

2022

Hybridization of Biologically Inspired Algorithms for Discrete Optimisation Problems

Elihu Essien-Thompson

Technological University Dublin, Ireland

Follow this and additional works at: <https://arrow.tudublin.ie/scschcomdis>



Part of the [Computer Engineering Commons](#)

Recommended Citation

Essien-Thompson, E. (2022). Hybridization of Biologically Inspired Algorithms for Discrete Optimisation Problems. [Technological University Dublin].

This Dissertation is brought to you for free and open access by the School of Computer Science at ARROW@TU Dublin. It has been accepted for inclusion in Dissertations by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, gerard.connolly@tudublin.ie, vera.kilshaw@tudublin.ie.



This work is licensed under a [Creative Commons Attribution-Share Alike 4.0 International License](#).

Hybridization of Biologically Inspired Algorithms for Discrete Optimisation Problems



Elihu Essien-Thompson

A dissertation submitted in partial fulfilment of the requirements of
Technological University Dublin for the degree of
M.Sc. in Computer Science (Advanced Software Development)

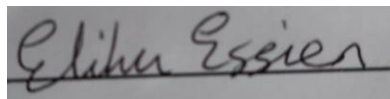
June 2022

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Advanced Software Development), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Technological University Dublin and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed:

A handwritten signature in dark ink, appearing to read 'Elisha Essien', is written over a horizontal line.

Date:

15/06/2022

ABSTRACT

In the field of Optimization Algorithms, despite the popularity of hybrid designs, not enough consideration has been given to hybridization strategies. This paper aims to raise awareness of the benefits that such a study can bring. It does this by conducting a systematic review of popular algorithms used for optimization, within the context of Combinatorial Optimization Problems. Then, a comparative analysis is performed between Hybrid and Base versions of the algorithms to demonstrate an increase in optimization performance when hybridization is employed.

Keywords: *Biologically Inspired Optimization Algorithms, Combinatorial Optimization Problems, Machine Learning, Swarm Intelligence, Mathematical Modelling*

ACKNOWLEDGEMENTS

I would love to express my sincere thanks to my supervisor Jack O'Neill and Dr Luca Longo for the time, talent and expertise they gave me from the preparation to the completion of this project. Their insights and support were what made undergoing this dissertation possible.

I also want to express my appreciation to all of the staff members of TU-Dublin, for all the support given throughout my years in the college. The invaluable knowledge and guidance that I have received from them have proven an incredible source of encouragement and inspiration all through my master's course.

I owe inexpressible gratitude to the members of my family for their continual love and support despite the many hours that I spent working on this project and ignoring them.

TABLE OF CONTENTS

ABSTRACT	II
ACKNOWLEDGEMENTS.....	III
TABLE OF CONTENTS.....	IV
TABLE OF FIGURES	VII
TABLE OF TABLES.....	X
TABLE OF EQUATIONS	XI
1. INTRODUCTION	1
1.1 BACKGROUND	1
1.2 RESEARCH PROJECT	2
1.3 RESEARCH OBJECTIVES.....	3
1.4 RESEARCH METHODOLOGIES	4
1.5 SCOPE & LIMITATIONS	5
1.6 DOCUMENT OUTLINE	5
2. LITERATURE REVIEW	6
2.1 COMBINATORIAL OPTIMIZATION PROBLEMS.....	6
2.2 OPTIMIZATION METHODOLOGY	8
2.2.1 <i>Search Space</i>	8
2.2.2 <i>Search Method</i>	10
2.2.3 <i>Evaluation Metric</i>	11
2.2.4 <i>Statistical Analysis</i>	12
2.3 STATE-OF-THE-ART IN BIOLOGICAL OPTIMIZATION	14
2.3.1 <i>Genetic Algorithm overview</i>	14
2.3.2 <i>The Selection Operator: Fitness Function Variants</i>	16
2.3.3 <i>The Breeding Operator: Crossover variants</i>	21
2.3.4 <i>Genetic Algorithm Enhancers</i>	21
2.3.5 <i>Particle Swarm Optimization Overview</i>	22
2.3.6 <i>Particle Swarm Optimization in Discrete Domains</i>	24

2.3.7	<i>Modified Particle Swarm Optimization</i>	26
2.3.8	<i>Ant Colony Optimization Overview</i>	27
2.3.9	<i>The Ant System</i>	30
2.3.10	<i>The Max-Min Ant System</i>	31
2.3.11	<i>The Ant Colony System</i>	32
3.	DESIGN & METHODOLOGY	34
3.1	DATA GENERATION.....	34
3.2	LANGUAGES USED	34
3.3	ALGORITHM IMPLEMENTATION & CONFIGURATION.....	35
3.3.1	<i>Genetic Algorithm Implementation Specifics</i>	36
3.3.2	<i>Particle Swarm Optimization Implementation Specifics</i>	38
3.3.3	<i>Ant Colony Optimization Implementation Specifics</i>	39
3.3.4	<i>Overview of Implementation Queries for Research Objective 3:</i>	41
3.3.5	<i>Hybridization Methodology</i>	42
3.3.6	<i>Benchmark</i>	43
3.3.7	<i>Chosen Statistical Test</i>	43
4.	RESULTS, EVALUATION & DISCUSSION	45
4.1	EXPERIMENT 1: GENETIC ALGORITHM	45
4.1.1	<i>Part 1 – Which Fitness Function to use?</i>	45
4.1.2	<i>Part 2 – Which Enhancer to use?</i>	50
4.2	EXPERIMENT 2: PARTICLE SWARM OPTIMIZATION	53
4.2.1	<i>Part 1 – Which Original Particle Swarm Configuration to Use?</i>	53
4.2.2	<i>Part 2 – Which Modified Particle Swarm Configuration to Use?</i>	55
4.2.3	<i>Part 3 – Which Particle Swarm Representative Model to Use?</i>	56
4.3	EXPERIMENT 3: ANT COLONY OPTIMIZATION	58
4.3.1	<i>Part 1 – Which Pbest value is the most appropriate for the Ant Colony variants?</i>	58
4.3.2	<i>Part 2 – Which of the Ant Colony variants performs the best?</i>	60
4.3.3	<i>Interpretation</i>	61
4.4	EXPERIMENT 4: HYBRIDS VS BASE	61
4.5	ALGORITHM RUNTIMES	66
5.	CONCLUSION	68

5.1	PROBLEM DEFINITION & RESEARCH OVERVIEW	68
5.2	STUDY OUTCOME	68
5.2.1	<i>Methodological Understanding</i>	69
5.2.2	<i>Experimental Understanding</i>	70
5.2.3	<i>Theoretical Understanding</i>	70
5.3	LIMITATIONS AND IMPROVEMENTS	71
5.4	FUTURE WORK & RESEARCH	71
6.	BIBLIOGRAPHY	73
7.	APPENDIX	80

TABLE OF FIGURES

FIGURE 1: EXAMPLE TRAVELING SALESMAN PROBLEM	7
FIGURE 2: EXAMPLE SEARCH SPACE/LANDSCAPE (MIRJALILI, 2019B)	9
FIGURE 3: LOCAL VS GLOBAL OPTIMUMS	10
FIGURE 4: ALGORITHM LEARNING RATE	12
FIGURE 5: GENETIC ALGORITHM FLOWCHART	16
FIGURE 6: ROULETTE WHEEL SAMPLING.....	17
FIGURE 7: STOCHASTIC UNIVERSAL SAMPLING.....	19
FIGURE 8: RANK-BASED SAMPLING	20
FIGURE 9: Crossover OPERATOR (MIRJALILI, 2019A)	21
FIGURE 10: PARTICLE SWARM OPTIMIZATION FLOWCHART	24
FIGURE 11: APPLYING MOVEMENT VECTOR TO POSITION VECTOR FOR A PARTICLE	25
FIGURE 12: ACO FLOWCHART	30
FIGURE 13: EXAMPLE DATASET FOR A MAP OF 10 CITIES	34
FIGURE 14: SOLUTION DISPLAY PROGRAM (CITY COUNT 10 AND 50)	35
FIGURE 15: POPULATION COMPOSITION FOR THE GENETIC ALGORITHM	36
FIGURE 16: GA BREEDING FOR THE TSP.....	37
FIGURE 17: GA MUTATION FOR THE TSP.....	37
FIGURE 18: ACO EXAMPLE DISTANCE MATRIX FOR A CITY COUNT OF 5	39
FIGURE 19: SEQUENTIAL HYBRIDIZATION STRATEGY FOR THE PARTICLE SWARM AND GENETIC ALGORITHM	43
FIGURE 20: ALGORITHM RESULTS SHOWING OUTLIERS.....	44
FIGURE 21: LINE PLOT SHOWING MODEL SCORE BY ITERATION COUNT FOR VARYING VALUES OF THE TOURNAMENT SAMPLING NOISE PARAMETER δ . A δ VALUE OF 1, INDICATING NOISE-FREE TOURNAMENT SAMPLING CONSISTENTLY OUTPERFORMS THE ALTERNATIVES.....	46
FIGURE 22: BOX PLOT SHOWING MODEL SCORE FOR VARYING VALUES OF THE TOURNAMENT SAMPLING NOISE PARAMETER δ	46
FIGURE 23: LINE PLOT SHOWING MODEL SCORE BY ITERATION COUNT FOR THE GENETIC ALGORITHM FITNESS FUNCTIONS.	47
FIGURE 24: BOX PLOT SHOWING MODEL SCORE PER GENETIC ALGORITHM FITNESS FUNCTIONS.	48

FIGURE 25: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR VARYING VALUES OF THE TOURNAMENT SAMPLING NOISE PARAMETER Δ , FOR AN ENLARGED EXPERIMENT OF 200 ITERATIONS FOR MAP SIZES OF 50 CITIES	49
FIGURE 26: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR GENETIC ALGORITHM FITNESS FUNCTION, FOR AN ENLARGED EXPERIMENT OF 800 ITERATIONS FOR MAP SIZES OF 50 CITIES.....	49
FIGURE 27: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR VARYING ELITE PERCENTAGES	50
FIGURE 28: BOX PLOT SHOWING MODEL SCORE PER ELITE PERCENTAGE	51
FIGURE 29: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE THREE ENHANCERS (ELITISM, STEADY-STATE, AND LOCAL STEADY-STATE)	52
FIGURE 30: BOX PLOT SHOWING MODEL SCORE PER GENETIC ALGORITHM ENHANCER	52
FIGURE 31: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR VALUES GIVEN FOR MODELS IMPLEMENTING PARTICLE INERTIA.....	54
FIGURE 32: BOX PLOT SHOWING MODEL SCORE FOR VALUES GIVEN FOR MODELS IMPLEMENTING PARTICLE INERTIA.	54
FIGURE 33: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE CONFIGURATIONS OF THE MODIFIED PARTICLE SWARM	55
FIGURE 34: BOX PLOT SHOWING MODEL SCORE FOR THE CONFIGURATIONS OF THE MODIFIED PARTICLE SWARM.....	56
FIGURE 35: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE ORIGINAL AND MODIFIED PARTICLE SWARM ALGORITHM.....	57
FIGURE 36: PARTICLE SWARM VARIANT BOX PLOT	57
FIGURE 37: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE PBEST VARIABLE OF THE MAX-MIN ANT SYSTEM.	59
FIGURE 38: BOX PLOT SHOWING MODEL SCORE FOR THE PBEST VARIABLE OF THE MAX-MIN ANT SYSTEM.....	59
FIGURE 39: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR EACH ANT COLONY VARIANT.	60
FIGURE 40: BOX PLOT SHOWING MODEL SCORE FOR EACH ANT COLONY VARIANT. ...	61
FIGURE 41: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE HYBRID VS BASE ALGORITHMS.	62

FIGURE 42: BOX PLOT SHOWING MODEL SCORE FOR THE HYBRID VS BASE ALGORITHMS.....	62
FIGURE 43: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE CROPPED HYBRID VS BASE ALGORITHMS.	63
FIGURE 44: BOX PLOT SHOWING MODEL SCORE FOR THE CROPPED HYBRID VS BASE ALGORITHMS.....	63
FIGURE 45: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE HYBRID VS BASE ALGORITHMS (CITY COUNT 20).	64
FIGURE 46: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE HYBRID VS BASE ALGORITHMS (CITY COUNT 30)	65
FIGURE 47: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE HYBRID VS BASE ALGORITHMS (CITY COUNT 50)	65
FIGURE 48: LINE PLOT SHOWING MODEL SCORE PER ITERATION FOR THE SUPPLEMENTAL EXPERIMENT TESTING THE EFFECTS OF AN INCREASED INERTIA WEIGHT ON THE PSO/GA SEQUENTIAL HYBRID ALGORITHM.	66
FIGURE 49: GA SOLUTION FOR TSP MAP (50 CITIES)	80
FIGURE 50: PSO SOLUTION FOR TSP MAP (50 CITIES)	80
FIGURE 51: ACO SOLUTION FOR TSP MAP (50 CITIES)	81
FIGURE 52: ACO/GA SOLUTION FOR TSP MAP (50 CITIES)	81
FIGURE 53: PSO/ACO SOLUTION FOR TSP MAP (50 CITIES).....	82
FIGURE 54: PSO/GA SOLUTION FOR TSP MAP (50 CITIES)	82
FIGURE 55: GREEDY-OPTIMIZER SOLUTION FOR TSP MAP (50 CITIES)	83

TABLE OF TABLES

TABLE 1: TOURNAMENT SAMPLING DELTA AUC	45
TABLE 2: FITNESS FUNCTION AUC	47
TABLE 3: ELITISM AUC.....	50
TABLE 4: ENHANCER AUC	51
TABLE 5: INERTIA WEIGHT AUC	53
TABLE 6: MODIFIED PARTICLE SWARM CONFIGURATION AUC.....	55
TABLE 7: PARTICLE SWARM VARIANT AUC	57
TABLE 8: MAX-MIN ANT SYSTEM – PBEST AUC	58
TABLE 9: ACO VARIANT AUC	60
TABLE 10: HYBRID VS BASE AUC FOR TSP CITY SIZE (10).....	62
TABLE 11: ALGORITHM RUNTIMES	67

TABLE OF EQUATIONS

EQUATION 1: TSP DISTANCE CALCULATION.....	7
EQUATION 2: TSP OBJECTIVE FUNCTION.....	8
EQUATION 3: ROULETTE WHEEL SELECTION PROBABILITY	17
EQUATION 4: MIN-MAX NORMALIZATION	18
EQUATION 5: CONVERTING RANGES	18
EQUATION 6: FITNESS SCORE.....	18
EQUATION 7: PARTICLE VELOCITY CALCULATION	23
EQUATION 8: WEIGHT APPLIED ON PARTICLE SWAP SEQUENCE.....	26
EQUATION 9: ADAPTED PARTICLE MOVEMENT CALCULATION	26
EQUATION 10: ANT PHEROMONE DEPOSIT	29
EQUATION 11: EDGE PHEROMONE UPDATE.....	29
EQUATION 12: PATH DISTANCE SCORE.....	30
EQUATION 13: ANT STOCHASTIC DECISION RULE	31
EQUATION 14: MAX-MIN PHEROMONE UPDATE	32
EQUATION 15: MAX-MIN CLAMP OPERATOR.....	32
EQUATION 16: LOCAL PHEROMONE UPDATE	32
EQUATION 17: ANT DETERMINISTIC DECISION RULE	33
EQUATION 18: T-MAX CALCULATION	40
EQUATION 19: T-MIN CALCULATION	40

1. INTRODUCTION

1.1 Background

Biologically Inspired Algorithm is a term used to denote a family of algorithms that arose from an analysis of nature's solutions to common problems. They are further subcategorized by their general methodologies like *evolutionary algorithms* (using the concept of genetic crossovers) and *swarm intelligence* (modelled after the behaviours of creatures that operate in swarms like birds, fish and bees; using a team of multiple simplistic agents working together to solve a complex problem), among many others.

Originally developed sometime in the 1960s, one of the earliest occurring members of these Biologically Inspired Algorithms in history is the *Genetic Algorithm* inspired by Charles Darwin's theory of evolution through natural selection (Coley, 1999). Progressing on through the latter quarter of the nineties marked revolutionary findings in the development of more AI technologies like evolutionary computation (Back et al., 1997) and the Artificial Neural Network (Jain et al., 1996) modelled after the inner workings of the brain. These algorithms have found great application in a variety of fields, but few findings made during that time have brought as many revolutionary insights to AI as the emergence of *swarm intelligence* (Kennedy et al., 2001).

Swarm intelligence is a method developed to allow exploitation of social behaviours by splitting the computational requirements for performing complex tasks and calculations across a group, or swarm, of simplistic inter-communicating individual agents. Inspiration for the design was taken from the collective behaviour of social organisms such as ants, termites, bees, birds, and fish. Two of the most popular algorithms that arose from implementations of swarm intelligence are the *Ant Colony Optimization* and the *Particle Swarm Optimization* algorithms (Blum & Li, 2008).

Ant Colony and Particle Swarm Optimization have both found success in application to discrete and continuous domains respectively. Ant Colony Optimization has been used as an approach to feature selection (Chen et al., 2010), heart disease prediction and classification (Khourdifi & Bahaj, 2019) and real-time routing problems (Samà et al., 2016). Particle Swarm Optimization has been used for multi-objective optimization

(Delgarm et al., 2016), clustering for high dimensional datasets (Esmin et al., 2015) and scalable optimization through social learning (Cheng & Jin, 2015). Work has also been done to bridge the gap in application domains between the two algorithms by introducing variations in design that allow the Ant colony to operate in continuous domains (Socha & Dorigo, 2008), and facilitating operation in discrete domains for the Particle swarm (Zhong et al., 2007). Along with this, comparative analysis has been performed on these algorithms to draw a better understanding of their strengths and weaknesses (Castillo et al., 2012; Selvi & Umarani, 2010). An age-old method used for general comparative analysis is the combinatorial optimization problem: *The Traveling Salesman Problem*; a study for which Ant Colony Optimization has had great accomplishments (Dorigo & Gambardella, 1997; Lin, 1965).

Through all of this use and analysis, advantages and drawbacks have been highlighted over the years in these algorithms which have led to the development of algorithm variants being built that try to address them. Hybrids have also been built, through which the methodologies of the given algorithms are combined in an effort to merge their strengths. A study done by Huang *et al.* (2013) demonstrated some of the techniques through which hybrid models can be built and, through the example of an Ant Colony and Particle Swarm hybrid optimization algorithm.

1.2 Research Project

Unfortunately, with the exception of Huang *et al.* (2013), studies conducted in the body of literature concerning hybrid optimization algorithms only ever document a single hybrid construction methodology. Perhaps through further research into this field, patterns and heuristics can be gleaned to direct the choice of hybridization methods justified by highlighted characteristics found in the base algorithms used. Extracting these patterns could, like the revolutionary Swarm Intelligence, open up new avenues for our understanding of AI.

This project aims to raise interest in this field of research by demonstrating the value that can be gained from hybridization despite the state-of-the-art advancements made in base algorithm versions over the years. This will be done through the research question:

“Can hybridization methods applied to biologically inspired optimization algorithms improve their efficiency in approximating a solution to the Travelling Salesman Problem?”

The chosen optimization algorithms used in this study are the *Genetic Algorithm* (GA), *Particle Swarm Optimization* (PSO) and *Ant Colony Optimization* (ACO), along with the 3 hybrid models created by combining their methodologies [ACO/GA, PSO/GA and PSO/ACO]. This inquiry was further divided into some more granular research sub-questions:

- *Methodological Understanding* – What is the State-Of-The-Art methodology for the chosen optimization algorithms? Are there any improvements that can be made?
- *Experimental Understanding* – What is the best configuration model for each algorithm given the problem domain used? What statistical test should be used to justify the conclusions drawn? When hybrid algorithms are compared against their base versions, which optimization algorithm is better?
- *Theoretical Understanding* – What further understandings of algorithm and general optimization methodology can be drawn from the experiment data?

1.3 Research Objectives

The hypothesis that this research project would aim to prove is:

“Using the Traveling Salesman Problem, when a Mann-Whitney U Test is done on the results between hybrid and base models, a hybrid algorithm will offer the best performance, when given standardized population size, number of maximum iterations, and a statistical significance threshold of 0.05”

The null hypothesis argues that since it was designed specifically to tackle combinatorial optimization problems like the Traveling Salesman Problems, then:

“Using the Traveling Salesman Problem, when a Mann-Whitney U Test is done on the results between hybrid and base models, no hybrid algorithm will offer the best performance, when given standardized population size, number of maximum iterations, and a statistical significance threshold of 0.05”

To test these hypotheses, a fair test between the best performing representative of all base algorithms operating in comparison with the hybrid models would have to be conducted. This comes with the sub-objective of determining the best representative for the base algorithms. To achieve all of this the following goals were defined:

1. To research each of the chosen Biologically Inspired Algorithms, exploring the different variations in implementation that have arisen over the years, to find the State-Of-The art variants to use for the experimentation.
2. To establish and justify an appropriate statistical test to be used for comparative analysis.
3. To define the parameters and methodologies that create the best performing representative, for each of the algorithm classes in this project domain, to be used in the final comparative analysis
4. To construct hybrid models based on the methodologies used in the base representatives.
5. To answer the research question by performing a comparative analysis between the results drawn from the hybrid and base algorithms.
6. To extract any extra methodological or theoretical understandings that can be gained from the experiment results.

1.4 Research Methodologies

To fulfil the research goals from Section 1.3, two research methods are utilized: secondary research (through a literature review) and empirical research (through implementation and evaluation of the findings from the review). The breakdown of the approach taken to solve those research goals mentioned in Section 1.3 is as follows:

1. Perform a literature review to research the chosen BIAs in order to find and understand the State-Of-The-Art variants in their design
2. Review also some of the most commonly used statistical tests to understand and justify any statistical tests performed in the study
3. For each of the main variations in algorithm design extracted from step 1, conduct empirical by implementing them in Python and running them against randomly generated Traveling Salesman maps to find the 3 best representative models for each of the algorithms that would be used in the final experiment.

4. Create the 3 Hybrid algorithms using the optimum methodologies extracted from step 3.
5. Using the test chosen from step 2 and representatives chosen from steps 3 and 4, conduct the final comparative analysis of the algorithms documenting any conclusion drawn.

1.5 Scope & Limitations

This study touches on interesting topics in the theory of computation like discrete and single-objective optimization, graph algorithm analysis, and the theory of randomized search heuristics. It also discusses machine learning theories, like artificial intelligence, biologically inspired optimization, multi-agent reinforcement learning and evolutionary algorithms. Finally, mathematical topics are also touched on, like mathematical modelling and optimization.

Unfortunately, due to monetary limitations over quarantine, it was decided to carry out the study using a borrowed college laptop having an Intel® Core™ i5-10210U CPU @1.60GHz 2.11GHz processor, a 16BG ram capacity, and a 64-bit Operating System. These computational constraints limited the experiment to problem graphs of size 50.

1.6 Document Outline

The sections in this dissertation are organised as follows:

- Chapter 2: A history and overview of the 3 chosen algorithms are presented, along with a description of the problem domain to which they will be applied. Also given are examples of the current state-of-the-art variations in their algorithm design, as well as details about the hybrids built from them.
- Chapter 3: The experimental design and research methods employed are discussed including an outline of the dataset used, configurations set, and the sub-topics focused on for the experiments.
- Chapter 4: The results and findings discerned from the experiments completed, structured by the subtopics extracted from the study, are reported.
- Chapter 5: A discussion of the results drawn from the experiments concerning the motive for this research project as well as its implication for future work is given.

2. LITERATURE REVIEW

This chapter presents a history and overview of three of the most popular algorithms in the biologically inspired algorithm family, along with a description of the problem domain they will be applied. Also given are examples of the current state-of-the-art variations in their algorithm

2.1 Combinatorial Optimization Problems

A combinatorial optimization problem is a problem of arranging a set of discrete variables (e.g., attributes, states, or values) in such a way that it minimizes, or maximizes, the desired result. In some cases, that goal includes eliminating some of those components, meaning that the number of elements to rearrange also becomes part of the problem (Kennedy et al., 2001). Combinatorial Optimization problems all come with a goal that is optimized towards an objective function, through which the solutions proposed can be critiqued. With the example of a company, having a machine that drills holes into printed circuit boards, that wants the machine to complete its job as fast as possible by minimizing the time taken to move the drill from one point to another, the problem can be explained as “what is the most efficient route for the machine to take?”, and the objective function would correspondingly be a measure of the distance travelled for any route/solution proposed. That is because, in this example, the total distance travelled serves as the metric through which a given solution can be critiqued against the optimization goal (Korte & Vygen, 2012).

Some examples of combinatorial optimization problems are *Bin-Packing*: organising items into a finite set of containers (Delorme et al., 2016); *Job-Shop Scheduling*: efficiently allocating shared resources over time to competing activities (Zhang et al., 2011); and *Boolean Satisfiability*: determining if there exists an interpretation that satisfies a given Boolean formula (Soeken et al., 2010). However, one of the most well-known Combinatorial Optimization Problems is the *Traveling Salesman Problem* (TSP) (Yousefikhoshbakht, 2021). The challenge of the TSP can be defined by the question: “Given a map of cities to visit and the distances between each pair of cities, what is the shortest round trip that can be made from a given origin city, visiting each city on the map exactly once, and returning to your starting position?”. Figure 1 is an example of

this, showing the problem map to solve (on the left) and the solution route (on the right).

The problem is characterised by two main conditions:

1. Each city must be visited exactly one time
2. The trip must conclude with a loop back to the starting position

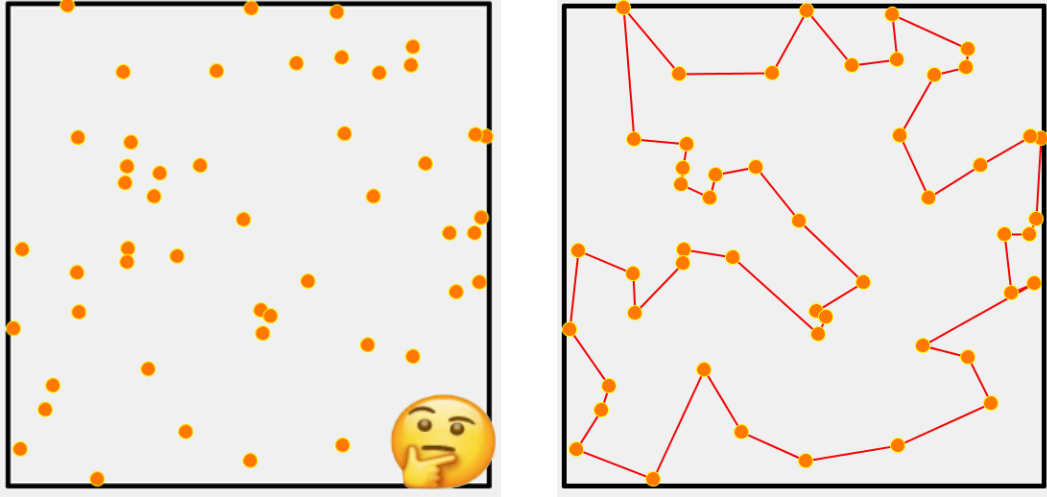


Figure 1: Example Traveling Salesman Problem

With this in mind, the optimum route/solution to the TSP can be described as ordering an itinerary of cities to visit $S_p = \{c_1, c_2, \dots, c_n\}$ in such a way that the sum of distances traversed while following the itinerary returns the smallest possible value.

$$f(S_p) = \sum_{i=1}^n d(S_p[i], S_p[(i + 1) \bmod n])$$

Equation 1: TSP Distance calculation

Where $d(c, c')$ means the distance between cities c and c' , if the location of city $c = (x, y)$ and $c' = (x', y')$, then $d(c, c') = \sqrt{(x - x')^2 + (y - y')^2}$. $S_p[x]$ refers to the city at position x on the itinerary S_p . The introduction of the modulo operator enables the loop to return to the beginning after all cities on the itinerary have been addressed. Since the goal is to find the smallest possible total distance, the calculation in Equation 1 may be inverted so that the answer returned can be used as a score for the proposed solution.

$$f'(S_p) = \frac{1}{f(S_p)}$$

Equation 2: TSP Objective Function

According to Equation 2, the smaller the total distance travelled in a given TSP solution, the larger the score that would be awarded to that solution. Both Equations 1 and 2 are usable for the objective function and are both compatible with the algorithms used. However, Equation 2 was chosen for this study because it best encapsulates the meaning of the objective function: the smaller, the better.

