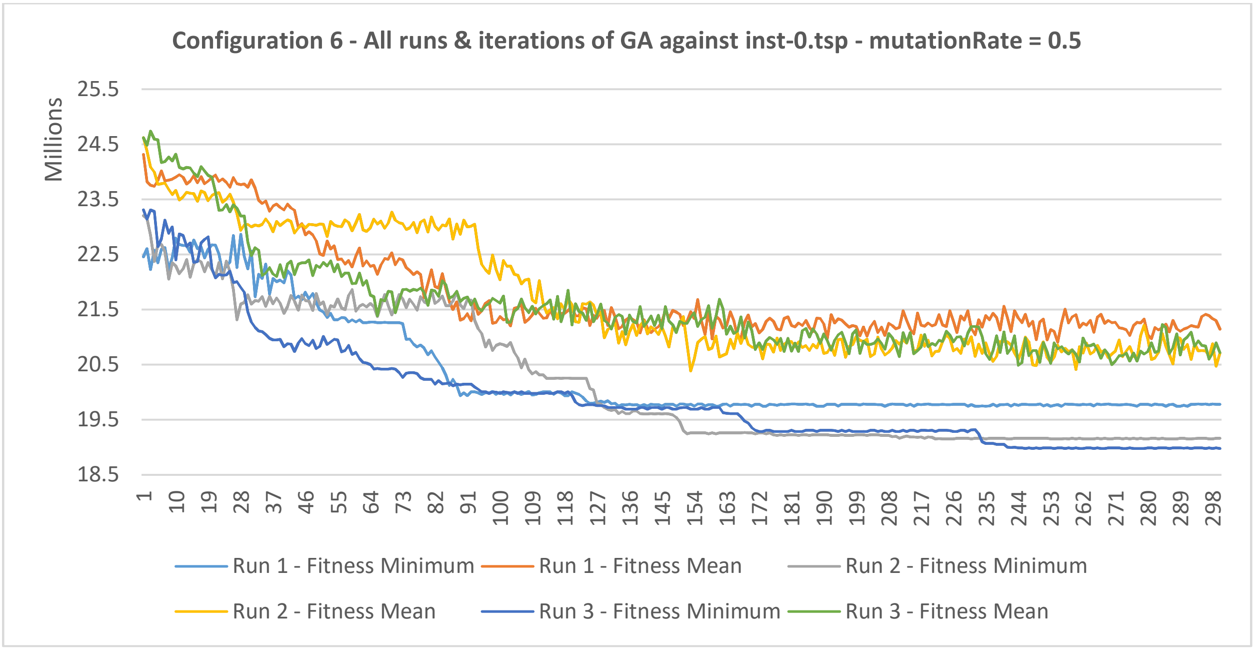


Figure 10 – Configuration 6 all runs & iterations: GA best & mean fitness value against inst-0.tsp with mutation rate of 0.5

Another parameter that remained constant throughout the testing of all previous configurations was the population size. In all previous configuration tests the population size was fixed to 100 individuals, to test the impact of population size this value was increased to 300 and run against configuration 3. Although configuration 6 gives the best results, the selection method would render any change to the population inconsequential as only the two best parents from the mating pool are selected and used for the entire run across all iterations.

`

The first noticeable impact to increasing the population size is the dramatic increase in time taken to complete each iteration and all runs. This increase is a result of the implementation of loops in the GA, even though each iteration only makes use of two parents at a time, processing on the entire population must be regularly carried out so this has meant between run times of almost 4 times as long even though the population size is 3 times larger.

Run #	Total Run Time (s)	Avg. run time per iteration (s)
1	69.33545	0.231118
2	69.38232	0.231274
3	69.88239	0.232941

Table 22 - *Configuration 3 run time information for inst-0.tsp with population size of 100*

Run #	Total Run Time (s)	Avg. run time per iteration (s)
1	265.8860185	0.886286728
2	277.3388302	0.924462767
3	293.0290973	0.976763658

Table 23 - *Configuration 3 run time information for inst-0.tsp with population size of 300*

