

---

**An investigation into solving the travelling  
salesman problem using nature inspired  
techniques**

---

Sonas MacRae -  
40277542

Submitted in partial fulfilment of  
the requirements of Edinburgh Napier University  
for the Degree of  
BSc (Hons) Computing Science

School of Computing

April 8, 2021

## **Authorship Declaration**

I, Sonas MacRae, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

*Signed: Sonas MacRae*

*Date: 07/04/2020*

*Matriculation no: 40277542*

## **General Data Protection Regulation Declaration**

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

## **Abstract**

Optimisation problems have troubled computer scientists for decades, with the common over-appreciation of the capabilities of modern computing power, optimisation problems such as the travelling salesman problem are often over looked as an elementary problem. As you look into this fantastically complex problem you begin to realise the issues to overcome. Algorithms are used to produce routes, although as more nodes are added to the problem, the number of possible solutions grows exponentially. This paper reviews the effectiveness of two stochastic and one deterministic for this problem. The evolutionary algorithm was chosen due to its renowned success on optimisation problems. From this the ant colony optimisation algorithm was selected, keeping with the theme of nature inspired stochastic algorithms and the fact it was initially designed to solve the travelling salesman problem. To evaluate the performance of the stochastic algorithms, they were compared against a nearest neighbour paired with a 2-opt algorithm, to answer the question whether the time taken to run the stochastic algorithms will be made up by the potential improvement of performance. It was found that the time required to run the stochastic algorithms weren't relevant due to the fact that the nearest neighbour paired with the 2-opt algorithm returned the best results throughout the final tests. It was concluded that the results don't paint a clear picture of the usefulness of the algorithms since the parameter tuning wasn't as precise as it could be and the number of operators that were tested for the evolutionary algorithm could have been more. Future work on this project would include improving the algorithms and extending the testing period, running the algorithms more times on benchmark problems to gain a better insight into the reliability and potential of the algorithms.

## Contents

<b>1</b>	<b>Introduction</b>	<b>11</b>
1.1	Background . . . . .	11
1.2	Aims and objectives . . . . .	12
<b>2</b>	<b>Literature review</b>	<b>14</b>
2.1	The travelling salesman problem . . . . .	14
2.2	Benchmark problems . . . . .	15
2.3	Deterministic techniques for solving the travelling salesman problem . . . . .	15
2.3.1	Brute force algorithm . . . . .	15
2.3.2	2-opt algorithm . . . . .	16
2.3.3	Nearest neighbour algorithm . . . . .	16
2.4	Evolutionary algorithms . . . . .	18
2.4.1	Representation . . . . .	18
2.4.2	Fitness . . . . .	20
2.4.3	Population . . . . .	20
2.4.4	Parent selection . . . . .	20
2.4.5	Crossover . . . . .	21
2.4.6	Mutation . . . . .	21
2.4.7	Survivor selection . . . . .	22
2.5	A comparison of mutation methods . . . . .	22
2.5.1	Inversion mutation . . . . .	23
2.5.2	Displacement mutation . . . . .	23
2.5.3	Pairwise swap mutation . . . . .	23
2.5.4	Results . . . . .	24
2.6	A comparison of crossover methods . . . . .	24
2.6.1	Cycle crossover . . . . .	26
2.6.2	Partially mapped crossover . . . . .	27
2.6.3	Non-Wrapping Ordered Crossover (NWOX) . . . . .	27
2.6.4	Ordered Crossover . . . . .	28
2.6.5	Shuffle crossover . . . . .	28
2.6.6	results . . . . .	28
2.7	Ant colony optimisation . . . . .	29
2.7.1	How they work . . . . .	29
2.7.2	Applying an ant colony optimisation algorithm to bench- mark travelling salesman problems . . . . .	30
2.8	Summary . . . . .	31

<b>3</b>	<b>Implementation</b>	<b>32</b>
3.1	Version control . . . . .	33
3.2	Evaluating algorithms for the travelling salesman problem . .	34
3.3	Command line application design . . . . .	35
3.3.1	Generating the problem . . . . .	36
3.3.2	Changing the algorithm . . . . .	36
3.3.3	Settings . . . . .	36
3.3.4	Optimal tour . . . . .	38
3.3.5	Running the algorithm . . . . .	38
3.3.6	Help . . . . .	40
3.4	Writing the evolutionary algorithm . . . . .	40
3.4.1	Representation . . . . .	40
3.4.2	Fitness evaluation . . . . .	40
3.4.3	Selection . . . . .	41
3.4.4	Crossover . . . . .	41
3.4.5	Mutation . . . . .	42
3.5	Tuning the evolutionary algorithm . . . . .	42
3.5.1	Tuning the population size . . . . .	44
3.5.2	Tuning the probability of mutation . . . . .	44
3.5.3	Mutation operator comparisons . . . . .	45
3.5.4	Crossover operator comparisons . . . . .	45
3.6	Implementing the ant colony optimisation algorithm . . . . .	46
3.6.1	Count . . . . .	48
3.7	Tuning the ant colony optimisation algorithm . . . . .	48
3.7.1	Tuning Alpha . . . . .	48
3.7.2	Tuning Beta . . . . .	48
3.7.3	Tuning Rho . . . . .	49
3.7.4	Tuning Elite . . . . .	50
3.7.5	Tuning Q . . . . .	50
3.7.6	Tuning T . . . . .	50
3.7.7	Tuning Count . . . . .	52
3.8	Writing the nearest neighbour algorithm . . . . .	52
3.8.1	Applying 2-opt to the nearest neighbour algorithm . .	52
3.9	Testing the evolutionary algorithm . . . . .	53
3.10	Comparing the algorithms on the benchmark problems . . . .	53
3.10.1	Stochastic algorithm comparison . . . . .	53
3.10.2	Comparing the EA against the Nearest neighbour paired with 2-opt . . . . .	56

<b>4</b>	<b>Evaluation</b>	<b>57</b>
4.1	Has the Project met its Aims and Objectives? . . . . .	57
4.2	Future work . . . . .	58
4.3	Personal statement . . . . .	58
	<b>Appendices</b>	<b>62</b>
<b>A</b>	<b>Project Overview</b>	<b>62</b>
<b>B</b>	<b>Second Formal Review Output</b>	<b>62</b>
<b>C</b>	<b>Project management</b>	<b>62</b>

## List of Tables

1	Population size test results . . . . .	44
2	Mutation rate test results . . . . .	45
3	Comparing mutation operators test results . . . . .	45
4	Crossover comparison test results . . . . .	46
5	Alpha test results . . . . .	48
6	Beta test results . . . . .	49
7	Rho test results . . . . .	49
8	Elite test results . . . . .	50
9	Q test results . . . . .	51
10	T rate test results . . . . .	51
11	Count test results . . . . .	52
12	Evolutionary algorithm test results . . . . .	60
13	Nearest neighbour final results . . . . .	60
14	Nearest neighbour + 2opt final results . . . . .	60
15	Ant colony optimisation final results . . . . .	61
16	EA and ACO t test results . . . . .	61



## List of Figures

1	Nearest neighbour vs hybrid method results [Nuraiman et al., 2018]	17
2	Evolutionary algorithm cycle <a href="https://www.sicara.ai/blog/2017-08-29-was-darwin-great-computer-scientist">https://www.sicara.ai/blog/2017-08-29-was-darwin-great-computer-scientist</a> . . . .	19
3	Two point crossover example <a href="https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm">https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm</a> . . . . .	21
4	Inversion mutation . . . . .	23
5	Displacement mutation . . . . .	23
6	Pairwise mutation . . . . .	24
7	Mutation findings [Chieng and Wahid, 2014] . . . . .	25
8	Cycle crossover step-1 . . . . .	26
9	Cycle crossover step-2 . . . . .	27
10	Cycle crossover step-3 . . . . .	27
11	Cycle crossover step-4 . . . . .	27
12	[Dorigo and Gambardella, 1997] ACO results . . . . .	30
13	App main menu . . . . .	35
14	Generate problem menu . . . . .	36
15	Preset benchmark problems selection . . . . .	37
16	Updating the algorithm in use menu . . . . .	37
17	ACOPY settings menu . . . . .	38
18	kroD100 optimal route . . . . .	39
19	Calculating fitness . . . . .	41
20	Inversion mutation code . . . . .	43
21	Calculating fitness . . . . .	59
22	eil76 score over time . . . . .	62
23	Berlin52 score over time . . . . .	63
24	Berlin52 BoxPlot EA vs ACO . . . . .	64
25	Eil51 BoxPlot EA vs ACO . . . . .	64
26	kroA100 BoxPlot EA vs ACO . . . . .	65
27	att48 BoxPlot EA vs ACO . . . . .	65
28	bays29 BoxPlot EA vs ACO . . . . .	66
29	gr120 BoxPlot EA vs ACO . . . . .	66
30	kroD100 BoxPlot EA vs ACO . . . . .	67
31	eil76 BoxPlot EA vs ACO . . . . .	67
32	st70 BoxPlot EA vs ACO . . . . .	68
33	IPO - part 1 . . . . .	69
34	IPO - part 2 . . . . .	70
35	Week 9 report - part 2 . . . . .	71
36	Week 9 report - part 2 . . . . .	72

37	Project gantt chart . . . . .	74
----	-------------------------------	----

## **Acknowledgements**

I would like to thank Kevin Sim for being my supervisor, helping out with issues and offering support and encouragement to get the most out of this project. I would also like to thank Simon Powers, the second marker, for sharing his advice on writing papers and conducting experiments and for also taking this project on.

# 1 Introduction

## 1.1 Background

Routing problems are very relevant today, improving the search methods that find efficient routes for delivery drivers can save money with shorter routes requiring less labour and fuel costs, which as a result is better for the environment. The potential difficulty of a routing problem is often underappreciated, with exponential scaling it would be very easy to produce a problem with more solutions than the predicted number of atoms in the visible universe. The processing power required to solve such a problem with absolute certainty is far beyond modern computing and thus gives the licence for algorithms to play their part. Until an algorithm, or configuration of an already existing algorithm is capable of finding the optimal path for any routing problem within an acceptable time it could always be argued that algorithms can be optimised further. The purpose of this project is to explore techniques and compare their effectiveness at solving the travelling salesman problem. A literature review will be written up using information from established articles and books that have explored this problem before. An evolutionary algorithm, ant colony optimisation, nearest neighbour and a brute force technique will all be implemented into a command line application, all of which will be available to work on benchmark travelling salesman problems. Tests will be carried out where these algorithms will be applied to a host of benchmark problems, where possible these results will be compared to those gathered in the literature review.

This project will look at both deterministic and stochastic algorithms, and for that reason different tests will have to be configured for each. Stochastic algorithms are unpredictable, therefore they will require more rigorous testing than the deterministic methods implemented. Evolutionary and ant colony optimisation algorithms both have an array of parameters, these parameters will have to be tuned in attempt of achieving better results. In addition, evolutionary algorithms can be highly customised, this project will investigate crossover and mutation operators, with the remaining operators of the evolutionary algorithm being chosen using preliminary experience. Given the time assigned to complete this project, it isn't realistic to carry out extensive testing on every operator within the evolutionary algorithm, especially given the fact that an ant colony optimisation algorithm is to be implemented as well. Deterministic algorithms are those that return the same result every time on a given problem, because of this one test will have to be run for every deterministic algorithm.

## 1.2 Aims and objectives

### Aims

- To develop an app with algorithms which are strong at solving travelling salesman benchmark problems
- This application will also allow the tuning and testing of algorithms for the TSP
- To come to a conclusion regarding the usefulness of the algorithms for solving TSP problems
- To conclude whether or not the time taken to run stochastic algorithm is worth it, given they outperform the deterministic algorithm.

### Objectives

- Based on the literature review conclusion, certain algorithms will be studied and implemented.
- Design the app in such a way that the algorithm's parameters can be updated using the GUI. Implement features which allow the user to run a problem several times, the results gathered will be used to carry out statistical analysis, as a result efficient testing will be possible.
- Using the application, a series of tests will be conducted on each algorithm, the results will be graphed and displayed in tables, these results will be used to conclude the usefulness and accuracy of the algorithms.
- The scores obtained by the algorithms will be considered, if the stochastic algorithms fail to beat the deterministic method, the time taken is irrelevant. If this isn't the case then the time taken and the difference in scores will be discussed, there isn't a definitive answer since different scenarios offer resource restraints in the real world.

The main aim of this project is to gain an understanding of how the selected algorithms work, where possible investigating different configurations and how they perform on benchmark problems. The implementation will involve designing and coding a command-line application which will allow algorithms to be tested on benchmark problems, where automated statistical evaluation will be carried out and presented to the user at the end of the testing process. In addition to this, the user will be able to manually tune the parameters of the ant colony and evolutionary algorithm from the

user interface of the application, with the best recorded run per algorithm per problem recorded with their respective parameters. Another goal of this project is to compare the effectiveness of stochastic algorithms on benchmark problems against a deterministic method, where a conclusion will be drawn up detailing the usefulness of the stochastic methods in comparison to the results gathered using the deterministic algorithm.

This naturally leads to the research questions: Will the stochastic algorithms produce better results than the selected stochastic approach? If so, is the time required to run the stochastic algorithm made up by its improvement over the deterministic method?

## 2 Literature review

This section will review academic papers surrounding techniques tailored to solve the TSP. The outcome will present research questions, the better performing algorithms reviewed will be used for the implementation stage.

### 2.1 The travelling salesman problem

The travelling salesman problem presents the following, given a list of  $N$  locations, what is the most efficient path that visits each location exactly once before returning to the original location. “Researchers have recognized it as an NP-hard problem” [Shim et al., 2011], meaning it can’t be solved in polynomial time. The issue with this problem is how computationally expensive it becomes the more locations you add, if we represent a potential solution as an array of coordinates that each represent a location within a search domain, we can measure the size of this search domain as  $(n-1)!/2$  [Larrañaga et al., 1999]. This means the magnitude of the problems increase exponentially as you add more locations, and there comes a point in which modern computer architecture can’t definitively solve these large problems within a suitable time frame.

To return the best possible solution to a travelling salesman problem, we must generate every possible permutation of the co-ordinates and search through them to return the solution with the highest fitness. There are four main classes of algorithms that can offer answers to the travelling salesman problem, namely exact, heuristic, approximation, and meta heuristic approach. [Kizilates and Nuriyeva, 2013].

Exact methods include the brute force method, even though given enough time it will always return the most efficient route, its downfall is the fact that the time required to produce a solution scales exponentially as you add more nodes.

Heuristic approaches include the ant colony optimisation algorithms, which is covered in this project. A benefit of heuristic algorithms is that they have the ability of finding good enough solutions to large problems within a suitable time frame. Although they don’t always promise good results, the size and shape of the problem can have a big effect on this as some problems are more suitable to heuristic algorithms than others.

The nearest neighbour algorithm is an example of an approximation technique to solve the travelling salesman problem, this algorithm relies heavily

on the setup of the problem to produce good solutions. The nearest neighbour finds solutions relatively quickly compared to some stochastic techniques, although it is common for this algorithm to struggle on larger problems.

Meta heuristics are designed to select, generate or find a heuristic that may be able to find a good solution to an optimisation problem. If successful meta-heuristic techniques can be applied to a wide range of optimisation problem instances, although a downfall of them is that they are complex and can be time consuming to set up.

## **2.2 Benchmark problems**

There are a host of benchmark problems readily available for the travelling salesman problem. The most efficient path for these problems are available to view, meaning that they offer the ability to compare solutions against the optimum. Benchmark problems will be used to determine the effectiveness of the algorithms. To ensure fair experiments on configurations of the evolutionary algorithm, the number of calls to the fitness function will remain constant for every experiment. The deterministic algorithms implemented will always give the same answer, because of this they only need to be ran once on every benchmark problem. The problems used for this project were extracted from TSPLIB.

## **2.3 Deterministic techniques for solving the travelling salesman problem**

Deterministic techniques are those that will output the same answer to a problem every time they are used, they are predictable whilst also being very reliable. Deterministic algorithms are used in this project because they can be implemented quickly and aren't computationally expensive to run, by comparing the results they give against the stochastic techniques, a conclusion can be drawn up whether or not the time spent implementing and running a stochastic technique is worth while.