The TSP has been said to be easy to describe but difficult to solve (Hoffman & Padberg, 2001). While ACO was developed specifically to tackle problems like this (Dorigo & Stützle, 2019), over the years, the GA has also been used to accomplish this (Braun, 1991; Moon et al., 2002). The PSO on the other hand, posed the greatest challenge because it was not designed for combinatorial optimization. Nevertheless, work has been done to adapt the algorithm so that it can handle the TSP (Wang et al., 2003; Yousefikhoshbakht, 2021).

2.2 Optimization Methodology

With a clear understanding of a problem domain complete, this section offers a breakdown of the general optimization methodology used to tackle the problem domain.

2.2.1 Search Space

When dealing with optimization problems, the array of possible valid solutions is often illustrated as a *search space*, or *search landscape*, which exists on an n -dimensional plane, for which each point on that search space represents a possible valid solution and the dimensions of the plane correspond to the different variables existing in that problem domain (Mirjalili, 2019b). Solutions existing in relatively close locations to one other in the search space would receive similar scores from the objective function because of the close proximity of their input variable values which determine their dimensional

location. Figure 2 demonstrates an example search space showing a plane of possible solutions. On the left, the combination of input variables x and y are represented as the x and z axes, and the score given from the objective function for those possible inputs $f(x, y)$ is used as the y axis. On the right, only the x and y axes are used while the score is represented by the colour. The gaps in the search space depicted would stand for solution regions that are not valid given the constraints of the objective function.

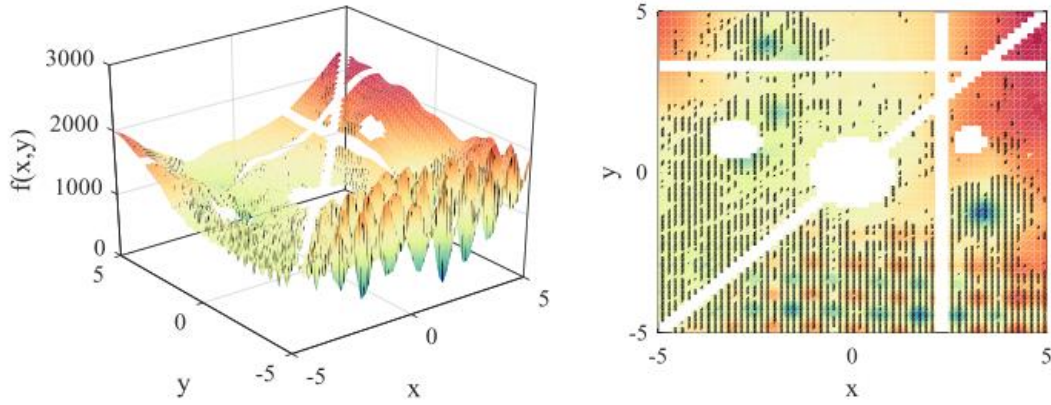


Figure 2: Example Search Space/Landscape (Mirjalili, 2019b)

In Section 2.1, a mathematical model demonstrating how a TSP solution can be tested was constructed. The objective function in Equation 2, takes in a possible solution (sequence of cities) as an input and returns a score, through which the efficacy of that solution can be measured. Hence, the role that optimization algorithms play with respect to this objective function is to devise an input solution, to supply to the function, which returns the highest score possible. Given the conditions included in the TSP problem definition and the nature of combinatorial optimization, the number of possible valid solutions that can be accepted into the objective function is finite. So in other words, the role of the optimization algorithms is to traverse the finite search space seeking the highest peak (a location/position for which the objective function returns the highest score) (Blum & Li, 2008).

2.2.2 Search Method

The algorithms operate by locating and exploring promising regions within the search space. But, when a peak is found, how is it determined whether this location is the highest in the entire search space? This important consideration of *Local Optimum* vs *Global Optimum* is critical to the optimization algorithm development process as it determines the adequacy of a given algorithm design. The term *local optimum* refers to the best solution found in a specific region of the search space while the *global optimum* is the best solution in the entire search space as demonstrated in Figure 3.

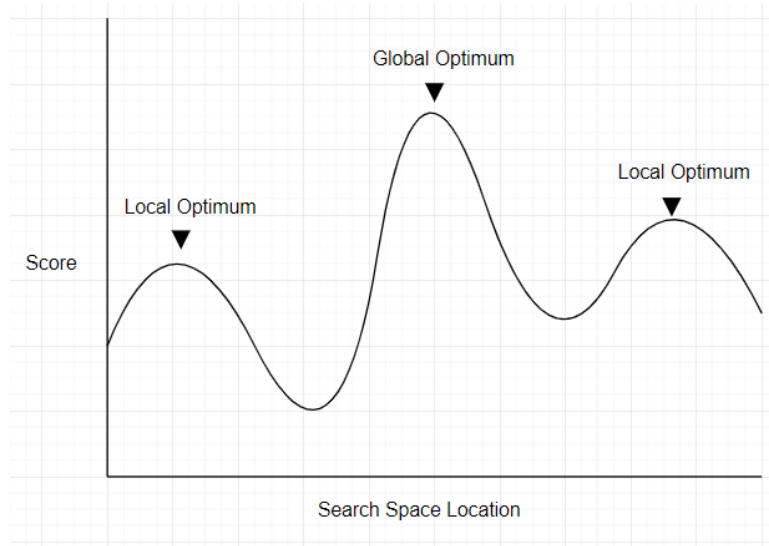


Figure 3: Local vs Global Optimums

As each optimization algorithm sends out its agents to search various regions of the search space, this consideration is addressed in through the balancing of two important mechanisms: *Diversification* and *Intensification* (Mirjalili, 2019b; Thangaraj et al., 2011). As suggested by the name, diversification involves exploring the regions of the search space. This can be accomplished by making frequent or large changes to the composition of the algorithm agents to scatter their positions on the search space. This added stochastic element prevents these agents from converging prematurely on optimum location, allowing them the opportunity to find other potentially better avenues of the search space. On the other hand, the intensification mechanism offers the opposite behaviour, through which all members of the algorithm converge towards the optimum location found so far. In this way, the opportunity is given to reassess a given optimum before algorithm convergence.

In the GA, *Selection Pressure* refers to the “degree to which the better individuals are favoured: the higher the selection pressure, the more the better individuals are favoured” (Miller & Goldberg, 1995). The selection pressure is the driving force for improvement over succeeding generations in the GA and it is a primary influence when it comes to GA convergence. In this case, diversification is implemented by lowering the selection pressure and intensification is done by raising it. The Swarm Intelligence algorithms PSO and ACO on the other hand, still use these two mechanisms but under different names: *Exploration* (diversification) and *Exploitation* (intensification) (Thangaraj et al., 2011).

For all algorithms traversing the search space in seeking an optimum, their search is brought to a conclusion when some pre-determined criteria are reached, and the best solution found at that point is returned. The two most popular criteria for search termination are convergence, when the majority of the members of the algorithm’s population converge on a single solution (Miller & Goldberg, 1995), and through use of a search counter, when the maximum number of algorithm iterations allowed is reached (Ahmadi & Dincer, 2010). It should be noted that this ‘best solution’ does not always mean the global optimum, but rather the best optimum found when the stop criteria were reached.

2.2.3 Evaluation Metric

Two common metrics used for critiquing optimization algorithms are the best solution scores found, and the number of iterations used to find them (Samà et al., 2016). In this study, however, the research question aimed to find the algorithm offering the best performance which, in popular optimization analysis, is a composite metric blending the earlier two metrics (Mandloi & Bhatia, 2016). A line-graph mapping between the best solution scores found against the algorithm iterations taken is commonly called the algorithm’s *speed-time* curve (Luan et al., 2019; Yao et al., 2008) and the performance metric can be denoted as the *Area Under the Curve* (AUC) as shown in Figure 4 and

calculated using the trapezoidal function¹. Since the chosen objective function for this study (Equation 2) aims to be maximized, when comparing the performance of multiple algorithms, the algorithm with the overall highest performance value is the best performing algorithm.

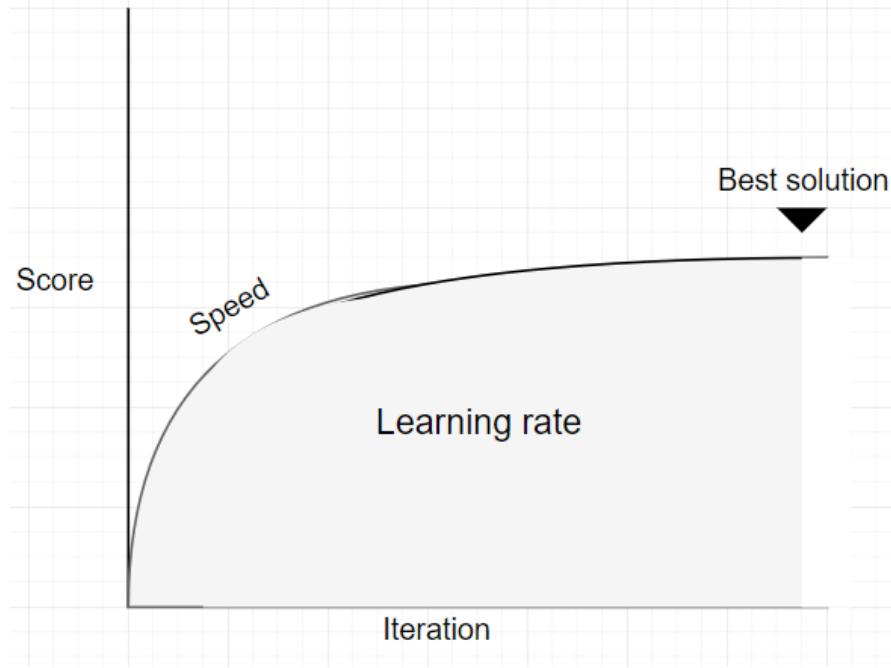


Figure 4: Algorithm Learning Rate

2.2.4 Statistical Analysis

As the algorithms construct solutions to the problems given, performance or AUC data is drawn, however for the sake of this project, a method through which a winner can be declared and statistically justified using this collected data needed to be established. This introduces the concept of *Statistical Significance* which aims to measure the degree of data conformance to the null hypothesis through the metric p -value. The most popular threshold used is 0.05 such that ' $p < 0.05$ ' shows a lack of support for the null

¹ An explanation of how to use the trapezoidal function can be found here:

<https://math24.net/trapezoidal-rule.html>

<https://www.rdocumentation.org/packages/pracma/versions/1.9.9/topics/trapz>

hypothesis by the data given, and ' $p \geq 0.05$ ' suggests that the data given is not statistically strong enough to reject the null hypothesis (Di Leo & Sardanelli, 2020).

A *statistical test* refers to an established method used to calculate this p -value, and many have been developed over time. Each of these tests comes with an, often unique, set of dataset assumptions or preconditions that need to be fulfilled for valid use. All algorithms used in this project operate on TSP maps, which are statistically labelled the *independent variable*, and the output metric used is their performance AUC, which is statistically labelled the *dependent variable*. Even so, the AUC returned by each algorithm occurs independently of any other algorithm run in that comparative test. Finally, the aim of the analysis is not one of correlation but rather to declare the winning algorithm by establishing a difference between their result data. These dataset characteristics preserve three popular pathways for statistical analysis: the independent *T-test*, *Analysis of Variance (ANOVA)* and the *Mann-Whitney U test*.

The classical *T-test*² is a means of comparing the data between two groups and it is presently still widely in use (Kelter, 2020; Kim, 2015). The assumption required for use of the independent T-test is that there should be exactly two independently homogeneous groups to compare, each having an approximately normal distribution with no significant outliers. Homogeneity here refers to two datasets that are using the same metric and coming from similar sources. An example of this would be measuring the heights of the two groups: male and female. In this case, the heights recorded for the male group would also be independent or un-reliant on the heights recorded for the female group (as long as the participants aren't blood-related of course).

Similarly, the *ANOVA*³ test assumes a normal distribution of the dataset as well as independency between the classes of the tested dataset (Vázquez et al., 2001). It targets the variance (the spread between numbers in the dataset) to identify if there is a

² See: <https://statistics.laerd.com/spss-tutorials/independent-t-test-using-spss-statistics.php>

³ See: <https://statistics.laerd.com/spss-tutorials/one-way-anova-using-spss-statistics.php>

significant difference between the means of three or more independent groups (Demšar, 2006).

Another alternative would be the *Mann-Whitney U test*⁴. Originally worked out by Mann & Whitney (1947) and then further refined by Wilcoxon (1992), this test is often called the Wilcoxon sum of ranks test (Nachar, 2008). The assumptions required for this test are, firstly, that two independent samples are homogeneous. The variables have to be continuous variables with similar distributions. When comparing the Independent T-test and the Mann-Whitney U test, Nachar (2008) notes that when averages are the same between the independent datasets used but their variances increasingly differ from one another, then the t-test becomes an increasingly more reliable method. Nonetheless, if the postulates can be met, they found the Mann-Whitney U test to be an excellent alternative. Demšar (2006) adds that due to the use of rank rather than value, outliers have less of an effect on tests like this and the Wilcoxon signed-ranks test, which gives them more accuracy than the standard T-test and ANOVA test in this case.

2.3 State-Of-The-Art in Biological Optimization

2.3.1 Genetic Algorithm overview

GA begins with an encoding of the problem domain as a list of chromosomes, representing an initial population, as an arbitrary set of trial solutions. These opening solutions were classically obtained through simple randomized generation to provide unique starting points for each member of the population within the search space (Johnson & Rahmat-Samii, 1997). In time, extra techniques like the Gaussian random distribution have also been implemented at this step to maximise diversity in the initial population (Mirjalili, 2019a). Moreover, due to the robustness and diversity in construction offered by the completed algorithm, simple random generation remains the most common approach (Katoch et al., 2021).

⁴ See: <https://statistics.laerd.com/spss-tutorials/mann-whitney-u-test-using-spss-statistics.php>

After initialization, a combination of two techniques called *evaluation* and *fitness allocation* is used to award each member a measure of ‘attractiveness’ (also called fitness) in such a way that those chromosomes which represent a better solution to the target problem are given more chances to ‘reproduce’ than those chromosomes which are poorer solutions. Evaluation, performed through the Evaluation Function, provides a means to measure the performance of a given individual regarding a set of parameters extracted from the problem domain (i.e., the TSP objective function given in Equation 2). Fitness Allocation, performed through the Fitness Function, then transforms that evaluated score into an allocation of reproductive opportunities. The ‘attractiveness’ of any given individual is typically assigned relative to the current population (Whitley, 1994). Using this combined process (evaluation and fitness allocation), a *selection operator* chooses the best individuals from the population and compiles them into a mating pool. It is then the *breeding operator’s* task then to mix the genetic components of those chromosome members in that mating pool to make the next generation.

At this point, the issue of selection pressure, discussed in Section 2.2, comes into play. Emphasis must not be overly placed on these best individuals when allocating mating opportunities until it is more likely that their chromosome patterns represent global, rather than local, optima. This is especially apparent after the initialization step because of the reasonably low chances of finding the Global Optimum through random initialization. Rather, the selection and breeding operators aim to progressively extract and combine favourable parts of the genetic codes of the population while discarding the unfavourable. As the generations go by, through this iterative process, these favourable chromosome components would gradually become more prominent in the population set until a consensus is eventually made on an optimum component set.

As part of the last step of the *breeding operation* before the creation of the next generation is finalized, a very important component of the GA is introduced: *mutation*. So far, the GA process begins with a varied initial population and, through its selection and breeding mechanisms, isolates desirable gene sequences within the chromosomes to focus on, making these components gradually more prominent as generations progress. However, it should be noted that there is no guarantee of having the globally best genetic components within the initial populations of the algorithm; hinting toward the

importance of mutation. Mutation can be seen as the operator charged with maintaining the genetic diversity of the population as it aims to preserve the diversity embodied in the initial generation. It does this by introducing new information into the genetic sequence, allowing the population to ‘leapfrog’ over potential sticking points. As a concluding step, mutations are randomly assigned under an appropriately low percentage to allow more variability in the search space (Coley, 1999; Whitley, 1994). Figure 5 displays a flowchart reviewing the main structure of a GA.

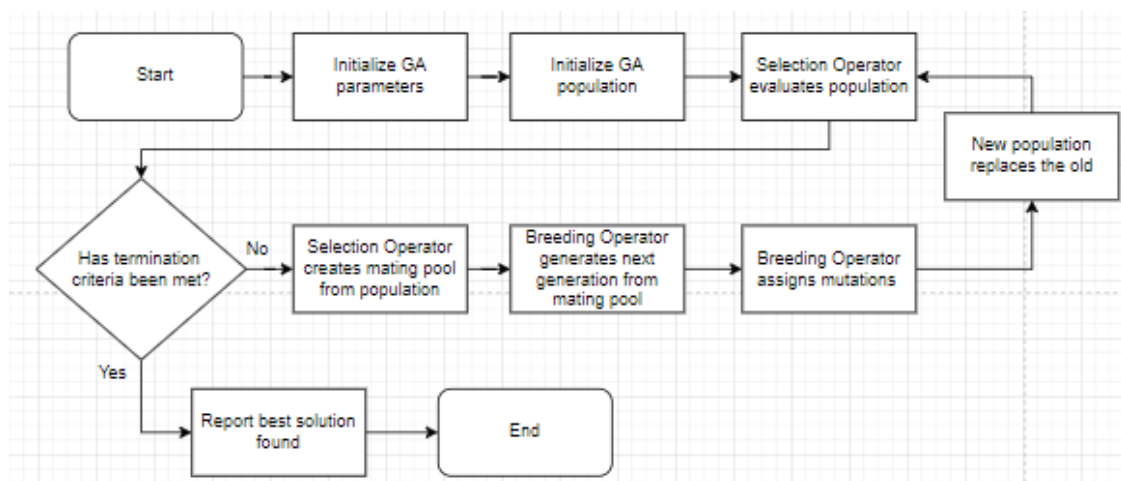


Figure 5: Genetic Algorithm Flowchart

2.3.2 The Selection Operator: Fitness Function Variants

The main variation in GA composition techniques occurs with the Fitness Function of the Selection Operator. The preceding section highlighted that the role of this function is to convert the scores given from population evaluation into an allocation of reproductive opportunities. In other words, to convert the evaluation score into a measure of *fitness* (or attractiveness) as a new *fitness score*. This can be done in a number of ways:

2.3.2.1 Roulette Wheel Selection

As the title suggests, the concept of natural selection is simulated using a roulette wheel type selection process. The *fitness score* in roulette wheel selection refers to the number of slots allotted on the wheel to each member of the population, and it is calculated

relative to each member's evaluation score. The probability of selecting a member of the population in roulette wheel selection, shown in Equation 3, can be viewed as the probability of the selection pointer landing on that member after a roulette wheel, with the number of slots for each member proportional to their fitness score, is spun. Figure 6 is a depiction of this (Razali & Geraghty, 2011).

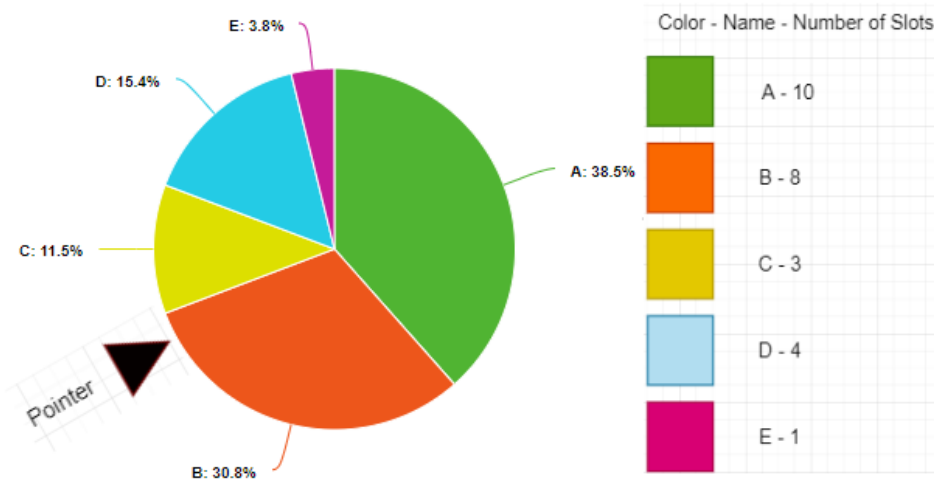


Figure 6: Roulette Wheel Sampling

With the list of fitness values for all members of the population f_1, f_2, \dots, f_n the selection probability for any individual i is:

$$\frac{f_i}{\sum_{j=1}^n f_j}$$

Equation 3: Roulette Wheel Selection Probability

To calculate that fitness score, the evaluation scores of all members of the population are usually normalized for algorithm consistency (Equation 4), and then scaled according to how large the wheel is desired to be. For example, if 10 slots are the largest that can be allotted on the wheel and 1 is the smallest, then those normalized values ranging between 0 and 1 can be scaled up to the range 1-10 using Equation 5.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Equation 4: Min-Max Normalization

$$CR(x, cur, new) = \min(new) + \left(length(new) * \frac{x - \min(cur)}{length(cur)} \right)$$

Equation 5: Converting Ranges

In Equation 5, x is the value to convert, 'new' and 'cur' are respectively the new and current ranges that x should be mapped across, $\min()$ returns the lower boundary value of the given range, and $length()$ returns the length of the given range.

Whitley (1994) offered the suggestion, which was used in this study, to deal with any remainder values generated after using Equations 4 and 5. He suggested using those remainders as a probability for offering a bonus slot to that member. Equation 6 details how this could be done.

$$f_i = floor(f_i) + (1 * prob(r))$$

Equation 6: Fitness Score

Here, f_i is the fitness score, r is its decimal value, $floor()$ returns the value given rounded down, and $prob()$ returns 1 with a probability of the value given or else it returns 0.

2.3.2.2 Stochastic Universal Sampling

Over time, weaknesses were highlighted in the workings of the roulette wheel selection, and variations of that fitness function emerged to solve those problems. One of those was the problem of inefficiency, for which the stochastic universal sampling function was developed to tackle. In the roulette wheel selection, there is a requirement for multiple spins of the wheel before a selected breeding pool can be compiled. Grefenstette (2013) described *stochastic universal sampling* as an $O(N)$ sampling algorithm that can

achieve N samples in a single traversal. It works in a similar manner to *roulette wheel sampling* but, by having multiple selection pointers evenly spaced around the wheel, multiple members can be selected simultaneously leading to significantly fewer or even a single spin. This technique is depicted in Figure 7.

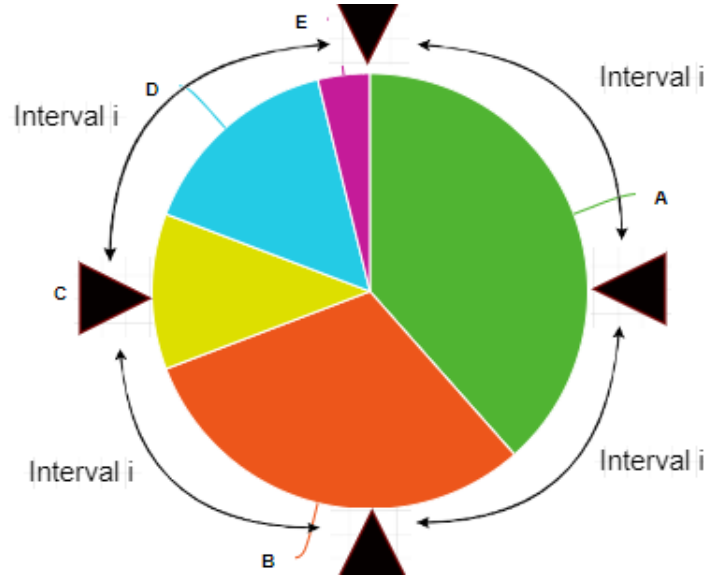


Figure 7: Stochastic Universal Sampling

2.3.2.3 Rank-Based Sampling

Another problem noted with the *roulette wheel sampling* technique was its selection pressure which was arguably too high (Razali & Geraghty, 2011). As seen in Figures 5 and 6, because of the great scores found with individuals A and B when compared to the others, they were assigned portions that nearly dominate the entire wheel, leaving little room for selection chances for the other individuals. The goal of *rank-based sampling* (also known as Linear Rank Selection) (Mirjalili, 2019a) is to tackle this by performing the allotment proportional to each individual's 'rank' rather than their evaluation score directly. Using their evaluation scores, all members of the population are ranked from 1st to Nth and then fitness is distributed using those assigned ranks, presenting a more evenly distributed wheel to select from. In this case, Equations 6 and 7 can be used on the member's rank, rather than their evaluation score, to convert it to a fitness score. Figure 8 demonstrates the new proportions given when rank-based wheel allocation is used.

