Advocating Hybridization Strategies using Biologically Inspired Optimization Algorithms and the Traveling Salesman Problem



Elihu Essien-Thompson

A dissertation submitted in partial fulfilment of the requirements of Technological University Dublin for the degree of

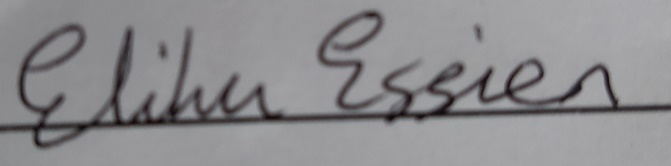
M.Sc. in Computer Science (Advanced Software Development)

**April 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 test 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:** 

**Date: 16/05/2022**

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](#_Toc105434681)

[ACKNOWLEDGEMENTS iii](#_Toc105434682)

[TABLE OF CONTENTS iv](#_Toc105434683)

[TABLE OF FIGURES vii](#_Toc105434684)

[TABLE OF TABLES ix](#_Toc105434685)

[TABLE OF EQUATIONS x](#_Toc105434686)

[1. Introduction 1](#_Toc105434687)

[1.1 Background 1](#_Toc105434688)

[1.2 Research Project 2](#_Toc105434689)

[1.3 Research Objectives 3](#_Toc105434690)

[1.4 Research Methodologies 4](#_Toc105434691)

[1.5 Scope & Limitations 5](#_Toc105434692)

[1.6 Document Outline 5](#_Toc105434693)

[2. LITERATURE REVIEW 6](#_Toc105434694)

[2.1 Combinatorial Optimization Problems 6](#_Toc105434695)

[2.2 Optimization Methodology 8](#_Toc105434696)

[2.3 History of the chosen algorithms 11](#_Toc105434697)

[2.3.1 The Genetic Algorithm 11](#_Toc105434698)

[2.3.2 Particle Swarm Optimization 12](#_Toc105434699)

[2.3.3 Ant Colony Optimization 14](#_Toc105434700)

[2.4 State-Of-The-Art in Biological Optimization 16](#_Toc105434701)

[2.4.1 Genetic Algorithm overview 16](#_Toc105434702)

[2.4.2 The Selection Operator: Fitness Function Variants 18](#_Toc105434703)

[2.4.3 The Breeding Operator: Crossover variants 22](#_Toc105434704)

[2.4.4 Genetic Algorithm Enhancers 23](#_Toc105434705)

[2.4.5 Particle Swarm Optimization overview 24](#_Toc105434706)

[2.4.6 Particle Swarm Optimization in Discreet Domains 26](#_Toc105434707)

[2.4.7 Modified Particle Swarm Optimization 28](#_Toc105434708)

[2.4.8 Ant Colony Optimization Overview 28](#_Toc105434709)

[2.4.9 The Ant System 31](#_Toc105434710)

[2.4.10 The Max-Min Ant System 32](#_Toc105434711)

[2.4.11 The Ant Colony System 33](#_Toc105434712)

[2.5 Hybridization Methodology 34](#_Toc105434713)

[3. Design & Methodology 36](#_Toc105434714)

[3.1 Data Generation 36](#_Toc105434715)

[3.2 Languages Used 36](#_Toc105434716)

[3.3 Algorithm Implementation & Configuration 38](#_Toc105434717)

[3.3.1 Genetic Algorithm Implementation Specifics 39](#_Toc105434718)

[3.3.2 Particle Swarm Optimization Implementation Specifics 41](#_Toc105434719)

[3.3.3 Ant Colony Optimization Implementation Specifics 42](#_Toc105434720)

[3.3.4 Overview of Questions to Investigate: 45](#_Toc105434721)

[3.3.5 Benchmark 45](#_Toc105434722)

[3.4 Statistical Analysis 45](#_Toc105434723)

[4. Results, Evaluation & Discussion 48](#_Toc105434724)

[4.1 Experiment 1: Genetic Algorithm 48](#_Toc105434725)

[4.1.1 Part 1 – What Fitness Function to use? 48](#_Toc105434726)

[4.1.2 Part 2 – What Enhancer to use? 51](#_Toc105434727)

[4.2 Experiment 2: Particle Swarm Optimization 54](#_Toc105434728)

[4.2.1 Part 1 – What Original Particle Swarm Configuration to Use? 54](#_Toc105434729)

[4.2.2 Part 2 – What Modified Particle Swarm Configuration to Use? 56](#_Toc105434730)

[4.2.3 Part 3 – What Particle Swarm Representative Model to Use? 57](#_Toc105434731)

[4.3 Experiment 3: Ant Colony Optimization 59](#_Toc105434732)

[4.4 Experiment 4: Hybrids vs Base 62](#_Toc105434733)

[4.5 Algorithm Runtimes 66](#_Toc105434734)

[5. Conclusion 68](#_Toc105434735)

[5.1 Problem Definition & Research Overview 68](#_Toc105434736)

[5.2 Study Outcome 68](#_Toc105434737)

[5.3 Algorithm Methodology Findings 69](#_Toc105434738)

[5.3.1 Particle Inertia 69](#_Toc105434739)

[5.3.2 Tournament Sampling Selection Pressure 69](#_Toc105434740)

[5.3.3 Hybridization Strategy 71](#_Toc105434741)

[5.4 Optimization Methodology Findings 72](#_Toc105434742)

[5.5 Limitations and improvements 73](#_Toc105434743)

[5.6 Future Work & Research 73](#_Toc105434744)

[6. BIBLIOGRAPHY 74](#_Toc105434745)

[7. Appendix 81](#_Toc105434746)

TABLE OF FIGURES

[Figure 1: Example Search Space/Landscape (Mirjalili, 2019b) 8](#_Toc105434747)

[Figure 2: Local vs Global Optimums 9](#_Toc105434748)

[Figure 3: Double-Bridge Experiment 14](file:///C:\Users\C14460702\Dissertation\Documents\Dissertation.doc.docx#_Toc105434749)

[Figure 4: Different length bridges 15](file:///C:\Users\C14460702\Dissertation\Documents\Dissertation.doc.docx#_Toc105434750)

[Figure 5: Genetic Algorithm Flowchart 17](#_Toc105434751)

[Figure 6: Roulette Wheel Sampling 18](#_Toc105434752)

[Figure 7: Stochastic Universal Sampling 20](#_Toc105434753)

[Figure 8: Rank-Based Sampling 21](#_Toc105434754)

[Figure 9: Crossover Operator (Mirjalili, 2019a) 22](#_Toc105434755)

[Figure 10: Particle Swarm Optimization Flowchart 25](#_Toc105434756)

[Figure 11: Applying Movement Vector to Position Vector for a Particle 26](#_Toc105434757)

[Figure 12: ACO flowchart 31](#_Toc105434758)