### **2.3.1 Brute force algorithm**

Arguably one of the more simple ways of implementing an algorithm to solve the travelling salesman problem, brute force works by creating every possible permutation that a travelling salesman could make round all of the locations, calculate the length of each tour before returning the shortest path found. Whilst this may sound ideal, it happens to be very flawed as with every



new addition to the list of locations increases the number of permutations exponentially. The running time for a brute force approach when applied to a travelling salesman problem lies within a polynomial factor of  $O(n!)$ . If calculating the fastest route of a problem with 10 locations takes 1 second, 11 locations would take 11 seconds and 12 locations would take 132 seconds. The rate in which this time increases is so extreme that in the scenario in which 10 locations takes 1 second, 20 locations would take over 20,000 years to solve. The effectiveness of a brute force approach depends on the computational power available and the size of the problem. With more computing power comes the potential to perform more calculations within a time frame, in addition, if the problem is small enough that it can be handled using the computational power available, then a brute force approach can be deemed appropriate. The main advantage of a brute force approach is that its output will always be the most efficient route.

### **2.3.2 2-opt algorithm**

2-opt, first introduced by [Croes, 1958] is a simple local search algorithm [Nuraiman et al., 2018] and its function is to aid other algorithms by identifying inter-crossing routes and the locations they originate from in their solutions and rearranging them so that they don't intersect each other by creating two new edges from the locations identified.

### **2.3.3 Nearest neighbour algorithm**

The nearest neighbour is a greedy technique [Nuraiman et al., 2018], [Halim and Ismail, 2019], the algorithm works its way from the starting location, visiting the closest location which hasn't been visited on the current tour until there are no more locations un-visited, before returning to the starting location. Since there is comparatively low computational effort required to solve travelling salesman problem using the nearest neighbour against other techniques it is one that can be used to get results for large problems, although as documented by [Nuraiman et al., 2018] these results aren't optimal. Figure 1 illustrates the performance of the nearest neighbour algorithm in comparison to a hybrid method.

The hybrid method used applies a 2-opt algorithm to a solution that has been worked on by the nearest neighbour algorithm. From Figure 1 it is evident that the hybrid technique out performs the nearest neighbour on every problem they were tested on, therefore it can be said that the application of a 2-opt algorithm has the potential to greatly increase the effectiveness of

Case	$n$	Optimal Solution	Nearest Neighbor		Hybrid Method	
			<i>Tour length</i>	<i>Error (%)</i>	<i>Tour length</i>	<i>Error (%)</i>
berlin52	52	7542	8182.19	8.49	7713.03	2.27
eil76	76	538	612.65	13.88	569.104	5.78
kroA100	100	21282	24698.5	16.05	21390.8	0.51
bier127	127	118282	133971	13.26	122072	3.20
ch150	150	6528	8025.45	22.94	7656.96	17.29
hex162	162	1620	1803.54	11.33	1692.64	4.48
pr264	264	49135	54491.5	10.90	52084	6.00
pr299	299	48191	57901.3	20.15	49555.9	2.83
pcb442	442	50778	58953	16.10	53044.4	4.46
hex486	486	4860	6140.35	26.34	5582.28	14.86
rat783	783	8806	10709	21.61	9369.56	6.40

Figure 1: Nearest neighbour vs hybrid method results [Nuraiman et al., 2018]

an algorithm. The error rate displayed in Figure 1 represents the difference between the solution found against the optimal solution, when the nearest neighbour was applied to the kroA100 problem it had an error of 16.05%, whereas the hybrid method was as low as 0.51%, this is a significant improvement and indicates the potential that the 2-opt algorithm has. The average error for the nearest neighbour algorithm throughout this experiment held at 16.46%, the hybrid method managed to lower this rate down to 6.19%. This suggests that although the nearest neighbour is computationally light, the solutions it produces can be optimised, in this case they were improved by 10% on average. From this it can be said that the nearest neighbour is an ineffective method for finding optimal solutions, although with the inclusion of a 2-opt algorithm, these solutions can be viable.

## 2.4 Evolutionary algorithms

Evolutionary algorithms are based off Darwin's theory of evolution, they are powerful problem solvers and have a host of different uses in the real world from scheduling timetables to planning out routes for delivery drivers. The theory of evolution states that, given an environment with limited resources, such as food, there is a limit to the number of individuals within a population that can thrive there, "natural selection favours those individuals that can compete for the given resources most effectively" (Eiben, A. E., & Smith, J. E. (2003)). Fitter individuals, those who are better adapted to the environment have a higher chance of survival and to reproduce. On the other hand, less fit individuals have a lower chance of surviving and thus are less likely to reproduce and pass on their genes to the next generation. The evolutionary process is illustrated in Figure 2.

Evolutionary computing takes inspiration from the survival of the fittest mechanism which takes place in nature by hosting a population of solutions, combining their traits to create new solutions with crossover operators, which then have a chance to be mutated to introduce new traits to the population before the entire population undergoes survivor selection to keep the population size constant.

As explained by [Larrañaga et al., 1999], evolutionary algorithms are made up of a set of components, namely:

- Representation
- Fitness evaluation
- Population
- Parent selection
- Recombination
- Mutation
- Survivor selection

### 2.4.1 Representation

Representation is the definitions of the individuals within the population, mapping from the real world to the evolutionary algorithm world. In evolutionary computing, a population is a set of solutions to the problem at

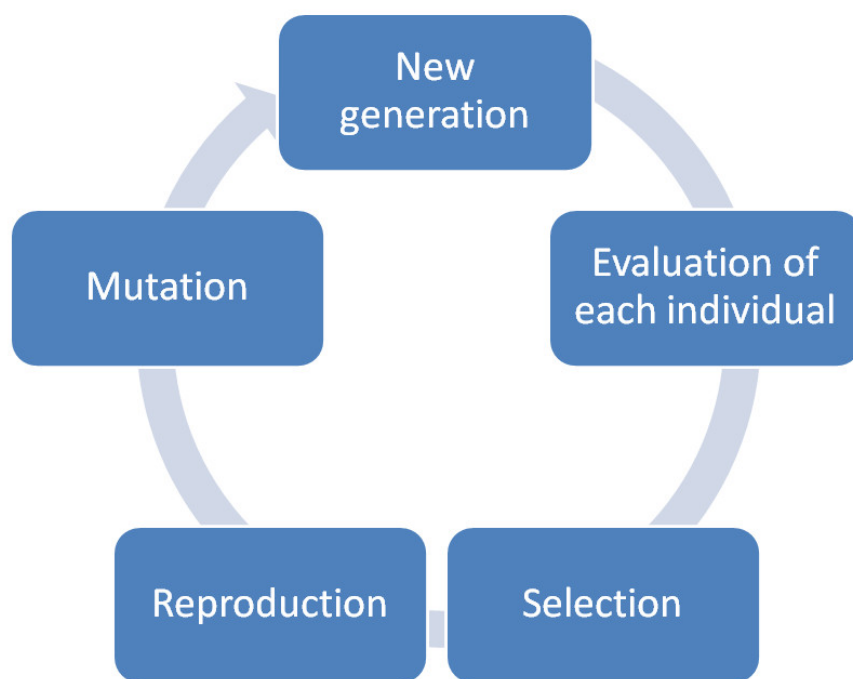


Figure 2: Evolutionary algorithm cycle <https://www.sicara.ai/blog/2017-08-29-was-darwin-great-computer-scientist>

hand, they must be built in such a way that they can be evaluated and assigned a fitness based on their performance. If we look at the travelling salesman problem, we can represent a solution in the real world as a route between locations on a map, whereas within an evolutionary algorithm this would be represented as a list of co-ordinates, a format which can easily be manipulated and will serve the purpose of representing a chromosome. “Objects forming possible solutions within the original problem context are referred to as phenotypes, while their encoding, that is, the individuals within the EA, are called genotypes computing” [Eiben, 2003a] . As explained by [Larrañaga et al., 1999], individuals within a population are often referred to as chromosomes, this term will be used for the duration of this paper.

#### **2.4.2 Fitness**

Every new chromosome needs to be assigned a fitness based on their performance in relation to the problem, this fitness will be necessary for selection and crossover. When solving the travelling salesman problem, the fitness of a chromosome is the length of the tour it represents, in this case a smaller tour length translates to a higher fitness score. The higher fitness an individual has, the more likely they are of surviving and passing on their genes to the next generation, although lower fitness individuals still have a chance of reproducing since their chromosomes may contain quality genes which could aid the algorithm in finding better solutions.

#### **2.4.3 Population**

The population is the collection of chromosomes that are currently being evaluated. A search space is the collection of all of the possible solutions mapped out with an axis for every trait that effects the performance of the chromosome, problems which require tuning two traits would have a three dimensional search space, with the z axis representing the fitness of a particular combination of the two traits. With the travelling salesman problem, every location requires an additional dimension within the search space, meaning the algorithm could be sifting through search spaces with hundreds of dimensions. The highest peak of a search space is known as the global optimum, this is ultimately what the evolutionary algorithm is searching for, the lower peaks are known as local optima.

#### **2.4.4 Parent selection**

Parent selection takes place before crossover, this is the process of choosing which chromosomes from the population are viable to pass on their character-

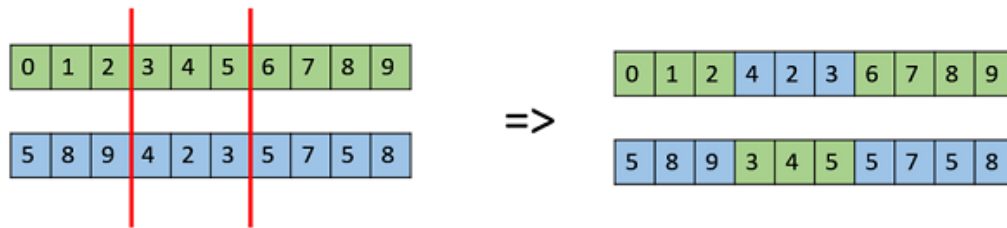


Figure 3: Two point crossover example [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_crossover.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm)

istics and traits onto the next generation of solutions. The selection process can be based on fitness, with fitter chromosomes given a better chance of being selected, although this process is stochastic, which helps the algorithm from getting stuck at local optima within the search space as it pushes the algorithm to explore more of the search space, this counter acts premature convergence, where solutions are centered around local optima.

#### 2.4.5 Crossover

The crossover function usually works by taking two solutions from the population and creating one or two child solutions. Figure 3 illustrates the outcome of two-point crossover that was applied to two solutions, each child takes characteristics from both of their parents' chromosomes. When working with permutation optimization problems such as the travelling salesman problem, the crossover operators used have to work in such a way that the outcome of their process doesn't render the solution invalid, for example where the same city appears more than once.

#### 2.4.6 Mutation

Mutation is applied to new chromosomes, "it is applied to one genotype and delivers a (slightly) different modified mutant" [Eiben, 2003b]. Mutation is stochastic, there is a chance that a chromosome will be mutated, there are many ways of applying mutation, such as swapping two random cells or applying a small probability to each cell that they will be slightly altered. Mutation aids the algorithm as it enhances exploration, increasing the area of the search space that is looked at by the algorithm. By increasing the size of the search, the algorithm has a lower chance of being stuck at a low-fitness local optimum. The mutation chance, that is how likely mutation will be applied to an individual, along with the mutation rate, which defines how mutated a child solution can become are both set by the algorithm designer. Certain

mutation techniques become invalid depending on the type of representation, for example in permutation based problems, such as the travelling salesman problem the outcome of a mutation must be a valid permutation, there can't be repeated genes throughout the chromosome. Whereas in situations where an individual can be represented using an array of integers, where genes can be repeated then single genes can be mutated, such as adding values to single genes. A common method for conducting mutation to a solution is to assign a chance for each gene to be mutated individually. These individual mutations include altering the index of the gene or altering its value.

#### **2.4.7 Survivor selection**

The survivor selection aspect of an evolutionary algorithm is used to decide which solutions are strong enough to represent the next generation. This enforces the survival-of-the-fittest rule, leaving behind poor chromosomes in favour of those that are stronger. Without a survivor selection mechanism, the population would ever increase, this process would continually require more memory, ultimately leading to a crash when the available memory runs out.

### **2.5 A comparison of mutation methods**

There are many ways that an evolutionary algorithm can be configured, which can cause issues as the designer is faced with many decisions, such the type of mutation, crossover and the way that the individuals are represented. To add to this, the parameters can be tuned which can give better results, the parameters generally include the number of iterations the algorithm will run for, the size of the population, mutation rate and the chance that an individual will be mutated. [Chieng and Wahid, 2014] compared different mutation methods on a travelling salesman problem, the experiments focuses on comparing the mutation operators to determine which operator finds the best solution and the computation time taken to find its optimal solution. The mutation methods used are GA-inversion mutation, GA-displacement mutation, GA-pairwise swap mutation and GA-inversion and they are applied to problems which have 10, 20, 30 and 40 nodes. Since the travelling salesman problem is permutation based, the mutation methods have to alter a chromosome in such a way that no gene appears more than once in the mutated chromosome.

<b>Before:</b>	1	3	2	6	5	4	7
<b>After:</b>	1	4	5	6	2	3	7

Figure 4: Inversion mutation

<b>Before:</b>	1	3	2	6	5	4	7
<b>After:</b>	3	2	6	5	1	4	7

Figure 5: Displacement mutation

### 2.5.1 Inversion mutation

Inversion mutation works by inverting a subset of a chromosome between 2 selected points (genes). If an individual is represented as  $\{1,5,4,2,3,6\}$  and genes  $\{5\}$  and  $\{3\}$  are selected, the sub-string  $\{5,4,2,3\}$  becomes  $\{3,2,4,5\}$ , thus altering the chromosome to become  $\{1,3,2,4,5,6\}$ . This is illustrated in Figure 4.

### 2.5.2 Displacement mutation

Displacement mutation selects a gene from the chromosome and relocates it within the same chromosome therefore altering the order of the genes. This helps to retain structures within the chromosome, this can be effective given the nature of travelling salesman problems since a set of neighbouring genes could represent an efficient path round a section of the problem. Displacement mutation potentially offers protection to these subsets as well as potentially shifting where they lie within the chromosome. An example of how displacement mutation works is shown in figure 5, where the first gene is displaced between the genes at indices 4 and 5.

### 2.5.3 Pairwise swap mutation

Pairwise swap mutation works by simply switching 2 genes within a chromosome. This is commonly used in permutation based encodings. This mutation typically works by iterating over the chromosome, where at each



<b>Before:</b>	1	3	2	6	5	4	7
<b>After:</b>	1	3	2	5	6	4	7

Figure 6: Pairwise mutation

gene there is a chance that a mutation operation will be applied (swapping the current gene with another random gene within the chromosome).

#### 2.5.4 Results

As documented by [Chieng and Wahid, 2014] these three mutation techniques underwent testing to determine their efficiencies in regard to processing time and the shortest path they each found over 10 runs, from this Figure 7 was produced which illustrates the findings. The best results are highlighted in green. Based on the results from this experiment it can be said that inversion mutation produces the best average minimum tour distance, it can also be seen that pairwise swap performed particularly poorly throughout these tests. Section B of Figure 7 displays the average number of iterations (generations) needed to return an optimal solution. The combination of the three mutation techniques within the same algorithm produces results in far fewer iterations than by using any single mutation method. Adding up the average iterations across the different mutations used shows that pairwise swap comes out as the least efficient method using 2043.6 iterations on average across the 4 problems it was applied to. Followed by displacement mutation which consumed 1953.1 iterations on average. Inversion came second at 1662.3 on average, whereas by using a combination of all 3 returned an average of only 852.8 iterations. Observing section C of figure 2 shows the average computation time of each mutation technique, again using a combination of all three mutations yields a more efficient search process as this takes the least amount of computation time in comparison to using each mutation strategy independently.