In terms of improvements to the performance of the GA, altering the population size has not had any impact on the likelihood of fitness improvements or the best/mean fitness values. Tables 24 and 25 show the differences on GA performance when population sizes are changed. There is no discernible difference in the best fitness value see, or the max, range, mean or media values. There is a negligible reduction in the fitness improvement likelihood but this could be attributed to a better initial population when only using 100.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	22417625.84	27081759.31	4664133.47	24587684.93	24527981.27	0.666666667
2	22923653.19	27272518.34	4348865.147	25049434.39	25049562.45	0
3	22874793.71	27459901.04	4585107.337	25062850.66	25049449.52	0.333333333

Table 24 - Configuration 3 high level performance results for inst-0.tsp with population size of 100

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	22626188.61	27577266.34	4951077.738	25062619.12	25056328.26	0
2	22872505.74	28071017.48	5198511.741	25046455.88	25037995.67	0
3	22633179.93	27466757.98	4833578.052	25027862.33	25021749.74	0.666666667

Table 25 - Configuration 3 high level performance results for inst-0.tsp with population size of 300

Figure 10 further strengthens the findings of no impact to performance of the GA in terms of fitness values when looking at the results from run to run and iteration to iteration. The results found with a population size of 300 are the same as those in figure 5 with a population size of 100. There is a consistent trend across all iterations where the best fitness solution varies within a set range and the fitness media remains consistent throughout.

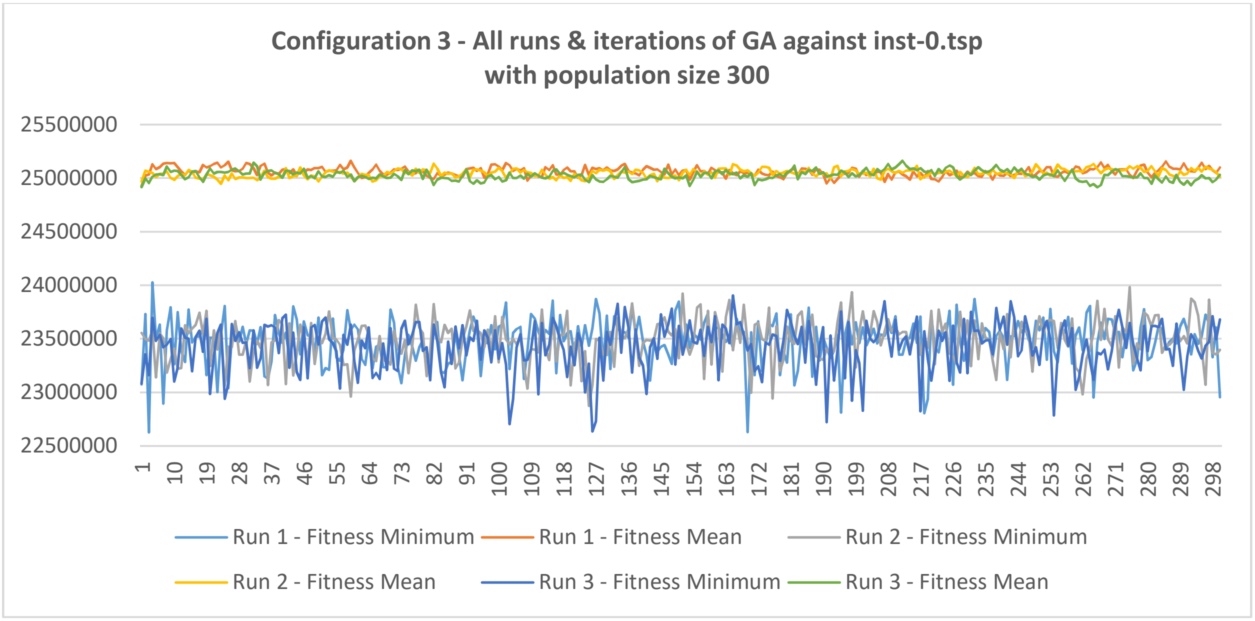


Figure 10 – Configuration 3 all runs & iterations: GA best & mean fitness value against inst-0.tsp with population size of 300

The last variation to GA implementation that as tested for impact is the concept of 'elite survival'. Elite survival is the process of taking the best X% of the old population and placing them into the new population. What percentage of the old population to maintain is entirely down to the GA implemented and the desired

`

outcome of the GA. In this case where there are a lot of individuals and a lot of genes in each individual's chromosome it may be considered acceptable to keep 20% of the best solutions from a previous run.