[Figure 13: Example Dataset for a map of 10 cities 36](#_Toc105434759)

[Figure 14: Solution Display Program (city count 10 and 50) 37](#_Toc105434760)

[Figure 15: TSP Objective Function in Python 37](#_Toc105434761)

[Figure 16: Example Scatter Plot generated in R 38](#_Toc105434762)

[Figure 17: Population Composition for the Genetic Algorithm 39](#_Toc105434763)

[Figure 18: GA Breeding for the TSP 40](#_Toc105434764)

[Figure 19: GA Mutation for the TSP 40](#_Toc105434765)

[Figure 20: ACO example Distance Matrix for a city count of 5 43](#_Toc105434766)

[Figure 21: Algorithm Learning Rate 46](#_Toc105434767)

[Figure 22: Algorithm Results Showing Outliers 47](#_Toc105434768)

[Figure 23: Tournament Sampling Line Plot 49](#_Toc105434769)

[Figure 24: Tournament Sampling Box Plot 50](#_Toc105434770)

[Figure 25: Fitness Function Line Plot 50](#_Toc105434771)

[Figure 26: Fitness Function Box Plot 51](#_Toc105434772)

[Figure 27: Elitism Line Plot 52](#_Toc105434773)

[Figure 28: Elitism Box Plot 53](#_Toc105434774)

[Figure 29: Enhancer Line Plot 53](#_Toc105434775)

[Figure 30: Enhancer Box Plot 54](#_Toc105434776)

[Figure 31: Inertia Weight Line Plot 55](#_Toc105434777)

[Figure 32: Inertia Weight Box Plot 55](#_Toc105434778)

[Figure 33: Modified Particle Swarm Line Plot 56](#_Toc105434779)

[Figure 34: Modified Particle Swarm Box Plot 57](#_Toc105434780)

[Figure 35: Particle Swarm Variant Line Plot 58](#_Toc105434781)

[Figure 36: Particle Swarm Variant Box Plot 58](#_Toc105434782)

[Figure 37: Max-Min Ant System Pbest Line Plot 60](#_Toc105434783)

[Figure 38: Max-Min Ant System Pbest Box Plot 60](#_Toc105434784)

[Figure 39: ACO Variant Line Plot 61](#_Toc105434785)

[Figure 40: ACO Variant Box Plot 61](#_Toc105434786)

[Figure 41: Hybrid vs Base Line Plot 63](#_Toc105434787)

[Figure 42: Hybrid vs Base Cropped Line Plot 63](#_Toc105434788)

[Figure 43: Hybrid vs Base Box Plot 64](#_Toc105434789)

[Figure 44: Hybrid vs Base Cropped Box Plot 64](#_Toc105434790)

[Figure 45: Hybrid vs Base Line Plot (city count 20) 65](#_Toc105434791)

[Figure 46: Hybrid vs Base Line Plot (city count 30) 65](#_Toc105434792)

[Figure 47: Hybrid vs Base Line Plot (city count 50) 66](#_Toc105434793)

[Figure 48: Supplementary Delta Test 70](#_Toc105434794)

[Figure 49: Supplementary Fitness Function Test 71](#_Toc105434795)

[Figure 50: Supplementary Hybridization Strategy 72](#_Toc105434796)

[Figure 51: GA solution for TSP map (50 cities) 81](#_Toc105434797)

[Figure 52: PSO Solution for TSP map (50 cities) 81](#_Toc105434798)

[Figure 53: ACO solution for TSP map (50 cities) 82](#_Toc105434799)

[Figure 54: ACO/GA solution for TSP map (50 cities) 82](#_Toc105434800)

[Figure 55: PSO/ACO solution for TSP map (50 cities) 83](#_Toc105434801)

[Figure 56: PSO/GA solution for TSP map (50 cities) 83](#_Toc105434802)

[Figure 57: Greedy-Optimizer solution for TSP map (50 cities) 84](#_Toc105434803)

TABLE OF TABLES

[Table 1: Tournament Sampling delta AUC 49](#_Toc105434804)

[Table 2: Fitness Function AUC 49](#_Toc105434805)

[Table 3: Elitism AUC 52](#_Toc105434806)

[Table 4: Enhancer AUC 52](#_Toc105434807)

[Table 5: Inertia Weight AUC 55](#_Toc105434808)

[Table 6: Modified Particle Swarm Configuration AUC 56](#_Toc105434809)

[Table 7: Particle Swarm Variant AUC 57](#_Toc105434810)

[Table 8: Max-Min Ant System – Pbest AUC 59](#_Toc105434811)

[Table 9: ACO Variant AUC 59](#_Toc105434812)

[Table 10: Hybrid vs Base AUC for TSP city size (10) 62](#_Toc105434813)

[Table 11: Algorithm Runtimes 67](#_Toc105434814)

TABLE OF EQUATIONS

[Equation 1: TSP Distance calculation 7](#_Toc105431955)

[Equation 2: TSP Objective Function 7](#_Toc105431956)

[Equation 3: Roulette Wheel Selection Probability 19](#_Toc105431957)

[Equation 4: Normalization 19](#_Toc105431958)

[Equation 5: Converting Ranges 19](#_Toc105431959)

[Equation 6: Fitness Score 19](#_Toc105431960)

[Equation 7: Particle Velocity Calculation 25](#_Toc105431961)

[Equation 8: Weight Applied on Particle Swap Sequence 27](#_Toc105431962)

[Equation 9: Adapted Particle Movement Calculation 27](#_Toc105431963)

[Equation 10: Ant Pheromone Deposit 30](#_Toc105431964)

[Equation 11: Edge Pheromone Update 30](#_Toc105431965)

[Equation 12: Path Distance Score 31](#_Toc105431966)

[Equation 13: Ant Stochastic Decision Rule 32](#_Toc105431967)

[Equation 14: Max-Min Pheromone Update 33](#_Toc105431968)

[Equation 15: Max-Min Clamp Operator 33](#_Toc105431969)

[Equation 16: Local Pheromone Update 33](#_Toc105431970)

[Equation 17: Ant Deterministic Decision Rule 34](#_Toc105431971)

[Equation 18: T-max Calculation 43](#_Toc105431972)

[Equation 19: T-min Calculation 44](#_Toc105431973)

# Introduction

## Background

Biologically Inspired Algorithms are a term used to denote a family of algorithms that each arose from an analysis of nature’s solution 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 (Coley, 1999), 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. 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.

Swarm intelligence was 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 great success in application to discrete and continuous domains respectively. Ant Colony Optimization has been used as a rough set 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 through some variations in their method (Socha & Dorigo, 2008; Zhong et al., 2007) and a comparative analysis has also been done on these algorithms (Selvi & Umarani, 2010), for example, using the age-old combinatorial optimization problem: *The Traveling Salesman Problem*, a study for which Ant Colony Optimization has had great accomplishments.