## 2.6 A comparison of crossover methods

The paper [Abdoun and Abouchabaka, 2012] compares a host of crossover operators and how well they each perform on the travelling salesman problem. The evolutionary algorithm used to conduct this research represented individuals with an array of integers, with each integer representing a par-

Number of constrained visited city ( <i>n</i> )	Mutation operators			
	Inversion	Displacement	Pairwise Swap	Inversion + Displacement + Pairwise Swap
	Section A: Average of minimum tour distance			
10	105.1791303	109.6361094	112.1044998	116.8613086
20	174.3624949	178.0269224	186.5919048	184.9301064
30	222.1828426	237.0308001	254.4681519	228.5520741
40	255.7966691	291.6375579	318.0730783	261.728686
	Section B: Average iteration			
10	32.6	31.5	42.4	16.5
20	180.8	346.9	341.6	139.6
30	564.6	641.1	720.8	276.4
40	884.3	933.6	938.8	420.3
	Section C: Average of computation time (s)			
10	10.4	11.2	10.3	4.1
20	11.2	12.2	11.5	5
30	14.1	14.1	13.7	6
40	15.8	15.8	15.6	7.1

Figure 7: Mutation findings [Chieng and Wahid, 2014]

<b>Parent 1:</b>	8	4	7	3	6	2	5	1	9	0
<b>Parent 2:</b>	0	1	2	3	4	5	6	7	8	9

Figure 8: Cycle crossover step-1

particular city of a tour, the integers are placed in the list in correlation to the order in which they are visited by the salesman. The initial population was generated randomly, this is done by creating individuals and assigning each of them a permutation of the list of locations. This particular evolutionary algorithm uses a roulette wheel selection process, in which individuals have a chance of being selected that is directly proportional to their fitness. [Abdoun and Abouchabaka, 2012] looks into the following crossover operators:

### 2.6.1 Cycle crossover

The cycle crossover [Oliver et al., 1987] identifies a number of cycles between the two parent chromosomes. To form Child 1, the first cycle is copied from parent 1, then the second cycle is copied from parent 2, cycle 3 is copied from parent 1, and so on. Figure 8 displays the two parent chromosomes which will be used as an example of how this crossover operator works. We start by looking at Parent 1's first gene and drop down to the same position in Parent 2, which holds the value 0, from here we go to where this value appears in Parent 1, which is in index 9<sup>1</sup>, we go to where the value 9 appears in Parent 1 and drop down to the same position in Parent 2, which is 8. Since we started with 8, we've completed our cycle. Figure 9 highlights the genes selected for the cycle in green, the next cycle starts with the second gene of Parent 1, Figure 10 illustrates the result of this. After the second cycle we are left with one gene, this will make up the final cycle. Figure 11 shows what the children chromosomes would look like, Child 1 takes the first and third cycles from Parent 1 and the second cycle from parent 2. Child 2 is made up of the remaining cycles. When the children are being constructed from the parents chromosomes in cycle crossover, the value and position of the parents genes are transferred to the child.

---

<sup>1</sup>In programming, the first index of an array is 0 instead of 1

<b>Parent 1:</b>	8	4	7	3	6	2	5	1	9	0
<b>Parent 2:</b>	0	1	2	3	4	5	6	7	8	9

Figure 9: Cycle crossover step-2

<b>Parent 1:</b>	8	4	7	3	6	2	5	1	9	0
<b>Parent 2:</b>	0	1	2	3	4	5	6	7	8	9

Figure 10: Cycle crossover step-3

### 2.6.2 Partially mapped crossover

Partially mapped crossover [Goldberg and Lingle, 1985] works by splitting the 2 parents into 3 sections each, using the same indexes for both parents, from which these sections can be used to assemble the child's chromosome. When creating 2 children from 2 parents, the first and last portions of one parent along with the remaining genes from the other parent that aren't currently present in the newly created child.

### 2.6.3 Non-Wrapping Ordered Crossover (NWOX)

NWOX was first introduced by [Cicirello, 2006]. NWOX starts by initialising two children (C1, C2) with copies of the parent chromosomes (P1, P2). Two positions are selected at random (A, B), such that  $\{0 \leq A \leq B \leq \text{length of the parents chromosome}\}$ , iterating over the C1, any gene that appears in P2's subsection (P2[A] - P2[B]) is made null. After this the genes to the left and right of where the child's subsection would be (C1[A] - C1[B]) are shifted towards the edge, leaving a gap between the genes and the subsection. After this the subsection fills every null gene outside of the subsection from left to right before the subsection of the P2 fills the gap in the middle. This same process is used to create C2, using P1.

<b>Child 1:</b>	8	1	2	3	4	5	6	7	9	0
<b>Child 2:</b>	0	4	7	3	6	2	5	1	8	9

Figure 11: Cycle crossover step-4

#### **2.6.4 Ordered Crossover**

This method of crossover works by taking a sub-string from one of the two parents chromosomes at random and placing this sub-string in its corresponding indexes in the child. Fill the empty genes of the child from left to right with the genes of the second parent that don't appear in the sub-string, placing the second parents genes into the child's chromosome in the order they show up when iterating over the chromosome from left to right.

#### **2.6.5 Shuffle crossover**

"Hypothesizing that the positional bias of onepoint crossover is more likely to work against the GA than to aid it, we developed an alternative form of crossover, shuffle crossover, to eliminate the bias. Shuffle crossover is similar to one-point crossover except that it randomly shuffles the bit positions of the two strings in tandem before crossing them over and then unshuffles the strings after the segments to the right of crossover point have been exchanged. Thus, crossover no longer has a single, consistent positional bias because the positions are randomly reassigned each time crossover is performed." [Caruana et al., 1989] This quote claims that the shuffle crossover is an improvement over the one-point crossover since on-point is more likely to not aid the genetic algorithm due to the positional bias of the operator.

#### **2.6.6 results**

The results from the experiments carried out from the paper [Abdoun and Abouchabaka, 2012] concluded that the ordered crossover method returned the best results with the non-wrapping ordered crossover producing the second best results in terms of shortest distance discovered throughout the duration of the testing. It is stated that non-wrapping ordered crossover does not always produce similar results, with the standard deviation of the best of the final individuals on 50 different populations being higher than all other competitors. This suggests that this operator is more influenced by its initial population than the other operators used.

## 2.7 Ant colony optimisation

Natural problem solvers are fascinating, by studying how they work computer scientists have been able to replicate how they work in the form of algorithms, these algorithms can be applied to routing and scheduling problems amongst others. With over 3 billion years of experience, evolution has produced many solutions to problems, of which some have been adopted by science with great success. Examples of nature's influence on the modern world include that the bullet train was influenced by the shape of the kingfisher bird, plants with burrs inspired Velcro, even some robotic arm designs were modelled off of an elephant's trunk. Ant colony optimization is an algorithm which mimics the behaviour of an ant colony when foraging for food, the motive behind pursuing this field comes from the fact that ants can find the shortest path between the source and their nest [Beckers et al., 1992]. In addition to this, ants can adapt to changes in their environment, for example finding a new shortest path once an older route becomes infeasible [Beckers et al., 1992]. As summarized by [Li et al., 2018], ant colony optimization (ACO) is inspired by how ant colonies behave in nature, it is a type of swarm intelligence and is a technique that is widely used to solve combinatorial optimisation problems such as the travelling salesman problem.

### 2.7.1 How they work

Ants deploy a pheromone trail as they walk, which attracts other ants to follow this trail, the pheromone decomposes over time and as the shorter paths are sprayed more frequently than the longer paths they become more potent. The more concentrated the path is with the pheromone the more attractive it becomes for ants to follow, resulting in the population following the shortest path found. As explained by [Acan, 2004], "This ant-based optimization principle combined with a pheromone update strategy, to avoid premature convergence to locally optimal solutions, is transformed into a remarkable optimization algorithm, the ACO algorithm, and applied for the solution of a wide variety of combinatorial optimization problems". Premature convergence can cause difficulties when attempting to explore the search space for good solutions, local optima can halt an un-optimised algorithm as they can continuously look for better solutions since the algorithm may not be equipped to explore less fit areas in the hope of finding larger peaks that may be present. This is the issue of exploration against exploitation, exploration enhances the algorithm's ability to cover more area of the search space whereas exploitation aims to guide the algorithm up the peaks faster. In an ant colony optimisation algorithm, an artificial ant is a simple agent that has the task of searching for good solutions on a given routing problem. Within

Problem name	ACS ( $cl = 20$ ) best result (1)	ACS average	Optimal result (2)	%Error $\frac{(1)-(2)}{(2)}$	CPU secs to generate a tour
d198 (198-city problem)	15 888 [585 000]	16 054 [71.1]	15 780	0.68	0.02
pcb442 (442-city problem)	51 268 [595 000]	51 690 [188.7]	50 779	0.96	0.05
att532 (532-city problem)	28 147 [830 658]	28 522 [275.4]	27 686	1.67	0.07
rat778 (778-city problem)	9015 [991 276]	9066 [28.2]	8806	2.37	0.13
fl1577 (fl1577-city problem)	22 977 [942 000]	23 163 [116.6]	[22 137–22 249]	3.27 + 3.79	0.48

Figure 12: [Dorigo and Gambardella, 1997] ACO results

a tour, an artificial ant is faced with a decision at each edge, to select the next edge in the tour, the length of each edge along with the pheromone level they have. The basic construction of an ant colony optimisation algorithm goes as follows, the first step within each iteration, every ant constructs a stochastic solution (with aid from the pheromone and weights of the edges), before the paths produced by the ants are compared. Finally the pheromone levels on each edge are updated.

### 2.7.2 Applying an ant colony optimisation algorithm to benchmark travelling salesman problems

An ant colony optimisation algorithm was applied to a set of benchmark travelling salesman problems and the results were documented by [Dorigo and Gambardella, 1997], the results of which are displayed in figure 12. The second column shows the best result obtained after 15 runs, the number in the square brackets represents the number of tours taken to find the shortest path. The third column shows the average shortest path found over 15 runs of the algorithm with the number in square brackets being the standard deviation. The fourth column displays the length of the optimal solution to the corresponding benchmark problem, with the fifth column showing the error rate, that is the difference between the best solution found against the optimal solution. The last column shows the CPU cost to run the algorithm on the corresponding problem. It is evident that this algorithm is very efficient, with consistently good results, with low error rates that drop below 1% with some of the benchmark problems in addition to consistently low length average tours being discovered throughout the testing.

## 2.8 Summary

Given the results from the literature review, it is evident that the ant colony optimisation algorithm, nearest neighbour paired with 2-opt and evolutionary algorithms are very effective at solving the benchmark problems they were given. Although given how dynamic a real world travelling salesman problem can be, an algorithm must be able to return good results for a wide range of problems. For this reason these algorithms will be implemented into command line application written using Python which will host an array of benchmark problems where statistical analysis will be carried out on the effectiveness and reliability of these algorithms on the benchmark problems. The brute force algorithms will not be assessed simply because there isn't enough time or computational power available to carry out the tests.

Larger problems require more time for evolutionary algorithms to work on to return good results, given a good result isn't always stumbled upon quickly due to the stochastic nature of the algorithm. This raises the question, what is the trade off between time taken to run the algorithm and the end product? Would it be more beneficial to use deterministic techniques which can produce lower quality solutions in a fraction of the time that stochastic algorithms?

Based on the information gathered in the literature review, the evolutionary algorithm will incorporate cycle, non-wrapping ordered and ordered crossover methods. Based on the fact that non-wrapping ordered and ordered crossovers performed the best from the paper [Abdoun and Abouchabaka, 2012]. Cycle crossover was found to be particularly interesting in the manner it functions, in addition the paper [Abdoun and Abouchabaka, 2012] tests the crossover operators on the Berlin52 problem. An effective crossover operator must be able to produce quality results on an array of problems, it is unknown if cycle crossover will be especially useful for the other benchmark problems which will be used to test the algorithm without testing. To add to this, cycle crossover may have been less effective in this paper given the setup of the evolutionary algorithm, with different operators active it may become hugely beneficial. For these reasons cycle crossover was selected for implementation and testing.

Given the paper that was reviewed for the mutation operators only had 3 operators to compare, all 3 of them will be implemented. The reason is that they are simple, thus it wouldn't take much time to write and introduce them into the proposed application.



### 3 Implementation

This section of the project will talk about how the algorithms were implemented, tested and evaluated, where the strengths and weaknesses of each algorithm will be discussed. Some of the code used for this project was designed and written specifically for this project, whereas the rest came from libraries or online sources. Writing code for a project such as this one enhances your knowledge and understanding as you learn how the algorithms work at their core. Although, the nature of writing code is that logic errors can be overlooked, leading to software that outputs unexpected results. In the case of optimisation problems, including the travelling salesman problem, it can be especially difficult to spot these errors if the code manages to output valid routes. This is where using libraries helps, if they are tried and tested, the results from these libraries when used on the benchmark problems can be compared to the results acquired from the self written algorithms.

Three algorithms were implemented for this project, an evolutionary algorithm, ant colony optimisation algorithm and a nearest neighbour paired with a 2-opt algorithm. The evolutionary algorithm was selected due to the impressive results they can produce, this was found whilst reading through academic papers that were gathered to aid writing the literature review. Nature inspired technology is becoming ever more present in the world, and with still so much to discover and learn about the natural world it will take research and investment to ensure we get the most out of this abundance of knowledge to help shape the future of technology. Evolutionary computing derives from Darwin's theory of evolution, from this biological theory came a field of computing which has aided computer scientists with solving optimisation problems for decades.

On the topic of nature inspired technology, the ant colony optimisation algorithm is directly inspired by the behaviour of ant colonies when foraging for food. This is an example of how code can simulate the movement and communication of an animal to solve a problem in the form of an algorithm. This is fascinating simply because ants are the product of evolution, according to the theory at least. Evolution is a problem solver in the context that species compete to survive against the odds such as the environment and other creatures, where strong genetic code translates to desirable physical, behavioural and cognitive abilities. In addition to this, evolution created human beings, who then went on to theorize evolution and created algorithms which could simulate this process. Evolution has the potential to be very powerful and this is why it is a respected field in the computing industry.

Ant colonies are very efficient in finding short routes between their nests and a food source, which in turn reduces the amount of energy required to transport the food back to the nest. This principle can be applied to the travelling salesman problem, a library by the name ACOpy was used for for project, this allowed the results from the ant colony optimisation algorithm to be compared to the evolutionary algorithm on the same benchmark problems.

With both the evolutionary and ant colony optimisation algorithms being stochastic, a third deterministic algorithm was brought in in attempt of evaluating whether the reliability and unpredictability of the stochastic algorithms are significant or not. If the stochastic algorithms take longer than their deterministic counterpart yet produce worse results, it can be said this time taken to run was un-necessary. The nearest neighbour algorithm paired with the 2-opt algorithm was chosen to represent the deterministic algorithm for the project since it's relatively simple to implement yet still capable of producing respectable results on benchmark problems.

For the testing phase the algorithms will be run on a set of benchmark problems, this allow the results to be compared to the optimal routes. Some algorithms may perform better on certain problems than others, by testing each of them on various problems, an insight will be gained into the performance of the algorithms in different environments. These algorithms in the real world will be used to solve a range of problems, if an algorithm is designed specifically to solve one problem, it may not be as effective on others.

Python was selected as the language since it has a lot of support with various libraries, it is modern and thus has a plentiful amount of documentation as well as being very easy to setup and operate the app using a command line interface.

### **3.1 Version control**

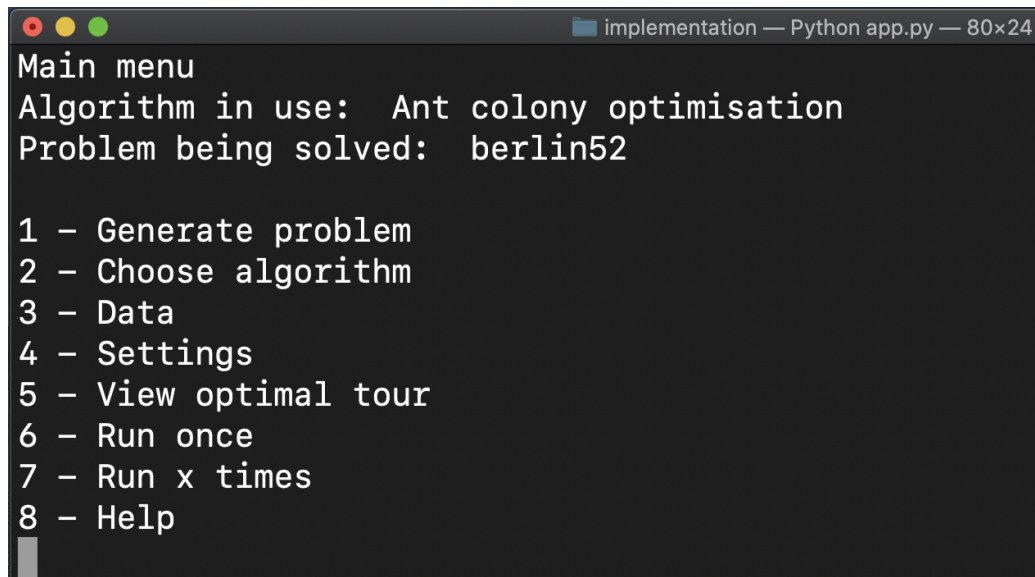
Whilst writing the code for this project, backups were made to GitHub. Pushing code to GitHub allowed previous versions to be revisited, this is handy when newly written code fails and previous states of the project are available to be retrieved. When developing the code, as the application became increasingly complex, it became more vulnerable to bugs, at times these bugs required more time to fix that what it would take to re-write. It was in these situations that having access to previous versions of the project allowed sections of code to be re-written without disrupting the rest of the

project.