Sticking with configuration 6, elite survival was implemented with a percentage rate of 20% between runs. Whilst it appears at first glance in table 26 that the performance of the GA has decreased with elite survival included, the advantages are seen between runs. The improvements to the fitness solution in run 1 is carried on to run 2, and from run 2 the improvements are carried on to run 3. This behaviour results in the GA always working on the best possible solution given all the previous runs and iterations, it won't start again when a new population is initialised, it will try to improve on the current best known solution.

Run #	**Fitness Minimum**	**Fitness Max**	**Fitness Range**	**Fitness Mean**	**Fitness Median**	**Fitness Improvement Likelihood %**
1	20118981.09	26271793.11	6152812.021	20923928.55	20149357	12.33333333
2	19996011.7	25749023.61	5753011.911	20382726.39	20123707.8	0.666666667
3	19996011.7	25972674.05	5976662.347	20333020.68	20027154.87	0

Table 26 - Configuration 6 high level performance results for inst-0.tsp with 20% elite survival

Figure 11 shows the impact of elite survival on configuration 6, whilst the improvements are great for the first run, the improvements for the second and third runs do not proceed like what was seen without elite survival. This behaviour can also be attributed to the selection method and mutation rate of 0.1, with some altering of parameters and selection method for configuration 6 there could be substantial improvements made to the GA performance.

`

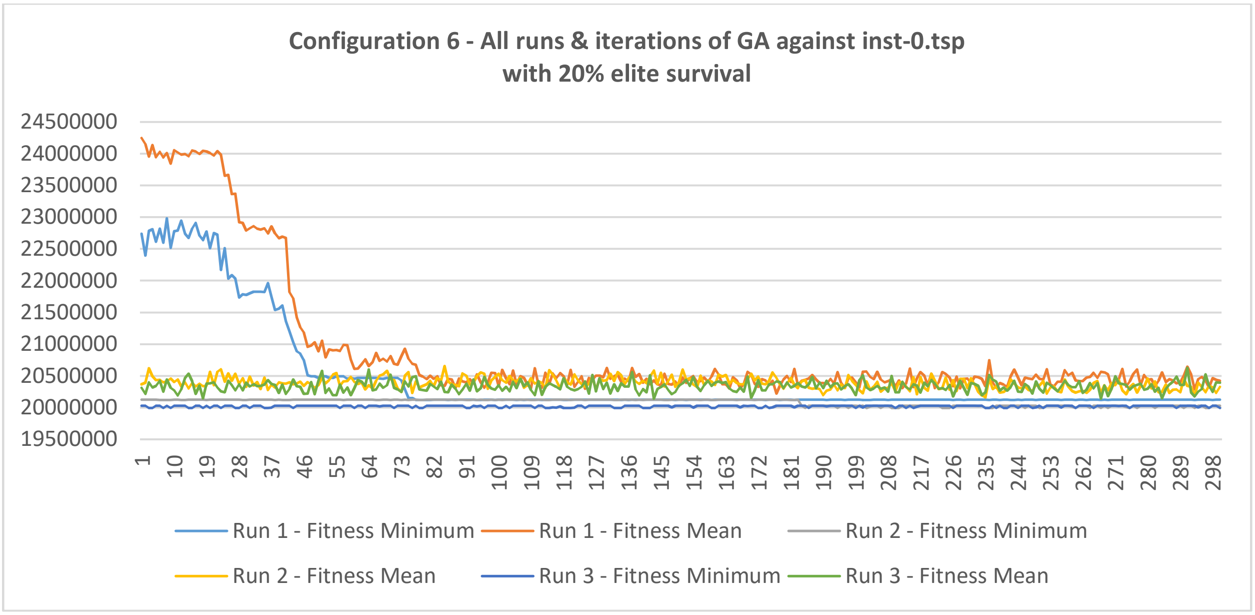


Figure 11 – Configuration 6 all runs & iterations: GA best & mean fitness value against inst-0.tsp with 20% elite survival

5. **Construct comprehensive description of the algorithms and an extensive evaluation of your results. And it should describe the experimental design, what experiments are and what they are intended to show. Use tables and figures where appropriate.**

During the course of running the various configurations and altering parameters within such as population size, mutation rates, and implementing elite survival, it became apparent that none of the configurations were ideal, but that an ideal solution existed with a combination of varying selection/crossover/mutation operators along with fine tuning GA variables like mutation rates and elite survival.