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 the ACO and PSO hybrid, demonstrated that these hybridization strategies each came with a different level of efficacy. Unfortunately, even though hybridization for these biologically inspired optimization algorithms has popularity in literature, not much attention has been drawn to analysing their hybridization strategies.

## Research Project

The majority of the literature read in this study that dealt with hybrid models has only ever considered and documented a single hybrid construction methodology. The study done by Huang et al. (2013) was the only one found that did otherwise. Perhaps through further research into this field, patterns and possibly 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.

Unfortunately, the requirements for such a study far outreach the scope of what this dissertation can accomplish. The goal of this project is not to develop new or experiment with hybridization strategies, this project simply aims to be an advocate of the value gained through hybridization in an effort to raise interest in this field of research. This will be done through the research question:

*“By comparing the performance of hybrid versus base algorithms using the Traveling Salesman Problem, which of the chosen Biologically Inspired Optimization Algorithms offers the best performance in terms of learning rate?”*

The chosen BIAs for this study are the *Genetic Algorithm* (GA), *Particle Swarm Optimization* (PSO) and *Ant Colony Optimization* (ACO), along with the 3 hybrid models created from mixing them [ACO/GA, PSO/GA and PSO/ACO].

## Research Objectives

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

*“Using the Traveling Salesman Problem, when a Wilcoxon Signed-Rank Test is done on the results between hybrid and base models, a hybrid algorithm will offer the best performance, in terms of learning rate, 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 Wilcoxon Signed-Rank Test is done on the results between hybrid and base models, the Ant Colony Optimization algorithm will offer the best performance, in terms of learning rate, 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 to 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.

## 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. Perform a literature review on 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.

## Scope & Limitations

This study touches on interesting topics in the theory of computation like discrete and single-objective optimization, graphic 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. Due to the number of variations needing to be tested, the experiments completed took a lot of time to run and, given also the learning curve of understanding the algorithms and programming languages used, the experiments often had to be repeated after any new algorithm discoveries were made that required a code change. It was decided very early on, for efficiency’s sake, that the earlier experiments would be completed using simpler maps (TSP maps of 10 cities to visit), and only the final experiment would be run on the more complex maps (TSP maps of 50 cities).

## 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 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 decerned 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.

# LITERATURE REVIEW

Detailed in this chapter is a history and overview of the 3 chosen algorithms, along with a description of the problem domain they will be applied to. Also given are examples of the current state-of-the-art variations in their algorithm

## Combinatorial Optimization Problems

When the goal is the optimization of problems occurring with qualitative, or discrete, variables (e.g., attributes, states, or values), the solution to the problem consists of arranging those components in such a way that it minimizes, or maximizes, the desired result. In some cases, that goal includes eliminating some of those components as well, meaning that the number of elements to rearrange also becomes part of the problem. This process of seeking the best possible solution within a finite set of possibilities is what is called combinatorial optimization and a problem solved through the arranging of its propositions is a combinatorial optimization problem (Kennedy et al., 2001). Combinatorial Optimization problems all come with a goal that is optimized towards and 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 (Delorme et al., 2016), Job-Shop Scheduling (Zhang et al., 2011) and Boolean Satisfiability (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 back to your starting position?”. 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

With this in mind, the optimum route/solution to the TSP can be described as ordering an itinerary of cities to visit in such a way that the sum of distances traversed while following the itinerary returns the smallest possible value.

Equation 1: TSP Distance calculation

Where means the distance between cities and , if the location of city and , then . refers to the city at position on the itinerary . 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.

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 I believe it best captures the meaning of the TSP with regard to the members of the algorithm’s population.

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, 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).

## Optimization Methodology

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 -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 denoted their dimensional location. **Figure 3** 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 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.

Chart

Description automatically generated

Figure 1: 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 be supplied to the function, that 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).

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.

Diagram

Description automatically generated with low confidence

Figure 2: Local vs Global Optimums

The algorithms focused on in this study, provide different solutions to this problem. 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. If the algorithm lends too much focus on the best individuals of a particular generation, (i.e., the selection pressure is too high), disregarding the potential gain from considering the other members, then the algorithm converges toward those members’ solutions ardently as the optimum. This ‘tunnel-vision’ risks convergence on a local rather than global optimum. So, care needs to be taken when analysing the selection pressure of a given GA design.

The Swarm Intelligence algorithms PSO and ACO on the other hand, operate by balancing between two mechanisms, *exploration* (diversification) and *exploitation* (intensification) (Mirjalili, 2019b; Thangaraj et al., 2011). As suggested by the name, *Exploration* involves diversifying the regions searched by the members of the swarm by making frequent or large changes to the solutions that they compose. This throws a stochastic element into the algorithm that pulls the members away from the currently targeted ‘best location’, allowing them the opportunity to find and explore other potentially better avenues of the search space. On the other hand, *Exploitation* is the mechanism that offers the opposite behaviour, through which all members of the algorithm converge towards the optimum solution.

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.

## History of the chosen algorithms

### The Genetic Algorithm

Charles Darwin’s Theory of Evolution through natural selection in his “On the Origin of Species” book (1876), though not completely factual as noted by the evolutionary biologist Stuart Newman (Mazur, 2008) and Gordons’s (1999) findings on “The Concept of Monophyly”, has inspired many through analysis of its applications. The theory is based on the observation: ‘*survival of the fittest*’, in which fitter and more capable individuals of a population naturally achieve higher survival rates in their given environments, providing them longer lifespans and more opportunities to pass on their superior genetic codes to the next generation. The weaker members of the population would typically achieve lower chances to pass on their inferior genetics, eventually being completely overwritten from the genetic history by fitter candidates over the progressing generations. Darwin’s theory hinged on the concept of variation; that there is a range of differences between the genetic makeup of the individuals in the population, which when accumulated through the principles above would be able to push organisms past the barrier of species toward something completely different, perhaps new, but ultimately better.

GAs are a family of computational models that draw inspiration from Darwin’s evolutionary theory, and they are known to have been originally introduced and investigated by the American engineer John Holland sometime in the 1960s (Coley, 1999; Holland, 1992). Mirjalili (2019) reports that Holland’s genetic algorithm was “one of the first population-based stochastic algorithms proposed in history”. Using a chromosome-like data structure and recombination operators to simulate the mechanics of DNA reproduction, these algorithms have been applied to a very broad range of problem domains.

The algorithm offers a wide application domain because, as long as the given problem can be encoded as a chromosome-based population and a function for evaluation of individual fitness (or attractiveness), the GA can be utilized. This flexibility of application is a reason why the GA remains one of the most popular evolutionary algorithms in literature (Mirjalili, 2019a) with various applications found like the automatic design of convolutional neural networks for image classification (Sun et al., 2020), as a solver for systems of second-order boundary value problems (Arqub & Abo-Hammour, 2014), optimization of cogeneration plant systems (Ahmadi & Dincer, 2010) and optimizing neural networks (Ding et al., 2011), among many others.