### **3.2 Evaluating algorithms for the travelling salesman problem**

Comparing the effectiveness of algorithms on a certain problem requires testing, the data gathered from these tests can be used to carry out statistical evaluations, which as a result gives an insight into the strengths and weaknesses of the algorithms. Evaluation for this type of problem isn't one dimensional, with multiple important factors to consider. Generally, the bigger the problem the more time it takes to solve, given a situation with limitless time to find an efficient route, you could run an algorithm until the goal is achieved. Although this isn't realistic since the use for path finding algorithms in the real world is time constrained. To illustrate this point, take a delivery company, it is likely that one of their vans will take a new route everyday with different customers requiring packages. An algorithm will have to at least solve the routing problem once per 24 hours to ensure an efficient route is ready to be used on the morning the vans set out. Now consider there's a road block, the route produced by the algorithm is now invalid, if the algorithm takes a few hours to process then all of the deliveries will be late and the company loses money. This example is a reason to argue that the time required for an algorithm to calculate a good enough route is a valid factor to consider. The reliability of an algorithm is also important, that is how often it will manage to find an efficient route, it may be the case that a stochastic algorithm will have to be run more than once to find a suitable route, the less reliable an algorithm is the more times it may have to be run. Another indicator of the performance of an algorithm on a routing problem is the shortest route it can find, this shows the potential of the algorithm. The ideal algorithm will find an efficient route every time it is run within a suitable time frame.

Measuring the efficiency of an algorithm, that is how long it takes to run isn't can't be used as a constant when comparing algorithms that are run on different computers. In evolutionary computing, having the same number of calls to the fitness function is the standard way to ensure fair comparisons between evolutionary algorithms. Take an distributed evolutionary algorithm (DEA), where the population is split up into isolated subsets, where individuals can only reproduce with others from their subset and migration occurs every  $n$  generations to ensure the spread of genes and to stall the rate of pre-mature convergence. Where a DEA would be compared with an EA using a fixed number of generations, the DEA would have multiple more calls to the fitness function, meaning it would undergo more crossover



```
implementation — Python app.py — 80x24
Main menu
Algorithm in use: Ant colony optimisation
Problem being solved: berlin52

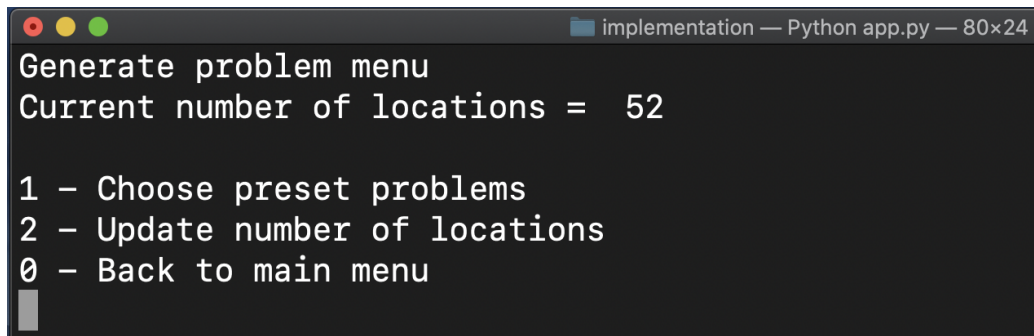
1 - Generate problem
2 - Choose algorithm
3 - Data
4 - Settings
5 - View optimal tour
6 - Run once
7 - Run x times
8 - Help
```

Figure 13: App main menu

and mutation operations as a result. This is assuming that the number of newly created solutions per island of the DEA is equal to that of the EA per generation.

### 3.3 Command line application design

To deal with the testing required for this project, the idea was to create a command line application which would make the process easier. This application would allow the user to test out the effectiveness of an ant colony optimisation, nearest neighbour and an evolutionary algorithm on 10 benchmark travelling salesman problems which range in complexity. By running an algorithm multiple times on a chosen problem, the standard deviation, error rate, mean and best scores are displayed. This allows large tests to be conducted unsupervised which means the user won't have to manually set up a test multiple times. A goal of this application was to allow a user to interact with the stochastic algorithms, such as updating the parameters of the evolutionary algorithm from a settings menu. Figure 13 shows how the main menu looks, it's is simple yet functional, it takes an integer between 1 and 8 as an input and runs the corresponding command. At the top of the main menu, the problem and algorithm in use are displayed. This subsection will focus on how the application works, without too much depth into how the code was written, the reason being that this project is centred around the science rather than the software development.



```
implementation — Python app.py — 80x24
Generate problem menu
Current number of locations = 52

1 - Choose preset problems
2 - Update number of locations
0 - Back to main menu
█
```

Figure 14: Generate problem menu

### 3.3.1 Generating the problem

Within the Generate problem sub-menu, shown in figure 14, the user has 3 options, the first is to enter "1" to choose from a selection of benchmark problems, this is illustrated in figure 15. The second option allows the user to decide how many nodes to be used when generating random co-ordinates, and thirdly entering "0" will return the user back to the main menu.

### 3.3.2 Changing the algorithm

The user is able to update the algorithm in use between runs, from the main menu they can access the sub-menu shown in figure 16 where 4 algorithms can be picked from, the chosen algorithm will be in use until another one is chosen or when the application is terminated.

### 3.3.3 Settings

The settings menu allows the user to update the EA and ACOpy settings. The EA settings include the mutation chance, population size, number of generations, the mutation operator and the crossover operator. The ACOpy settings include, Alpha, Beta, Rho, Elite, Q, T and number of ants (N), this is shown in figure 17. Default settings are applied to the algorithms when the application is launched, once the settings are updated, they are in use until they are changed or the application is terminated. This feature was designed to allow an easier process when tuning parameters and running tests using different settings, the alternative is to manually update the code, which is less fluid as the application has to be compiled and launched every time an update is made. The EA and ant settings are represented each with an object, where the objects attributes are the respective parameters for the algorithms. The objects attributes can be easily updated and passed

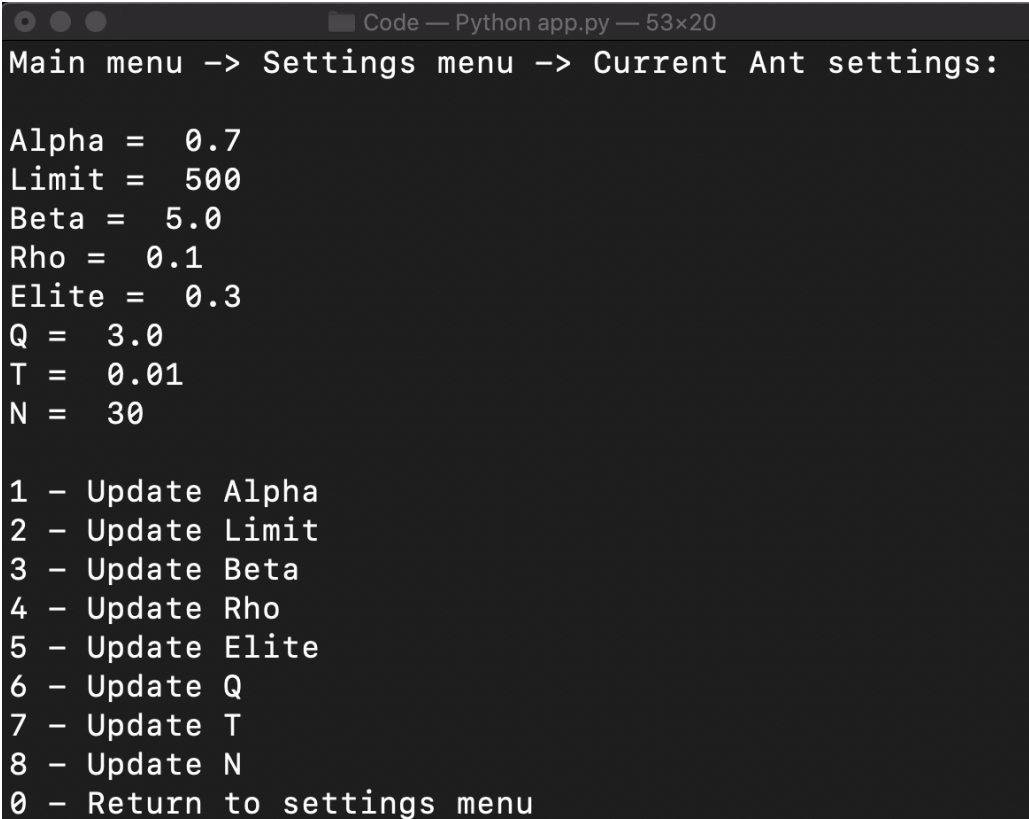
```
implementation — Python app.py — 80x24
Generate problem menu
Current number of locations = 52

1 - Choose preset problems
2 - Update number of locations
0 - Back to main menu
1
Preset problems
1 berlin52
2 eil51
3 kroA100
4 att48
5 bays29
6 gr120
7 pr76
8 kroD100
9 eil76
10 st70
█
```

Figure 15: Preset benchmark problems selection

```
implementation — Python app.py — 80x24
Algorithm in use: Ant colony optimisation
These are the algorithms available:
1 - Evolutionary algorithm
2 - Nearest neighbour
3 - Brute force
4 - Ant colony optimisation
█
```

Figure 16: Updating the algorithm in use menu

A screenshot of a terminal window titled "Code — Python app.py — 53x20". The terminal displays the ACOpy settings menu. At the top, it says "Main menu -> Settings menu -> Current Ant settings:". Below this, the current settings are listed: Alpha = 0.7, Limit = 500, Beta = 5.0, Rho = 0.1, Elite = 0.3, Q = 3.0, T = 0.01, and N = 30. A list of options follows: 1 - Update Alpha, 2 - Update Limit, 3 - Update Beta, 4 - Update Rho, 5 - Update Elite, 6 - Update Q, 7 - Update T, 8 - Update N, and 0 - Return to settings menu.

```
Code — Python app.py — 53x20
Main menu -> Settings menu -> Current Ant settings:

Alpha = 0.7
Limit = 500
Beta = 5.0
Rho = 0.1
Elite = 0.3
Q = 3.0
T = 0.01
N = 30

1 - Update Alpha
2 - Update Limit
3 - Update Beta
4 - Update Rho
5 - Update Elite
6 - Update Q
7 - Update T
8 - Update N
0 - Return to settings menu
```

Figure 17: ACOpy settings menu

into the algorithms where the attributes are used as the parameters for the algorithms. On application launch, default settings objects are created with initial attributes.

### 3.3.4 Optimal tour

For the 10 benchmark problems implemented, their optimal path is available to view, this is especially useful to visualise how close to a route is to the optimal, the data used to display the optimal tours are also used to calculate the difference between a single solution and the optimal. Figure 18 shows the optimal path for the kroD100 problem.

### 3.3.5 Running the algorithm

There are two ways to run an algorithm from the main menu, running the algorithm once on the selected problem, or to run the algorithm a set number of times on the problem. Running the algorithm once will return the best

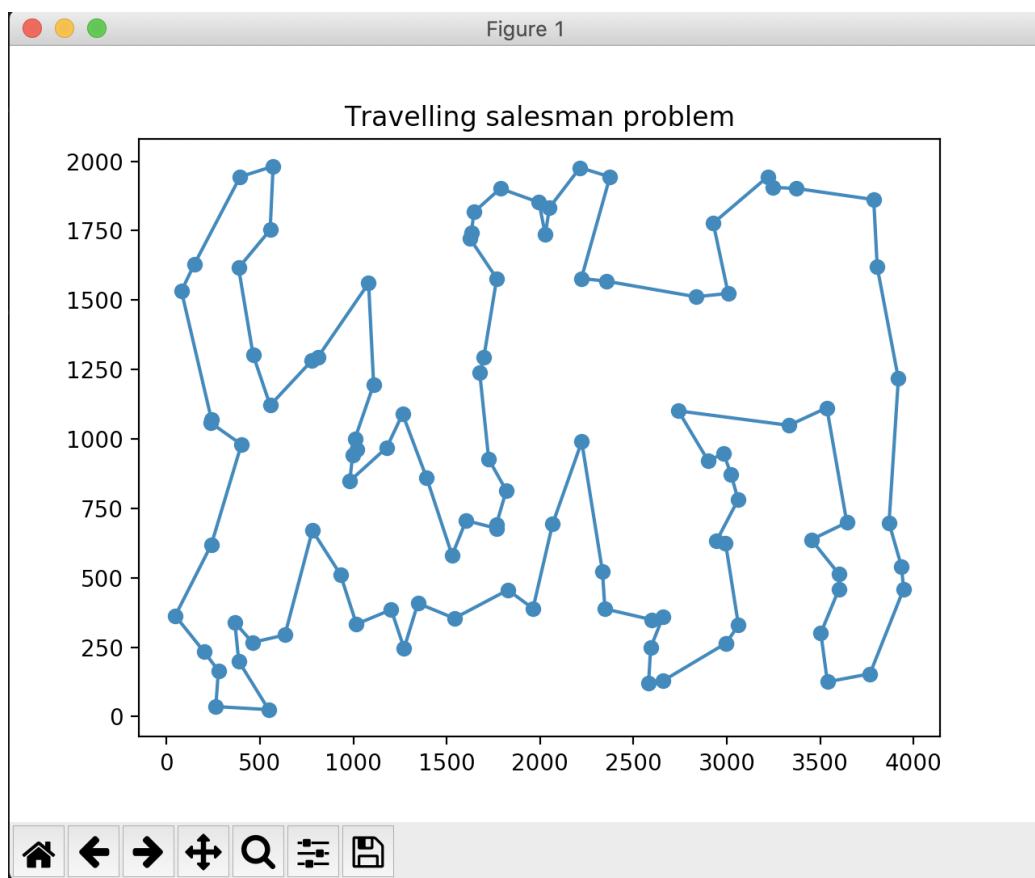


Figure 18: kroD100 optimal route



score obtained, whereas running the algorithm multiple times will have the added bonus of displaying the mean and standard deviation of the set of runs along with the best score obtained across all of the runs.

### **3.3.6 Help**

No this isn't a cry for help, when selecting the help option, information is displayed detailing how to operate the application. The text is printed character by character with a delay between each, this attempts to simulate the appearance that the text is being typed, this works by calling the sleep function between printing each character.

## **3.4 Writing the evolutionary algorithm**

The evolutionary algorithm has 6 main sections which had to be designed and implemented, namely representation, fitness evaluation, population, selection, recombination and mutation. Multiple crossover and mutation methods which were studied in the literature review were written to be tested in attempt of finding an efficient configuration for the algorithm. The remaining elements of the algorithm were chosen due to preliminary knowledge, testing couldn't be carried out for all aspects of the evolutionary algorithm due to time constraints.

### **3.4.1 Representation**

At the start of a run, random routes are initialised from the selected benchmark problem, these routes are stored in the form of an array within a 2-dimensional array, this 2-dimensional array represents the population. An individual is a tuple array, with each tuple consisting of two integer values, with the x and y co-ordinates of a node within a route being represented by the values in the tuple.

### **3.4.2 Fitness evaluation**

The fitness assigned to an individual from the fitness evaluation function is the length of the tour that the solution represents. The length of a tour is calculated using the euclidean distance between all of the connected nodes. The code for calculating the euclidean distance between two co-ordinates is displayed in figure 19, an individual is passed into the TotalDistance function, where the euclidean between each node is added up using the Distance function, which takes in two co-ordinates and returns the euclidean distance between them.

```

def Distance(a, b):
    tempA = (a[0] - b[0]) ** 2
    tempB = (a[1] - b[1]) ** 2
    return math.sqrt(tempA + tempB)

def TotalDistance(inputList):
    total = 0
    for x in range(len(inputList) - 1):
        total += Distance(inputList[x], inputList[x + 1])
    total += Distance(inputList[len(inputList) - 1], inputList[0])
    return total

```

Figure 19: Calculating fitness

### 3.4.3 Selection

The selection method used for the evolutionary algorithm is roulette, this works by assigning a probability of being chosen to each individual based on their score compared to the rest of the population. Roulette selection allows every individual a chance of being selected for crossover, lowering the selection pressure as a result, this in theory aids the algorithm to avoid wasting time stuck at local optima.