Before evaluating the entire GA and various configurations, a seventh configuration was implemented which made use of multiple aspects of different configurations. This configuration consisted of:

- Random initial population
- Best choice parent A / roulette wheel parent B
- Mutation rate of 0.5
- 20% elite survival
- 500 iterations / 10 runs

`

The population size was not altered as a population size of 100 is considered satisfactory for the current GA and TSP problem. The aim with the above configuration is to get the fitness improvement likelihood of configuration 6, but negate the effect of best/second best selection by varying the second choice using roulette selection. The mutation rate of 0.5 should negate the impact of inheritance issues by using the same parent for each iteration in a single run, and 20% elite survival should keep the improvements to the TSP solution continuing throughout the multiple runs.

After running configuration 7 with the above settings, a considerable improvement over all previous configurations could be seen, the results are presented in table 27. The best fitness solution was approximately 40 million lower than that found by configuration 6, configuration 7 resulted in a best solution of ~159 million, configuration 6 had a best solution of ~199million. Altering the mutation rate along with selecting the second parent for each iteration from the mating pool by roulette wheel selection had the desired effect on the fitness improvement likelihood percentage. Unlike configuration 6 and the tests with elite survival, the improvements to the fitness solution did not reduce to none after one or two runs, improvements could be seen right through until the 10th run.

Run #	Fitness Minimum	Fitness Max	Fitness Range	Fitness Mean	Fitness Median	Fitness Improvement Likelihood %
1	18558352.76	26252650.33	7694297.565	20686755.87	19959014.65	15.8
2	17795936.96	26288573.62	8492636.658	19384616.95	18743731.77	3.4
3	17629987.05	25820891.7	8190904.652	18966081.3	18223990.61	0.8
4	17349893.5	26002531.17	8652637.677	18855390.37	18149845.82	1.6
5	17178654.23	26032950.18	8854295.949	18568592.15	17822258.15	1.6
6	17005736.05	25979683.08	8973947.035	18449824.99	17728539.45	1.4
7	16957256.89	25774981.29	8817724.4	18361543.17	17612533.61	1
8	16733526.72	26038181.54	9304654.822	18183970.73	17428288.86	2.6
9	16418467.47	25861875.13	9443407.663	17952187.66	17191454.44	2.2
10	15940993.89	25846262.79	9905268.897	17681217.64	16915357.23	2.8

Table 27 - Configuration 7 high level performance results for inst-0.tsp

Figure 12 gives insights into the 10 runs carried out by configuration 7 and the 500 iterations of each. With each run there is a clear improvement in fitness solution, with the best solution for a given run being the

`

starting best solution for the next run. This behaviour is ideal as any improvements found apply to the whole GA process and not just on a per-run basis. Given more runs and iterations it is not unfeasible to see further improvements made to the best fitness solution, but with the trend patterns 'flattening' from run to run it can be said that it is likely that it will become more difficult to find better solutions as the runs progress.