### Particle Swarm Optimization

Arguably since the invention of the electronic computer (possibly even earlier), scientists and philosophers alike pondered over the similarities between computer programs and minds. Similar to minds, computers demonstrated the capability to process symbolic information, derive conclusions from premises, and recall stored information when appropriate. They reasoned that the capability of minds to host intelligence gave direction to the possibilities for computers, hence birthing the great quest for Artificial Intelligence (Kennedy et al., 2001).

Progressing from the latter quarter of the nineties marked revolutionary findings in the development of AI technologies like the GA, Evolutionary Computation (Back et al., 1997), and the Artificial Neural Network (Jain et al., 1996). Due to the variety in methods to approach many problems, building an intelligent computer program that finds the best choice required and motivated algorithm engineers to think of several clever techniques. They developed ‘logical shortcuts’, called *heuristics*, that speed up the process in a manner that was applicably reusable. The programs developed by the researchers were simply outstanding at problem-solving, calculation and memory retention but were found to fail “at the simple things” like conversation and face recognition (Kennedy et al., 2001). This was due to the continuously growing number of variables still needing to be addressed in the problem domains it was being applied to. There was always something else that could go wrong. The social psychologist James Kennedy and his associates (2001) observed the causing stereotype creeping into the general public perspective that was limiting the understanding of AI at the time. “Early AI researchers had made an important assumption, so fundamental that it was never stated explicitly nor consciously acknowledged” (Kennedy et al., 2001).

AI at the time was modelled on the vision of a single disconnected person capable of coolheadedly handling the situations posed to them using the information and logical reasoning stored in their brain. However, they argued that if human intelligence was the intended model, then this model of understanding was devoid of an important comportment/behaviour involved in human reasoning and development: *Socialization*.

In real social interaction, not only information but also rules, tips, and also methods of processing information are exchanged. They observed further social behaviours which were “the norm throughout the animal kingdom” in biological examples like Fish schooling, birds flocking and bugs swarming (Kennedy et al., 2001). These behaviours occurred not only for copulation purposes but included important necessities for the population like “having a thousand eyes” to keep watch for predators and searching for food, among other advantages.

Their book ‘Swarm Intelligence’ cited earlier, introduced the concept of exploiting social behaviours by splitting the computational requirements of a system across a group or ‘swarm’ of intercommunicating individuals. Gaining inspiration from the natural examples mentioned earlier, they proposed a model called *Particle Swarm Optimization* which differed from the popular evolutionary computation methods at the time because its population members, named *particles*, were first initialized with stochastically assigned positions and velocities, and then flown through the problem space in search of a solution. The stochastic mechanisms implemented in the algorithm gave it a “lifelike” appearance (Kennedy & Eberhart, 1995) as the particles buzzed around the search space resembling a swarm of mosquitoes. They believed that this observed behaviour, along with the description of each particle being, in essence, a mass-less and volume-less mathematical abstraction that can be called a point when stationary, deemed the terms *particle* and *Particle Swarm Optimization* fitting descriptors for their methodology.

Due to its simplistic and effective design, PSO has gained a lot of popularity over the years (Blum & Li, 2008; Eberhart & Shi, 2001; Kennedy & Eberhart, 1995) and has found applications in many domains such as scalable optimization for social learning (Cheng & Jin, 2015), clustering for high dimensional data sets (Esmin et al., 2015), and multi-objective optimization (Delgarm et al., 2016). Though targeted at continuous domains, studies have been undertaken to adapt the algorithm to discrete problem domains like the TSP (Wang et al., 2003; Yousefikhoshbakht, 2021; Zhong et al., 2007).

### Ant Colony Optimization

Around the same time as the PSO, the ACO algorithm was also developed. It gained inspiration from the study done by Pierre-Paul Grassé, the French entomologist in the 40s and 50s of the 20th century, who shone a light on some interesting findings observed in some species of termites. He observed the reactions of these termites to something he called “significant stimuli” and found that those reactions themselves could also operate as significant stimuli for other insects in the colony, including the insect that produced them. This special type of communication found in these species was termed *stigmergy* and it was described with two main characteristics (Dorigo et al., 2006; Salman et al., 2020):

* It is an indirect, non-symbolic form of communication using the environment as a medium (i.e., communication through modification of the environment).
* The stigmergic information created is local (i.e., can only be accessed when in the vicinity/locus in which it was released).

Diagram

Description automatically generatedSince then, stigmergy has been observed in many other species including ant colonies. In ant species, as the members travel in search of, or returning from a food source, they deposit a chemical along the trails they traverse called *pheromone*. Other ants, upon inspecting a trail, can perceive these pheromones and, as a response, tend to follow the trail containing higher pheromone levels. As they traverse their chosen path, they also add their own deposited pheromone trail to the path, further increasing its pheromone concentration and the ‘attractiveness’ of this trail to successive ants on arrival. The remarkable efficacy of this exploratory pattern was demonstrated by the thorough investigation performed by Deneubourg et al. (1990). In their, soon to be well known, “double bridge experiment” depicted in **Figure 3**, they introduced a diamond-shaped bridge between the ant nest and a chemically unmarked arena for the ants to explore. This provided the ants with a binary left/right choice in such a way that the “dynamics of their cumulative choice [could] be easily quantified”. They noted that the ant’s stigmergic system exploited the positive feedback loop such that it, “after initial fluctuation, rapidly leads to one of the two forks becoming more or less completely preferred to the other” and eventually the whole colony converges on the use of only one of the bridges.

Figure 3: Double-Bridge Experiment

Diagram

Description automatically generatedGoss et al.(1989) expanded on this study by adding a food source to the arena and differing the size of the two bridges in the experiment. Again, the axis of entry is the same 30° on both sides of the bridges (total 60° between bridges) to minimize ‘loop back’ ant journeys and so that the forager has no preference for one or branch or the other based on position. In their experiment, at first, the ants were choosing equally between the short and long but as the experiment was under way, quite abruptly, one of the bridges gained preference. The ants choosing stochastically the shortest bridge, were the first to reach the food source so, on return to the bridges, the probability that they take again take the shorter path is higher as there is yet no pheromone trail attracting them to the longer path until those on that path finally arrive. Their choice then reinforces that pheromone trail as they deposit more on the way back, positively affecting bias towards this trail for all successive ants (Blum & Li, 2008). Their proposed model for that observed behaviour became the main source of inspiration for developing the Ant Colony Optimization algorithm we know today (Dorigo et al., 2006).