### 3.4.4 Crossover

The crossover methods for the evolutionary algorithm take in the population, the mutation rate and the mutation operator. The crossover operators select the parents as well as returning mutated children to the main function. Although it should be noted that each mutation and crossover operator have their own function, in hindsight it would have been simpler to call mutation and selection from the main method, this was realised after the EA was completed and thus it was deemed not worth the time to re-structure the code given it was able to produce good solutions to the benchmark problems. Non-wrapping ordered crossover was initially planned to be a part of this project, although with it being so complicated this threw up some errors when testing the code. After much effort and re writing of the code, it was deemed that this operator will not be used, and to move on and write the other methods instead.

### 3.4.5 Mutation

The plan for the mutation operators was to implement a separate function for each of them, with the mutation chance and individuals to mutate as parameters. Once all of the individuals have been mutated they are returned as a list back to the crossover function that the mutation function was called from. Figure 20 demonstrates the workings of the inversion mutation operator, which is a good example of how the mutation operators function for this project. The mutation functions have two inputs, the routes they are to mutate and a probability that each route will be altered. Figure 20 shows how genes are selected to represent the inner and outer bounds of the subsection that will be reversed within the child's chromosome, for the logic to work, the inner bound must come before the outer bound, by simply swapping their values if this is not the case allows the mutation to operate without compilation errors. Once the subsection is reversed, using python's `[::-1]` feature which can be used to reverse the order of lists, the subsection is re-introduced into the chromosome in its respective indices. Once all of the children have been operated on the children are returned and the process is complete.

### 3.5 Tuning the evolutionary algorithm

Before the evolutionary algorithm can be compared to the other algorithms implemented, it must be tuned. Parameter tuning consists of running the algorithm a number of times on a problem with different parameter settings in attempt of finding an optimal configuration. Because of the time restrictions of this project, not every configuration of the parameters could be tested. The method of tuning the parameters required choosing a configuration of the algorithm to test against, after each set of tests this configuration would be updated using the data acquired from that set of tests. This standard configuration included:

- Population size: 50
- Mutation chance: 5%
- Number of run times: 10
- Number of generations per run: 2000
- Mutation operator: Insertion
- Crossover operator: 2-point
- Benchmark problem: Berlin52

```

def InversionMutation(routes, probability):
    for y in range(len(routes)):
        x = routes[y]

        # Mutation probability
        if random.randint(0,100) <= probability:
            # Randomly choose two mutation points
            mutationPoint = random.randint(0, len(x) - 1)
            mutationPoint2 = random.randint(0, len(x) - 1)

            # Ensure the first point comes before the second
            if mutationPoint2 > mutationPoint:
                temp = mutationPoint
                mutationPoint = mutationPoint2
                mutationPoint2 = temp

            # Create a sub-section between the mutation points
            subChromosome = []
            for a in x[mutationPoint:mutationPoint2]:
                subChromosome.append(a)
            # Reverse the subsection
            subChromosome = subChromosome[::-1]
            # Introduce the reversed subsection into the chromosome
            for a in range(len(x[mutationPoint:mutationPoint2])):
                x[a] = subChromosome[a]

        routes[y] = x
    return routes

```

Figure 20: Inversion mutation code

Population Size	Mean score	Best score	Within optimal
10	13578.99	12217.10	61.94%
25	11356.93	10198.71	35.18%
50	10548.57	9627.52	27.61%
<b>75</b>	9743.29	<b>8672.32</b>	<b>14.95%</b>
100	<b>9666.53</b>	8763.13	16.15%

Table 1: Population size test results

The following sections will detail how the tests were conducted, and will look into the results from the tests to determine which configuration will be used for the remainder of this project.

### 3.5.1 Tuning the population size

Table 1 displays the results from a series of tests that were carried out using the standard configuration mentioned above. For each test, the population size was increased, the table shows the mean score, best score and how close to the known optimal this best score is for each population size. A test consisted of running the configuration on the Berlin52 benchmark problem 10 times. The number of generations was set to 2000 since if this was much larger, the time required to conduct the tests would be longer, where as if the number of generations was much smaller then this would give less of an indication of the performance of the configuration, the reason is that evolutionary algorithms are stochastic and they need time to find good solutions. From observing the results from table 1, it is evident that a population size of 75 returns the best result, although having a population size of 100 returned a better mean score, the best score overall was found with 75.

### 3.5.2 Tuning the probability of mutation

The next task was to tune the probability of a chromosome being mutated after it's been created, the table 2 shows the results of the mutation probability tuning. It can be said, based on the results that as the probability of mutating a chromosome is increased, the mean score and best score are improved, this is fascinating because traditionally in evolutionary computing it has been found that having a much lower mutation probability has been found to be more optimal. With a mutation probability of 75% the best score was found, finding a route within 6.35% of the known optimum. This is considerably better than the best route found when using a probability of 5% at 21.42%. Based on the results displayed in table 2, there is a correla-

Mutation chance	Mean score	Best score	Within optimal
5%	9966.85	9160.81	21.42%
10%	9457.29	8634.05	14.44%
25%	8855.73	8295.67	9.96%
50%	8976.01	8228.60	9.07%
75%	<b>8501.76</b>	<b>8024.54</b>	<b>6.36%</b>

Table 2: Mutation rate test results

Operator	Mean score	Best score	Within optimal
Insertion	<b>8781.94</b>	<b>8286.59</b>	<b>9.84%</b>
Inversion	18782.26	16779.93	122.42%
Swap-bit	9712.35	8941.65	18.52%

Table 3: Comparing mutation operators test results

tion between mutation probability and performance, with the mean scores improving as the probability is increased.

### 3.5.3 Mutation operator comparisons

Three mutation operators were written for the implementation stage for this project. Namely insertion, inversion and swap-bit mutation. Each operator was run 10 times on the Berlin52 benchmark problem, using a mutation probability of 75% and a population size of 75, using the 2-point crossover. Each operator was tested 10 times, with the best and mean scores being recorded. Table 3 displays the results from these tests. Inversion mutation performed the worst out of the three with a mean score of 18782.26, which scored 9068.91 worse than the mean of swap-bit mutation at 9712.91, this suggests that the inversion mutation operator is un-effective for the chosen parameters and current configuration of the evolutionary algorithm. The insertion mutation operator out performed the other two operators, with a considerable improvement over the swap-bit mutation, returning a best score which was 8.68% closer to the known optimal for the Berlin52 benchmark problem. An improvement of this magnitude is considerable and was chosen without hesitation to represent the evolutionary algorithm for the remainder of the project.

### 3.5.4 Crossover operator comparisons

Just like the mutation operators, three crossover operators were implemented. 2-point, cycle and ordered crossovers, they underwent similar testing to the

Operator	Mean score	Best score	Within optimal
2-point	8726.09	8050.55	6.70%
Cycle	8813.85	8336.99	10.50%
Ordered	<b>8484.79</b>	<b>8027.85</b>	<b>6.41%</b>

Table 4: Crossover comparison test results

mutation operators, using the same parameter settings. Since insertion mutation was deemed the most effective operator from those that were implemented, this was used for all of the crossover tests. The results from the experiments are displayed in table 4. It is clear that 2-point and ordered crossover return better results than the cycle crossover, with this operator failing to find a solution which was within 10% of the known optimal, whereas the remaining operators returned routes which were within 7%, which is impressive given the relatively small number of iterations the algorithm is run for each test. Ordered crossover, however, edges out 2-point crossover as it improves upon it's mean score by 241.3, with the best route found from 2-point being 0.28% closer to the known optimal. For these reasons, the conclusion is that 2-point crossover was the most suitable crossover method from those that were implemented for the evolutionary algorithm and thus will be used for the comparisons against the other algorithm in the final testing stage.

### 3.6 Implementing the ant colony optimisation algorithm

The library <https://pypi.org/project/acopy/> [?] was used for the ant colony optimisation algorithm. Contact was made with the author of ACOPY, where full permission was granted to use ACOPY to conduct the experiments needed for this project. The library is easy to set up, with an option of inputting values for a range of parameters, these parameters will be discussed and tuned, with the goal of finding high performing values for all those which will be tuned for this project.

**Alpha**, this represents how much pheromone matters is always kept greater than zero. The higher this value is, the more likely an ant will follow a trail with a higher concentration of pheromone, if this value is maxed out, an ant will always choose the path with the highest pheromone count, which restricts the exploration rate since most of the ants will end up following the shortest paths discovered at the beginning of the process. On the other hand if the alpha value is set to zero this eliminates the need for pheromone in the algorithm, as a result there becomes a much lower chance of good quality

solutions being found given the ants will wander near-aimlessly, around the graph until a tour is created.

**Beta**, this value dictates how important the distance between nodes matter when the ants decide which node to travel to next. If this value is too high then the ants will more often than not go to the nearest node, which begins to mimic the nearest neighbour algorithm. If the beta value is too low then the edges with more pheromone will be chosen more often, if initially bad paths are created, they could become highly concentrated with pheromone quickly, resulting in a bad solution as there is little exploration and not many paths will be reviewed.

**Limit**, this parameter is the maximum number of iterations to perform, in order to keep experiments fair, the number of calls to the fitness function will remain constant between the ant colony and evolutionary algorithms.

**Rho**, the level of rho specified effects how quickly the pheromone evaporates. If the rate of pheromone decay is too high, it will evaporate too quickly to become useful, as a result the ants have less guidance and may struggle to produce good solutions. If the rate of pheromone decay is too low then ants can become stuck in absolute solutions.

**Elite**, this value controls how much pheromone the best performing ant from each iterations gets to deposit. This in theory is very useful as it intensifies the amount of pheromone around the current best solution, which will mean more ants will explore more closely to this route.

The **Q** value represents the pheromone capacity of every ant, of course the more pheromone an ant can carry the more it can deposit. The Q parameter, along with Rho and Alpha alters the effect the pheromone will have on the ants when creating routes.

**T** is the amount of pheromone that can be present at each edge, this stops certain edges from having too much pheromone. This is necessary as it enforces exploration since if few edges had such high concentrations of pheromone then they will be visited much more frequently than other edges, resulting in increased exploitation.



Alpha rate	Mean score	Best score	Within optimal
0.5	10080.76	10025.70	32.89%
0.6	9880.19	9652.89	27.95%
<b>0.7</b>	<b>9692.37</b>	9180.26	21.68%
0.8	9721.10	9356.07	24.01%
0.9	9818.09	9390.75	24.47%
1.0	9725.13	<b>9083.02</b>	<b>20.39%</b>

Table 5: Alpha test results

### 3.6.1 Count

**Count** represents how many ants to use in each iteration. More ants require more processing power as more calculations are needed, although with more ants more routes can be explored. Tuning the number of ants to use will give an insight into how it effects the results.

## 3.7 Tuning the ant colony optimisation algorithm

Although the ant colony optimisation algorithm came from a library it still required parameter tuning as the base settings produced results which weren't able to compete with the evolutionary algorithm. The tuning consisted of running tests on the alpha, beta, rho, elite, Q, T, count and the limit settings.

### 3.7.1 Tuning Alpha

The alpha value is responsible for how much influence the pheromone has over the ants decisions when deciding which node to visit next. Interesting, based on the tuning results from table 5 it's evident that as the alpha value increases towards 1.0, the algorithm manages to find better scoring routes, with exception for 0.7 which could be argued as the most effective settings for the alpha value given it returned the best mean score. With similar results for the best score between 0.7 and 1.0, the fact that 0.7 had the best mean score was the reason why it was deemed the best setting, even though 1.0 did perform perfectly well to be considered.

### 3.7.2 Tuning Beta

From observing table 6 it is immediately clear that a beta setting of 5.0 is most beneficial for the algorithm since it achieved the lowest mean and best scores across all of the tests. The distance between the nodes is an important

Beta rate	Mean score	Best score	Within optimal
1.0	16876.59	15875.63	110.43%
1.5	13988.62	13375.76	77.29%
2.0	11977.71	11539.28	52.95%
2.5	10466.82	9926.34	31.57%
3.0	9783.47	9266.93	22.83%
3.5	9316.86	8948.80	18.61%
4.0	8858.21	8389.44	11.20%
4.5	8722.27	8568.56	13.56%
<b>5.0</b>	<b>8507.73</b>	<b>8170.62</b>	<b>8.30%</b>

Table 6: Beta test results

Rho rate	Mean score	Best score	Within optimal
<b>0.1</b>	<b>8401.85</b>	<b>7974.08</b>	<b>5.70%</b>
0.2	8525.33	7982.61	5.81%
0.3	8525.43	8312.56	10.18%
0.4	8465.41	8195.58	8.63%
0.5	8544.37	8323.98	10.33%
0.6	8413.21	8160.23	8.16%
0.7	8505.23	8207.20	8.78%
0.8	8493.92	8228.15	9.06%
0.9	8531.63	8141.65	7.92%
1.0	8522.77	8133.45	7.81%

Table 7: Rho test results

heuristic according to this result, given that better results are obtained where the beta value is turned up.

### 3.7.3 Tuning Rho

Rho represents how quickly the pheromone evaporates, the results from tuning this parameter are displayed in table 7. 0.1, being the lowest Rho level that was tested produced the best results, indicating that slowly evaporating the pheromone aided the algorithm more than quick evaporation. It can be said that there isn't a direct correlation between increasing the Rho rate and the performance of the algorithm given the mean scores fluctuate. This indicates that the Rho level isn't hugely significant, although since 0.1 performed the best in the tests it was deemed the most effective for this project.

Elite rate	Mean score	Best score	Within optimal
0.1	8475.06	8295.72	9.96%
0.2	8462.23	8208.99	8.81%
<b>0.3</b>	<b>8432.69</b>	8178.10	8.40%
0.4	8526.57	8356.52	10.76%
0.5	8518.31	8215.79	8.90%
0.6	8482.93	8156.92	8.12%
0.7	8511.10	8302.52	10.05%
0.8	8521.06	8191.91	8.58%
0.9	8434.85	8296.29	9.97%
1.0	8484.92	<b>8153.90</b>	<b>8.08%</b>

Table 8: Elite test results

### 3.7.4 Tuning Elite

Table 8 holds the results from tuning the Elite parameter, although 1.0 produced the best score across all of the Elite levels at 8151.90, the mean score returned by 1.0 is bested by 5 other Elite settings. Whereas 0.3 had the lowest mean score along with a respectable best score in comparison to the other results. For these reasons 0.3 was selected for the final configuration.

### 3.7.5 Tuning Q

Looking at table 9 it can be argued that the algorithm performs better as the Q rate is increased towards 3 and 4, and performs worse as the rate is increased beyond 4. This suggests a sweet spot between 3 and 4, with 3 producing the lowest mean score and 4 the best score overall. a Q level of 3 was chosen given it had the lowest mean score, lower mean scores indicate consistency, given ACOPy is stochastic, being able to find efficient routes frequently is a desirable quality to have.

### 3.7.6 Tuning T

The T value was tuned between 0.01 and 2.0, as displayed in table 10. Interestingly, based on the results, there doesn't seem to be a direct correlation between the T value and performance. Given how close the mean scores are, 0.1 managed to produce the best overall score, within 7.25% of the optimal route, for this reason it was chosen.