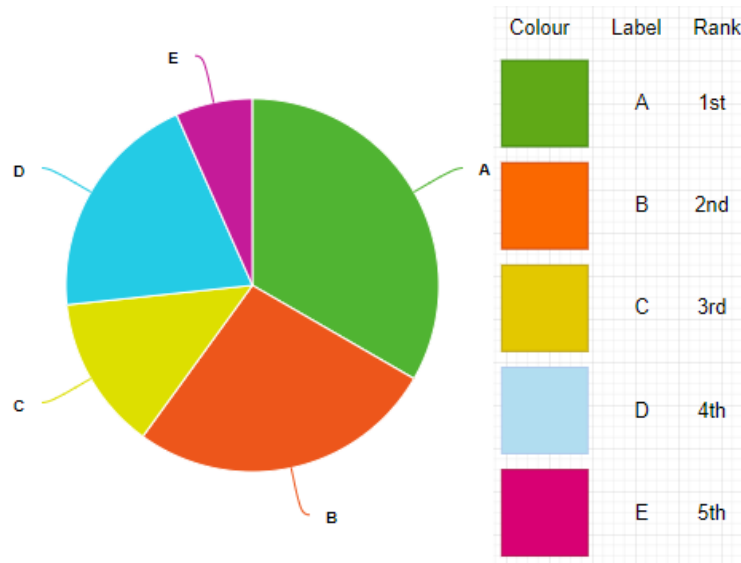


Figure 8: Rank-Based Sampling

2.3.2.4 Tournament Sampling

Unlike the variations mentioned above which followed the pattern of the roulette wheel selection, tournament sampling uses a completely different mechanism for selecting a mating pool for breeding. In the tournament sampling, pairs of individuals, each chosen randomly from the population, are put against each other in a tournament. The deterministically selected winner of that tournament is then copied directly into the mating pool for breeding. The winner of a competition is selected by comparing the evaluated scores of each member's proposed route (Back et al., 2000). Although benefits have been found with tournament sampling when used on small problem sets (Razali & Geraghty, 2011), it has been highlighted that tournament sampling also runs into a similar problem as the roulette wheel selection: that its' selection pressure is too high. Different techniques have been tested over the years to try to remedy that. For example, Miller & Goldberg (1995) experimented with the effects of noise in the tournament sampling applied to the scores of the members before each competition in order to test their convergence approximation equations.

2.3.3 The Breeding Operator: Crossover variants

In the natural inspiration for the GA, chromosomes in the genes of a male and female are combined to produce the children's chromosomes. The same technique is employed by the GA through the crossover operator. Though not as fully-fledged as the variations found in the previous section 2.3.2, there exist some minor variations in the way genetic crossovers can be implemented in the GA. The two most common methods are the single- and double-point crossover techniques which are depicted in Figure 9. In the single-point crossover technique, a random swap point is chosen along the chromosomes of the 2 parents and their genetic code from that point onwards is swapped in order to create 2 children. Double-point crossover operates the same except that the genetic code between two points is swapped. Other example variations include Uniform Crossover, 3 Point Crossover, and Cycle Crossover (Mirjalili, 2019a).

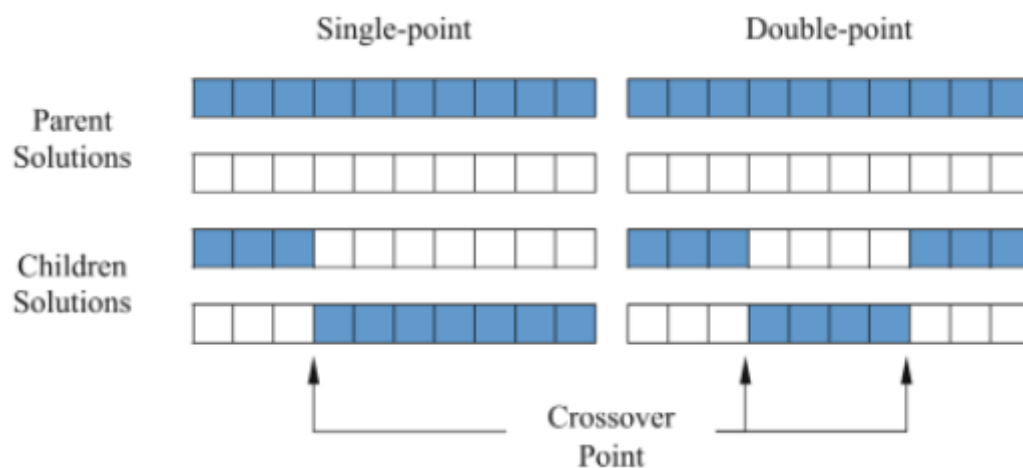


Figure 9: Crossover Operator (Mirjalili, 2019a)

2.3.4 Genetic Algorithm Enhancers

Many extensions that layer over the basic GA operation to improve its functionality have been implemented such as assigning dominant and recessive genes, and the concept of niche and speciation. Two of the most popular ones in use today are *Elitism* (Ahn & Ramakrishna, 2003) and *Steady-State* (Johnson & Rahmat-Samii, 1997). The natural inspiration for these ties back into the idea of 'survival of the fittest' in which fitter individuals are *preserved*, carrying on for longer than their weaker contemporaries.

2.3.4.1 Elitism

Due to the stochastic nature of the GA, it is possible for the next generation to have the best individual with lower fitness than the preceding generation's best representative. Elitism is a technique, developed to address this concern, in which the fittest 'elite percentage' of a generation is retained into the next generation. In this experiment, when elitism is used, for each iteration, the members of the population are evaluated and ordered by their score. Then, the top-scoring group, whose size is decided by the elite percentage assigned for the operation, is kept intact while the others are replaced by their children (Johnson & Rahmat-Samii, 1997).

2.3.4.2 Steady-State

This function takes the elitist approach even further and can be thought of as the overlapping of generations. In the steady-state mechanism, when offspring are made, rather than replacing their parents, they replace the members of the population that are the lowest in fitness. "The result is a more aggressive search that in practice is often quite effective" (Whitley, 1994). There are a few methods of implementing the steady-state function. One method is by storing the new child generation in a separate list and then copying over by overriding selected weaker parents with the fitter children. Another method is by appending the children to the end of the parent's list, temporarily creating an enlarged population size. Then, using their evaluation scores, weaker members of this extended population are removed until the population size returns to its origin. It should also be noted that since Elitism also aims to retain the best of each generation for the next, the combined use of Steady-State and Elitism brings redundancy (Johnson & Rahmat-Samii, 1997).

2.3.5 Particle Swarm Optimization Overview

Original models of the PSO aiming to imitate bird movement, found that their models were too rigid. The flocks they studied were able to follow the general flow of the group but were found to often change directions suddenly through observed scattering and regrouping behaviours. Simply programming particles to follow one another could not capture this element of "craziness" because then the group would quickly settle on a unanimous, unchanging direction (Kennedy & Eberhart, 1995). Through refinement,

Kennedy & Eberhart (1995) settled on the two most important PSO variables still in use today: *pbest* and *gbest*.

In the PSO, each member of the swarm is composed of 3, D-dimensional vectors (D being the number of dimensions within the given search space) (Poli et al., 2007). These vectors store the particle's current position \vec{x}_i , the personal best position or *pbest* found in the particle's history \vec{p}_i and the particle's current velocity \vec{v}_i . At the start of the algorithm, the particles are initiated at random locations within the search space and, using these 3 local variables, along with a 4th vector shared by all particles storing the global best position or *gbest* found in the entire algorithm history \vec{p}_g , the particle navigates its search space. For each iteration of the algorithm, the velocity for each particle is calculated relative to its current velocity (inertia), the distance from its *pbest*, and the distance from its *gbest*. Then that velocity is used to update the position of the particle within the search space. Finally, at the end of each iteration, each particle's new solution is assessed (Equation 2) and the *pbest* and *gbest* variables are adjusted accordingly. The classical velocity calculation equation for the PSO is detailed in Equation 7 (Das et al., 2008).

$$\vec{v}_i = \omega * \vec{v}_i + c_1 r_1 (\vec{p}_i - \vec{x}_i) + c_2 r_2 (\vec{p}_g - \vec{x}_i)$$

Equation 7: Particle Velocity Calculation

Here, ω is the inertia weight and c_1 and c_2 are weights respectively managing the balance between exploration, known as “self-confidence”, and exploitation, also known as “swarm-confidence”. Included in this calculation are the variables r_1 and r_2 which are both random numbers between [0,1], generated at each iteration, introducing a stochastic element to the search.

The *pbest* variable serves as the particle's memory, and it is used to simulate independent thinking for each particle. Particle *exploration* is carried out through the combined use of this variable and the application of particle inertia. The *gbest*, on the other hand, is the collective best solution found globally in the algorithm's history across all particles. This variable is used in the *exploitation* process allowing particles to converge on the

optimum solution (Thangaraj et al., 2011). As discussed in Section 2.2, these processes of exploration and exploitation need to be balanced as a concern shown toward local vs global optimums.

Particle Inertia is an important concept in the workings of the PSO. Das *et al.* (2008) argue that the concept of velocity, used for calculating particle movement through the search space, is rendered completely void if there is no inertia included in the calculation. As suggested by its title, inertia is a mechanism, through which a particle keeps some record of its past velocity (speed and direction of travel) to be applied when calculating its new velocity. This mechanism is managed by the inertia weight ω , which is typically set to higher values (≥ 3) like 0.8 (Shi & Eberhart, 1998). Techniques like simulating raising the viscosity of the environment traversed by the particles, by linearly decreasing from a higher $\omega = 0.9$ to a lower $\omega = 0.4$, have also been found to be effective (Shi & Eberhart, 2001). Figure 10 displays a flowchart reviewing the main structure of a PSO algorithm.

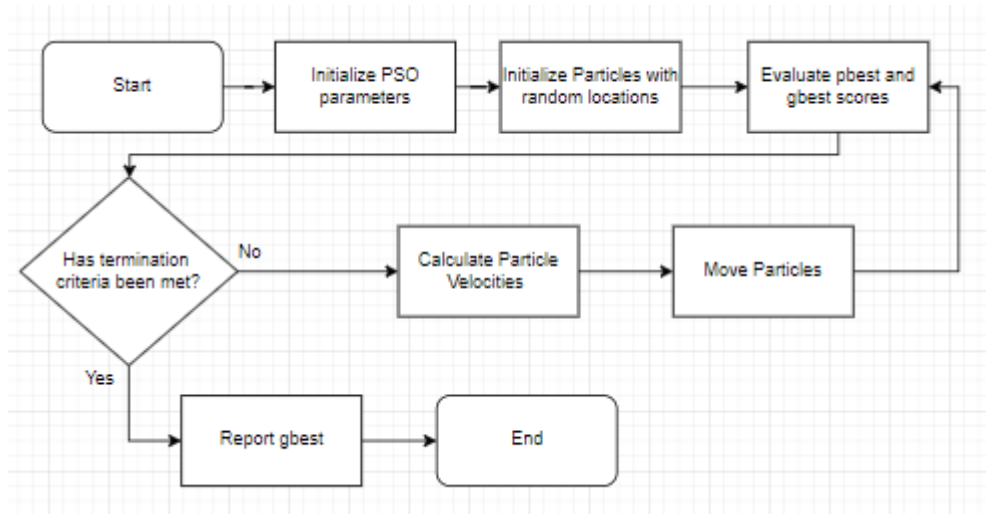


Figure 10: Particle Swarm Optimization Flowchart

2.3.6 Particle Swarm Optimization in Discrete Domains

For the TSP with a finite discrete search space, the classical PSO methodology had to be adapted because it was originally designed for continuous domains. Research has been done to find ways in which this adaptation can be made (Zhong et al., 2007). An interesting solution was the one proposed by Wang *et al.* (2003). There, they represented

the position vectors, used within the TSP domain, as a sequence of cities to visit $\vec{x}_i = \{i_1, i_2, i_3 \dots i_c\}$, where c is the number of cities on the map. The movement vectors are then represented by swap operators defined as $SO(i_1, i_2)$ such that, when applied to the position vector \vec{x}_i , it swaps the location that the cities i_1 and i_2 within the vector sequence. This creates a completely new sequence \vec{x}_j which can be treated as the new location vector for the particle after the movement vector SO was applied to it, $\vec{x}_j = \vec{x}_i \oplus SO$. A velocity vector can contain any number of swap operators compiled together as a *Swap Sequence*, $\vec{v}_i = \{SO_1, SO_2, SO_3 \dots SO_n\}$, which can then be applied on any location vector to bring it to another position within the search space. Figure 11 illustrates how a movement vector \vec{v}_i is applied on a position vector \vec{x}_i creating a new position vector \vec{x}_j . With this understanding in mind, the velocity needed to bring an example particle from its current location to its *pbest*, $(\vec{p}_i - \vec{x}_i)$, can be understood as the question: What swaps to my current sequence of cities are needed until it becomes the *pbest* sequence?

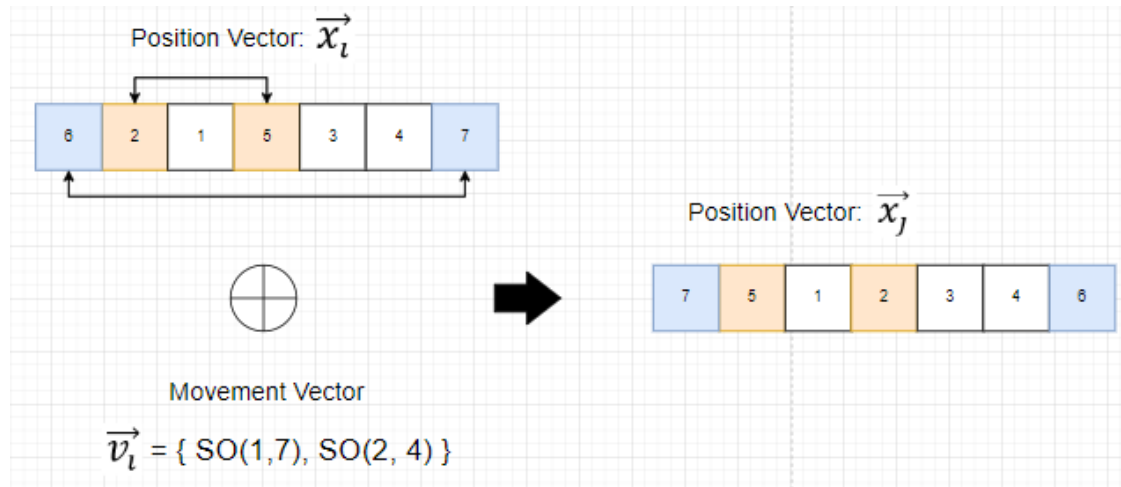


Figure 11: Applying Movement Vector to Position Vector for a Particle

Another big consideration in this application of the algorithm is how the weights are represented. In normal velocity vectors, simple vector scaling is done by multiplying it by the weights assigned. However, with our new representation of the velocity vector, the application of weights needs to be rethought. To continue the development of their proposed model, Wang *et al.* (2003) repurposed the weights used in the algorithm as

inclusion probabilities for each of the Swap Operators. When a weight is applied to a movement vector or Swap Sequence, the weight stands for the probability of keeping each of the Swap Operators in that Swap Sequence. Any Swap Operator failing the probability check is deleted from the Sequence. This is demonstrated in Equation 8 where $P(SO|\omega)$ is an operator that returns SO with a probability of ω , otherwise, it returns nothing.

$$\omega * \vec{v}_i = \sum_{SO \in \vec{v}_i} P(SO|\omega)$$

Equation 8: Weight Applied on Particle Swap Sequence

Finally, Wang *et al.* (2003) proposed an update to the *Particle Swarm* vector calculation given in Equation 7 that encapsulated all the concepts stated above. Originally, the movement vectors represented simple directions of travel, which could be scaled in magnitude by weights c_1 and c_2 , until the particle eventually reached its desired location. However, in this new adaptation, the movement vectors encapsulate the complete transformation needed to move between positions in the search space. As such, the magnitude weights lose their meaning, and so, were removed from the model proposed by Wang *et al.* (2003). Their updated velocity calculation is given in Equation 9. However, it was noted that no justification was given for the removal of the inertia from the calculation.

$$\vec{x}_i = \vec{x}_i \oplus r_1 * (\vec{p}_i - \vec{x}_i) \oplus r_2 * (\vec{p}_g - \vec{x}_i)$$

Equation 9: Adapted Particle Movement Calculation

2.3.7 Modified Particle Swarm Optimization

Yousefikhoshbakht, (2021) found optimization problems with the application of the PSO adaptation from Section 2.3.6 to industry services, in which PSO has been applied (Qolomany et al., 2020), due to premature convergence on local optimums. Some of the application challenges highlighted in that domain were: the large size of problems that

managers face daily, the importance rankings of the different problems based on user/customer attention, and the consistency in answers returned from the various manager and customer problems. A balance needed to be found between local searches for susceptible areas and global best searches, which they tackled through their proposed PSO variant named *modified particle swarm optimization*. There, they introduced another important variable called *gcbest* which refers to the best solution found across all particles for the current iteration.

To track the use of *gcbest* a variable a_1 , bound between a min (*%alpha*) and max (*%beta*), was used along with an accompanying inverse $a_2 = (1 - a_1)$. At the start of the algorithm, the variable a_1 begins with a value of *%alpha*, and as the iterations increase, it linearly progresses towards the value of *%beta*. In this technique, a_1 refers to the probability that the original *gbest* will be used for this iteration's movement vector calculation step, while its inverse a_2 is the probability of using the new *gcbest* instead. Yousefikhoshbakht, (2021) found that modifications to the classical version increased the algorithm quality, obtaining "excellent answers".

2.3.8 Ant Colony Optimization Overview

Dorigo & Blum (2005) defined the framework of the basic ACO as an iterative method in which exploration of the optimization problem search space is done using model ants constructing solutions by exploiting a given pheromone model. The algorithm was built to operate on combinatorial graph-like problems and the ants generated by the algorithm are tasked to traverse the graph's edges constructing solutions (S_p) to the problem based on their paths taken, updating the pheromone levels for each path traversed. Once complete, the solutions returned from an ant, can come in the form of a sequence of edges that the ant used when traversing the graph, $S_p = \{e_{ij}, e_{jk}, e_{kl} \dots e_{mn}\}$ where i, j, k, m and n are vertices on the map, and e_{xy} denotes an example edge connecting 'from' vertex x 'to' vertex y . In this case, the vertex location of an ant at any given time is the 'to' vertex of the current last edge of the solution it is constructing (i.e., the last edge that it travelled on) (Dorigo et al., 2006). The ant's solutions can alternatively be stored as the sequence of vertices reached as the ant traversed the map, $S_p = \{v_1, v_2, v_3, \dots, v_n\}$. In which case, the vertex location of an ant at any given time is the

last vertex in its current solution sequence. Though the first method is more prevalent in literature, both are logically equivalent, so it does not matter which one is chosen. The ACO formulas presented in this dissertation will use the second representation (solutions as a list of cities).

Based on the layout of the map and connections between vertices (e.g., directed/undirected graph), there is a finite set of valid choices that an ant can make from a given location on the map $N(S_p)$. The combinatorial optimization problem that the algorithm is tackling, in this case, the TSP, can also play a part in determining the validity of a solution component choice. For example, condition 1 of the TSP was that no repeats within the solution sequence were allowed.

For each iteration of the algorithm, the set of m ants initially starts with empty solutions and are each given some arbitrary starting vertex i , from which to begin building their solutions $S_p = \{i\}$. Then, for each step in constructing a solution for that iteration, the ant chooses the next valid vertex to visit $j \in N(S_p) \subseteq V$, and appends it to its current solution. Again, $N(S_p)$ represents the list of valid vertex choices, given the current solution list of the ant S_p , which is a subset of the complete list of vertexes on the map V and of which the solution component j (or in this example case: vertex j) is an element. If there are no more valid solution components that can be chosen $N(S_p) = \emptyset$, then the ant's solution can be treated as complete and some extra checks may also be made to ensure the validity of the completed solution within the problem domain of the study (Dorigo & Stützle, 2019).

The final step of the algorithm is the *pheromone update*. The goal of the pheromone update is to make the solution components belonging to good solutions when encountered, more attractive to future ants. However, with consideration of local vs global optimums, the pheromone update should avoid causing a too rapid convergence of the algorithm towards a local, sub-optimal, region of the search space. To accomplish this, two mechanisms are put into play. First is *pheromone deposit*, where pheromones are added to edges traversed by each ant with a pheromone strength relative to how good their completed solution was. Usually used for this, is an evaluation function that awards ants performing better, a higher score than those that are lower in fitness (Dorigo &

Stützle, 2019). Fortunately, the TSP objective function chosen in Section 2.1 does exactly this, so its value returned can be used as the pheromone strength for each ant. Equation 10 demonstrates this using τ_{ij} as the pheromone level of the edge from vertex i to vertex j and $f'(S_p)$ as the pheromone level deposited by the ant on that edge, taken from Equation 2.

$$\tau_{ij} = \tau_{ij} + f'(S_p)$$

Equation 10: Ant Pheromone Deposit

The second mechanism used is *pheromone evaporation*, where the pheromone levels are reduced across all edges. This serves as a method through which the algorithm gradually ‘forgets’ previous best solutions, favouring exploration of new areas of the search space (Dorigo & Stützle, 2019; Socha & Dorigo, 2008). For this purpose, an evaporation rate ρ is used to simulate pheromone evaporation across all edges. The complete equation for the pheromone levels of each edge after the ant solution construction phase is over, incorporating both mechanisms mentioned, is detailed in Equation 11 where S_{ij} is a set of all completed valid solutions, returned by the ants after the solution construction stage is complete, that used the edge going from i to j . Note that in Equation 11 evaporation is applied on the pheromone levels of the edges *before* the new pheromones are deposited. This is in line with the purpose of evaporation, which is to gradually forget *older* solutions. Figure 12 displays a flowchart reviewing the main structure of an ACO algorithm.

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \sum_{S_p \in S_{ij}} f'(S_p)$$

Equation 11: Edge Pheromone Update

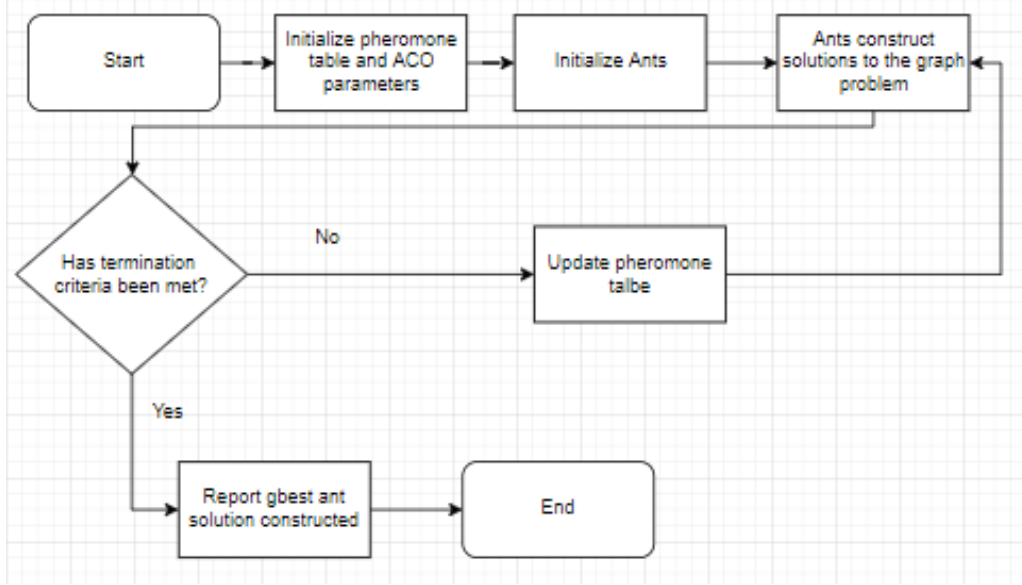


Figure 12: ACO flowchart

2.3.9 The Ant System

There are a few ways in which a choice from the list of valid solution components $N(S_p)$ can be made. The most widely used method, taking close inspiration from the mathematical model proposed by Goss *et al.* (1989), is classic the *Ant System* which was also the first proposed model for the ACO (Dorigo et al., 2006; Dorigo & Stützle, 2019).

Here, two mechanisms come into play that influences the attractiveness of a given valid choice to an ant. Naturally, the first is the level of the pheromone τ_{ij} that has been deposited on the path from its current position i to the choice j . The second is the heuristic information about that choice direction, by which the individual ant can make an independent assessment of the choice. This heuristic is a score for the length of the chosen path demonstrated through Equation 12 where $d(i, j)$ is the distance between vertices i and j .

$$d'_{ij} = \frac{1}{d(i, j)}$$

Equation 12: Path Distance Score

Similar to the workings of the *Particle Swarm*, through a balance of these two mechanisms, using weights α and β respectively for pheromone importance (i.e., swarm-confidence) and heuristic importance (i.e., self-confidence), a measure of attractiveness for a given choice can be quantified. The choice of a solution component from the list of valid choices is carried out probabilistically for each construction step. Each choice in the list of valid choices is given a choice probability weighted by their levels of attractiveness and, for each ant. This weighted probability choice adds a stochastic element to the algorithm, allowing the possibility (though less likely) of an ant to venture off course by choosing a less attractive path. This completed stochastic decision rule for choosing a vertex $j \in N(S_p)$ given a current position vertex i is given in Equation 13.

$$p(j|i) = \frac{\tau_{ij}^{\alpha} * d'_{ij}{}^{\beta}}{\sum_{k \in N(S_p)} \tau_{ik}^{\alpha} * d'_{ik}{}^{\beta}}$$

Equation 13: Ant Stochastic Decision Rule

2.3.10 The Max-Min Ant System

Though still effective, further research has shown that the performance of the classic *Ant System* could be further improved through stronger exploitation of the best solutions found during the search. By allowing all ants to update pheromone levels, better solutions were not as apparent until later iterations, however, using a greedier approach to the search provoked the problem of premature convergence. The *Max-Min* approach to the Ant System aims to solve this by combining an improved exploitation mechanism with an effective early search stagnation avoidance mechanism (Stützle & Hoos, 2000), shown in Equation 14.

The *Max-Min Ant System* uses the same pheromone update method specified in Equation 11 to update the edges, however, only the best performing ant for the iteration is considered. The value allowed for the pheromone levels of all edges is also bound between a maximum value, limiting the effects of best-performing ants on any given edge, and a minimum value, preserving a small level of pheromone on all edges. Together, they incorporate a small level of constant attractiveness to all edges to

encourage minor exploration to counteract the problem of premature convergence brought by the removal of distraction from trails left by weaker performing ants. The max-min formula for pheromone update calculation for an edge used by the best performing ant is explained through Equation 14 where S_p^{Best} is the solution returned by the best performing ant, and τ_{max} and τ_{min} respectively are the upper and lower bounds imposed on the pheromones (Dorigo et al., 2006). The workings of the operator $[x]_b^a$ is defined in Equation 15. The paper by Stützle & Hoos (2000) also offered guidelines, through which, the values used for τ_{max} and τ_{min} can be empirically configured.

$$\tau_{ij} = [\tau_{ij} * (1 - \rho) + f'(S_p^{Best})]_{\tau_{min}}^{\tau_{max}}$$

Equation 14: Max-Min Pheromone Update

$$[x]_b^a = \begin{cases} a & \text{if } x > a, \\ b & \text{if } x < b, \\ x & \text{otherwise;} \end{cases}$$

Equation 15: Max-Min Clamp Operator

2.3.11 The Ant Colony System

The ant colony system algorithm, introduced by Gambardella & Dorigo (1996), blends the concepts posed in the *Ant System* and *Max-Min Ant System*, by having all ants update the pheromone through a local pheromone update while keeping the main global pheromone update to be done at the end by only the best performing ant. So again, a similar method to Equation 11 is used, but rather than waiting till the end where their completed route scores can be used for pheromone update, each ant deposits a tiny predetermined local pheromone level at each solution construction step as shown in Equation 16. This allows dynamic diversification as the iteration runs through.