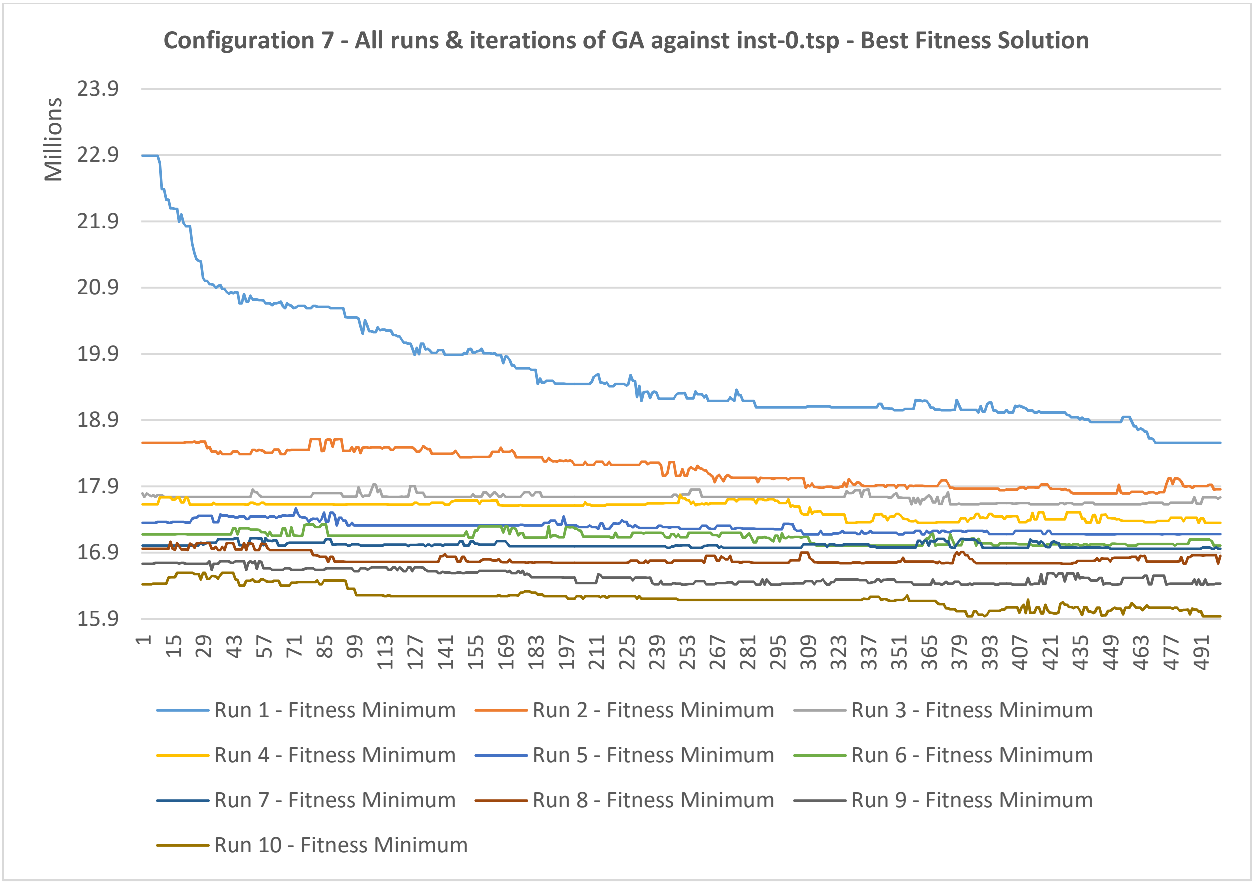


Figure 12 – Configuration 7 all runs & iterations: GA best fitness value against inst-0.tsp

A lot has been learned from the process of building and implementing these algorithms, then running them and fine-tuning to get the desired results, or as close to the desired results, as possible. There are a number of improvements that could be made to this GA in particular to make it perform more efficiently and produce better results.

The first improvement that could be made to this GA is to implement it as a multi-threaded program where all logical cores of the system it is running on are in use. Implementing it in this would also allow for the GA

`

to be further improved upon to use cloud computing sources to spread the processing load across multiple machines or containers.

Another major improvement that should be made to this GA implementation for the TSP solution is to start with a more suitable initial population. Whilst it could be seen throughout all of the configurations and runs against various datasets, the best solution found was often the randomly generated solution in the initial population. This may be satisfactory for this implementation of a GA, but in a real-world scenario a more suitable initial population would be required. A nearest neighbour insertion algorithm would be better suited to this GA as it would ensure a good first attempt at solving the problem which could then be improved upon through various runs and iterations of the GA. The importance of having good parents to start with cannot be understated, having poor parents will increase the amount of time required to get satisfactory solutions from a GA.