Q rate	Mean score	Best score	Within optimal
1	8469.37	8280.64	9.76%
2	8503.48	8107.85	7.47%
<b>3</b>	<b>8395.08</b>	8134.74	7.82%
4	8504.21	<b>8071.51</b>	<b>6.99%</b>
5	8517.85	8255.43	9.42%
6	8468.52	8172.14	8.32%
7	8504.00	8250.41	9.36%
8	8534.18	8251.73	9.38%
9	8424.30	8169.25	8.28%
10	8419.27	8274.57	9.68%

Table 9: Q test results

T rate	Mean score	Best score	Within optimal
0.01	8405.18	8148.55	8.01%
0.05	8473.19	8233.28	9.13%
<b>0.1</b>	8504.08	<b>8091.55</b>	<b>7.25%</b>
0.2	8455.72	8167.91	8.26%
0.5	8539.95	8309.04	10.13%
0.75	8505.51	8303.54	10.06%
1.0	<b>8402.65</b>	8121.51	7.65%
2.0	8543.16	8268.62	9.60%

Table 10: T rate test results

Count rate	Mean score	Best score	Within optimal
5	<b>8408.59</b>	8161.73	8.18%
10	8473.41	8239.83	9.22%
20	8529.60	8222.02	8.98%
<b>30</b>	8476.75	<b>8113.22</b>	<b>7.54%</b>
40	8492.95	8231.71	9.11%
50	8510.96	8179.38	8.42%

Table 11: Count test results

### 3.7.7 Tuning Count

The number of ants to use in each iteration was tuned between 5 and 50, the results from table 11 suggest that the number of ants used doesn't hugely alter the performance of the algorithm given the scores jump up and down as this value is increased. The mean scores acquired are all quite similar, with a difference between the best and worst being just 121.01, there wasn't much motivation behind selecting a count setting solely based off the mean. With 30 ants per iteration, the mean score is just 68.16 higher than the best at 8408.59 with 5 ants. Although with 30 ants the best score was within 7.54% of the optimal compared to 8.18% with 5 ants, since 30 ants achieved a respectable mean score and the best overall score it was deemed the most effective given the test results.

## 3.8 Writing the nearest neighbour algorithm

The nearest neighbour algorithm was written for this project, it works from a starting location, with an list of all of the un-visited nodes and an empty list. The algorithm adds the first node and the closest un-visited node from the starting point to the empty list. These two nodes are removed from the un-visited list, this process is repeated until the un-visited nodes list is empty, where the starting location is appended to close the route.

### 3.8.1 Applying 2-opt to the nearest neighbour algorithm

The nearest neighbour algorithm can find routes very quickly, although they aren't always good, the effectiveness of this deterministic algorithm depends on the problem. If the nodes are set up neatly then this algorithm is able to find the optimal route, although when the problems become more complex this algorithm begins to struggle. 2-opt takes the output from the nearest neighbour and improves upon it randomly swapping edges until all instances where paths cross in the solution are untangled, this in theory can vastly

improve a solution.

### **3.9 Testing the evolutionary algorithm**

Evolutionary algorithms take time to run, the amount of time required depends on the efficiency of the code, the magnitude of the problem and the power available from the computer that it's running on. To ensure time wasn't going to be wasted throughout the testing stage, initial tests were conducted where the evolutionary algorithm was left to run for 10000 generations on every benchmark problem, where the mean score was recorded after each generation, the results were graphed and analysed and the conclusion was that after around 4000 generations, the improvement considerably slows down. Figure 22 displays the best score obtained from each generation over the course of 10000 generations, it is clear that the algorithm starts to slow down significantly after 2000 generations, with a healthy drop between 0 and 2000 generations. Figure 23 shows the same test applied to the Berlin52 problem, here the algorithm slows down before 2000 generations, this suggests that the evolutionary algorithm finds efficient routes quickly. Although it could be argued that a lack of diversity throughout the population causes pre-mature convergence. After preliminary testing, it was found that running the algorithm for 5000 generations for the testing is manageable, this is above the point in which the algorithm slows down considerably, where it can take hundreds or even thousands of generations to find a better solution than the last, but helps to ensure that the algorithm will hit this stage before it is terminated.

### **3.10 Comparing the algorithms on the benchmark problems**

This section will look at the results from where the three algorithms were tested using 9 benchmark problems. Leading up to this section the evolutionary algorithm was parameter tuned, the mutation and crossover operators were selected after testing, the ACOPY algorithm's selected parameters were tuned also. It's known that the parameters for these algorithms aren't optimal, that is an optimisation problem in its self, but it can be said that given the parameters were selected based off testing, so they have enough quality for this project.

#### **3.10.1 Stochastic algorithm comparison**

With both the EA and ant colony optimisation algorithms being stochastic, it makes sense to compare their performance on the benchmark problems. Fair experiments were ensured by allowing the same number of fitness calls

for each algorithm. The EA requires 565,000 calls to the fitness function in 5000 generations, the next step is to take this and work out how many iterations the ant colony has to run for until this number is matched. Given T iterations, A ants and N nodes, the fitness function is called  $(T * A * N * N/2)$  times. The number of iterations to run for the Berlin52 benchmark problem can be worked out as follows:

1.  $565000 = T * A * N * N/2$
2.  $565000 = T * 30 * 52 * 26$
3.  $565000/30 * 52 * 26 = T$
4.  $565000/30 * 52 * 26 = T$
5.  $T = 14$

This calculation was applied to every benchmark problem before tests were conducted on them using the ant colony algorithm. The results from the ant colony tests are presented in table 15, which will be compared with the EA final test results from table 12. In addition to this, boxplots were created using the data from the EA and ant colony tests on the benchmark problems, the data from the tables along with the boxplots will be used to statistically analyse the performance of the two algorithms. The box plots were created using the same software which was used for displaying the graphs, Matplotlib. The results will then be compared to the performance of the nearest neighbour paired with the 2 opt before a final conclusion will be drawn up.

From observing 12 it can be argued that the EA manages to perform very well on some problems and not so well on others. The EA was within 2.28% of the optimal on the att48 problem, whereas it only manages to come within 28.73% of the optimal on the kroD100. This shows the inconsistency of the algorithm across the benchmark problems, even though it highlights the potential the algorithm has given the few very good scores. On the other hand, from looking at table 15 it could be argued that the results are quite similar across the board. To gain a deeper understanding of how these algorithms match up the boxplots will be compared.

Figure 24 illustrates how the EA and ant colony performed, it's clear that the EA performed best since its first and third quartiles are lower than the ant colonies. In addition to this, the best score recorded by the ant colony is worse

than the EA's first quartile. Figure 25 shows how the algorithms performed on the Eil51 problem, here the EA manages to produce much better results than the ant colony. The tables turn when Figure 26, where the ant colony algorithm produces much more consistent results, with a standard deviation of 924.32, compared to the EA with a standard deviation of 2211.51. This shows that for this problem, the ant colony is able to produce good results much more consistently. By observing Figure 27 and Figure 28 the conclusion that the EA is the better algorithm could be made as it performed best on these two problems. Figure 29, being the largest benchmark problem these algorithms are tested on it's interesting to see that whilst the EA managed to find higher fitness solutions, the range of scores it returned is much higher than the ant colony. Comparing these algorithms from Table 12 with Table 15 for this problem, we see that the ant colony had a standard deviation of 35.11 compared to the EA at 132.69. On such a large problem (compared to the others in this project) it's impressive that the ant colony managed to produce similar results so consistently. The EA is better suited to this problem, being able to return much better routes than the ant colony, despite how consistent the ant colony optimisation is. On the kroD100 problem, another relatively large problem the ant colony comes out on top, Figure 30 shows it's very low median, this demonstrates how the algorithm managed to consistently produce high fitness solutions, all of the ant colonies scores between its quartiles are better than the best score recorded by the EA. For the eil76 problem, the EA is clearly better, Figure 31 shows this. With a better mean and best score, the EA is clearly more suitable for this problem than the ant colony optimisation. The results for the final problem the algorithms were tested on, the st70 problem, shown in Figure 32 suggests the EA is capable of producing better results with less consistency. It should be noted that the median from the EA is significantly lower than the ant colonies, meaning there's a high concentration of high performing solutions coming from the EA, which is very promising. The conclusion is that the EA is the most suitable for this problem.

With the EA performing best on 7 out of the 9 problems, namely Berlin52, Eil51, att48, bays29, gr120, eil76 and st70, the conclusion is that the EA is more effective than the ant colony optimisation. The next section will further compare the EA against the nearest neighbour, with the intent of finding out whether or not the deterministic technique can out perform the stochastic methods.

**Analysing T-test results**, the P values from table 16 are useful, where they drop below 0.05 we can say the two data sets have a significant differ-



ence between their values. Where this is the case we can say with confidence, the better scoring algorithm is significantly better, the P values highlighted in **bold** are those that fall below 0.05, champion represents the best performing algorithm for the particular benchmark problem. kroD100 is the only benchmark problem where the ACO performs best and the P value is below 0.05. This means there is only one instance where we can say with confidence, the ACO is significantly better than EA. On the other hand there are 4 problems where the EA performs best with a P value of less than 0.05. With confidence that the EA is significantly better at solving almost half of the benchmark problems, this further backs up the statement that this EA is more suitable for solving these sets of problems.

### 3.10.2 Comparing the EA against the Nearest neighbour paired with 2-opt

The nearest neighbour with the 2-opt performed surprisingly well, so well in fact that a 280 node problem was implemented into the app just to test how effective this deterministic technique is on larger problems, the result can be seen in Figure 21, coming at at 5.15% within the optimal solution, which for a problem with  $((279)!/2)$  solutions is impressive.  $((279)!/2)$  is so unfathomably large it can't be described for this project. All of the scores obtained by the nearest neighbour, available to view in table 13, happen to be lower than all of the mean scores obtained by the EA. This means the EA often fails to produce scores better than the deterministic method. From observing the scores obtained by the nearest neighbour it becomes increasingly clear how effective this algorithm is. For example it managed to find a route which was within 1.21e-14%, which is so close to the optimal the difference is negligible. Noticeable differences between the algorithms include that the nearest neighbour came within 2.96% of the optimal for the kroA100 problem, which is leagues ahead of the EA at 25.70%, suggesting the staggering under-performance of the stochastic algorithms for this project. To add to this the nearest neighbour found a route within 10.44% of the optimal for the kroD100 problem, notably it's furthest from the optimal on any problem, still this beats the EA at 28.73%, yet another significant improvement over the EA. In fact the nearest neighbour produced better results on the Berlin52, kroA100, bays29, gr120, kroD100, eil76 and the st70. Where the EA out-performed the nearest neighbour, for problems eil51 and att48, the difference isn't nearly as significant as the gaps where the nearest neighbour performed better.

## 4 Evaluation

In conclusion, neither the evolutionary algorithm nor the ant colony optimisation algorithm managed to out perform the nearest neighbour paired with the 2 opt. This does not mean that the deterministic method is better than the rest, it simply mean based on the tests conducted for this project, the evidence suggests that the nearest neighbour is superior. Based on these results, it can be argued that the time taken for the deterministic algorithms to run is irrelevant based on the fact a relatively simple deterministic approach will return better results. Whether running the ant colony and evolutionary algorithms for longer will yield better scores will only be told if this is worked on at a future date. Saying this, throughout the building and tuning of the algorithms, very good scores were formulated, which was very promising and gave hope of the deterministic technique falling behind.

The application serves its purpose, it allows a smooth testing process as well as allowing the parameter settings to be tuned from a settings page, it is user friendly as far as cheap command line applications go and lays out all of its functions neatly across menus. The application was used for all of the parameter tuning and final testing, this did make the process easier which was the goal of the app.

In the end, an algorithm was found which was able to produce some very good scores on a set of benchmark problems, which to some extent is the purpose of this project. Although it was the goal to produce stochastic algorithms which would perform well, by indicating builds of these algorithms which don't work so well is still valid science.

### 4.1 Has the Project met its Aims and Objectives?

The first aim was to implement several algorithms to solve the travelling salesman problem, three algorithms were successfully implemented, all of which were in full working order and managed to produce high scoring results on benchmark problems. The next objective was to create an application which housed the algorithms, in addition the app would be tailored for testing. The application hosted the algorithms, allowed their parameters to be updated via a settings page, displayed the routes returned by the algorithms and carried out statistical analysis on the results. The app added a level of comfort to the parameter tuning and final testing phases, and was a joy to create. A conclusion was drawn up surrounding the performance of all of the algorithms, where the results weren't expected, yet they gave an insight into the performance of the configuration of the stochastic algorithms

which is valid science. Finally it was a goal to be able to visualise the results for a better insight into how the algorithms perform. This was covered when creating the application, the application can visualise a route from an input list of co-ordinates. In addition the app stores the optimal routes for all of the benchmark problems, which can be displayed also. Overall, the project managed to meet the aims and objectives that were laid out prior to starting, meaning it was a success.

## **4.2 Future work**

The project is far from perfect, there are many ways in which the application, and the algorithms it hosts can be improved. The application wasn't built robustly, it has flaws such as crashing if the wrong parameter type is inputted. The user interface could use a refresh, adding more visuals such as a loading bar when the algorithms are working away and the graphs displaying the algorithm and problem used. A data feature could be implemented, this would show information such as the best score recorded per benchmark problem along with the parameter settings/operators used. The ant colony and evolutionary algorithms could be tuned more rigorously, by either tuning the parameters more finely or using an external application that tunes the algorithm, as a result this could improve the algorithms which could lead to better results. Talking of the algorithms, the evolutionary algorithm is relatively bare in terms of the number of operators, replacement operators could be coded along with more crossover, selection and mutation operators. With more operators present more tests can be conducted, increasing the chance of more possible configurations that will be able to produce high scoring routes to problems. With the nearest neighbour paired with the 2-opt outperforming the stochastic algorithms in this project, it makes sense to experiment further with it. An example would be to use the product of a nearest neighbour run as a seed when running the evolutionary algorithm, this could prove beneficial as it provides a good starting, potentially accelerating the rate of improvement at the beginning of a run. The 2-opt algorithm could make use of the final paths returned by the stochastic algorithms, with many of the routes containing edges that cross each other. Finally, although it doesn't affect the performance of the app or the algorithms, the code could be cleaned up, putting duplicated code in functions and spreading the code into appropriate files as well as using classes more frequently.

## **4.3 Personal statement**

Artificial intelligence initially seemed otherworldly to me before getting my hands dirty with it, lots of new and fantastic opportunities came to light,

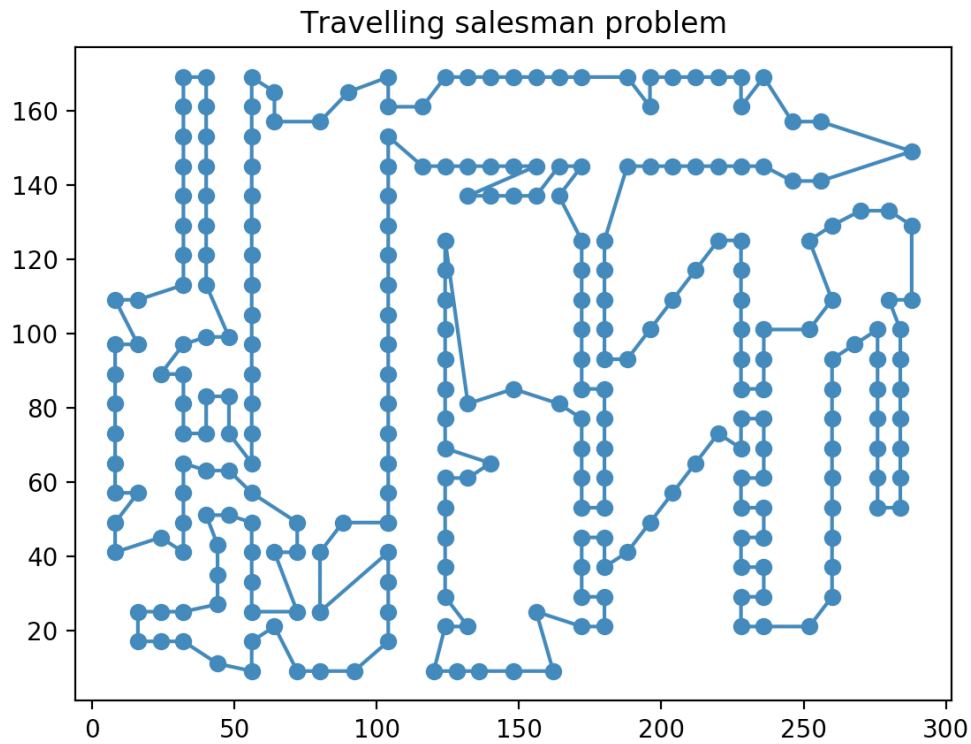


Figure 21: Calculating fitness

amongst those was evolutionary computing. Once a daunting subject became a very useful tool and helped to accelerate my confidence and knowledge of artificial intelligence. It was an easy decision to explore evolutionary computing for my honours, from there the other algorithms were chosen due to reading literature. This interest became motivation to read academic papers, books and online articles. A lot of time was spent writing the paper, when writing became mundane, I had the application to work on, admittedly the application needs work, but it functions well enough for this project and did its job very well. Overall, I am very happy with this project, it was a lot of fun and the most intense learning experience of my life so far.