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + (\rho * \tau_0)$$

Equation 16: Local Pheromone Update

Here, τ_0 is the initial value of the pheromone which Gambardella & Dorigo (1996) suggests should be set to $(n * L_{nn})^{-1}$, where n is the number of cities on the map, and L_{nn} a rough approximation of the optimal tour length. After the solution construction stage, the final pheromone update performed by the best ant is done just like in the *Max-Min* (Equation 14) except without the clamps τ_{max} and τ_{min} .

Another notable change that the *Ant Colony System* brings is to the ants' decision rule. They introduced two new variables q and q_0 , directing ants' decision-making method. q is a uniformly distributed random number between $[0,1]$ and q_0 is a pre-set parameter such that if $q \leq q_0$ then the ant would use the stochastic weighted probability choice detailed in Equation 13 as their decision rule, otherwise, they would just deterministically choose the most attractive path through Equation 17.

$$j = \mathit{arg\,max}_{k \in N(s_p)} \{ \tau_{ik}^\alpha * d'_{ik}{}^\beta \}$$

Equation 17: Ant Deterministic Decision Rule

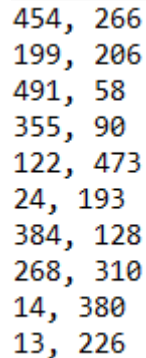
Though either the global or iteration's best ant can be used as the best representative for pheromone calculations, for the *ant colony system*, the global best ant is typically used, while the *max-min ant system* focuses on the use of the iteration's best. Of course, a mixed strategy, procedurally alternating between the global or iteration's best ants, can also be employed (similar to the mechanism of the *modified particle swarm optimization*) (Stützle & Hoos, 2000).

3. DESIGN & METHODOLOGY

This chapter discusses the experimental design and research methods employed, including an outline of the dataset used, coding languages used, algorithm configurations, and the sub-topics explored in the experiments.

3.1 Data Generation

Due to the simplicity of the data set used, a list of n vectors, the data used for Traveling Salesman tests in this study was self-generated. For the study, 4 datasets were generated having maps of 10, 20, 30, and 50 cities. Each dataset consisted of 100 maps containing its respective city amount. Each city was stored as a randomly assigned vector(x, y) that exists on a 500x500 map. Only the dataset having a city count of 10 was used for the preliminary rounds of analysis while the final analysis was run against all datasets. Figure 13 demonstrates how an example TSP map having 10 cities looks in storage with the left column meaning the x coordinate, and the right column meaning the y , for each city location on the map.



454,	266
199,	206
491,	58
355,	90
122,	473
24,	193
384,	128
268,	310
14,	380
13,	226

Figure 13: Example Dataset for a map of 10 cities

3.2 Languages Used

For data generation, the Java-based language ‘Processing’ (P3D) was used because of its simplistic and visual-based language. Processing was also used to develop another small program to display any generated TSP solutions for visual inspection. Figure 14 shows example images generated using Processing that displays a solution returned from an optimization algorithm as a connected graph on a map. The figure on the left displays

a solution to a map of a city-size 10, and on the right is a map of a city-size 50. Finally, a 3rd miniature program was developed in Processing for formatting; to clean up all generated result data returned from the optimization algorithms before data analysis.

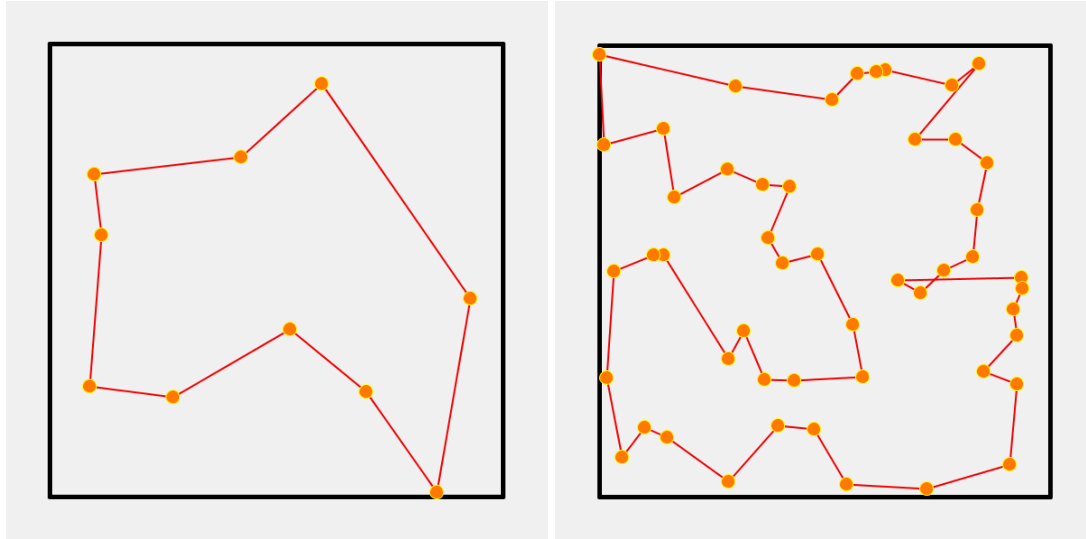


Figure 14: Solution Display Program (city count 10 and 50)

The second programming language used was Python which is a high-level but also general-purpose programming language that emphasizes code readability. All optimization algorithms used were developed using python. Specifically, the Anaconda Navigator's Jupyter Notebook was used to develop these programs. Finally, the analytical programming language R, developed for statistical computing and graphics, was used for all data analysis conducted in this study. The language offers very easy-use tools for data analysis and the colours automatically selected for the generated graphs are quite pleasant to the eye. Communication between languages was done through lightweight text files.

3.3 Algorithm Implementation & Configuration

In Section 2, research objective 1 was addressed and objective 2 was touched on. This section aims to answer Research Objective 3: *'To define the parameters and methodologies that create the best performing representative, for each of the algorithm classes in this project domain, to be used in the final comparative analysis.'* using the information drawn from Section 2. Any implementation queries found that needed further experimentation before this research objective could be completed is answered in Section 4. These implementation queries can be understood as supplementary

research questions linked to research objective 3, that arose as a result of tackling the previous research objectives.

3.3.1 Genetic Algorithm Implementation Specifics

The GA's application requirement was that the problem domain is presentable as a list of chromosomes and an evaluation function. With regards to the TSP, genes were symbolized as cities, a chromosome (a sequence of genes) was correspondingly symbolized as a sequence of cities, and continually, a population is simply understood as a group of chromosomes. These concepts are illustrated in Figure 15.

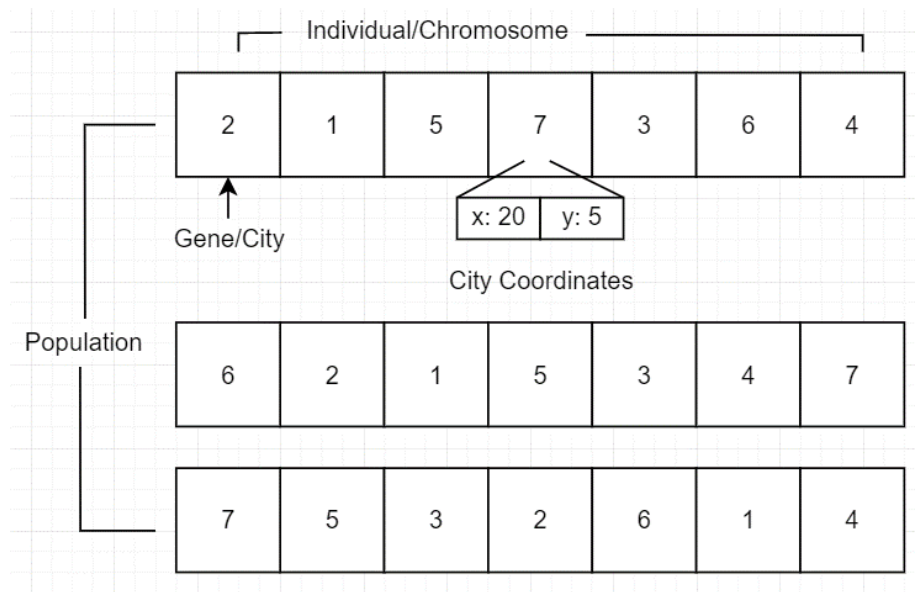


Figure 15: Population Composition for the Genetic Algorithm

By enforcing the gene sequence to comply with the TSP condition 1 detailed in Section 2.1 (no repeated cities) each full chromosome also becomes a complete solution to the TSP when the cities are visited in the sequence directed by the chromosome. For the evaluation function, the TSP Objective Function in Equation 2 was used.

In showing consideration to TSP condition 1 (no repeats), the method for breeding and mutation also had to be slightly adjusted. For breeding two parent chromosomes, after a swap point was chosen, the first section of the parent's genes was copied over to the children. Then, following the alternate parent's gene sequence, genes are copied over to complete each child's gene sequence only if they do not already exist within that

sequence. This process is demonstrated in Figure 16. Mutation, on the other hand, was treated as swaps between two cities chosen randomly along the chromosome sequence occurring at a rate denoted by the mutation-rate variable as illustrated in Figure 17. In this way, the states of all members of the population remain constantly valid concerning TSP solution requirements.

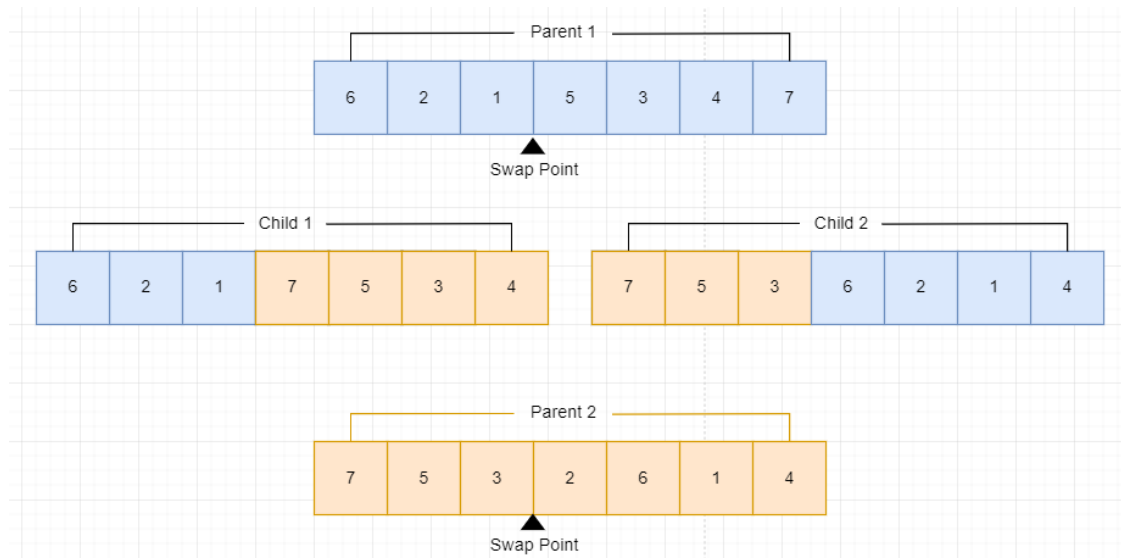


Figure 16: GA Breeding for the TSP

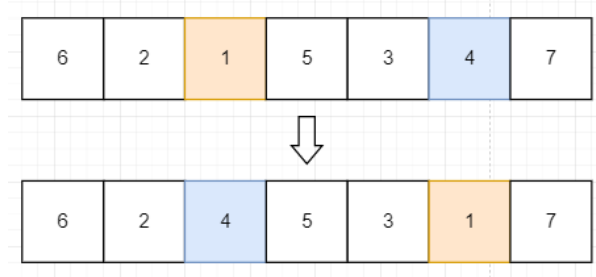


Figure 17: GA Mutation for the TSP

The mutation probability for the GA was set to 0.6% as recommended by Mirjalili (2019a). However, the variations in the State-Of-The-Art GA design highlighted in Sections 2.3.2-4 came with some implementation queries that needed to be investigated.

- Which fitness function variant for the GA performs the best?
- If elitism is used, what elite percentage works best?

- Is there any gain that can be found from the use of enhancers over using just the base algorithm? (Base Version vs. Elitism vs. Steady-State)

Along with this was the small concern found with the *tournament sampling* fitness function approach highlighted in Section 2.3.2.4: its selection pressure was potentially too high. To attempt to solve this, in this project a new approach was devised, borrowing inspiration from ACO's *ant colony system* explained in Section 2.3.11, where a new *delta* variable was introduced δ and used to probabilistically decide whether the winner of a tournament is the member with the higher or lower score. For example, $\delta = 0.7$ means a 70% chance that the member with the higher score would be declared the winner of the tournament, while the member with the lower score has a 30% chance of winning. Reverting back to the original tournament sampling mechanism can be done by setting the delta value $\delta = 1$. The introduction of this variable lowered the selection pressure of tournament sampling but also brought along the implementation query:

- What delta setting is the best for tournament sampling?

Another consideration was brought up when examining the Steady-State enhancer and its seemingly too-greedy mechanism detailed in Section 2.3.4.2. Attempting a solution to this, a new approach was constructed to localize its effect while at the same time striving to preserve its essence. This new *local steady-state* function limited the power of high-performing children found by allowing them only to replace their direct parents if better, rather than any other, possibly weaker, members of the population. Of course, this also brought along the implementation query:

- Does this new local steady-state function result in any improvement over the original steady-state function?

3.3.2 Particle Swarm Optimization Implementation Specifics

The studies performed by Yousefikhoshbakht (2021) and Wang *et al.* (2003) detailed an intriguing method for adapting the PSO to the TSP, detailed in Sections 2.3.6 and 7. Out of interest, their method of implementation was followed in this study. However, it was found that their proposed model was missing particle *inertia* which seemed a crucial error when considering other sources. Therefore,

- Could their model be improved by re-introducing particle inertia?

Yousefikhoshbakht (2021) introduced some new variables, detailed in Section 2.3.7, to use to configure his *modified particle swarm optimization* model improved from the one proposed by Wang *et al.* (2003). He carried out a test on 15 possible combinations to determine an optimal configuration.

- But what configuration suits the Traveling Salesman Problem?
- Does this modified version suit this project more than the original?

3.3.3 Ant Colony Optimization Implementation Specifics

The first things implemented in the ACO mechanism were two matrices used to store the pheromone and city distance data. Because the TSP used in this study was an undirected graph of city vertices allowing edge connections between any two cities, these matrices used were symmetric along the diagonal, having both the row and column able to represent the ‘from’ and ‘to’ cities for any edge and the data for each edge stored in its corresponding matrix cell. Figure 18 is an example of this.

		To/From				
		1	2	3	4	5
To/From	1	0	50.235	17.923	34.274	83.43
	2	50.235	0	5.343	98.374	87.426
	3	17.923	5.343	0	45.763	98.242
	4	34.274	98.374	45.763	0	78.503
	5	83.43	87.426	98.242	78.503	0

Figure 18: ACO example Distance Matrix for a city count of 5

The structure of the algorithm developed closely followed the descriptions posed in Sections 2.3.8-11. To avoid the divide by zero error, it should also be noted that the pheromone matrix should be initialized to store trivially small, non-zero values.

Following common practice (Gambardella & Dorigo, 1996; Stützle & Hoos, 2000), the alpha and beta weights used in the ACO for this project were $\alpha = 1$ and $\beta = 2$, and the evaporation rate was set to $\rho = 0.9$. Moreover, Stützle & Hoos (2000) and Gambardella & Dorigo (1996) also expounded on the method used for calculating the τ_{max} and τ_{min} variables in the *Max-Min Ant System* and *Ant Colony System* algorithms. Equations 18 and 19 adopting the *global* best ant solution S_p^{gbest} rather than the *iteration's* best S_p^{ibest} were used.

$$\tau_{max} = \frac{1}{\rho} * \frac{1}{f'(S_p^{gbest})}$$

Equation 18: T-max Calculation

$$\tau_{min} = \frac{\tau_{max} * 1 - (\sqrt[n]{Pbest})}{\left(\frac{n}{2} - 1\right) * \sqrt[n]{Pbest}}$$

Equation 19: T-min Calculation

For Equation 19, n represents the number of components used to create a complete TSP solution, i.e., the number of cities on a complete route or map. Stützle & Hoos (2000) detailed an experimentation process through which the appropriate configuration for the $Pbest$ variable used in this equation could be found. However,

- What $Pbest$ value is the most appropriate for the ACO variants?

Initialization of the pheromone tables for the 3 ACO variants, each operated differently. For the *Ant System*, simply initializing the table to trivial, non-zero values worked. However, for the *Max-Min Ant System*, a specialized pheromone initialization was done

after iteration 1 was complete, as pheromone levels for each edge were initialized to the calculated τ_{max} value gained from iteration 1 (Stützle & Hoos, 2000).

Similar to the *Max-Min*, the specialized *Ant Colony System* pheromone update was calculated after iteration 1 was complete, where the pheromone levels for each edge were initialized to the ants' local pheromone update strength calculated for the first iteration $\left(n * f(S_p^{gbest})\right)^{-1}$ (Gambardella & Dorigo, 1996). In keeping with the direction of Gambardella & Dorigo (1998), note here that the TSP Equation 1 for calculating the tour length is used rather than Equation 2 for the tour score. All of this left only the final question:

- Which of the three ACO variants performs the best?

3.3.4 Overview of Implementation Queries for Research Objective 3:

GA –

1. What is the best delta setting for tournament sampling?
2. Which fitness function performs the best?
3. If elitism is used, what elite percentage works best?
4. Is there any gain that can be found from the use of enhancers over using just the base algorithm? (Base Version vs. Elitism vs. Steady-State)
5. Does the new *local steady-state* function bring any merit over the original?

PSO –

6. Could the PSO model be improved by re-introducing particle inertia?
7. What configuration for the *modified particle swarm optimization* suit this study?
8. Does the modified version suit this project more than the original?

ACO –

9. What *Pbest* value is the most appropriate for the Ant Colony variants?
10. Which of the Ant Colony variants performs the best?

3.3.5 Hybridization Methodology

Tackling the 4th research objective: '*To construct hybrid models based on the methodologies used in the base representatives*', required that hybrid algorithms be devised using the base algorithms developed. Hybrid models have been built for mixing the ACO and GA models (Luan et al., 2019; Yang & Yoo, 2018), mixing the PSO and GA models (Moradi & Abedini, 2012; Omidinasab & Goodarzimehr, 2019; Thangaraj et al., 2011), and mixing the ACO and PSO models (Khourdifi & Bahaj, 2019; Mandloi & Bhatia, 2016; Shelokar et al., 2007). It was even found that 34% of all studies done using a PSO hybrid between the years 2001-2010, used the PSO and GA hybrid (Thangaraj et al., 2011). It was observed that in the studies cited here, only Luan *et al.* (2019) give some sort of justification for their choice of hybridization strategy by using a speed-time curve to track the point within the iterations of the program where the best performance benefit can be brought by switching the algorithm methodology.

This dissertation was inspired by the study performed by Huang *et al.* (2013), where it was discovered that sequential hybridization (running the algorithms one after the other) produced better results than parallel hybridization (running algorithms side by side) when mixing the ACO and PSO algorithms on a continuous scale. Due to this found success, the sequential hybridization method employed by Huang *et al.* (2013) was again applied to build the hybrid models used in this project. For each iteration of the hybrid algorithm, the program first operated through one of the base methodologies, using the data collected. The program then ran using the second algorithm methodology before returning the final data collected for that iteration. Each algorithm constructed was named according to the sequential order in which their methodologies were run (e.g. ACO/GA means ACO first, then GA for each iteration). Demonstrated in Figure 19 is an example strategy for constructing the PSO/GA hybrid model. Unlike the study done by Huang *et al.* (2013), however, the hybridization methodology used in this project was applied to the discrete versions of the algorithms as well as the yet untested PSO/GA and ACO/GA sequential hybrid models.

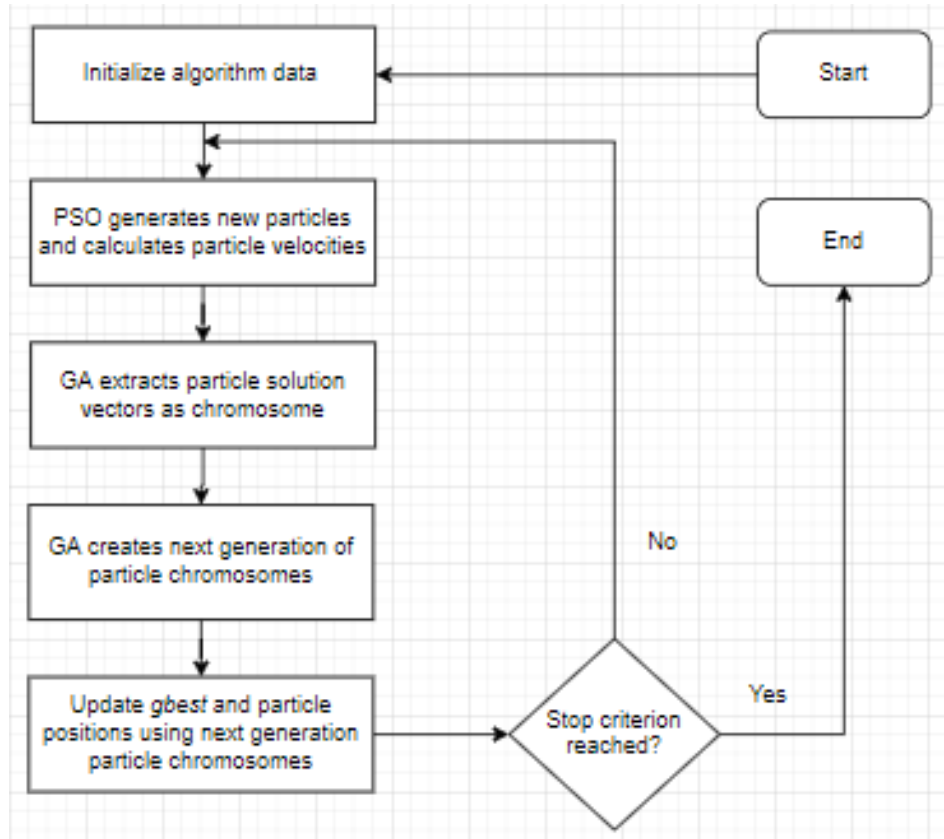


Figure 19: Sequential Hybridization Strategy for the Particle Swarm and Genetic Algorithm

3.3.6 Benchmark

As a comparative baseline against which to critique the performance of these algorithms, a benchmark *greedy optimization algorithm* was developed. Its mechanism was quite trivial in that it first started at a random city position on the map and, for each solution construction phase, simply went to the closest city it could find that was not already visited. After a complete traversal of the TSP map, the algorithm returns the sequence of cities it encountered on its journey. Because no iteration or improvement occurred in its mechanism its performance was quite literally a stagnant baseline.

3.3.7 Chosen Statistical Test

Though generally normally distributed, due to the stochastic nature of the algorithms used as well as variance in the layout of the randomly generated maps, analysis of the AUC data drawn from early experiments done in this project drew some outliers. Figure

20 displays the AUC data drawn from a test done to compare the fitness functions of the GA. The top image displays a scatter plot showing outliers found in all algorithms, and the 3 images below are histograms displaying the approximately normal distribution of the AUC data. Nonetheless, these outliers were still valid results obtained from the algorithms rather than simple un-representing mistakes, so removing them was not an option from a statistical point of view. Due to their presence, the Mann-Whitney U test was chosen over the widely used t-test and ANOVA test, completing Research Objective 2: ‘*To establish and justify an appropriate statistical test to be used for comparative analysis*’. For all analyses done, the statistical threshold of 0.05 was also chosen because it is the most common threshold used in statistical analysis.

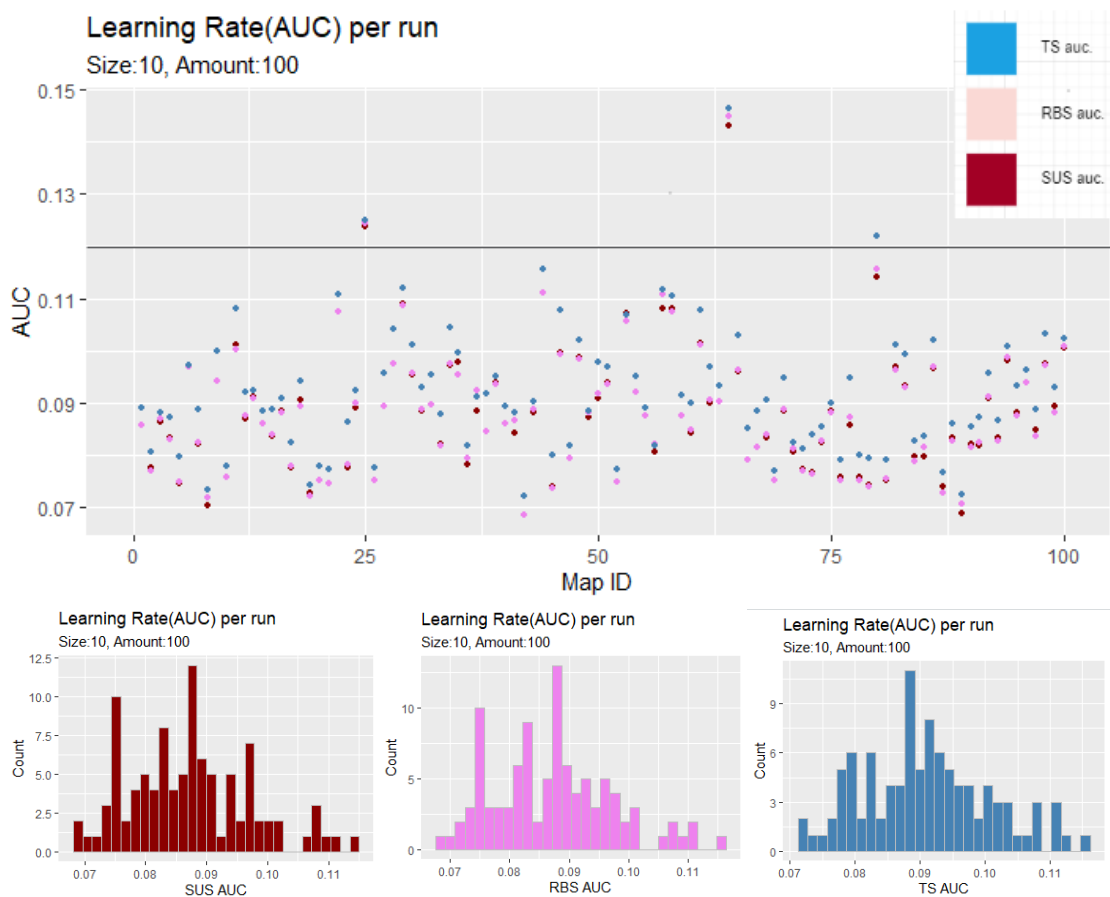


Figure 20: Algorithm Results Showing Outliers

4. RESULTS, EVALUATION & DISCUSSION

4 experiments were performed in this study. The first 3 were preliminary experiments aiming to determine the best configuration for each algorithm to use as a representative algorithm model, and the final experiment addresses the main objective of this project: the Hybrid vs Base comparative analysis. For all experiments done, unless specified otherwise, it can be assumed that the TSP maps used contain only 10 cities, the population size used was 50 and the maximum number of iterations allowed was 100.