Figure 4: Different length bridges

Over time, ACO has become one of the most popular biologically inspired algorithms in literature (Blum & Li, 2008) and has been used to solve many graph-based or graph adapted combinatorial optimization problems. It has found applications in areas like feature selection using a rough set approach (Chen et al., 2010), heart disease prediction and classification (Khourdifi & Bahaj, 2019), scheduling problems (Deng et al., 2019), and real-time routing problems (Samà et al., 2016). Work has also been done to increase the applications of the algorithm to include problems based on continuous domains (Socha & Dorigo, 2008).

## State-Of-The-Art in Biological Optimization

### Genetic Algorithm overview

GA begins with an encoding of the problem domain as a list of chromosomes representing an initial population. These chromosomes are an arbitrary set of trial solutions often randomized to provide unique starting points for each member of the population within the search space. Mirjalili (2019a) notes that techniques like Gaussian random distribution can be used to maximize diversity in the initial population while others like Johnson & Rahmat-Samii (1997) do not find that extra step necessary given the robustness offered by the complete algorithm.

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), Selection Operator chooses the best individuals from the population and compiles them into a mating pool. It is then 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, mentioned 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 certain that their chromosome patterns are optimum. 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, is the introduction of a very important component of the GA: *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.

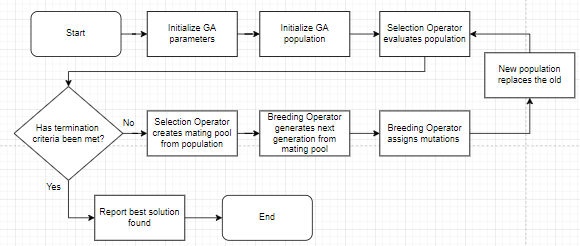


Figure 5: Genetic Algorithm Flowchart

### The Selection Operator: Fitness Function Variants

The main variation in GA composition techniques occurs with the Fitness Function of the Selection Operator. In **Section 2.4.1**, it was 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:

#### 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 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 as depicted in **Figure 6** (Razali & Geraghty, 2011).

Chart, pie chart

Description automatically generated

Figure 6: Roulette Wheel Sampling

With the list of fitness values for all members of the population the selection probability for any individual is:

Equation 3: Roulette Wheel Selection Probability

To calculate that fitness score, often times the evaluation scores of all members of the population are first normalized for algorithm consistency, 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**.

Equation 4: Normalization

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.

Equation 6: Fitness Score

Here, 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.

#### 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 a sampling algorithm that can achieve samples in a single traversal. It works the similar 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.

Chart, pie chart

Description automatically generated

Figure 7: Stochastic Universal Sampling

#### 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 6** and **7**, 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 till Nth and then fitness is distributed using those assigned ranks, presenting a more evenly distributed wheel to select from. In this case, **Equations 5** and **6** can be used on the member’s rank, rather than their evaluation score, to convert it to a fitness score.

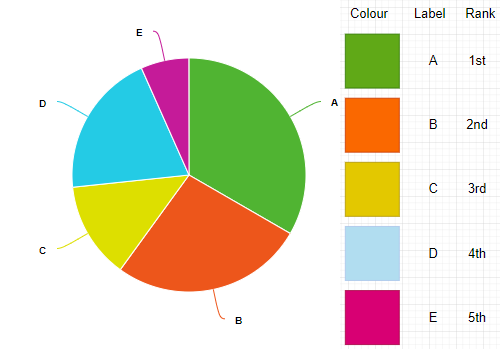


Figure 8: Rank-Based Sampling

#### 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.

### 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 **Section 2.4.2**, there exist some minor variations in the way genetic crossovers can be implemented in the GA. The two most common methods are single- and double-point crossover. 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).

Diagram

Description automatically generated

Figure 9: Crossover Operator (Mirjalili, 2019a)

### 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 and Steady-State (Assareh et al., 2010; 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.

#### 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).

#### 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 affective” (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 the 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).

### 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*.

Pbest is the best solution found by the particle throughout its own history. It 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 mentioned briefly in **Section 2.3**, 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 to be applied when calculating its current velocity. This mechanism is managed by the inertia weight , which is typically set to higher values 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 to a lower , have also been found effective (Shi & Eberhart, 2001).

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 , the personal best position or *pbest* found in the particle’s history and the particle’s current velocity . 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 , 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).

Equation 7: Particle Velocity Calculation

Here, is the inertia weight and and 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 and which are both random numbers between [0,1], generated at each iteration, introducing a stochastic element to the search. **Figure 10** displays a flowchart reviewing the main structure of a PSO algorithm.

Diagram

Description automatically generated

Figure 10: Particle Swarm Optimization Flowchart

### Particle Swarm Optimization in Discreet 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 , where is the number of cities on the map. The movement vectors are then represented by swap operators defined as such that, when applied to the position vector , it swaps the location that the cities and within the vector sequence. This creates a completely new sequence which can be treated as the new location vector for the particle after the movement vector was applied to it, . A velocity vector can contain any number of swap operators compiled together as a *Swap Sequence,* , 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 is applied on a position vector creating a new position vector . With this understanding in mind, the velocity needed to bring an example particle from its current location to its *pbest,* , can be understood as the question: What swaps to my current sequence of cities are needed until it becomes the *pbest* sequence?

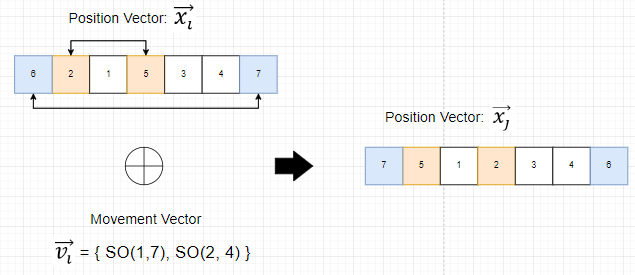


Figure 11: Applying Movement Vector to Position Vector for a Particle

Another big consideration in this application of the algorithm is how to 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. Continuing with 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 is an operator that returns with a probability of , otherwise, it returns nothing.

Equation 8: Weight Applied on Particle Swap Sequence

This adaptation explained by Wang et al. (2003) also brought about a change to the *Particle Swarm* vector calculation given in **Equation 7**. Originally, the movement vectors represented simple directions of travel, which could be scaled in magnitude by weightsand, until the particle eventually reached its desired location. However, in this 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**. It was, however, noted that no justification was given for the removal of the inertia from the calculation.

Equation 9: Adapted Particle Movement Calculation

### Modified Particle Swarm Optimization

Yousefikhoshbakht, (2021) found optimization problems with the application of the PSO adaptation from **Section 2.4.6** to industry services, 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 , bound between a min () and max (), was used along with an accompanying inverse . At the start of the algorithm, the variable begins with a value of , and as the iterations increase, it linearly progresses towards the value of . In this technique, refers to the probability that the original *gbest* will be used for this iteration’s movement vector calculation step, while its inverse is the probability of using the new *gcbest* instead.