Benchmark problem	Mean score	Best score	Within optimal	Standard deviation
Berlin52	8644.47	8144.61	7.96%	386.45
eil51	466.17	446.20	3.77%	13.56
kroA100	29443.08	26757.60	25.70%	2211.51
att48	36901.07	34287.71	2.28%	1650.93
bays29	9418.41	9120.34	-1.84%	210.35
gr120	2145.06	1879.18	12.76%	162.56
kroD100	29033.20	27412.77	28.73%	805.40
eil76	633.28	585.72	7.39%	27.98
st70	827.80	765.29	12.77%	53.19

Table 12: Evolutionary algorithm test results

Problem	Score	Within optimal
Berlin52	8182.19	8.45%
eil51	505.77	17.63%
kroA100	24698.50	16.03%
att48	39236.88	17.04%
bays29	9964.78	7.25%
gr120	1850.26	11.03%
a280	3095.17	19.65%
kroD100	24855.80	16.73%
eil76	612.66	12.33%
st70	761.68	12.24%

Table 13: Nearest neighbour final results

Problem	Score	Within optimal	Improvement
Berlin52	7544.36	1.21e-14%	637.83
eil51	450.04	4.66%	55.73
kroA100	21916.07	2.96%	2782.43
att48	35697.74	6.48%	3539.14
bays29	9111.61	-1.93%	853.17
gr120	1717.57	3.06%	132.69
a280	2720.53	5.17%	374.64
kroD100	23518.10	10.44%	1337.7
eil76	584.93	7.25%	27.73
st70	709.84	4.60%	51.84

Table 14: Nearest neighbour + 2opt final results

Problem	Mean score	Best score	Within opti- mal	SD
Berlin52	8567.00	8498.35	12.64%	286.74
eil51	507.75	487.31	13.33%	15.96
kroA100	27946.37	26473.70	24.37%	924.32
att48	39108.39	37580.62	12.10%	717.14
bays29	9983.74	9603.09	3.35%	231.45
gr120	2202.40	2144.78	28.70%	35.11
kroD100	27111.03	26277.75	23.40%	1003.42
eil76	680.93	642.48	17.80%	21.00
st70	838.73	794.44	17.07%	24.30

Table 15: Ant colony optimisation final results

Problem	P value	T value	Champion
Berlin52	0.6169	0.5091	EA
eil51	<b>&lt;0.0001</b>	6.2785	EA
kroA100	0.0639	1.9746	ACO
att48	<b>0.0011</b>	3.8779	EA
bays29	<b>&lt;0.0001</b>	5.7161	EA
gr120	0.2900	1.0903	EA
kroD100	<b>0.0002</b>	4.7242	ACO
eil76	<b>0.0004</b>	4.3072	EA
st70	0.5618	0.5911	EA

Table 16: EA and ACO t test results

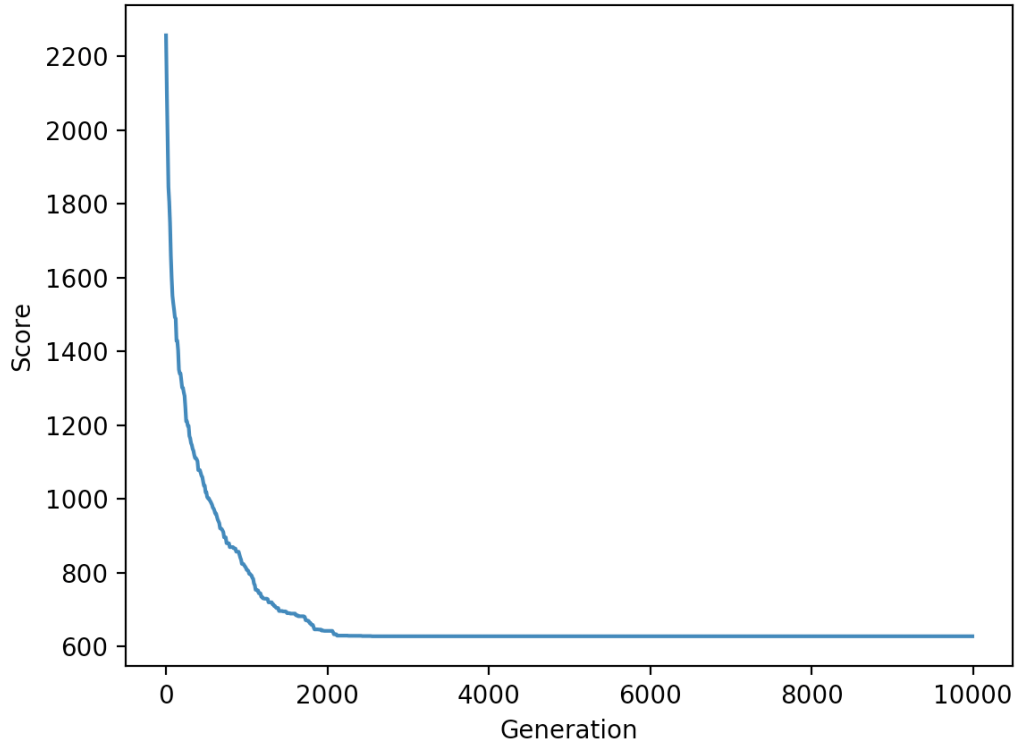


Figure 22: eil76 score over time

## Appendices

### A Project Overview

The initial project overview, which involved looking over the progress made and the plan for the rest of the project took place with the marker and second marker. This is displayed across Figures 33 and 34.

### B Second Formal Review Output

The second formal review is split between Figure 35 and Figure 36.

### C Project management

A gantt chart (Figure 37) was produced as a plan for when to start and finish certain milestones of the project. It starts with formulating the idea for the project and ends with completing the implementation stage. This chart doesn't cover every aspect of the project, just the key tasks.