4.1 Experiment 1: Genetic Algorithm

Condensing the GA implementation queries drawn from this study posed in Section 3.3, 2 overarching concerns were drawn: which fitness function and which enhancer? This splits the experiment into two parts:

4.1.1 Part 1 – Which Fitness Function to use?

As a first step, the effect of the introduced delta variable on the *tournament sampling* technique was examined and it was found that the algorithm model using the original tournament sampling ($\delta = 1$) performed the best with the highest average AUC of the group, as shown in Table 1. Figure 21 displays a line plot of the average global best score per iteration, also confirming this observation. Examining the AUC using a box plot shown in Figure 22 records a normal distribution for all learning rate (AUC) data, distinguishable by their approximately even spaced box and whiskers relative to their mean line, and it also confirms the win of the original tournament sampling. Only comparisons of the original tournament sampling against the tournament sampling with $\delta = 0.7$ beat the Mann-Whitney U test with a p-value of 0.014. Models with delta values of 1, 0.9 and 0.8 returned too-similar AUC values for there to be declared a statistical winner. These combined results suggested that not much gain could be drawn from the introduction of the delta variable.

Table 1: Tournament Sampling delta AUC

GA - TS delta	0.5	0.6	0.7	0.8	0.9	1
Mean	0.0593	0.0618	0.0646	0.0664	0.067	0.0674
Standard Deviation	0.0083	0.0076	0.0084	0.0086	0.008	0.0083

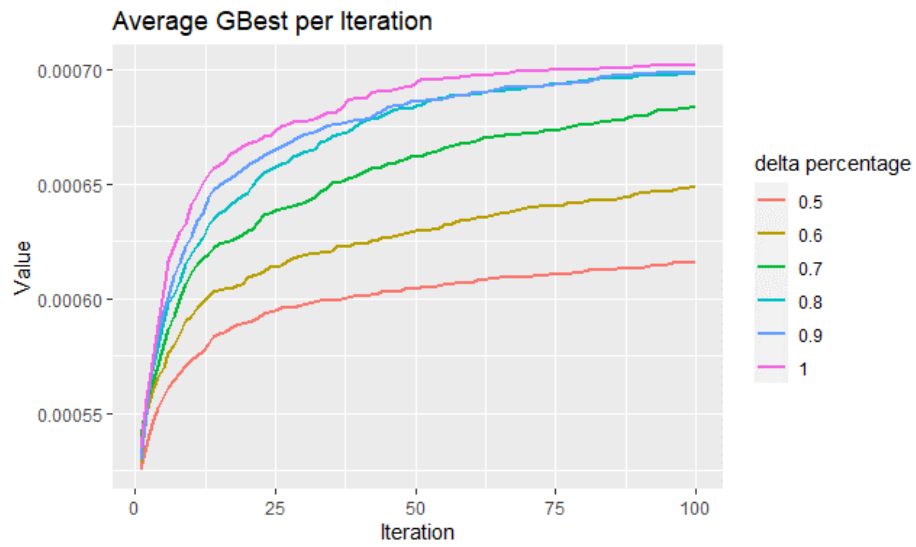


Figure 21: Line plot showing model score by iteration count for varying values of the tournament sampling noise parameter δ . A δ value of 1, indicating noise-free tournament sampling consistently outperforms the alternatives

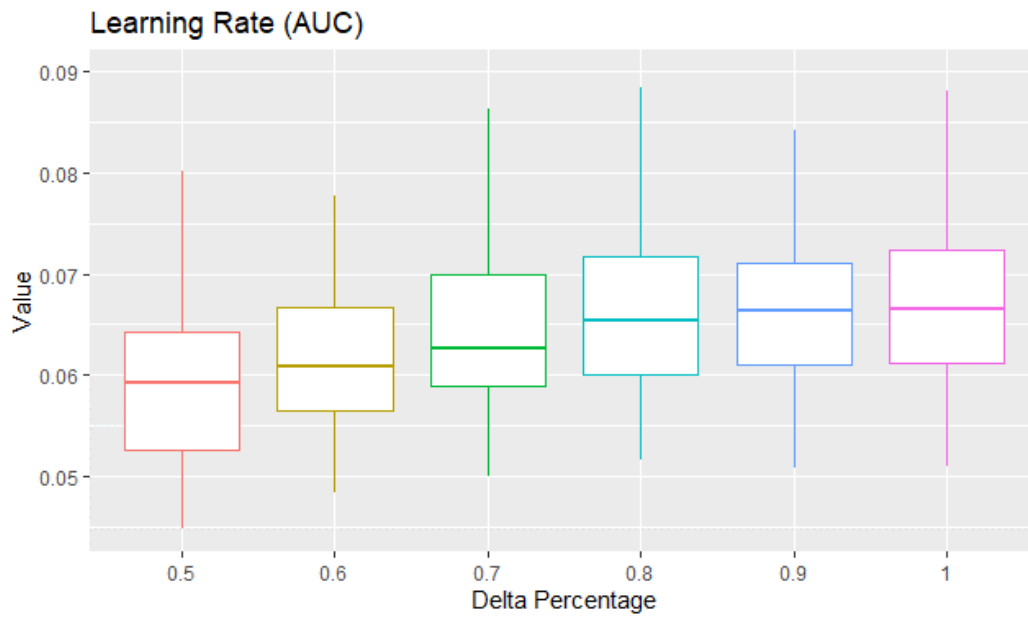


Figure 22: Box plot showing model score for varying values of the tournament sampling noise parameter δ .

After that, a comparison could be made to distinguish the best performing fitness function for the GA (*Stochastic Universal Sampling*, *Rank-Based Sampling* or the original *Tournament Sampling*). A plot of their best scoring solutions found shows the close rivalry between the rank-based sampling and tournament sampling as shown in Figures 23 and 24. In fact, both of those algorithms offer the same average AUC as shown in Table 2, though the rank-based sampling had a slightly lower standard deviation. Analysis of the results declared it essentially a tie between these two algorithms and either model was a valid representative choice. Due to its significantly faster runtime speed, *tournament sampling* was chosen as the optimum sampling technique for the GA in this project.

Table 2: Fitness Function AUC

GA - Fitness Function	SUS	RBS	TS
Mean	0.0667	0.0674	0.0674
Standard Deviation	0.0082	0.0083	0.0084

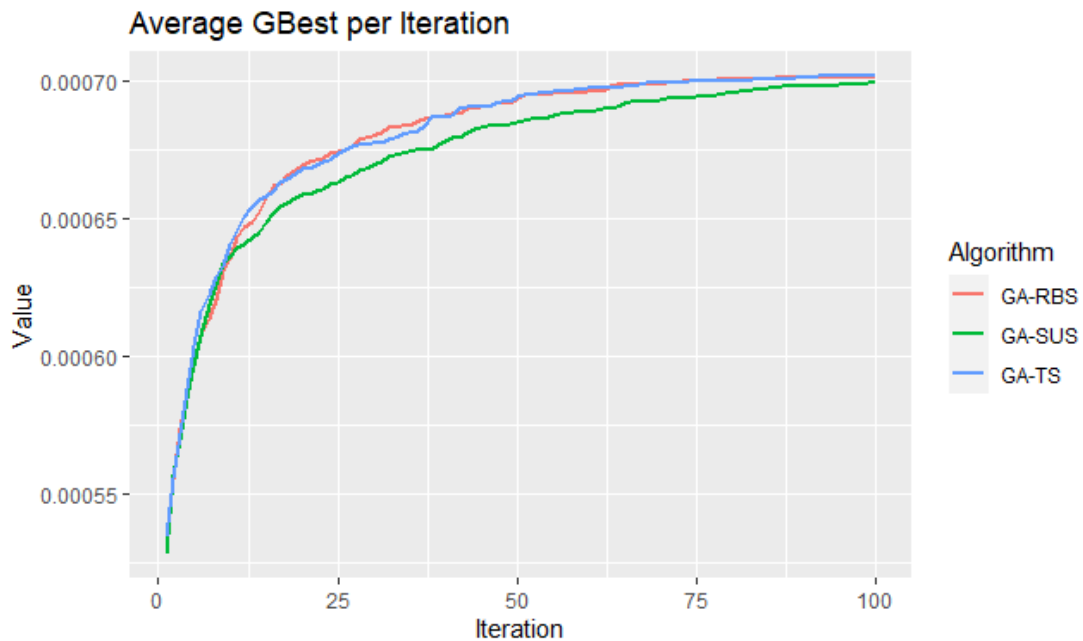


Figure 23: Line plot showing model score by iteration count for the Genetic Algorithm fitness functions.

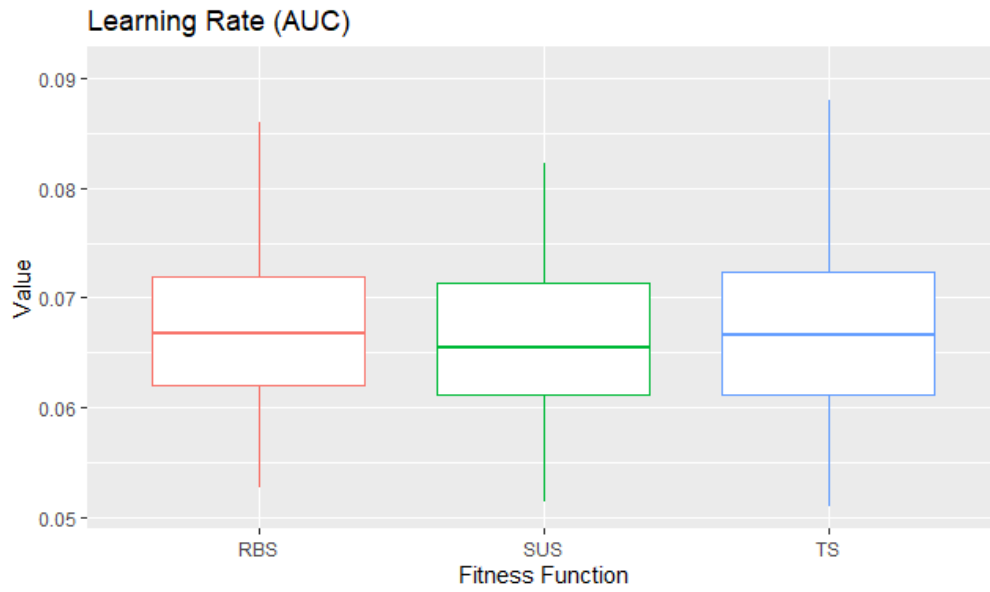


Figure 24: Box plot showing model score per genetic algorithm fitness functions.

These results agree with the findings of Razali & Geraghty (2011) who also found tournament sampling to be the best performing methodology at smaller problem sizes. However, they also observed that, as problem difficulty levels increased, tournament sampling became more prone to premature convergence and would eventually be overtaken by the other fitness function variants. The argument raised was that the selection pressure of the tournament sampling technique was too high when compared to the other fitness functions. With this in mind, similar to what was noted with particle inertia, it was originally concluded that perhaps the introduction of the delta variable to lower the selection pressure of tournament sampling was not as unfruitful as the results of these experiments have shown. Its gain may have been in trading performance rates (AUC) for alleviating this tendency for premature convergence, eventually bringing back a return on investments as problem sizes increase. However, the results of a supplementary experiment done to test this conclusion by observing the effect of the delta percentages against the larger map sizes of 50 cities, shown in the line graph of Figure 25, revealed that this was not the case even as the number of maximum iterations was doubled to 200. The original tournament sampling method remained the best performing methodology by an increasing margin, disproving any efficacy theorised from the introduction of the delta variable.

Additionally, when attempting to recreate the observations of Razali & Geraghty (2011) through another supplementary experiment, the results drawn, detailed in Figure 26, still find tournament sampling remaining as the best performing method as the allowed number of iterations increased. These discoveries run contrary to their observations and suggest that the selection pressure associated with the tournament sampling technique is satisfactory. Further research, comparing the detailed composition of their model against the one used in this project, would have to be done to resolve this discrepancy.

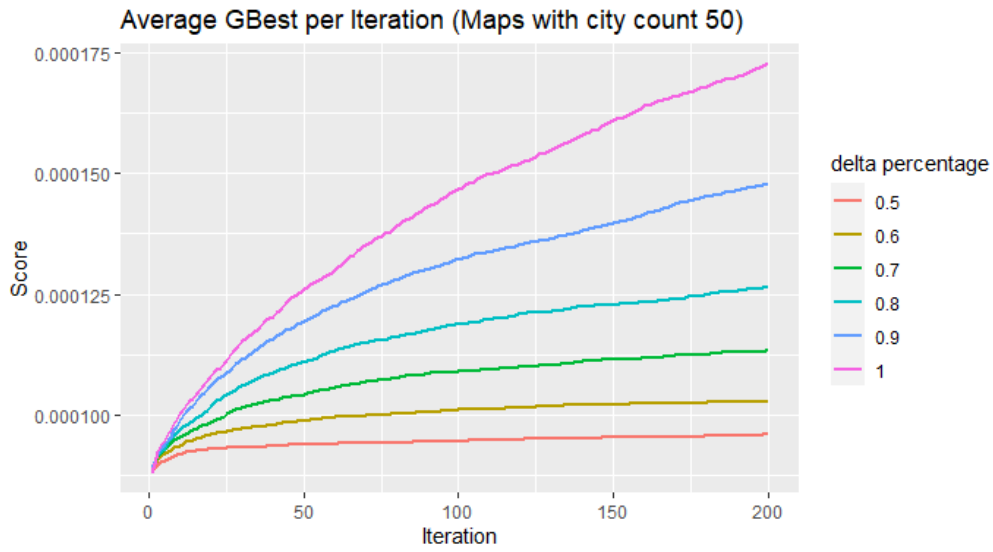


Figure 25: Line plot showing model score per iteration for varying values of the tournament sampling noise parameter δ , for an enlarged experiment of 200 iterations for map sizes of 50 cities

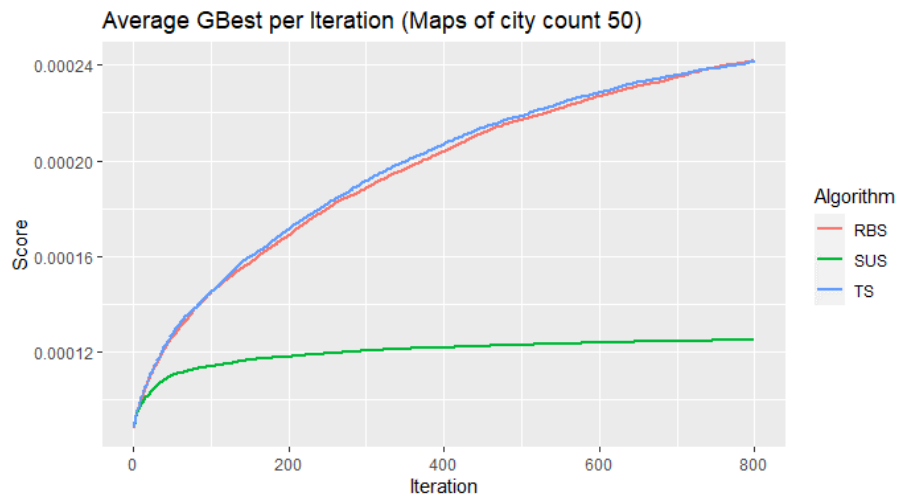


Figure 26: Line plot showing model score per iteration for genetic algorithm fitness function, for an enlarged experiment of 800 iterations for map sizes of 50 cities

4.1.2 Part 2 – Which Enhancer to use?

Both the *Elitism* and *Steady-State* enhancers needed some experimentation. Elitism required an elite percentage specified before use, so as a first step, an optimum setting for this needed to be found. Tracking the AUC using the box plot in Figure 28 showed normal distributions for all of the top-performing datasets and the mapping of average scores in Figure 27 found that, despite the close competition, an elite percentage of 10% was the best. This was confirmed when observing the AUC statistics in Table 3. Statistical analysis using the Mann-Whitney U test was inconclusive about a winner for this experiment. Nevertheless, the elitist model using an elite size of 10% was chosen as the winner.

Table 3: Elitism AUC

GA - Elitism	0%	10%	20%	30%	40%	50%
Mean	0.0678	0.0679	0.0672	0.0677	0.065	0.0671
Standard Deviation	0.0082	0.0085	0.0082	0.0082	0.008	0.0077

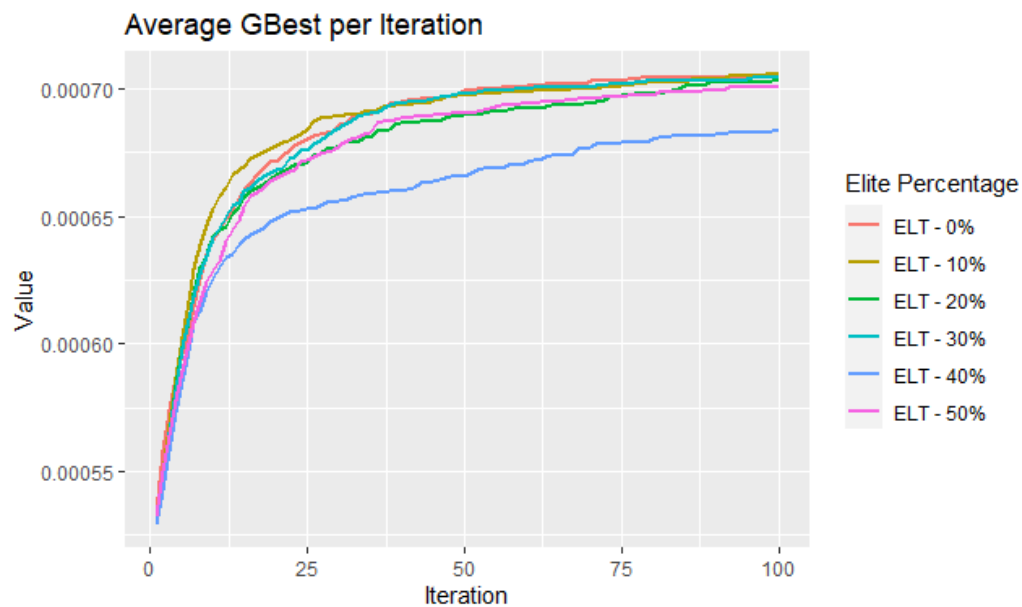


Figure 27: Line plot showing model score per iteration for varying elite percentages

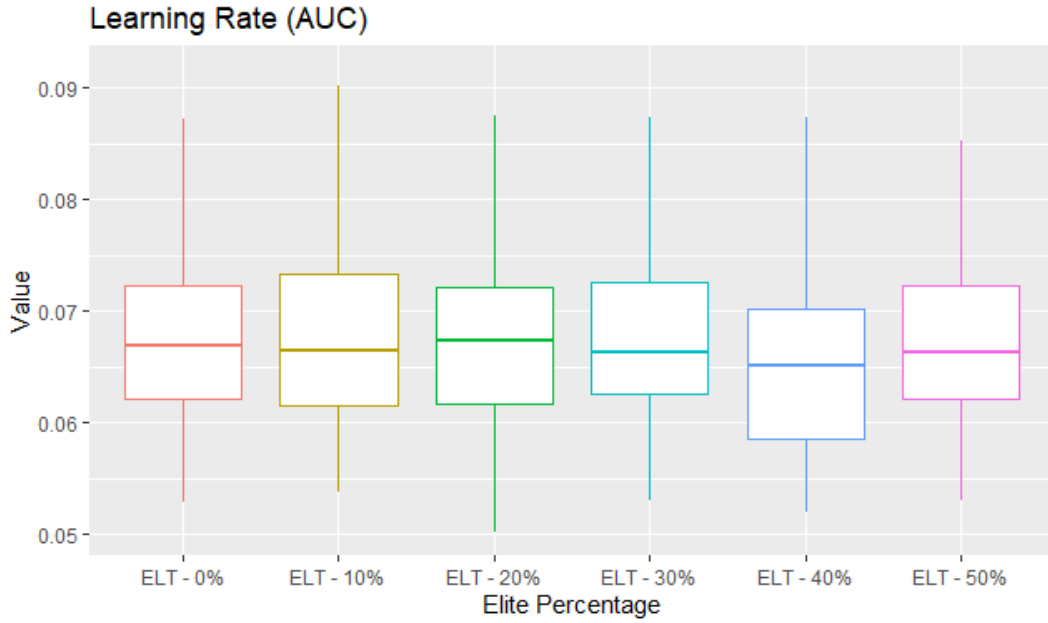


Figure 28: Box plot showing model score per elite percentage

Finally came the comparison of the GA enhancers. The *steady-state* technique and the proposed *local steady-state* variation, both performed marginally better than the elitist algorithm. Again, a winner could not be statistically justified, because of how close their performance was when examining the results found in Figures 29 and 30. Nevertheless, the AUC statistics given in Table 4 show that the original steady-state technique scored the highest average AUC and, for this reason, it was chosen as the winner for this comparison.

In conclusion, compiling all the results drawn from Experiment 1 reveals that the best results for the GA were achieved through the use of the classic Tournament Sampling fitness function combined with the original Steady-State enhancer.

Table 4: Enhancer AUC

GA - Enhancer	ELT	SS	LSS
Mean	0.0678	0.0682	0.0679
Standard Deviation	0.0082	0.0084	0.0084

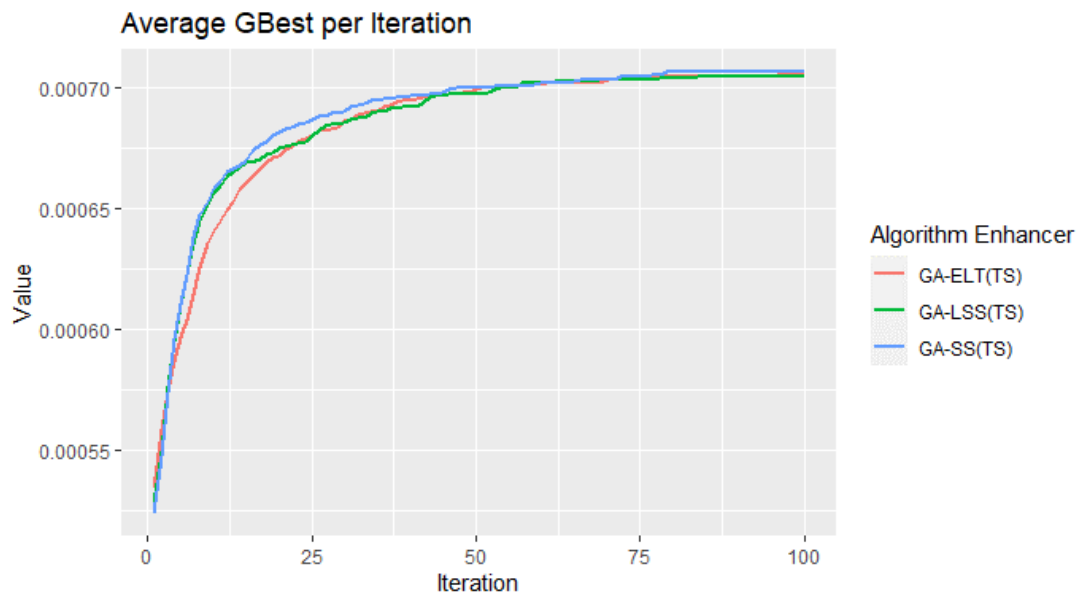


Figure 29: Line plot showing model score per iteration for the three enhancers (Elitism, Steady-State, and Local Steady-State)

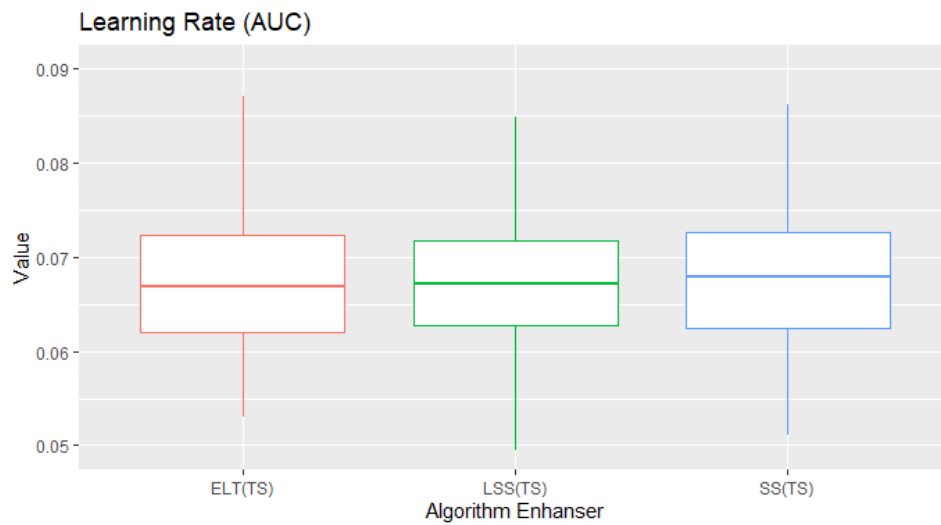


Figure 30: Box plot showing model score per genetic algorithm enhancer

4.2 Experiment 2: Particle Swarm Optimization

Consideration of the PSO implementation questions shaped this experiment into 3 parts. Analysis of the original version, analysis of the modified version and then a comparison between them.

4.2.1 Part 1 – Which Original Particle Swarm Configuration to Use?

The first consideration with using the PSO adapted for discrete domains was whether the particle inertia missing from the velocity calculation should be re-introduced. PSO models built using inertia retained a stochastically chosen portion of their previous velocities, calculated by their inertia weight, to be used to calculate their new velocities. When comparing PSO models using different inertia weights as shown in the line and box charts in Figures 31 and 32, the models using inertia weights of 0.4, 0.5 and 0.6 were found to be the best performing PSO models, outperforming the model without inertia ($w = 0$) with statistical significance values of 0.013, 0.011, and 0.015 respectively. Actually, the model not using inertia was found to be the worst-performing PSO model in the group. One thing to note in the line chart of Figure 31 is that the models with high inertia weights like 0.8 and 0.9, though having a lower AUC than the others, avoid premature convergence. They are shown to still be climbing in optimization scores returned, even overtaking the others, during the final iterations of the algorithm. This finding suggests that higher inertia weights would eventually offer better performance as problem sizes increase. This finding is also in line with and gives justification for, the popularity of using higher inertia weights like 0.8 (Shi & Eberhart, 1998).

Despite this, in keeping true to the analytical process determined for this study, the model having an inertia weight of 0.5 was chosen as the winner of this comparison because it offers the highest average AUC. These results support the argument pro inertia of Das *et al.*(2008) and highlight the shortcomings of the algorithm design proposed by Wang *et al.* (2003).