The selection methods in use in the 6 configurations were found to be unsuitable for this use-case, this may have been more noticeable because the initial population was randomly generated. With a better initial selection method, such as best selection & roulette wheel selection combined, returned much better results across all iterations and runs. The plan for a GA is to simulate biological procedures and imitate the real-world process of survival of the fittest, so it is important to balance inheritance, variation, and competition without hindering the performance of the GA too much. Implementing best/second best selection whilst provided very good results across few runs, may be viewed as not entirely in line with the objective of a genetic algorithm. It was found that very quickly the results were stifled because of a lack of variation in the parents. Increasing the mutation value from the default 0.1 was necessary to reduce the impact of having the same parents in use for an entire run. It is also worth noting that configuration 7, whilst found to be the most powerful solution, it may not be suitable for all optimisation tasks. It works for the TSP and the random nature of the initial population, but the improvements to the best solution were occurring less and less as the runs progressed. This may also be the case for the cycle crossover operation, it was found to be unsuitable for the GA TSP solution as it seemed to be more effective at producing maximum fitness values instead of minimum fitness values. Due to the random nature of the initial population the poor performance

`

of the cycle crossover operator may be due to not having good parents, however it is not possible to confirm or deny this without further and more robust testing on this operator in isolation and with different configurations.

From running through the various configurations and altering the GA variables to impact the functionality of a respective GA configuration, it was surprising to see that changing a certain value too much would have detrimental effects on the GA performance. Variables such as pool size, iterations, mutation rate, run count, and elite survival can be altered from run to run to increase or decrease the performance of a GA. The impact of these variables can only be evaluated for this particular GA implementation, it is not to assume that their behaviour in this GA will be reflected in other GAs in the same manner. A perfect example of this is mutation rate, it was found that the mutation rate was better at 0.5 (50% mutation rate) than 1.0 (100% mutation rate). Before running the various configurations and altering mutation rate the writer was under the impression that a higher mutation rate is always better so the variation in children is greater, thus providing more chances of finding an optimal solution. Instead what was found that 50% mutation rate was better than 100%, the exact reasons are unknown as more testing would be required, but from a high level it could be assumed that given two good parents, if mutation occurs every time it may take a good child solution and mutate it into something worse more often than it mutates into something better. However, as already stated, this behaviour could be isolated to this GA and different behaviour could be seen from a different GA implementation.

Elite survival had a very powerful impact on the performance of the GA configurations when implemented, however it was of not performing at its best when using a selection method such as best/second best. Elite survival worked best in this scenario when only one of the parents was selected by best fitness and the other was via randomisation or selection probability with roulette wheel selection. By keeping the top x% of a previous run and inserting it into the new population built for the next run, the performance of the GA was always focused on improving the best known solution across all runs. Without elite survival enabled the GA would regularly find a best solution in one run, then start the next run with a much worse solution and not get anywhere near the initial best solution. This kind of behaviour is counter-productive for the TSP

`

GA implementation as it can be considered a waste of processing power and time. When these kind of problems are known for their complexity and time required to solve, it is better to keep the best solution from run to run so as not to repeat fitness calculations for substandard solutions.

Another observation made during the course of implementing this GA is the suitability of the roulette wheel selection for scenarios where there are a lot of individuals where the fitness values are very large and there is very little between them in comparison. For example, a difference of 3 million may seem like a vast number, but when there are 100 individuals with fitness values of 21-26 million, it makes the selection probabilities for roulette wheel selection almost inconsequential. The selection process may as well be random, when the individuals' probabilities are all calculated with probability offset included there is very little to differentiate the best solution from the worst solution in terms of selection probability. There were only three selection methods implemented in this GA with a fourth constructed by merging two other selection methods, so there may be other selection methods such as tournament selection which could be better suited to this GA implementation to provide the best parents possible for a given iteration.

All the configurations whilst very similar in their steps all yield completely different results, and altering a single step or altering a particular variable within a GA may further alter the performance of the fitness solutions. It is this point which affirms the assumption that GAs are very much a stand-alone solution, there is no one size fits all GA. They are very much constructed and fine-tuned to best fit the problem at hand, and it may take multiple runs and iterations before any of the best possible constituent parts of a GA; selection method, crossover operator, mutation operator, can be found.