### 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 () 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, where and are vertices on the map, and denotes an example edge connecting ‘from’ vertex ‘to’ vertex . 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, . 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 methods are viable. To demonstrate this, 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 . 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 , from which to begin building their solutions . Then, for each step in constructing a solution for that iteration, the ant chooses the next valid vertex to visit , and appends it to its current solution. Again, represents the list of valid vertex choices, given the current solution list of the ant , which is a subset of the complete list of vertexes on the map V and of which the solution component (or in this example case: vertex ) is an element. If there are no more valid solution components that can be chosen , 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 unto 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 as the pheromone level of the edge from vertex to vertex and as the pheromone level deposited by the ant on that edge, taken from **Equation 2.**

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 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 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 to . 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.

.

Equation 11: Edge Pheromone Update

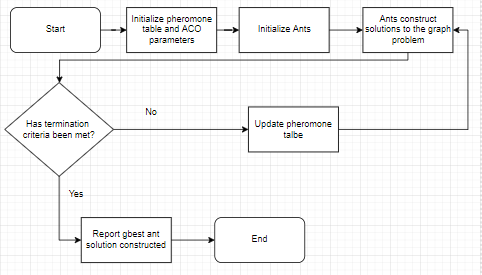


Figure 12: ACO flowchart

### The Ant System

There are a few ways in which a choice from the list of valid solution components can be made. The most widely used method, taking close inspiration from the mathematical model proposed by Goss et al. (1989), is that of the first algorithm model proposed: the *Ant System* (AS) (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, first is the level of the pheromone that has been deposited on the path from its current position to the choice . 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 is the distance between vertices and .

Equation 12: Path Distance Score

Similar to the workings of the *Particle Swarm*, through a balance of these two mechanisms, using weights and 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 adventure off course by choosing a less attractive path. This completed stochastic decision rule for choosing a vertex given a current position vertex is given in **Equation 13.**

Equation 13: Ant Stochastic Decision Rule

### 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).

In *Max-Min Ant System*, for each iteration, only the best performing ant is allowed to update the pheromone table with its solution trail. To avoid premature convergence, the value of the pheromones on all edges used is also bound between a maximum and minimum value. The max-min formula for pheromone update calculation is explained through **Equation 14** where is the solution returned by the best performing ant, and and respectively are the upper and lower bounds imposed on the pheromones (Dorigo et al., 2006). The workings of the operator is defined in **Equation 15**. The paper by Stützle & Hoos (2000) also offered guidelines, through which, the values used for and can be empirically configured.

Equation 14: Max-Min Pheromone Update

Equation 15: Max-Min Clamp Operator

### 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. This local update is performed by the ants, for each step of the solution construction stage, as they deposit small amounts of pheromone on the paths that they use to traverse the graph. This helps to further diversify subsequent searches performed on the graph. This local update is calculated in **Equation 16**.

Equation 16: Local Pheromone Update

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

Another notable difference that the ant colony system brings is the ant decision rule. They introduced two new variables and , directing ants’ decision-making method. is a uniformly distributed random number between [0,1] and is a pre-set parameter such that if 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**.

Equation 17: Ant Deterministic Decision Rule

Though either the globalor 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 globalor iteration’s best ants, can also be employed (similar to the mechanism of the *modified particle swarm optimization*) (Stützle & Hoos, 2000).

## Hybridization Methodology

As a requirement for tackling the primary, the underlying research question for the project (which optimization algorithm is the best between hybrid and base versions), hybrid algorithms needed to 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.

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. For this reason, the sequential hybridization strategy 1 used by Huang et al. (2013) was employed in this project, but of course, with the discrete versions of the algorithms discussed in the earlier sections of this study.

The nature of the GA also seems to lend itself to the sequential approach as well because the genetic selection and breeding process, which is the core mechanism of the algorithm, is tasked with taking in the information of a population in order to create a next, fitter generation. When considering hybridization, naturally one is inclined to simply swap out the words “a population” from this definition, with “a population of ants” or “a population of particles”. The names for each of the hybrid models built were assigned using the sequential order, by which, the algorithm structures occurring in the hybrid model operate.

# Design & Methodology

The experimental design and research methods employed are discussed including an outline of the dataset used, coding languages used, algorithm configurations set, and the sub-topics explored in the experiments.

## 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. As highlighted in **Section 1.5**, 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, the right column meaning the *y*, for each city location on the map.

Text, letter

Description automatically generated

Figure 13: Example Dataset for a map of 10 cities

## 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 an example image generated using Processing that displays a solution returned from an optimization algorithm as a connected graph on a map. 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.

Chart, line chart

Description automatically generated A picture containing diagram

Description automatically generated

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

The second programming language used was Python which is a high-level but also a 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. **Figure 15** demonstrates how the TSP’s objective function was coded in python.

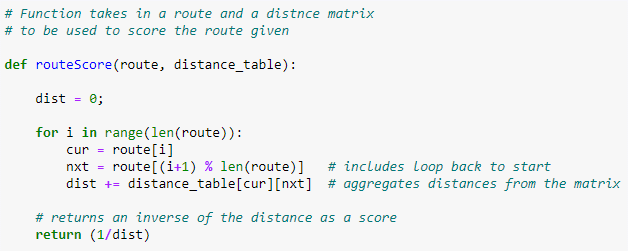


Figure 15: TSP Objective Function in Python

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 to use tools for data analysis and the colours automatically selected for the generated graphs are quite pleasant to the eye. **Figure 16** renders an example scatterplot generated in R demonstrating an appealing use of colour and graph composition.

Chart, scatter chart

Description automatically generated

Figure 16: Example Scatter Plot generated in R

Communication between languages was done through lightweight text files. Both the maps and results generated to be read from and/or written to in this study were of the extension “.txt”.

## Algorithm Implementation & Configuration

Detailed in this section is how the knowledge collected in the literature review in **Section 2**,influenced the design of the optimization algorithms used in this study. A detailed State-Of-The-Art implementation description was already given in **Section 2.4**, the majority of which, was used in this study. This chapter aims only to offer the project-specific details of the algorithms developed. Here also, the implementation gaps found in the extracted body of knowledge from **Section 2** that needed personal review are posed as questions which were the foundation for all preliminary tests conducted to determine optimal algorithm configurations.

### 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 17**.

Diagram

Description automatically generated

Figure 17: 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.

*Individual/Chromosome = Sequence of Genes/Cities = TSP Route/Solution*

(Synonymous within this context)

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 18**. 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 19**. In this way, the states of all members of the population remain constantly valid concerning TSP solution requirements.