Table 5: Inertia Weight AUC

PSO Inertia Weight	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Mean	0.0613	0.0623	0.0637	0.0637	0.0643	0.0644	0.064	0.0633	0.0636	0.0628	0.0628
Standard Deviation	0.0089	0.0083	0.0085	0.0099	0.0087	0.0077	0.0085	0.0083	0.008	0.0081	0.0081

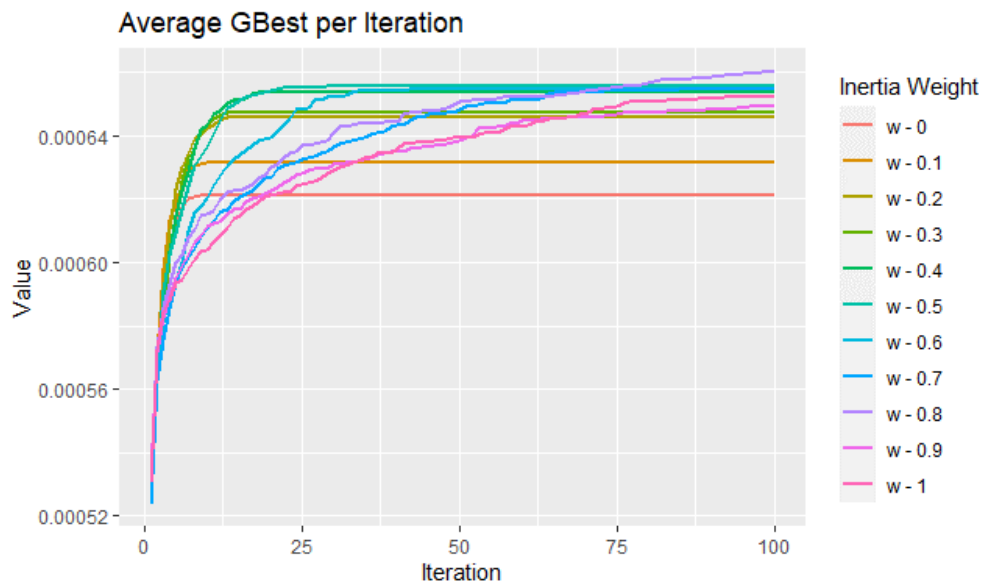


Figure 31: Line plot showing model score per iteration for values given for models implementing particle inertia.

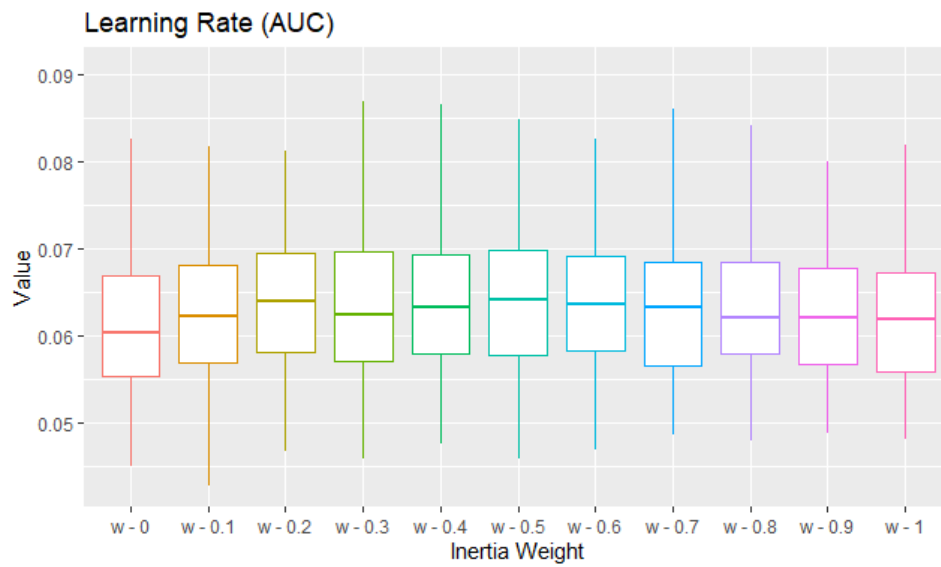


Figure 32: Box plot showing model score for values given for models implementing particle inertia.

4.2.2 Part 2 – Which Modified Particle Swarm Configuration to Use?

Following the example of Yousefikhoshbakht (2021), 7 test configurations were devised for the *modified particle swarm optimization* algorithm. As demonstrated in line and box graphs of Figures 33 and 34, the best performing algorithm were those with (alpha = 30%, beta = 70% and an iteration percent = 50% or 100%) and (alpha = 20%, beta = 80% and an iteration percent = 50%). The AUC statistics are given in Table 6 and the model with the highest average AUC (alpha = 30%, beta = 70% and an iteration percent = 50%) was chosen as the winner.

Table 6: Modified Particle Swarm Configuration AUC

MPSO settings	(0-5,5)	(0-10,5)	(0-10,10)	(2-8,5)	(2-8,10)	(3-7,5)	(3-7,10)
Mean	0.0619	0.0615	0.0612	0.0642	0.0637	0.0654	0.0652
Standard Deviation	0.0086	0.0074	0.0082	0.0089	0.0084	0.0089	0.0087

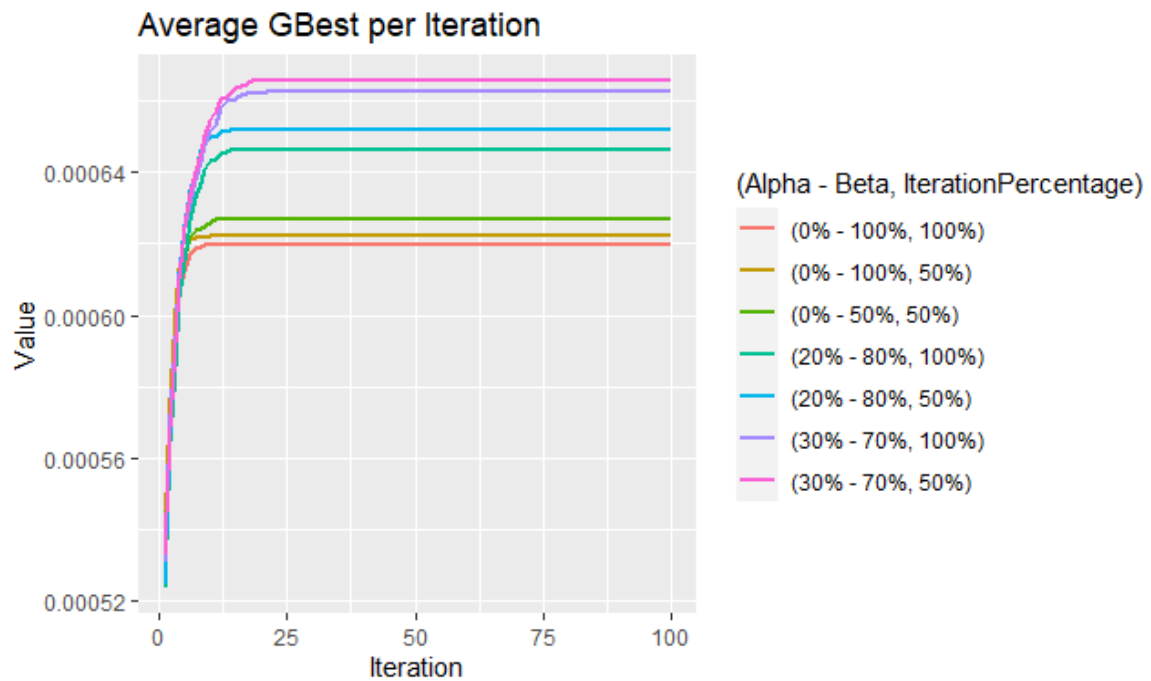


Figure 33: Line plot showing model score per iteration for the configurations of the Modified Particle Swarm

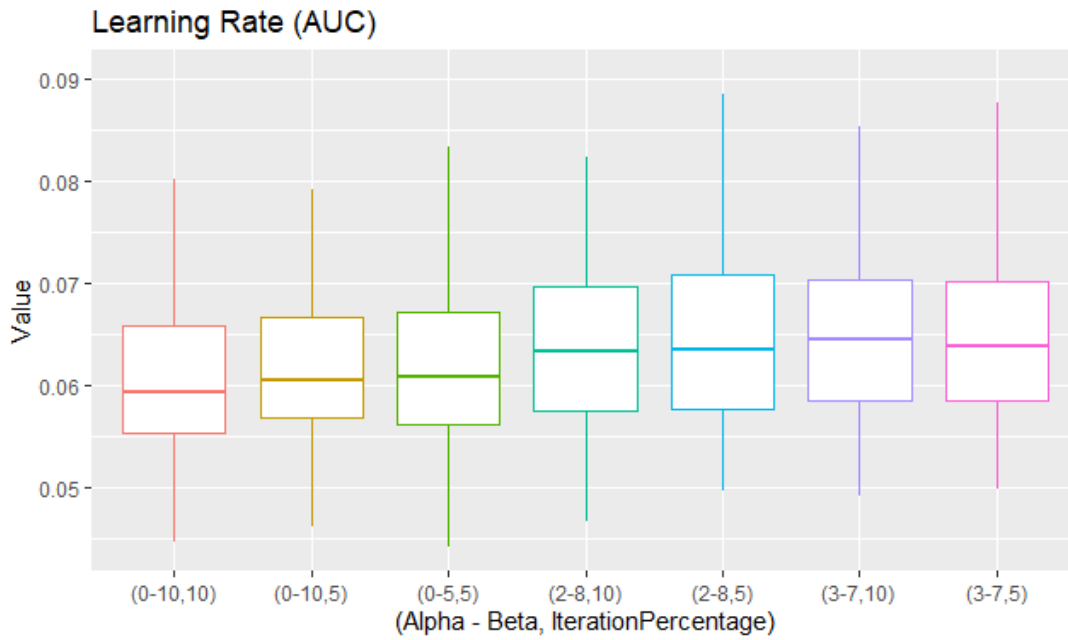


Figure 34: Box plot showing model score for the configurations of the Modified Particle Swarm

4.2.3 Part 3 – Which Particle Swarm Representative Model to Use?

Finally, came the comparison between modified and original PSO approaches to get the best Particle Swarm representative. When comparing the original PSO with an inertia weight of 0.5 with modified PSO with setting (alpha = 30%, beta = 70% and an iteration percent = 50%), it was found, in the line and box graphs of Figures 35 and 36, that modified offered the best performance, though not with a large enough margin to pass the statistical significance test ($p = 0.557$). This result is also reflected in the AUC statistics in Table 7.

In review, based on the results drawn from Experiment 2, the optimal PSO configuration found for this study was using the *modified particle swarm optimization* algorithm having an α variable linearly progressing from alpha of 30% to a beta of 70%, over the first 50% of the iterations, and with an inertia weight of 0.5.

Table 7: Particle Swarm Variant AUC

PSO Variant	PSO	MPSO
Mean	0.0644	0.0654
Standard Deviation	0.0094	0.0089

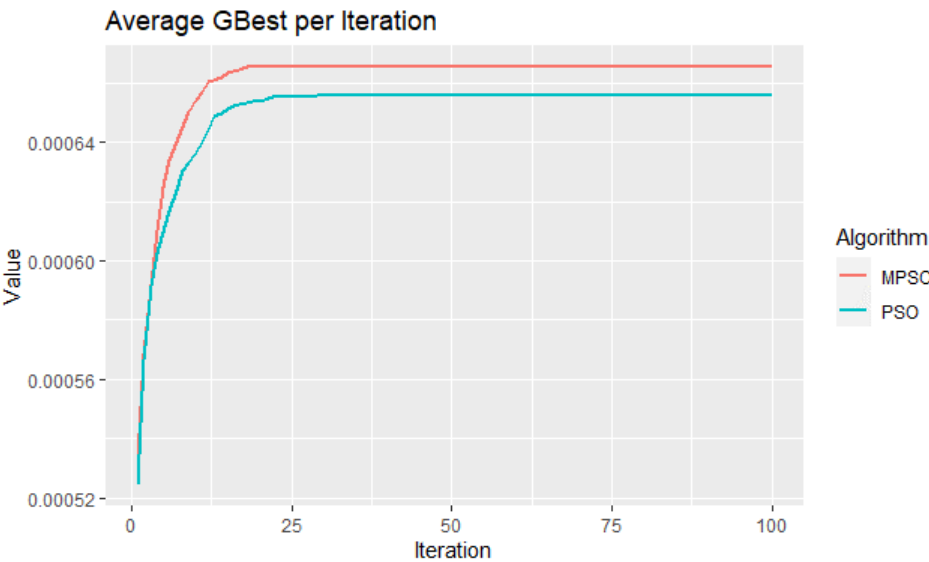


Figure 35: Line plot showing model score per iteration for the original and modified particle swarm algorithm

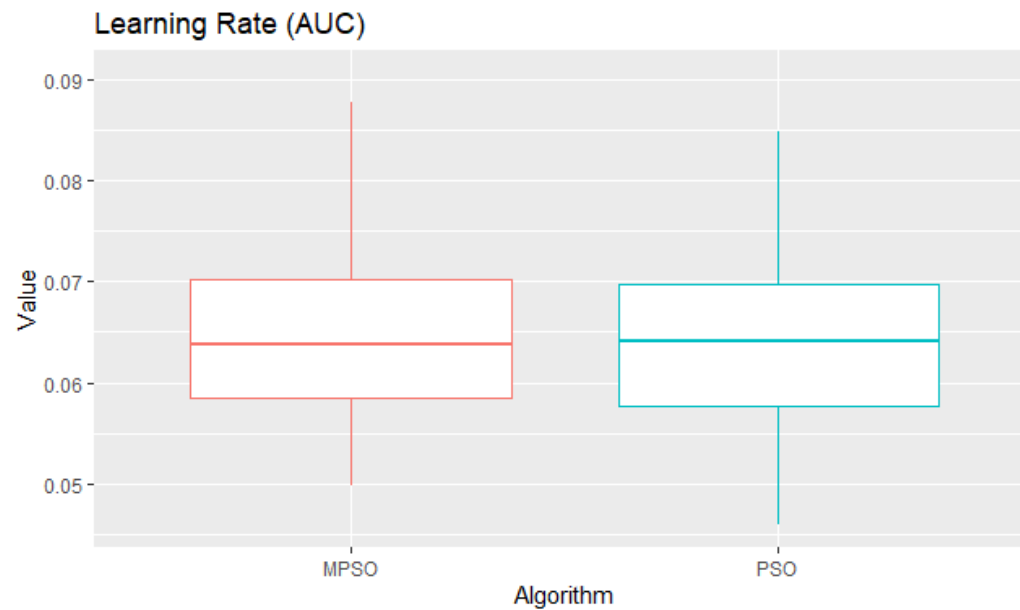


Figure 36: Particle Swarm Variant Box Plot

4.3 Experiment 3: Ant Colony Optimization

Experimentation for the ACO targeted the implementation queries, however, an unusual phenomenon occurred with the results drawn. All ACO models tested returned essentially the same results.

4.3.1 Part 1 – Which *Pbest* value is the most appropriate for the Ant Colony variants?

For the ACO, before the comparative analysis of its variants, the configuration for the *Pbest* value in the *Max-Min Ant System* variant would have to be decided. In their experiment, Stützle & Hoos (2000) tested *Pbest* values of 0, 0.5, 0.05, 0.005, and 0.0005. When a similar test was carried out in this study, not much of a difference was shown between them. Looking at the AUC statistics in Table 8, all models returned the same result of 0.070 (rounded to 0.07) and the same standard deviation. Not much difference could be seen when analysing the box plot in Figure 38 and a statistical winner was not declared. However, it seemed that the best model, winning by a minuscule margin based on the line graph in Figure 37, turned out to be the one having a *Pbest* of 0.0005. The test performed by Stützle & Hoos (2000) also found a *Pbest* of 0.0005 to be optimal, so that configuration was chosen as a result.

Table 8: Max-Min Ant System – *Pbest* AUC

MMAS - <i>PBest</i>	0	0.5	0.05	0.005	0.0005
Mean	0.07	0.07	0.07	0.07	0.07
Standard Deviation	0.0083	0.0083	0.0083	0.0083	0.0083

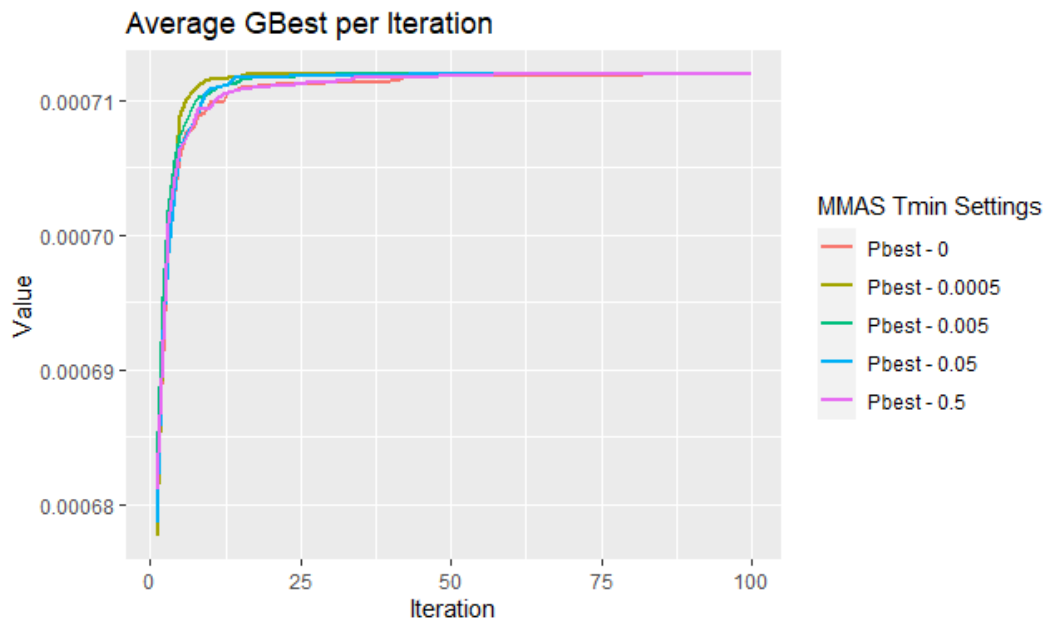


Figure 37: Line plot showing model score per iteration for the Pbest variable of the Max-Min Ant System.

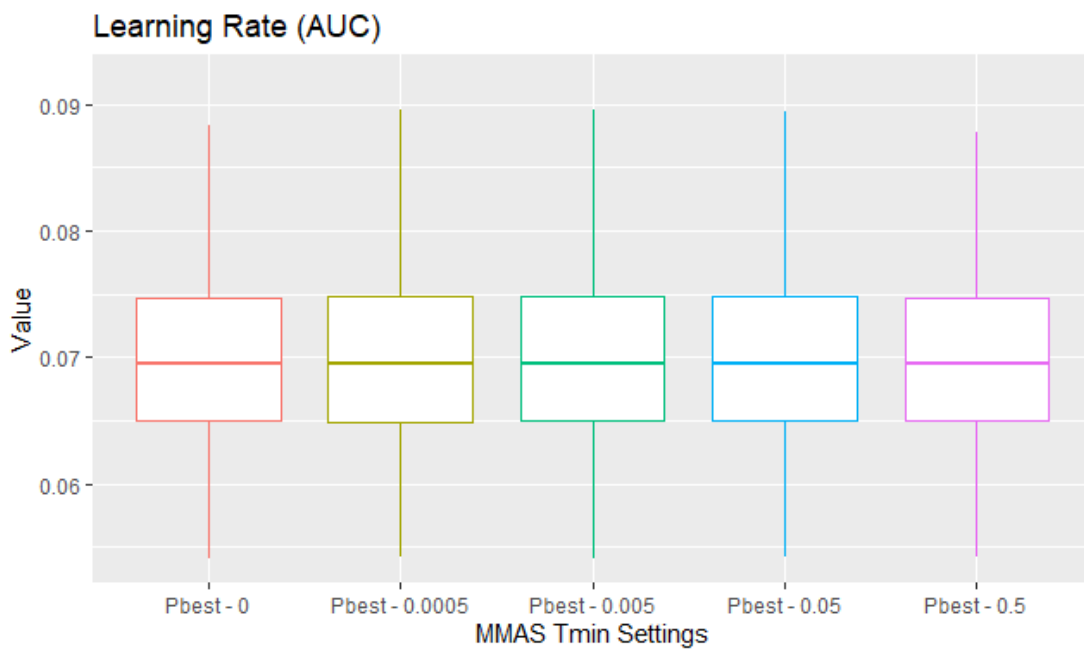


Figure 38: Box plot showing model score for the Pbest variable of the Max-Min Ant System.

4.3.2 Part 2 – Which of the Ant Colony variants performs the best?

With that configuration set, a comparative analysis could be done on the ACO's variants: the *Ant System*, *Max-Min Ant System* and *Ant Colony System*. Again, little was found to differentiate the performances of the 3 Algorithms as shown in the line and box graphs of Figures 39 and 40. The AUC statistics in Table 9 also revealed that the difference was negligible and statistical tests done on the data set offered no clear winner for the comparison. The line graph of Figure 39 does, however, suggest that the simple Ant System was the best performing algorithm by that minute margin, so that model was chosen as the final ACO representative for the study.

Table 9: ACO Variant AUC

ACO variant	AS	MMAS	ACS
Mean	0.07	0.07	0.07
Standard Deviation	0.0083	0.0083	0.0083

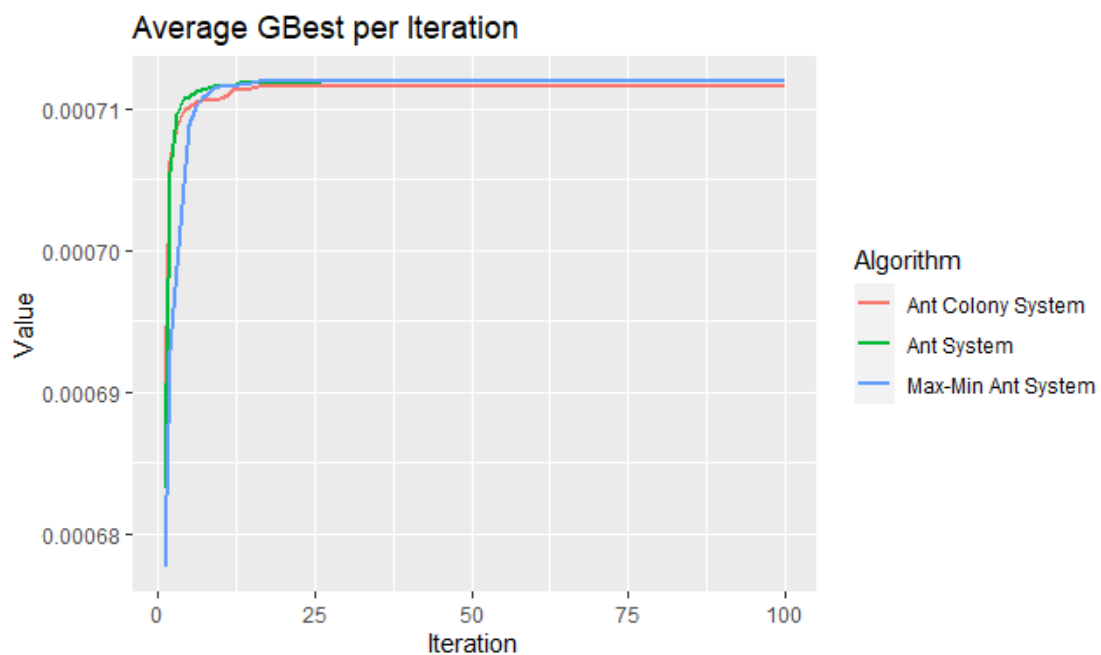


Figure 39: Line plot showing model score per iteration for each ant colony variant.

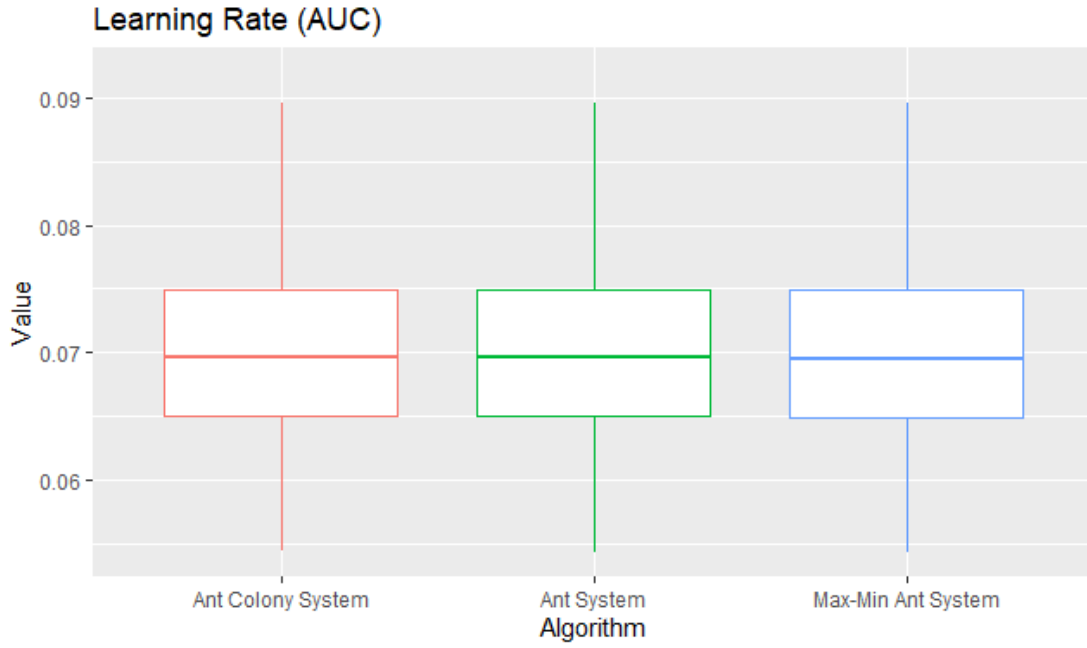


Figure 40: Box plot showing model score for each ant colony variant.

4.3.3 Interpretation

All ACO models tested offered approximately identical performance results. This phenomenon was theorised to be caused by the outstanding effectiveness of the general ACO methodology in this problem domain. Perhaps, the maps used in this experiment were much too simple for variations in model performance to become more apparent.

4.4 Experiment 4: Hybrids vs Base

After the best representative from each algorithm examined was compiled, the hybrid models were developed by mixing those base model configurations using a sequential hybridization technique, and a comparative analysis of their performance was performed. It was found that all algorithms performed better than the benchmark Greedy-Optimization algorithm, illustrated in the line graph in Figure 41. Though seemingly close, as shown in line graphs of Figures 41 and 42, the winner was the ACO/GA hybrid, as observed by the mean values in Table 10, which surprisingly beat the ACO base version with a statistical significance of $7.089e-12$.