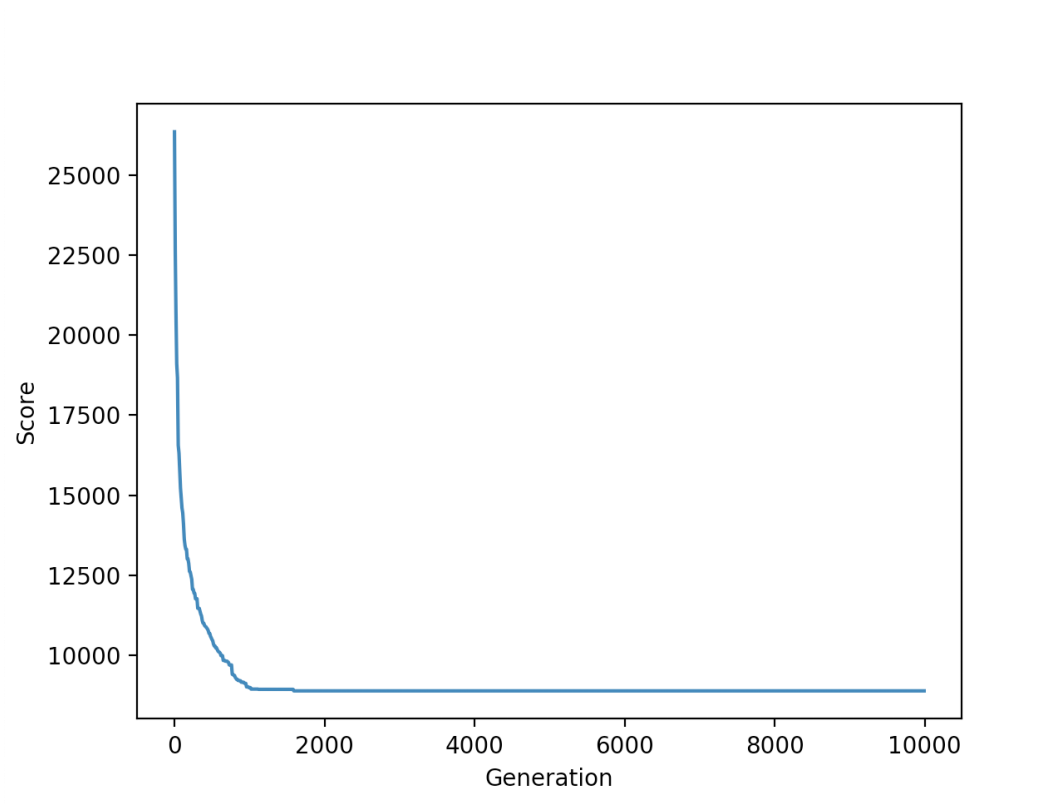


Figure 23: Berlin52 score over time



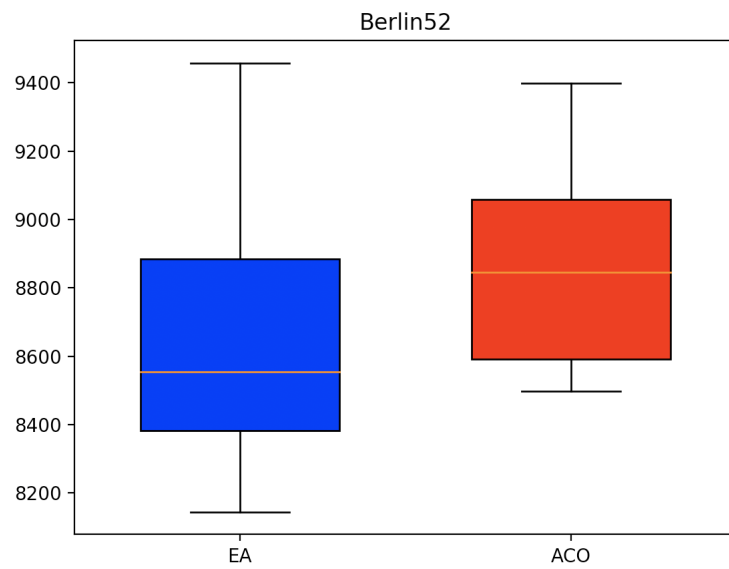


Figure 24: Berlin52 BoxPlot EA vs ACO

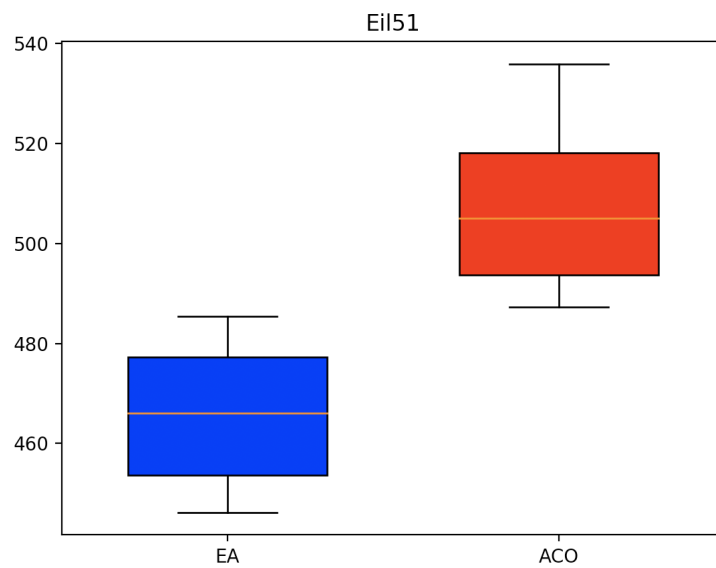


Figure 25: Eil51 BoxPlot EA vs ACO

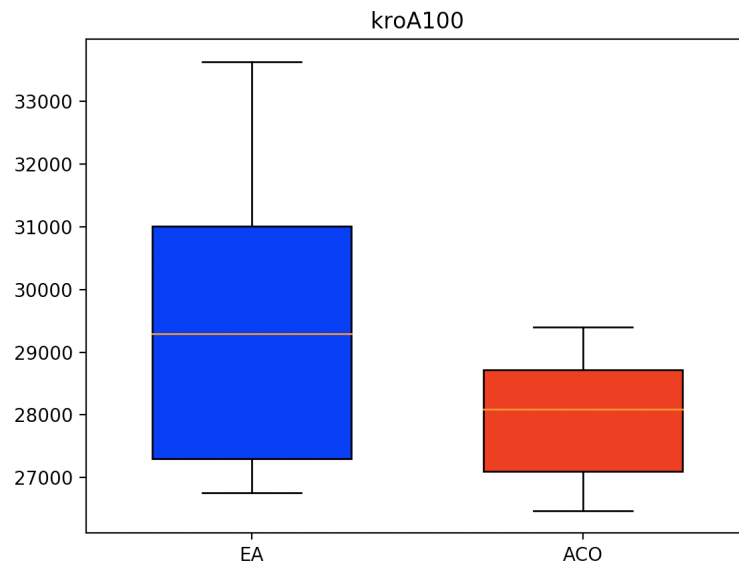


Figure 26: kroA100 BoxPlot EA vs ACO

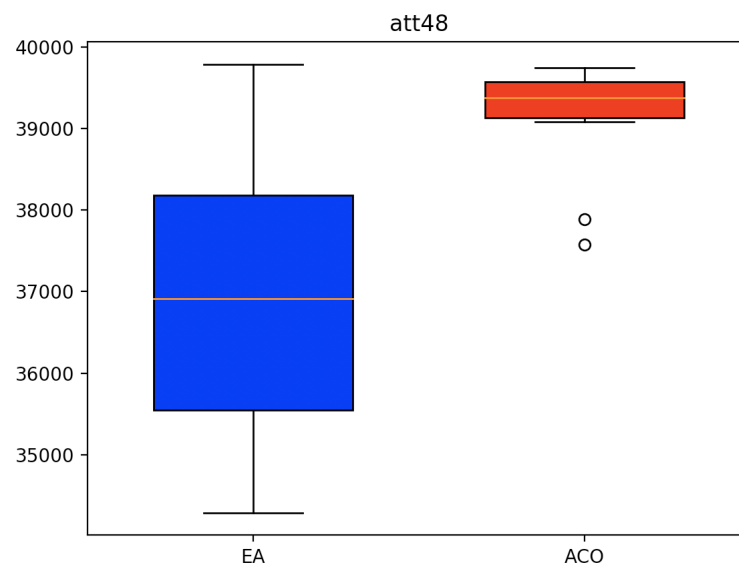


Figure 27: att48 BoxPlot EA vs ACO

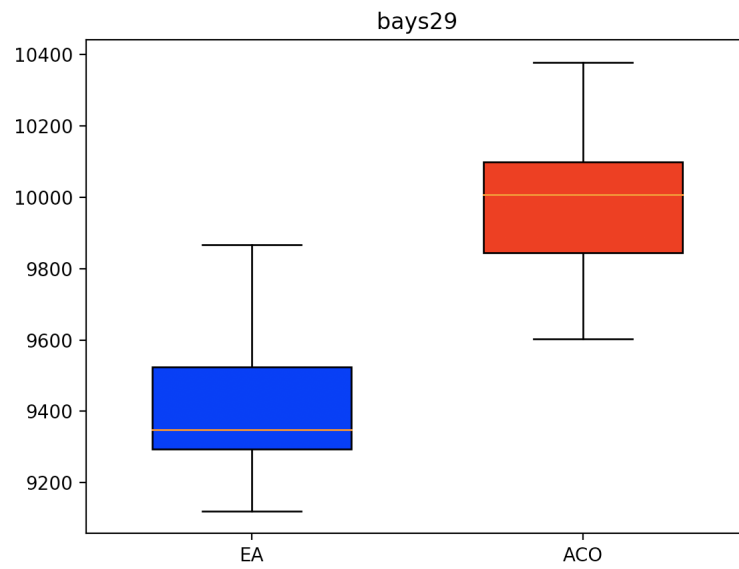


Figure 28: bays29 BoxPlot EA vs ACO

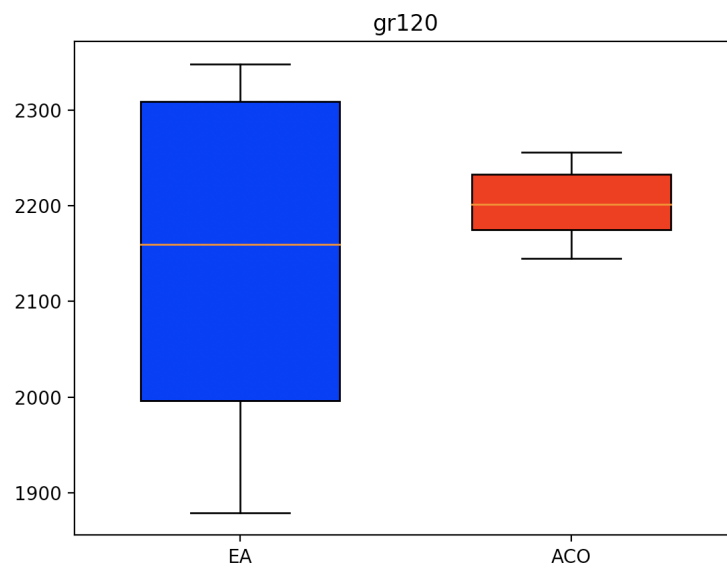


Figure 29: gr120 BoxPlot EA vs ACO

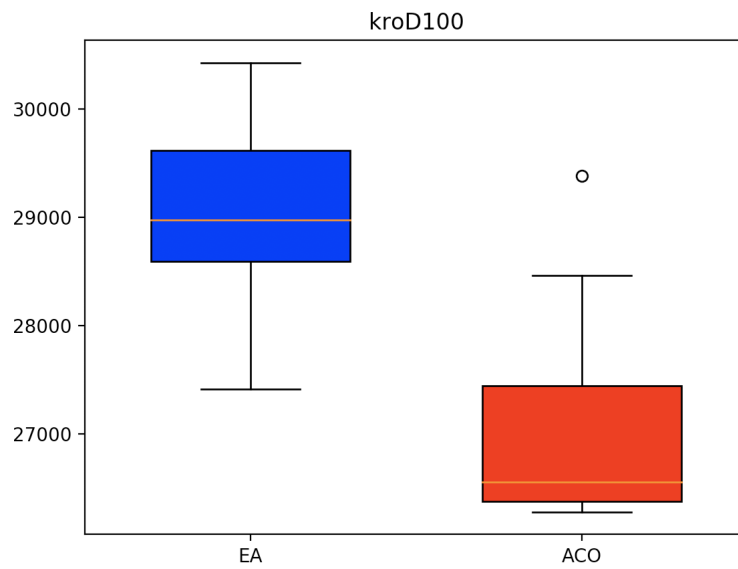


Figure 30: kroD100 BoxPlot EA vs ACO

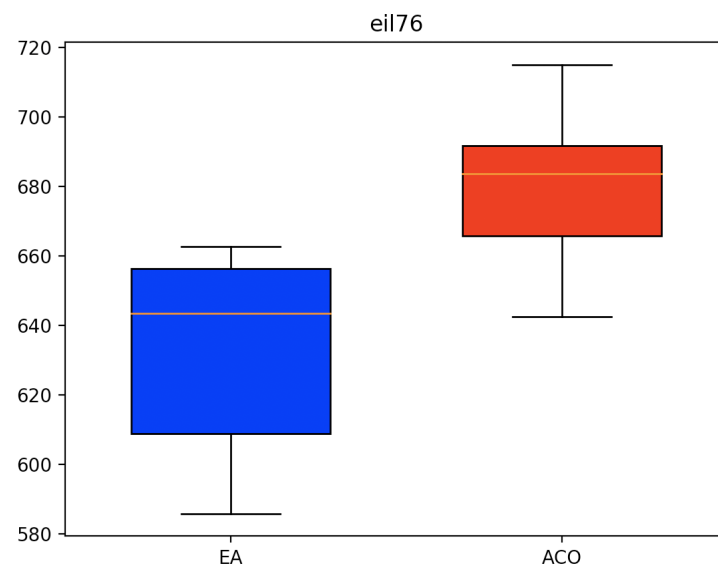


Figure 31: eil76 BoxPlot EA vs ACO

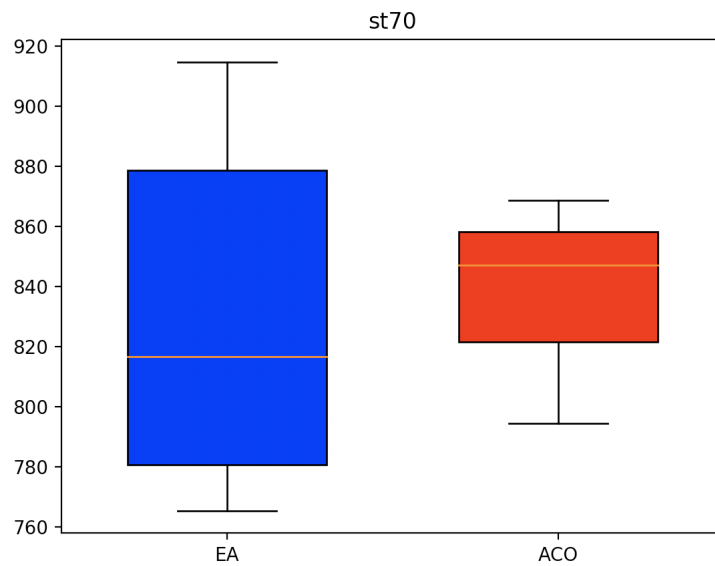


Figure 32: st70 BoxPlot EA vs ACO

The notes gathered from the weekly meetings were summarized into monthly objectives:

### September

- Research algorithms for the TSP
- Look into stochastic and deterministic methods
- Read up about EA
- Begin thinking about the application design and features
- Gather papers

### October

- Start writing app
- Write basic EA
- Start writing literature review
- Continue researching the topic

### November

- Add more operators to the EA

## **Initial Project Overview**

### **SOC10101 Honours Project (40 Credits)**

**Title of Project:** A comparison of the effectiveness of algorithms for solving the travelling salesman problem

#### **Overview of Project Content and Milestones**

To research methods for solving the travelling salesman problem.

To develop an application that hosts an array of methods for finding solutions for the travelling salesman problem, using this app to compare the effectiveness of the algorithms on a set of sample tsp routes and drawing conclusions from this analysis.

To visualise the results using graphs and to investigate and explain why some methods are more effective than others.

**The Main Deliverable(s):** Comparison of different methods for solving the TSP including complete solvers, heuristics and stochastic search techniques such as nature inspired algorithms. An insight into how these techniques work and a comparison of their effectiveness will be undertaken, and results presented using recognised statistical methods.

**The Target Audience for the Deliverable(s):** Students, researchers, anyone with an interest in nature inspired artificial intelligence.

#### **The Work to be Undertaken:**

- Investigation of methods of solving the travelling salesman problem
- Design and implement algorithms for the travelling salesman problem, test and analyse their effectiveness
- Compare the results of the different algorithms used throughout the project

- Writing a paper about the algorithms use

**Additional Information / Knowledge Required:** Benchmark problems. Methods for solving the travelling salesman problem

**Information Sources that Provide a Context for the Project:**

<http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html>

Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985, July). *Genetic algorithms for the traveling salesman problem*. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications* (Vol. 160, No. 168, pp. 160-168). Lawrence Erlbaum.

**The Importance of the Project:**

The travelling salesman problem has been studied extensively for decades. The problem is still unsolved in polynomial time and very much an active area of research. Finding better solutions for np-hard problems such as this one benefits a lot of people in society, more efficient path finding algorithms can save time and money among industries such as delivery companies and taxi drivers. This project will document the attempt of trying to improve on already established solutions for solving the travelling salesman problem as well as giving an insight into their effectiveness.

**The Key Challenge(s) to be Overcome:**

Understanding the theory behind and how to implement evolutionary algorithms and ant colony optimization on optimization problems. Research into a third nature inspired technique and how to implement it using code. A good understanding of all three is needed to be able to write a literature review, this will need lots of time devoted to reading books and papers on the subjects.

Figure 34: IPO - part 2

## SOC10101 Honours Project (40 Credits)

### Week 9 Report

**Student Name:** Sonas MacRae

**Supervisor:** Kevin Sim

**Second Marker:** Simon Powers

**Date of Meeting:** 12/11/2019

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

Please comment on the progress made so far

A fairly substantial literature review has been written. A range of operators and algorithms have been examined, and the report demonstrates good technical understanding. A prototype implementation has been produced, which is pleasing to see.

Is the progress satisfactory? **yes**

Can the student articulate their aims and objectives? **yes**

If **yes** then please comment on them, otherwise write down your suggestions.

The aims and objectives were not crystal clear from the report (especially objectives, which should be measurable). You should state explicit research questions. What algorithms will you compare and how will you compare them?

Does the student have a plan of work? **yes \***

If yes then please comment on that plan otherwise write down your suggestions.

\* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Figure 35: Week 9 report - part 2



A Gantt chart is not shown in the report (this should be shown in the final version, including evidence of rescheduling where appropriate).

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

This was not crystal clear from the report. Will the algorithms be evaluated on a fixed computational budget (fixed number of calls to the evaluation function)? Also consider extra work involved in calibrating the evolutionary algorithm compared to ACO.

Any other recommendations as to the future direction of the project

Add a conclusion to the literature review. Add a Gantt chart and reschedule where appropriate. Add numbered objectives. Finish coding this trimester to allow adequate time for experimentation. Allow adequate time for appropriate statistical tests.

Signatures: Supervisor Simon Powers

Second Marker Kevin Sim|

Student

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

\* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Figure 36: Week 9 report - part 2

- Start creating the application - interface etc
- Carry on writing literature review, citing the papers

## **December**

- Research ant colony optimisation
- Find an ACO library and implement it into the application
- Start testing the algorithms on benchmark problems
- Add to the literature review - explain the algorithms more

## **January**

- Begin writing the implementation stage
- Finish the literature review
- Brush up the app, add all of the proposed features, fix bugs
- Implement the Nearest neighbour

## **February**

- Conduct the parameter tuning for the EA and ACO
- Add the 2-opt algorithm to the nearest neighbour
- Add screenshots of the app to the project, talk about the app

## **March**

- Do the comparisons between EA and ACO
- Create tables, boxplots from comparisons
- Finish paper - write conclusion, app appendices, finish implementation stage
- Brush over the paper, fix grammar mistakes and fix the layout

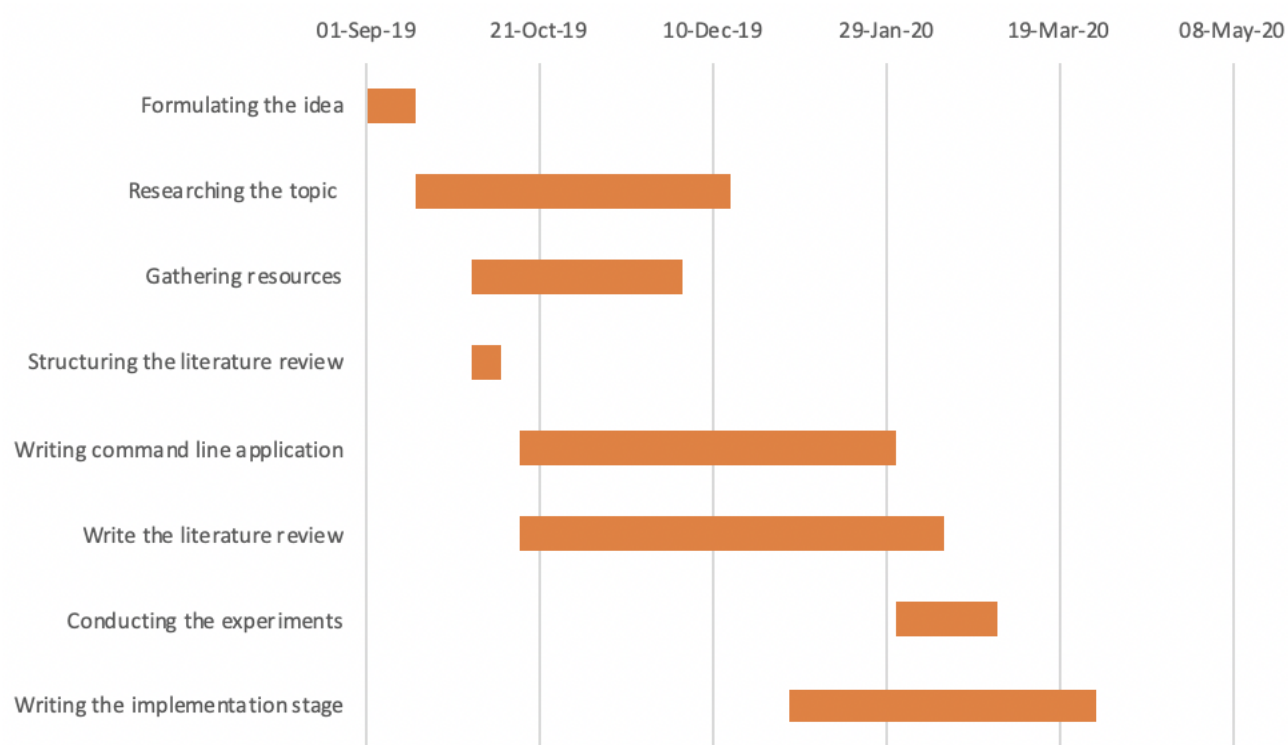


Figure 37: Project gantt chart

## References

- [Abdoun and Abouchabaka, 2012] Abdoun, O. and Abouchabaka, J. (2012). A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. *CoRR*, abs/1203.3097.
- [Acan, 2004] Acan, A. (2004). An external memory implementation in ant colony optimization. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F., and Stützle, T., editors, *Ant Colony Optimization and Swarm Intelligence*, pages 73–82, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Beckers et al., 1992] Beckers, R., Deneubourg, J. L., and Goss, S. (1992). Trails and u-turns in the selection of a path by the ant *lasius niger*. *Journal of Theoretical Biology*, pages 397–415.
- [Caruana et al., 1989] Caruana, R. A., Eshelman, L. J., and Schaffer, J. D. (1989). Representation and hidden bias ii: Eliminating defining length bias in genetic search via shuffle crossover. In *Proceedings of the 11th international joint conference on Artificial intelligence-Volume 1*, pages 750–755. Morgan Kaufmann Publishers Inc.
- [Chieng and Wahid, 2014] Chieng, H. H. and Wahid, N. (2014). A performance comparison of genetic algorithm’s mutation operators in n-cities open loop travelling salesman problem. pages 89–97.
- [Cicirello, 2006] Cicirello, V. A. (2006). Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’06)*, pages 1125–1131. ACM Press.
- [Croes, 1958] Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.
- [Dorigo and Gambardella, 1997] Dorigo, M. and Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *biosystems*, 43(2):73–81.
- [Eiben, 2003a] Eiben, A. E., . S. J. E. (2003a). *Introduction to evolutionary computing*, volume 53.
- [Eiben, 2003b] Eiben, A. E., . S. J. E. (2003b). *Introduction to evolutionary computing*, volume 53.

- [Goldberg and Lingle, 1985] Goldberg, D. E. and Lingle, Jr., R. (1985). Alleles and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.
- [Halim and Ismail, 2019] Halim, A. H. and Ismail, I. (2019). Combinatorial optimization: Comparison of heuristic algorithms in travelling salesman problem. *Archives of Computational Methods in Engineering*, 26(2):367–380.
- [Kizilates and Nuriyeva, 2013] Kizilates, G. and Nuriyeva, F. (2013). On the nearest neighbor algorithms for the traveling salesman problem. *Advances in Intelligent Systems and Computing*, 225:111–118.
- [Larrañaga et al., 1999] Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170.
- [Li et al., 2018] Li, X., Keegan, B., and Mtenzi, F. (2018). Energy efficient hybrid routing protocol based on the artificial fish swarm algorithm and ant colony optimisation for wsns. *Sensors*, 18(10).
- [Nuraiman et al., 2018] Nuraiman, D., Ilahi, F., Dewi, Y., and Hamidi, E. A. Z. (2018). A new hybrid method based on nearest neighbor algorithm and 2-opt algorithm for traveling salesman problem. In *2018 4th International Conference on Wireless and Telematics (ICWT)*, pages 1–4.
- [Oliver et al., 1987] Oliver, I. M., Smith, D. J., and Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 224–230, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.
- [Shim et al., 2011] Shim, V. A., Tan, K. C., Chia, J. Y., and Chong, J. K. (2011). Evolutionary algorithms for solving multi-objective travelling salesman problem. *Flexible Services and Manufacturing Journal*, 23(2):207.