Diagram

Description automatically generated

Figure 18: GA Breeding for the TSP

Diagram

Description automatically generated

Figure 19: 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 2.4.2-4** came with further configurations 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 2.4.2.4**; that its selection pressure was potentially too high. To attempt a solution to this, an approach was devised, borrowing inspiration from ACO’s *ant colony system* explained in **Section 2.4.11**, where a new *delta* variable was introduced and used to probabilistically decide whether the winner of a tournament is the member with the higher or lower score. For example, 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 . The introduction of this variable lowered the selection pressure of tournament sampling but also brought along the question:

* 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 2.4.4.2**. To attempt a solution, a method was attempted to localize its effect while at the same time striving to preserve its essence. This *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 question:

* Does this new local steady-state function bring any merit over the original steady-state function?

### Particle Swarm Optimization Implementation Specifics

The studies perfomed by Yousefikhoshbakht (2021) and Wang et al. (2003) detailed an intriguing method for adapting the PSO to the TSP, detailed in **Section 2.4.6**. 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 2.4.7**, 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 this study?
* Does this modified version suit this project more than the original?

### Ant Colony Optimization Implementation Specifics

Since the ACO was designed for combinatorial problems like the TSP, not much work was needed to adapt it for this study. The only things introduced to its 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 20** is an example of this.

Other than this addition, the structure of the algorithm developed closely followed the descriptions posed in **Sections 2.4.8-11**. 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.

Calendar

Description automatically generated

Figure 20: ACO example Distance Matrix for a city count of 5

Following common practice (Gambardella & Dorigo, 1996; Stützle & Hoos, 2000), the alpha and beta weights used in the ACO for this project were and , and the evaporation rate was set to . Following the justifications given by Stützle & Hoos (2000) and Gambardella & Dorigo (1996), when it came to calculating the and variables used, for the *Max-Min Ant System* and *Ant Colony System* (algorithms detailed in **Sections 2.4.10** and **2.4.11**) **Equations 18** and **19** adopting the *global* best ant solution rather than the *iteration’s* best were used.

Equation 18: T-max Calculation

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*, as instructed, specialized pheromone initialization was done after iteration 1 was complete, as pheromone levels for each edge were initialized to the calculated value gained from iteration 1. Similarly, the specialized *Ant Colony System* pheromone update was calculated after iteration 1 was complete, where the pheromone levels for each edge were initialized using the value calculated for the ants’ local pheromone update strength for the first iteration (Gambardella & Dorigo, 1996). Note here that TSP **Equation 1** forcalculating the tour length is used rather than **Equation 2** for the tour score. All of this left only the final question:

* Which of these ACO variants performs the best?

### Overview of Questions to Investigate:

GA –

* What is the best delta setting for tournament sampling?
* Which fitness function 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)
* Does the new *local steady-state* function bring any merit over the original?

PSO –

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

ACO –

* What *Pbest* value is the most appropriate for the ACO variants?
* Which of the ACO variants performs the best?

### Benchmark

As a comparative baseline to critique the performance of these algorithms against, 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 learning rate was quite literally a stagnant baseline.

## Statistical Analysis

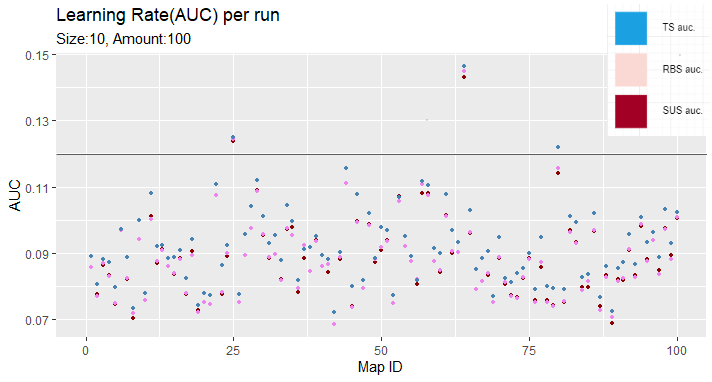
Two popular methods used for critiquing optimization algorithms are the score of the final solution given, and the number of iterations taken to achieve that score. In this study, however, the research question aimed to find the best performing algorithm in terms of *learning rate* which is a blend of the two. The learning rate denotes how fast and efficiently the algorithm searches its problem space to find an answer. On a line graph mapping the best-scoring solutions found by the algorithm over each iteration run, the learning rate is denoted by the *Area Under the Curve* (AUC) as shown in **Figure 19** and it is calculated using the trapezoidal function[[1]](#footnote-1).

Diagram

Description automatically generated

Figure 21: Algorithm Learning Rate

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 gotten from the algorithms rather than simple un-representing mistakes, so removing them was not an option from a statistical point of view.

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 22: Algorithm Results Showing Outliers

For statistical analysis of test results between multiple participants, Demšar’s (2006) recommendations about appropriate tests to use in the presence of outliers were followed. The Wilcoxon signed-rank test was chosen over the widely used t-test because it performs better when outliers are present. For all analyses done, the statistical threshold of 0.05 was chosen because it is the most common threshold used in statistical analysis.

# 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.

## Experiment 1: Genetic Algorithm

Condensing the GA composition questions drawn from this study posed in **Section 3.3.1**, 2 overarching concerns were drawn: what Fitness Function and what Enhancer? This splits the experiment into two parts:

### Part 1 – What 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 () performed the best with the highest average AUC of the group, as shown in **Table 1**. **Figure 23** displays a line plot of the average global best score per iteration, also confirming this observation. Examining the AUC using a box plot[[2]](#footnote-2) shown in **Figure 24**, 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 beat the Wilcoxon 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.

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 25** and **26**. 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 1: Tournament Sampling delta AUC

Table

Description automatically generated

Table 2: Fitness Function AUC

Table

Description automatically generated

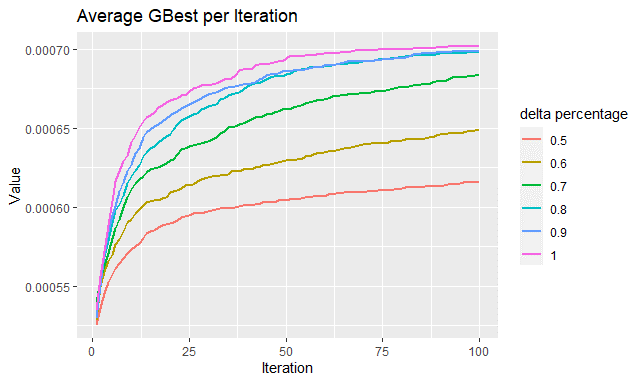


Figure 23: Tournament Sampling Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 24: Tournament Sampling Box Plot

Chart, line chart

Description automatically generated

Figure 25: Fitness Function Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 26: Fitness Function Box Plot

### Part 2 – What Enhancer to use?

*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, despite the close competition, that an elite percentage of 10% was the best. This was confirmed when observing the AUC statistics in **Table 3**. Unfortunately, statistical analysis using the Wilcoxon test was inconclusive about a winner for this experiment. Nevertheless, the elitist model using an elite size of 10% was chosen as the winner.