Table 10: Hybrid vs Base AUC for TSP city size (10)

Hybrid vs Base	GA	ACO	PSO	ACO/GA	PSO/ACO	PSO/GA
Mean	0.0681	0.0704	0.0654	0.0705	0.07	0.0655
Standard Deviation	0.0084	0.0083	0.0089	0.0083	0.0082	0.0082

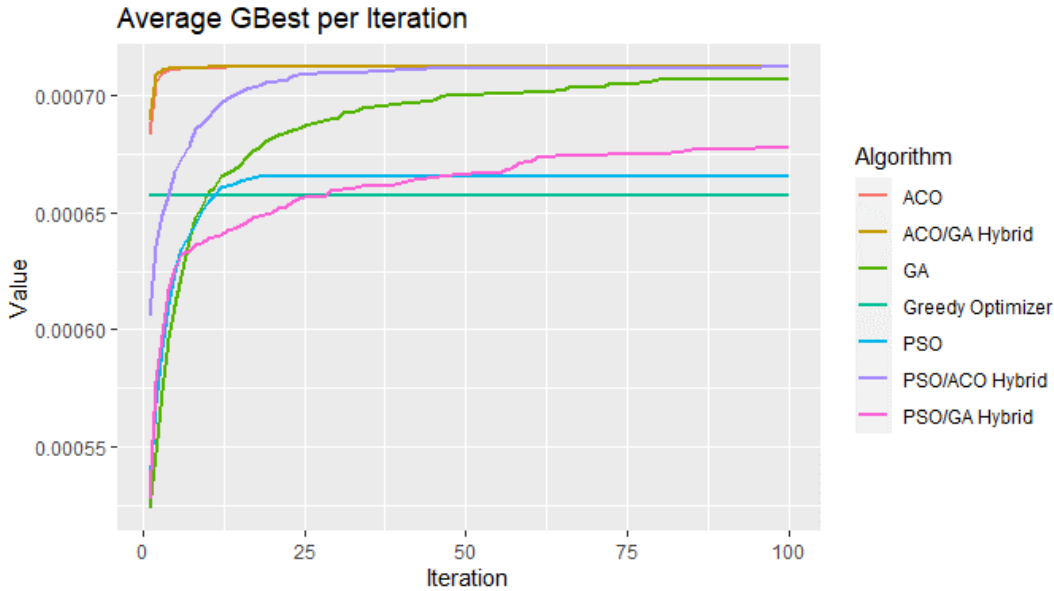


Figure 41: Line plot showing model score per iteration for the Hybrid vs Base algorithms.

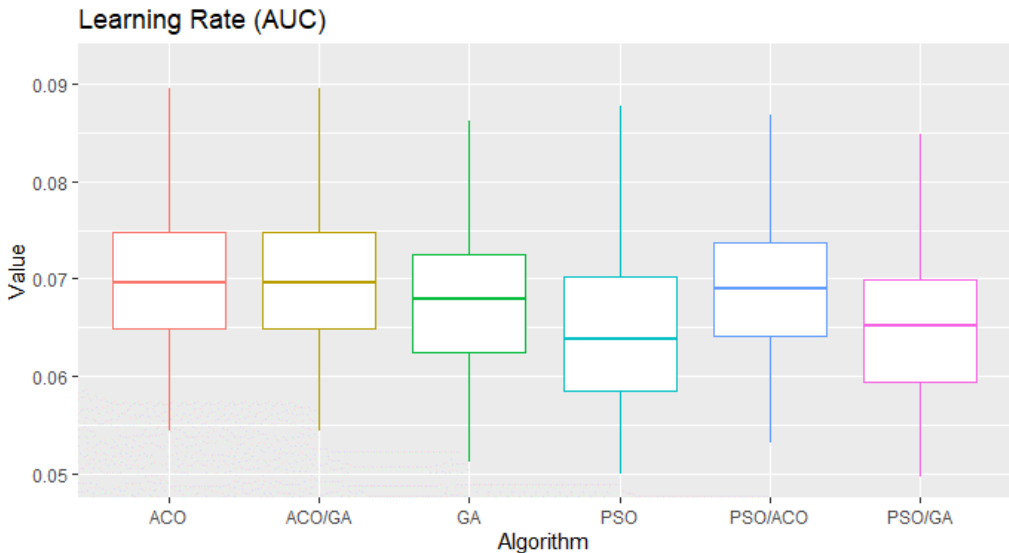


Figure 42: Box plot showing model score for the Hybrid vs Base algorithms.

This result brought a little scepticism due to how close they seemed in the line and box graphs. However, considering the consistency in results observed for all ACO models in the previous Section 4.3, it seemed that this mean difference, though small, was indeed unexpected and statistically significant. Shown in Figures 43 and 44 are cropped versions of the earlier figures, highlighting only the ACO and ACO/GA algorithms.

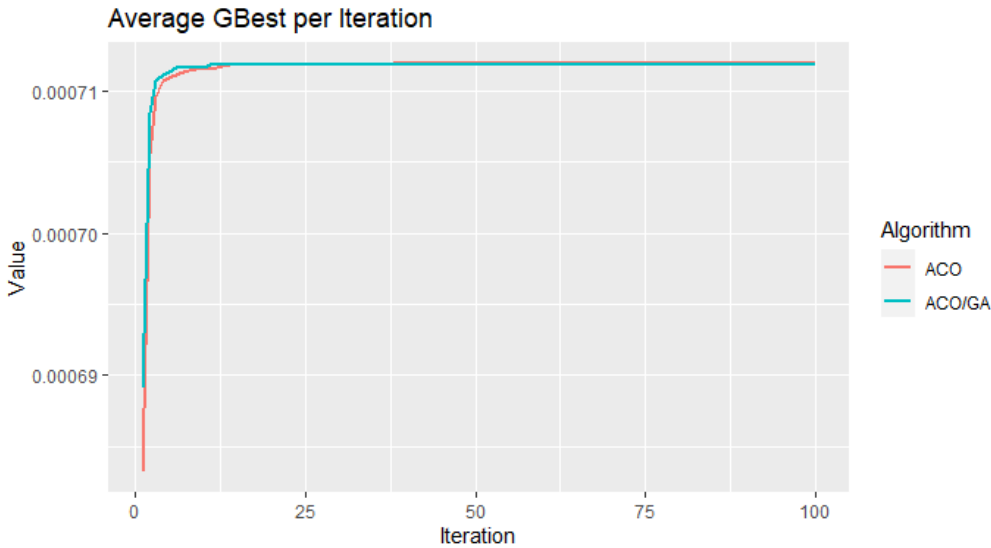


Figure 43: Line plot showing model score per iteration for the cropped Hybrid vs Base algorithms.

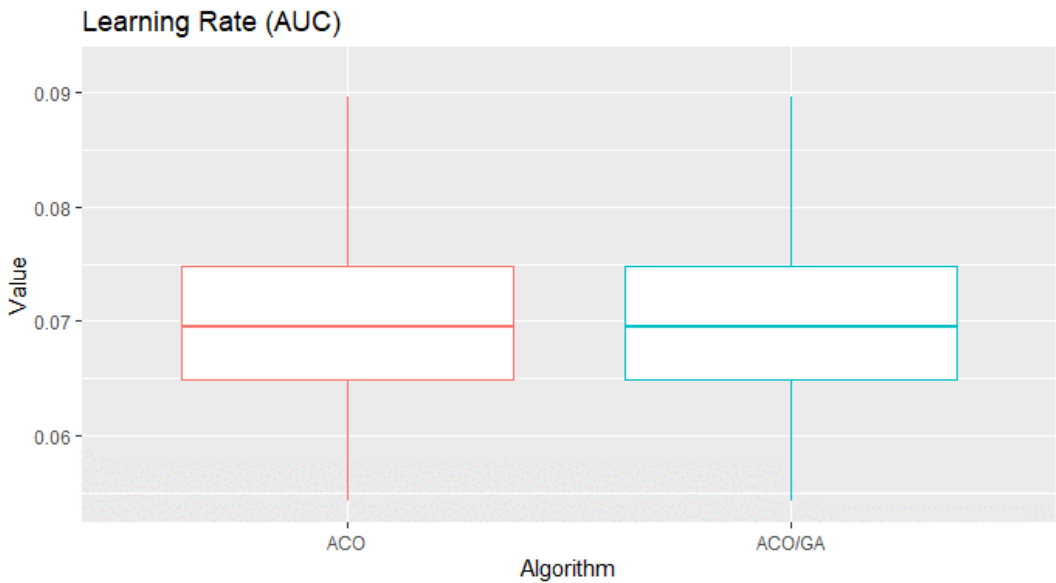


Figure 44: Box plot showing model score for the Cropped Hybrid vs Base algorithms.

In second place was the ACO algorithm and the PSO/ACO hybrid algorithm was 3rd. The 1st place ACO/GA and 2nd place ACO algorithms beat the 3rd place PSO/ACO model both with a statistical significance level of $p < 2.2e-16$.

When the test was expanded to maps having larger amounts of cities, the difference between the ACO/GA hybrid and the ACO algorithm became even more apparent as shown in Figures 45-47. It was also found that all other algorithms except these two were outperformed by the greedy optimization algorithm as map sizes increased. The worst-performing algorithm of the group as map sizes increased was revealed to be the PSO/GA hybrid.

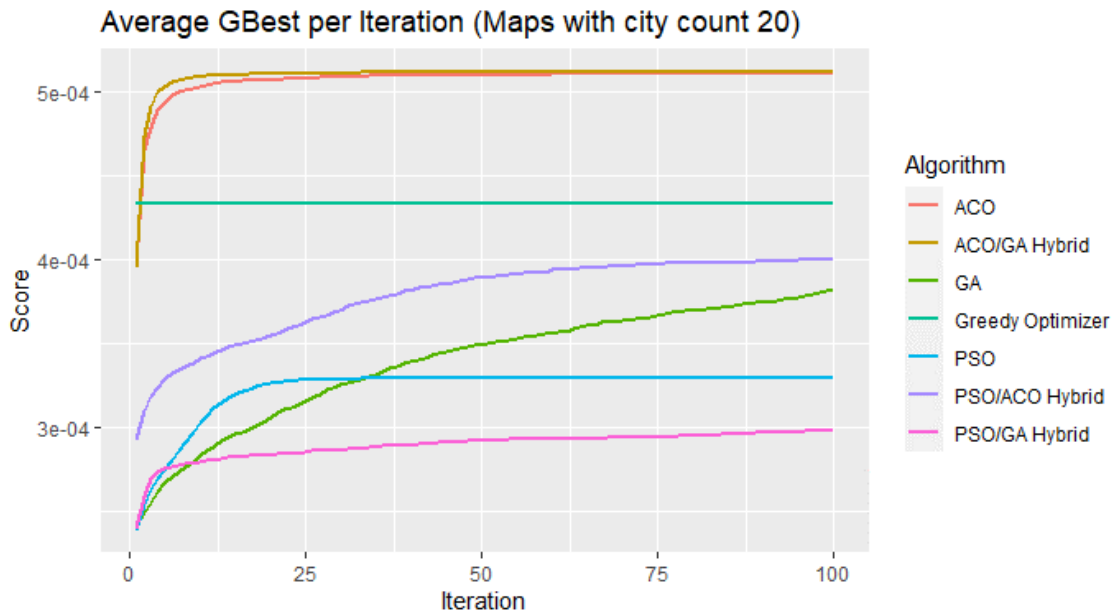


Figure 45: Line plot showing model score per iteration for the Hybrid vs Base algorithms (city count 20).

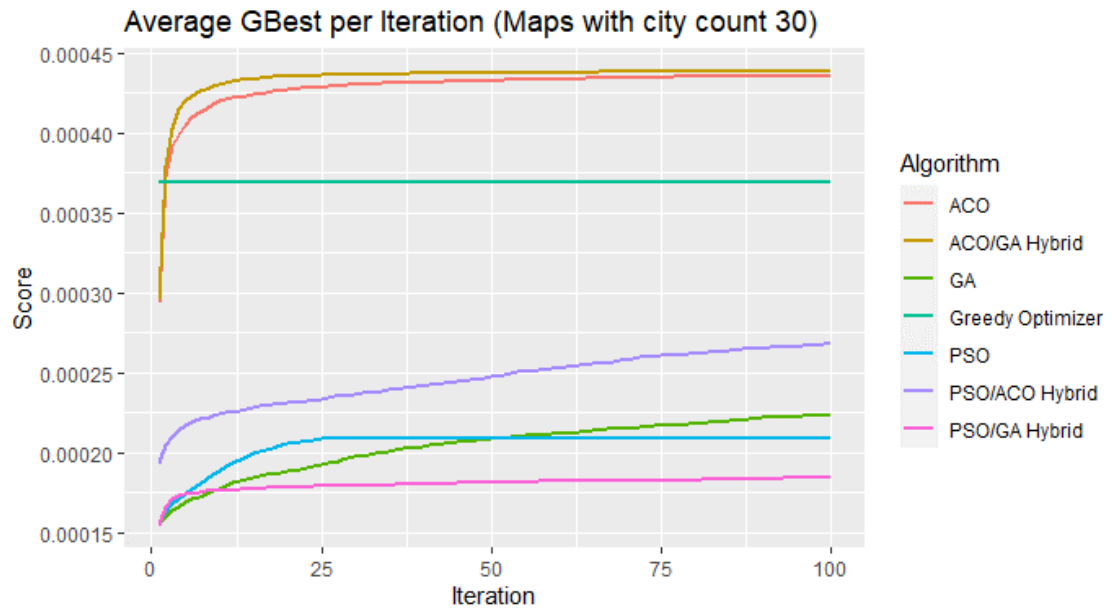


Figure 46: Line plot showing model score per iteration for the Hybrid vs Base algorithms (city count 30)

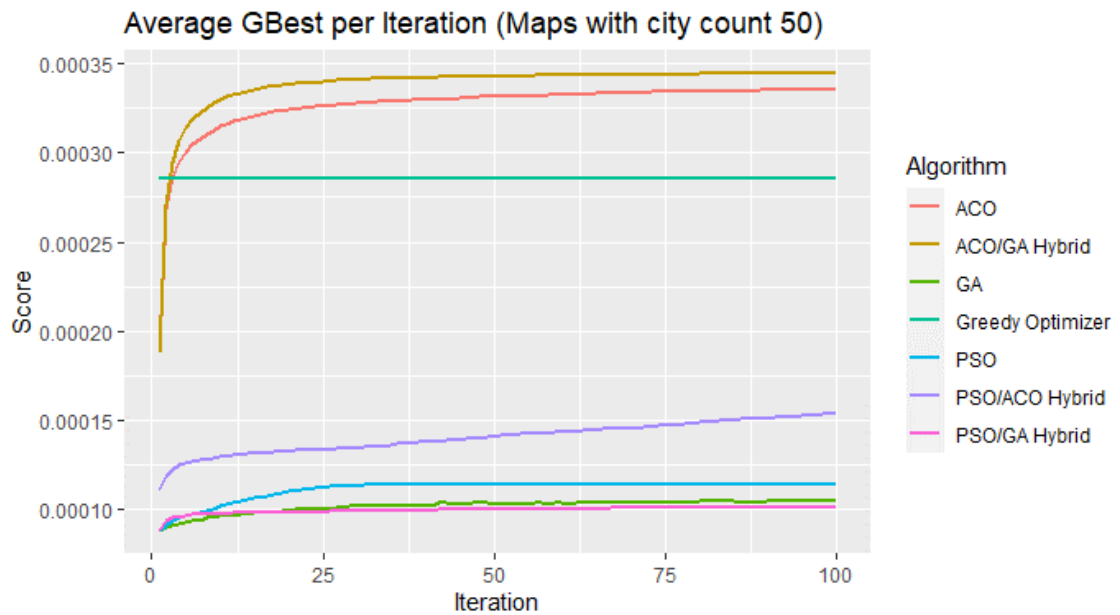


Figure 47: Line plot showing model score per iteration for the Hybrid vs Base algorithms (city count 50)

Also noted as map sizes increased, was that the PSO/GA hybrid algorithm was observed to be the worst-performing algorithm in the group. A possible explanation is that the already observed tendency for premature convergence in the particle swarm algorithm caused by the low inertia weight was aggravated by the too-effective Genetic Algorithm using the high selection pressure *tournament sampling* and *steady-state* techniques. In this case, the simple solution would only be to raise the inertia weight. Unfortunately, when a supplementary investigation was done into this proposed solution by raising the inertia weight used in the algorithm to the recommended 0.8, the results detailed in Figure 48 show that, though slightly better, not much gain in performance was observed. Another suggestion is that the sequential hybridization method is not a suitable method for combining the PSO and GA algorithms. Alternatively, the PSO and GA algorithms might simply just not work well together. Further research would have to be done to address a definite cause and solution for this observed behaviour.

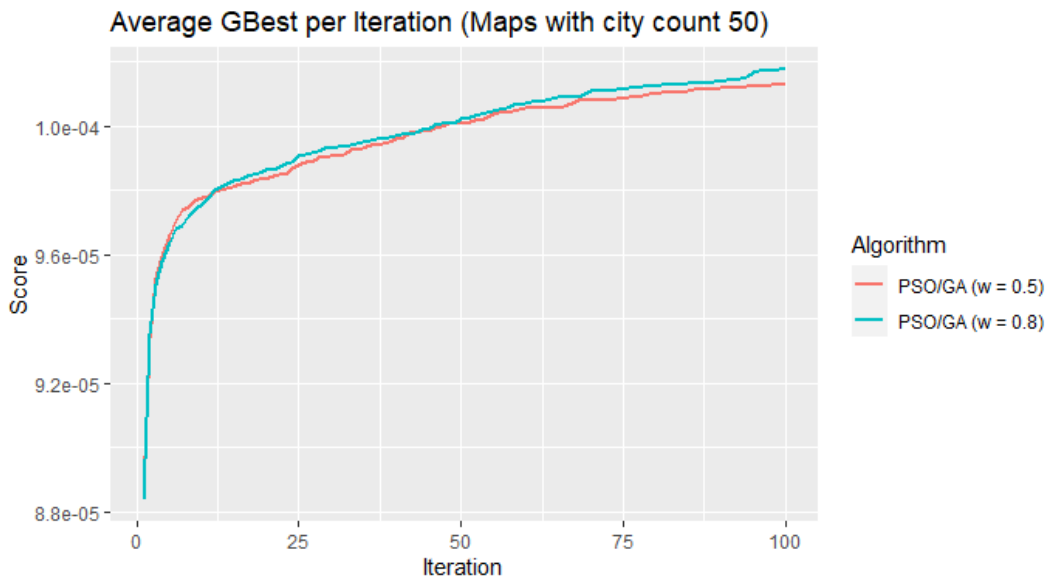


Figure 48: Line plot showing model score per iteration for the supplemental experiment testing the effects of an increased inertia weight on the PSO/GA sequential hybrid algorithm.

4.5 Algorithm Runtimes

As an extra point of interest, Table 11 was included, containing the average seconds per iteration for the algorithms run in this project. Though admittedly not entirely reliable, for improved accuracy the values shown in this table were averaged over 1,000 iterations for each algorithm used.

Some things to note in this table are, firstly, that the fastest algorithm at smaller TSP map sizes was the GA's plain tournament sampling technique (without the steady-state enhancement), but as map sizes increased, its speed was overtaken by both PSO algorithms. Of the GA algorithms, tournament sampling was the fastest running fitness function of the group and rank-based sampling was the slowest, both by a significant margin. Both of the PSO variations took the same time to run on average as there is only a slight variation in their methodologies. The ACO algorithms were the longest to run in the group leaving very mixed results as to which one was faster. This is most likely caused by the complexity of the calculations required. For each ants' solution construction step, it is required to check all valid paths for the most attractive next city to visit. This causes a mass of function loops as ants construct their solutions which, when completed, also adds more function loops for graph pheromone updates. This slowness shown by the ACO was also reflected in all hybrids using the ACO methodology. The fastest performing hybrid was the PSO/GA hybrid which stayed clear of the ACO methodology, and the slowest was the ACO/GA using the GA's added steady-state enhancement. Elitism was not included in this table because, in this project, it simply excluded the fittest members of the population from being overwritten without adding any significant complexity to the algorithm. So, in the event where runtime speeds gain priority, Elitism would be the superior enhancer choice over the Steady-State.

Table 11: Algorithm Runtimes

Average Seconds per Iteration		No. cities on the map			
		10	20	30	50
GA	Stochastic Universal Sampling	0.0032	0.0032	0.0038	0.0132
	Rank Based Sampling	0.02	0.0109	0.0133	0.0181
	Tournament Sampling	0.0005	0.0022	0.0013	0.0055
	+ Steady-State	0.0042	0.0107	0.0094	0.0188
PSO	Original Particle Swarm	0.0012	0.002	0.0029	0.0035
	Modified Particle Swarm	0.0012	0.002	0.0028	0.0035
ACO	Ant System	0.0183	0.0235	0.0512	0.3031
	Max-Min Ant System	0.0171	0.0231	0.0504	0.192
	Ant Colony System	0.0104	0.0436	0.1172	0.1518
Hybrid	ACO/GA	0.019	0.0483	0.0532	0.2431
	PSO/ACO	0.0066	0.0229	0.05	0.1296
	PSO/GA	0.002	0.0042	0.0061	0.0119

5. CONCLUSION

This section discusses the results drawn from the experiments concerning the objective of this research project.

5.1 Problem Definition & Research Overview

The main objective of this study was to establish the benefits of hybridization. It was expected that the Ant Colony Optimization algorithm would win by design, but this project aimed to prove otherwise. Its goal was to serve as an advocate for hybridization with the hypothesis that a hybrid algorithm can perform better than all the base algorithms used in this study. To do this, first, preliminary experiments had to be conducted to configure the best representative for each of the base algorithms used. After that, a comparative analysis between the best-performing representative of all base algorithms against the built hybrid models was conducted.

5.2 Study Outcome

In this project, the chosen biologically inspired algorithms were explored and the state-of-the-art variations in their design were expounded. A statistical test was established and used to define the composition of the best performing representative for each of the base algorithms experimented on. Hybrid models were constructed using the methodologies of the base representatives as a foundation and a comparative analysis was done between those hybrid models versus their base counterparts. The fundamental research question that this dissertation aimed to answer was: *‘Can hybridization methods applied to biologically inspired optimization algorithms improve their efficiency in approximating a solution to the Travelling Salesman Problem?’*. Through the results given in Section 4.4, this dissertation has successfully answered that question with a resounding ‘Yes, they can!’.

The ACO/GA hybrid algorithm was found to be the best performing algorithm when comparing its performance, represented by the AUC, against the base ACO which was the second-best algorithm. Using the Mann-Whitney U Test, it was found that the ACO/GA algorithm performed better with a statistical significance value of $7.089e-12$ for TSP maps of city count 10. This successfully passed the statistical threshold enforced

in this study (0.05) and confirmed the hypothesis of this study. This benefit in performance that the ACO/GA offered was further buttressed by the results drawn as the TSP difficulty increased.

To tackle the main research, question a series of granular research sub-questions were posed. The mass of information stored in the various Sections of this project has all contributed to answering these questions.

5.2.1 Methodological Understanding

Much research was done to highlight the state-of-the-art variations in algorithm methodology that have arisen over the years. An effort has been made to highlight, in great detail, some of the most popular algorithm variants and their mechanisms.

For the *Genetic Algorithm*, fitness function variants:

- Roulette Wheel Sampling
- Stochastic Universal Sampling
- Rank-Based Sampling
- Tournament Sampling

And the Enhancers:

- Elitism
- Steady-State

For the discrete *Particle Swarm Optimization Algorithm*:

- Base Particle Swarm
- Modified Particle Swarm

For the *Ant Colony Optimization Algorithm*:

- Ant System
- Max-Min Ant System
- Ant Colony System

5.2.2 Experimental Understanding

Despite gaining an understanding of the state-of-the-art, much experimentation was further done in this project to maximize base algorithm performances. For the Genetic Algorithm, an attempt was made to introduce a delta variable to the tournament sampling technique and to localize the effects of the Steady-State technique. Both attempts were unsuccessful leaving the standard versions as the best performing representatives. The Particle Swarm, however, saw improved performance levels with the re-introduction of particle inertia and this showed that its discrete methodology still had room for improvement. The results returned from all Ant Colony Algorithm models were rigidly consistent and were understood to suggest an already maximised performance within the problem domain difficulty level.

Despite finding the best performing variants, further effort was made to extract the optimum configurations that returned the best performance results for each of them. The optimum Genetic Algorithm composition found in this study comprised of using the original *tournament selection* algorithm along with the original *steady-state* enhancer. The optimal discrete Particle Swarm configuration found for this study used the *modified particle swarm* optimization algorithm methodology having an $a1$ variable linearly progressing from an alpha of 30% to a beta of 70%, over the first 50% of the iterations, along with an inertia weight of 0.5. The Ant Colony algorithm remained with the basic *ant system* methodology as its representative.

All experimentation results drawn were also evaluated statistically through the Mann-Whitney U test for significance and, after the hybrid algorithms were built and compared against these optimized base versions, the Ant Colony - Genetic Algorithm sequential hybrid algorithm was still found to statistically be the best performing algorithm overall.

5.2.3 Theoretical Understanding

Explored in Section 2.2 was the methodology for algorithm optimization, given a problem domain (or search space), aiming to avoid premature convergence on a local optimum while efficiently zoning in on the global optimum. The principal mechanism for optimization (*Intensification*), which was manifested in the GA as raising the

selection pressure and in the Swarm Intelligence algorithms as community exploitation, was viewed as a ‘two-edged sword’ which was safeguarded by counterbalancing mechanisms (*Diversification*). In this project, a few approaches were explored with an aim to dull this ‘sword’, however, the results of experiments done in Section 4 generally returned a lack of success, for example, the introduction of the delta variable to the Tournament Sampling technique (Section 4.1.1) or localizing the effects of the Steady-State technique (Section 4.1.2).

These findings reveal an understanding that the goal of any optimization algorithm design is actually to increase the intensification mechanism(s) of the algorithm as close as possible to a problem-specific threshold, past which, premature convergence becomes an issue. As long as this problem-specific threshold is not passed, weakening the intensification mechanism(s) only serves to slow down or possibly cripple the performance of the optimization algorithm. Conversely, the purpose of the diversification mechanism(s) can further be understood as being the safeguard to make sure that the threshold is not crossed. Unfortunately, a method to determine the value of this problem-specific threshold, other than simply through experimentation, has not been isolated from the results of this dissertation.

5.3 Limitations and improvements

This project suffered from hardware limitations which restricted the problem sizes that the algorithm models could be tested on. Some of the experiments run were statistically inconclusive which may not have been the case if the experiments were run using maps with larger numbers of cities, allowing more space for variance in optimization speeds. This point is demonstrated by comparing the difference in performance between the ACO/GA hybrid and the ACO algorithm observed in Figures 41 (map of size 10) and Figure 47 (map of size 50). For more conclusive results, a system upgrade, as well as a larger number of experiments, would have to be obtained.

5.4 Future Work & Research

Of the algorithms explored in this study, PSO seemed the ‘problem child’ in terms of learning rate. It was the weakest performing base algorithm and whenever its methodology was hybridized with the other base algorithms the resulting hybrid actually

produced worse results than if hybridization was not employed (Compare the ACO vs. PSO/ACO, and GA vs. PSO/GA in Figure 41). Of course, this conclusion reflects on the efficacy of the PSO adaptation for discrete domains used in this study rather than on the PSO methodology itself. Along with this, shortcomings in the design proposed in the literature were voiced when this adaptation was further analysed (particle inertia). These conclusions suggest that more work is needed to refine this methodology or to develop a completely new technique for discrete PSO adaptation.

Also scattered throughout this dissertation were many algorithm mechanisms enticing incentives to tinker and explore if given the time. For example, with the ACO pheromone update, it would be interesting to see what would happen if the route scores were first normalised to the range $[-1, 1]$ from worst to best scores. This would penalise a route for producing a worse-than-average result and make it more likely for new routes to be chosen.

Finally, demonstrated in this study is proof of the benefits that can be drawn through hybridization, and the ingenuity that exists in the field of algorithm development through exploration of the variations in techniques and approaches that have been devised for these algorithms over the years. It is hoped that this dissertation has provided an incentive for applying this ingenuity to the developing sector of hybridization strategies.

6. BIBLIOGRAPHY

- Ahmadi, P., & Dincer, I. (2010). Exergoenvironmental analysis and optimization of a cogeneration plant system using Multimodal Genetic Algorithm (MGA). *Energy*, 35(12), 5161–5172. <https://doi.org/10.1016/j.energy.2010.07.050>
- Ahn, C. W., & Ramakrishna, R. S. (2003). Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4), 367–385. <https://doi.org/10.1109/TEVC.2003.814633>
- Back, T., Fogel, D. B., & Michalewicz, Z. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. CRC Press.
- Back, T., Hammel, U., & Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1), 3–17. <https://doi.org/10.1109/4235.585888>
- Blum, C., & Li, X. (2008). Swarm Intelligence in Optimization. In C. Blum & D. Merkle (Eds.), *Swarm Intelligence: Introduction and Applications* (pp. 43–85). Springer. https://doi.org/10.1007/978-3-540-74089-6_2
- Braun, H. (1991). On solving travelling salesman problems by genetic algorithms. In H.-P. Schwefel & R. Männer (Eds.), *Parallel Problem Solving from Nature* (pp. 129–133). Springer. <https://doi.org/10.1007/BFb0029743>
- Castillo, O., Martínez-Marroquín, R., Melin, P., Valdez, F., & Soria, J. (2012). Comparative study of bio-inspired algorithms applied to the optimization of type-1 and type-2 fuzzy controllers for an autonomous mobile robot. *Information Sciences*, 192, 19–38. <https://doi.org/10.1016/j.ins.2010.02.022>
- Chen, Y., Miao, D., & Wang, R. (2010). A rough set approach to feature selection based on ant colony optimization. *Pattern Recognition Letters*, 31(3), 226–233. <https://doi.org/10.1016/j.patrec.2009.10.013>
- Cheng, R., & Jin, Y. (2015). A social learning particle swarm optimization algorithm for scalable optimization. *Information Sciences*, 291, 43–60. <https://doi.org/10.1016/j.ins.2014.08.039>
- Coley, D. A. (1999). *An Introduction To Genetic Algorithms For Scientists And Engineers*. World Scientific Publishing Company.
- Das, S., Abraham, A., & Konar, A. (2008). Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. In Y. Liu, A. Sun, H. T. Loh, W. F. Lu, & E.-P. Lim

- (Eds.), *Advances of Computational Intelligence in Industrial Systems* (pp. 1–38). Springer. https://doi.org/10.1007/978-3-540-78297-1_1
- Delgarm, N., Sajadi, B., Kowsary, F., & Delgarm, S. (2016). Multi-objective optimization of the building energy performance: A simulation-based approach by means of particle swarm optimization (PSO). *Applied Energy*, 170, 293–303. <https://doi.org/10.1016/j.apenergy.2016.02.141>
- Delorme, M., Iori, M., & Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1), 1–20. <https://doi.org/10.1016/j.ejor.2016.04.030>
- Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research*, 7, 1–30.
- Di Leo, G., & Sardanelli, F. (2020). Statistical significance: P value, 0.05 threshold, and applications to radiomics—reasons for a conservative approach. *European Radiology Experimental*, 4(1), 18. <https://doi.org/10.1186/s41747-020-0145-y>
- Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4), 28–39. <https://doi.org/10.1109/MCI.2006.329691>
- Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2–3), 243–278. <https://doi.org/10.1016/j.tcs.2005.05.020>
- Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53–66. <https://doi.org/10.1109/4235.585892>
- Dorigo, M., & Stützle, T. (2019). Ant Colony Optimization: Overview and Recent Advances. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 311–351). Springer International Publishing. https://doi.org/10.1007/978-3-319-91086-4_10
- Esmin, A. A. A., Coelho, R. A., & Matwin, S. (2015). A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data. *Artificial Intelligence Review*, 44(1), 23–45. <https://doi.org/10.1007/s10462-013-9400-4>

- Gambardella, L. M., & Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. *Proceedings of IEEE International Conference on Evolutionary Computation*, 622–627. <https://doi.org/10.1109/ICEC.1996.542672>
- Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12), 579–581. <https://doi.org/10.1007/BF00462870>
- Grefenstette, J. J. (2013). *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Psychology Press.
- Hoffman, K. L., & Padberg, M. (2001). Traveling Salesman Problem (TSP) Traveling salesman problem. In S. I. Gass & C. M. Harris (Eds.), *Encyclopedia of Operations Research and Management Science* (pp. 849–853). Springer US. https://doi.org/10.1007/1-4020-0611-X_1068
- Huang, C.-L., Huang, W.-C., Chang, H.-Y., Yeh, Y.-C., & Tsai, C.-Y. (2013). Hybridization strategies for continuous ant colony optimization and particle swarm optimization applied to data clustering. *Applied Soft Computing*, 13(9), 3864–3872. <https://doi.org/10.1016/j.asoc.2013.05.003>
- Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, 29(3), 31–44. <https://doi.org/10.1109/2.485891>
- Johnson, J. M., & Rahmat-Samii, V. (1997). Genetic algorithms in engineering electromagnetics. *IEEE Antennas and Propagation Magazine*, 39(4), 7–21. <https://doi.org/10.1109/74.632992>
- Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80(5), 8091–8126. <https://doi.org/10.1007/s11042-020-10139-6>
- Kelter, R. (2020). Analysis of Bayesian posterior significance and effect size indices for the two-sample t-test to support reproducible medical research. *BMC Medical Research Methodology*, 20(1), 88. <https://doi.org/10.1186/s12874-020-00968-2>
- Kennedy, J. F., & Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4, 1942–1948 vol.4. <https://doi.org/10.1109/ICNN.1995.488968>
- Kennedy, J. F., Eberhart, R. C., & Shi, Y. (2001). *Swarm intelligence*. Morgan Kaufmann Publishers.

- Khourdifi, Y., & Bahaj, M. (2019). Heart Disease Prediction and Classification Using Machine Learning Algorithms Optimized by Particle Swarm Optimization and Ant Colony Optimization. *International Journal of Intelligent Engineering and Systems*, 12. <https://doi.org/10.22266/ijies2019.0228.24>
- Kim, T. K. (2015). T test as a parametric statistic. *Korean Journal of Anesthesiology*, 68(6), 540–546. <https://doi.org/10.4097/kjae.2015.68.6.540>
- Korte, B., & Vygen, J. (2012). *Combinatorial Optimization: Theory and Algorithms* (Vol. 21). Springer Science & Business Media. <https://doi.org/10.1007/978-3-642-24488-9>
- Lin, S. (1965). Computer Solutions of the Traveling Salesman Problem. *Bell System Technical Journal*, 44(10), 2245–2269. <https://doi.org/10.1002/j.1538-7305.1965.tb04146.x>
- Luan, J., Yao, Z., Zhao, F., & Song, X. (2019). A novel method to solve supplier selection problem: Hybrid algorithm of genetic algorithm and ant colony optimization. *Mathematics and Computers in Simulation*, 156, 294–309. <https://doi.org/10.1016/j.matcom.2018.08.011>
- Mandloi, M., & Bhatia, V. (2016). A low-complexity hybrid algorithm based on particle swarm and ant colony optimization for large-MIMO detection. *Expert Systems with Applications*, 50, 66–74. <https://doi.org/10.1016/j.eswa.2015.12.008>
- Mann, H. B., & Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1), 50–60.
- Miller, B. L., & Goldberg, D. (1995). Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Syst.*, 9(3), 193–212.
- Mirjalili, S. (2019a). Genetic Algorithm. In S. Mirjalili (Ed.), *Evolutionary Algorithms and Neural Networks: Theory and Applications* (pp. 43–55). Springer International Publishing. https://doi.org/10.1007/978-3-319-93025-1_4
- Mirjalili, S. (2019b). Introduction to Evolutionary Single-Objective Optimisation. In S. Mirjalili (Ed.), *Evolutionary Algorithms and Neural Networks: Theory and Applications* (pp. 3–14). Springer International Publishing. https://doi.org/10.1007/978-3-319-93025-1_1
- Moon, C., Kim, J., Choi, G., & Seo, Y. (2002). An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of*

- Operational Research*, 140(3), 606–617. [https://doi.org/10.1016/S0377-2217\(01\)00227-2](https://doi.org/10.1016/S0377-2217(01)00227-2)
- Moradi, M. H., & Abedini, M. (2012). A combination of genetic algorithm and particle swarm optimization for optimal DG location and sizing in distribution systems. *International Journal of Electrical Power & Energy Systems*, 34(1), 66–74. <https://doi.org/10.1016/j.ijepes.2011.08.023>
- Nachar, N. (2008). The Mann-Whitney U: A Test for Assessing Whether Two Independent Samples Come from the Same Distribution. *Tutorials in Quantitative Methods for Psychology*, 4(1), 13–20. <https://doi.org/10.20982/tqmp.04.1.p013>
- Omidinasab, F., & Goodarzimehr, V. (2019). A Hybrid Particle Swarm Optimization and Genetic Algorithm for Truss Structures with Discrete Variables. *Journal of Applied and Computational Mechanics*, Online First. <https://doi.org/10.22055/jacm.2019.28992.1531>
- Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1(1), 33–57. <https://doi.org/10.1007/s11721-007-0002-0>
- Qolomany, B., Ahmad, K., Al-Fuqaha, A., & Qadir, J. (2020). *Particle Swarm Optimized Federated Learning For Industrial IoT and Smart City Services*. <https://doi.org/10.48550/arXiv.2009.02560>
- Razali, N. M., & Geraghty, J. (2011). *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. 6.
- Samà, M., Pellegrini, P., D'Ariano, A., Rodriguez, J., & Pacciarelli, D. (2016). Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological*, 85, 89–108. <https://doi.org/10.1016/j.trb.2016.01.005>
- Selvi, V., & Umarani, Dr. R. (2010). Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. *International Journal of Computer Applications*, 5. <https://doi.org/10.5120/908-1286>
- Shelokar, P. S., Siarry, P., Jayaraman, V. K., & Kulkarni, B. D. (2007). Particle swarm and ant colony algorithms hybridized for improved continuous optimization. *Applied Mathematics and Computation*, 188(1), 129–142. <https://doi.org/10.1016/j.amc.2006.09.098>

- Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 69–73. <https://doi.org/10.1109/ICEC.1998.699146>
- Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, 1, 101–106 vol. 1. <https://doi.org/10.1109/CEC.2001.934377>
- Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155–1173. <https://doi.org/10.1016/j.ejor.2006.06.046>
- Soeken, M., Wille, R., Kuhlmann, M., Gogolla, M., & Drechsler, R. (2010). Verifying UML/OCL models using Boolean satisfiability. *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 1341–1344. <https://doi.org/10.1109/DATE.2010.5457017>
- Stützle, T., & Hoos, H. H. (2000). MAX–MIN Ant System. *Future Generation Computer Systems*, 16(8), 889–914. [https://doi.org/10.1016/S0167-739X\(00\)00043-1](https://doi.org/10.1016/S0167-739X(00)00043-1)
- Thangaraj, R., Pant, M., Abraham, A., & Bouvry, P. (2011). Particle swarm optimization: Hybridization perspectives and experimental illustrations. *Applied Mathematics and Computation*, 217(12), 5208–5226. <https://doi.org/10.1016/j.amc.2010.12.053>
- Vázquez, E. G., Escolano, A. Y., Riaño, P. G., & Junquera, J. P. (2001). Repeated Measures Multiple Comparison Procedures Applied to Model Selection in Neural Networks. In J. Mira & A. Prieto (Eds.), *Bio-Inspired Applications of Connectionism* (pp. 88–95). Springer. https://doi.org/10.1007/3-540-45723-2_10
- Wang, K.-P., Huang, L., Zhou, C.-G., & Pang, W. (2003). Particle swarm optimization for traveling salesman problem. *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, 3, 1583–1585 Vol.3. <https://doi.org/10.1109/ICMLC.2003.1259748>
- Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4(2), 65–85. <https://doi.org/10.1007/BF00175354>

- Wilcoxon, F. (1992). Individual Comparisons by Ranking Methods. In S. Kotz & N. L. Johnson (Eds.), *Breakthroughs in Statistics: Methodology and Distribution* (pp. 196–202). Springer. https://doi.org/10.1007/978-1-4612-4380-9_16
- Yang, Q., & Yoo, S.-J. (2018). Optimal UAV Path Planning: Sensing Data Acquisition Over IoT Sensor Networks Using Multi-Objective Bio-Inspired Algorithms. *IEEE Access*, 6, 13671–13684. <https://doi.org/10.1109/ACCESS.2018.2812896>
- Yao, Z., Liu, J., & Wang, Y.-G. (2008). Fusing genetic algorithm and Ant Colony Algorithm to optimize virtual enterprise partner selection problem. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3614–3620. <https://doi.org/10.1109/CEC.2008.4631287>
- Yousefikhoshbakht, M. (2021). Solving the Traveling Salesman Problem: A Modified Metaheuristic Algorithm. *Complexity*, 2021, e6668345. <https://doi.org/10.1155/2021/6668345>
- Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4), 3563–3573. <https://doi.org/10.1016/j.eswa.2010.08.145>
- Zhong, W., Zhang, J., & Chen, W. (2007). A novel discrete particle swarm optimization to solve traveling salesman problem. *2007 IEEE Congress on Evolutionary Computation*, 3283–3287. <https://doi.org/10.1109/CEC.2007.4424894>

7. APPENDIX

GitHub link for dissertation resources: <https://github.com/elihuessien/Dissertation>

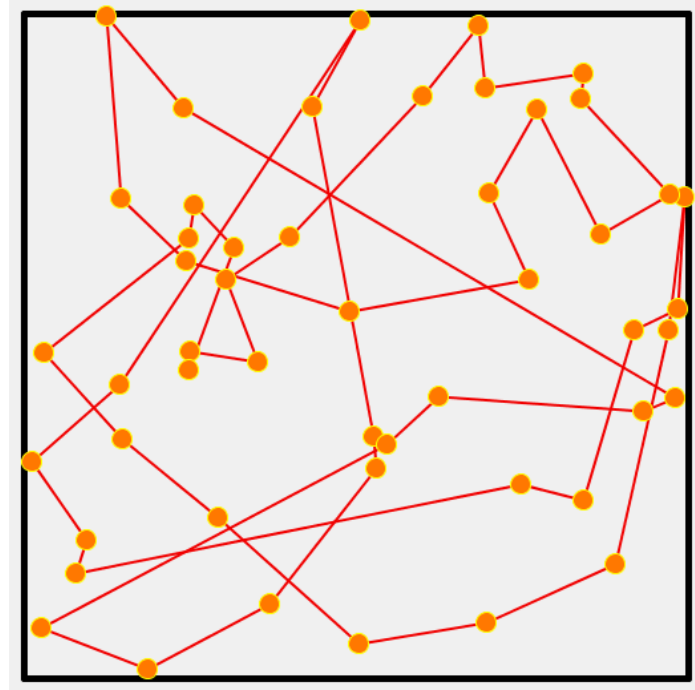


Figure 49: GA solution for TSP map (50 cities)

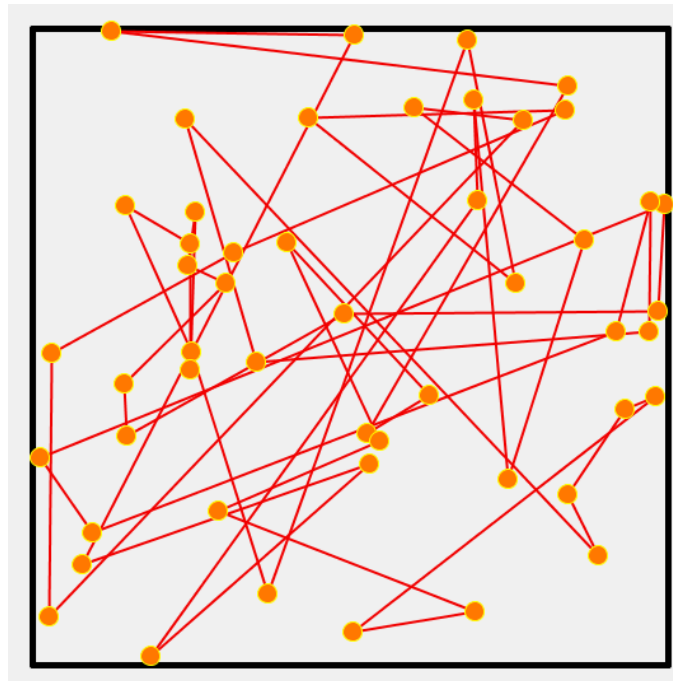


Figure 50: PSO Solution for TSP map (50 cities)

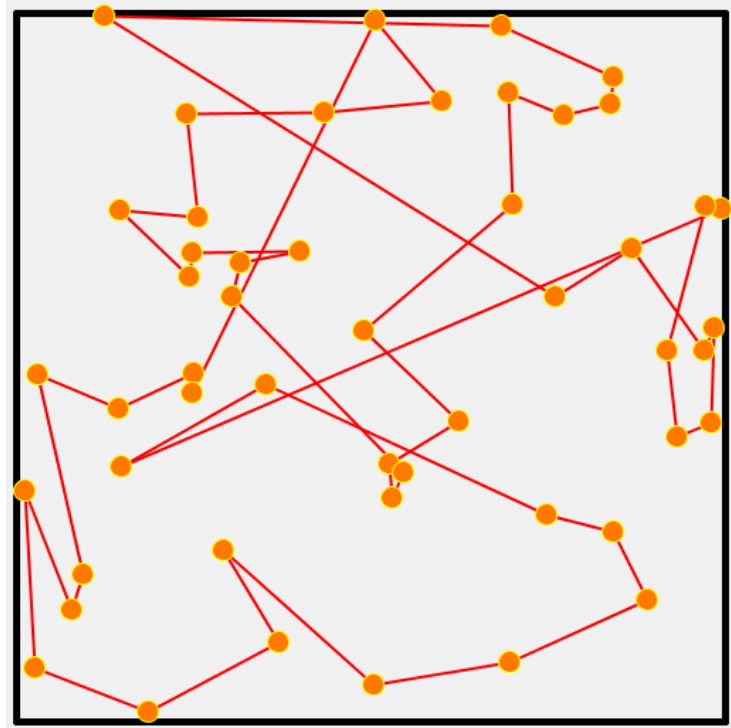


Figure 51: ACO solution for TSP map (50 cities)

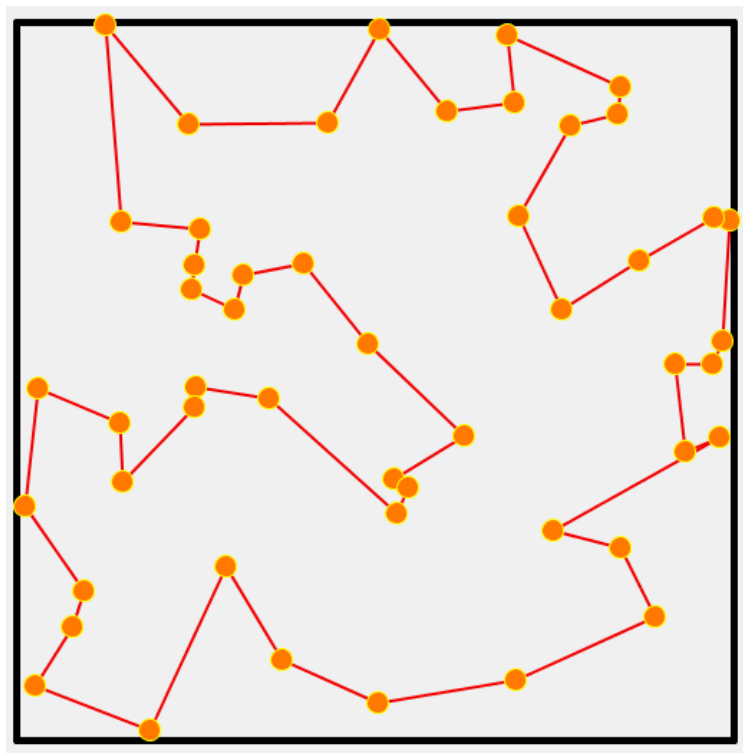


Figure 52: ACO/GA solution for TSP map (50 cities)

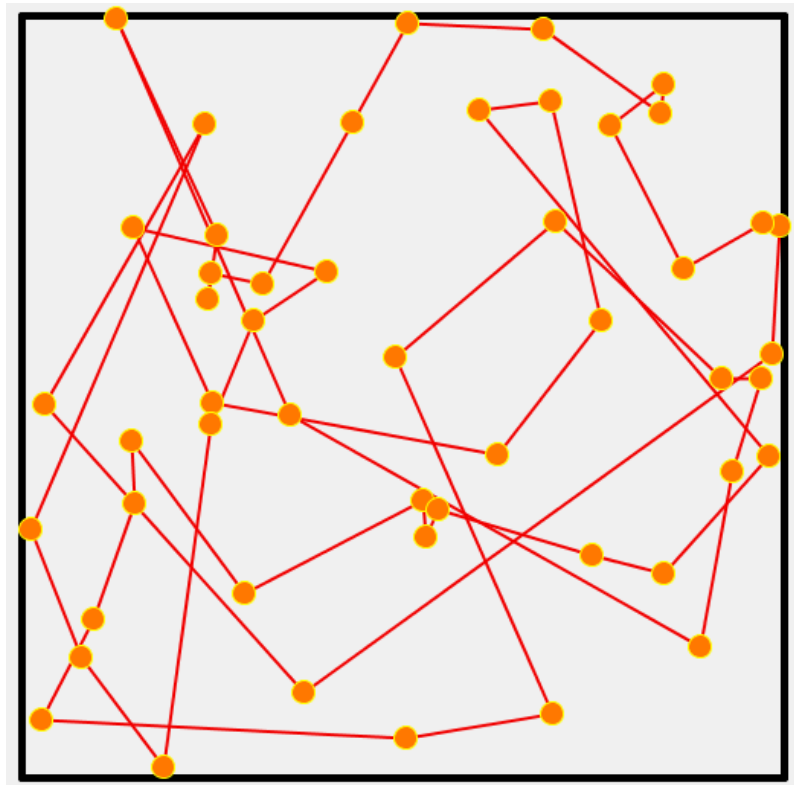


Figure 53: PSO/ACO solution for TSP map (50 cities)

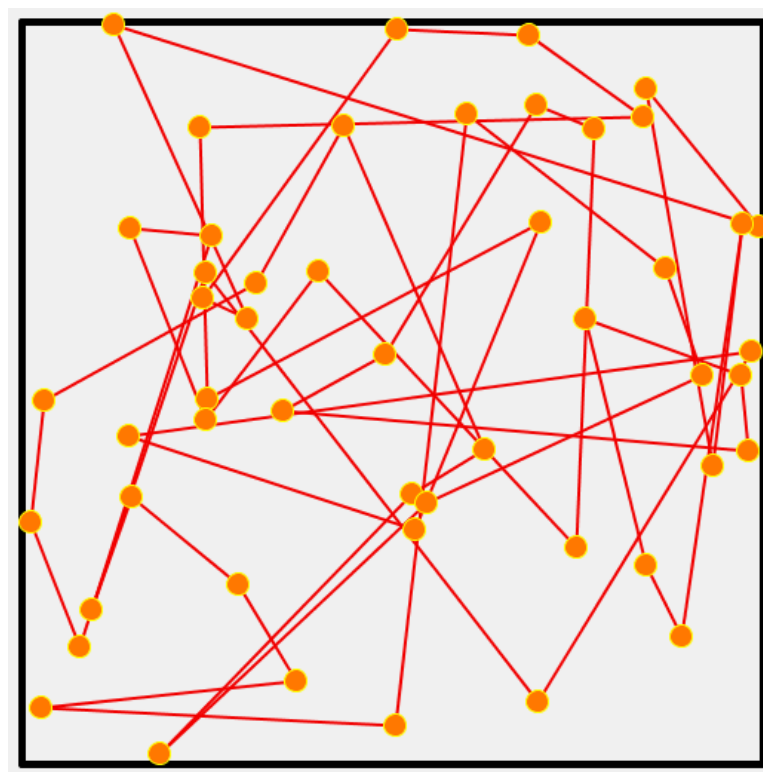


Figure 54: PSO/GA solution for TSP map (50 cities)

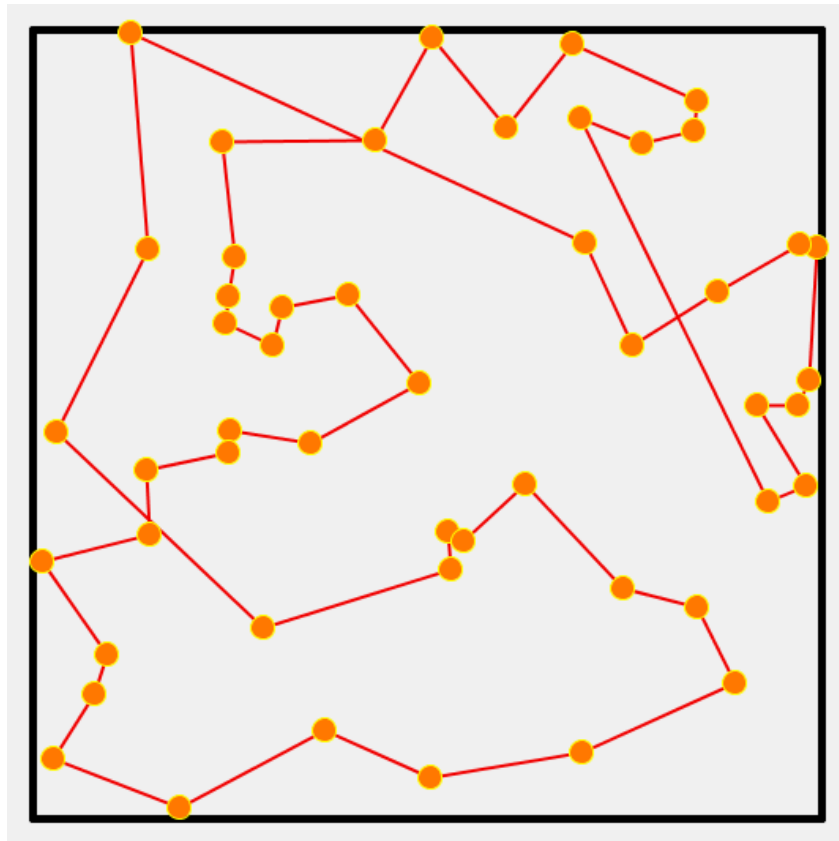


Figure 55: Greedy-Optimizer solution for TSP map (50 cities)