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 3: Elitism AUC

Table

Description automatically generated

Table 4: Enhancer AUC

Table

Description automatically generated

Chart, line chart

Description automatically generated

Figure 27: Elitism Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 28: Elitism Box Plot

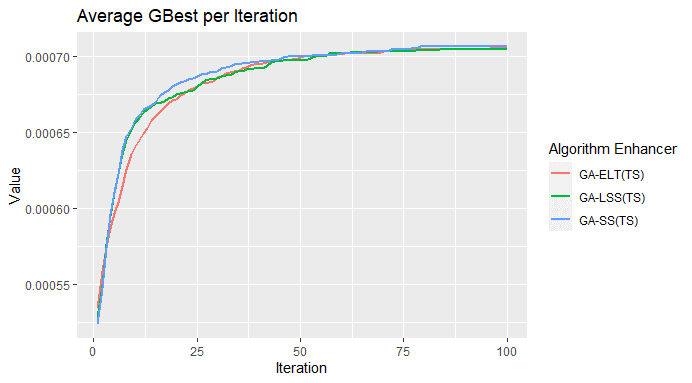


Figure 29: Enhancer Line Plot

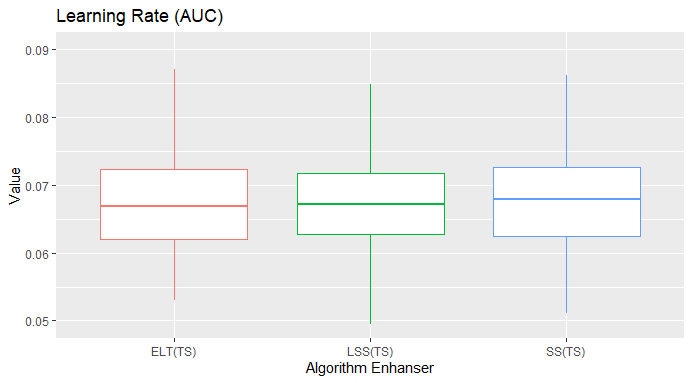


Figure 30: Enhancer Box Plot

## 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.

### Part 1 – What 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. 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. 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.

Table 5: Inertia Weight AUC

Table

Description automatically generated

Chart, line chart

Description automatically generated

Figure 31: Inertia Weight Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 32: Inertia Weight Box Plot

### Part 2 – What 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**, the model with the highest average AUC having (alpha = 30%, beta = 70% and an iteration percent = 50%) was chosen as the winner.

Table 6: Modified Particle Swarm Configuration AUC

Table

Description automatically generated

Chart

Description automatically generated

Figure 33: Modified Particle Swarm Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 34: Modified Particle Swarm Box Plot

### Part 3 – What 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 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

Table

Description automatically generated

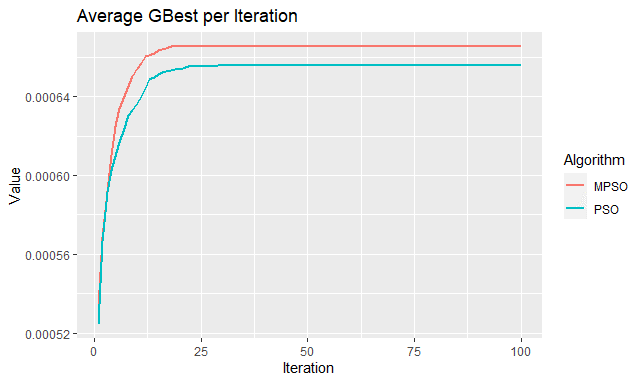


Figure 35: Particle Swarm Variant Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 36: Particle Swarm Variant Box Plot

## Experiment 3: Ant Colony Optimization

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 analyzing the box plot in **Figure 38** and a statistical winner was not declared. However, based on the line graph in **Figure 37**, it seemed that the best model, wining by a minuscule margin 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.

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 8: Max-Min Ant System – Pbest AUC

Table

Description automatically generated

Table 9: ACO Variant AUC

Table

Description automatically generated

Chart

Description automatically generated

Figure 37: Max-Min Ant System Pbest Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 38: Max-Min Ant System Pbest Box Plot

Chart

Description automatically generated

Figure 39: ACO Variant Line Plot

Chart, box and whisker chart

Description automatically generated

Figure 40: ACO Variant Box Plot

## 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 1**, which surprisingly beat the ACO base version with a statistical significance of 7.089e-12. 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. The line and box graphs in **Figures 42** and **44**, are a cropped version of **Figures 41** and **43**, trying to show a closer view of just the ACO and ACO/GA algorithms for comparison.

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.

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

Table

Description automatically generated

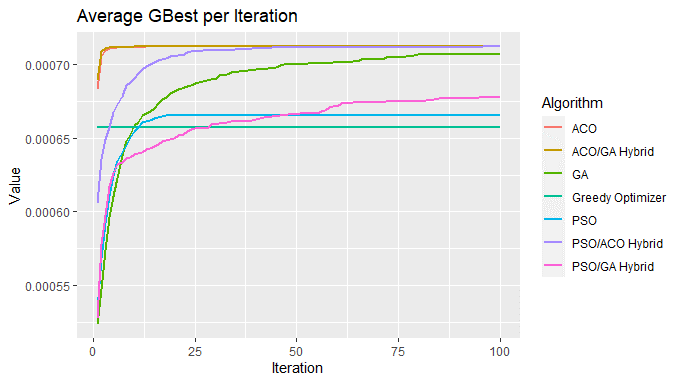


Figure 41: Hybrid vs Base Line Plot

Chart

Description automatically generated

Figure 42: Hybrid vs Base Cropped Line Plot

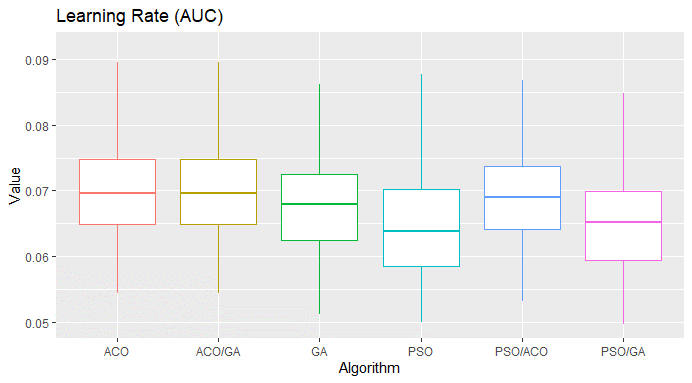


Figure 43: Hybrid vs Base Box Plot

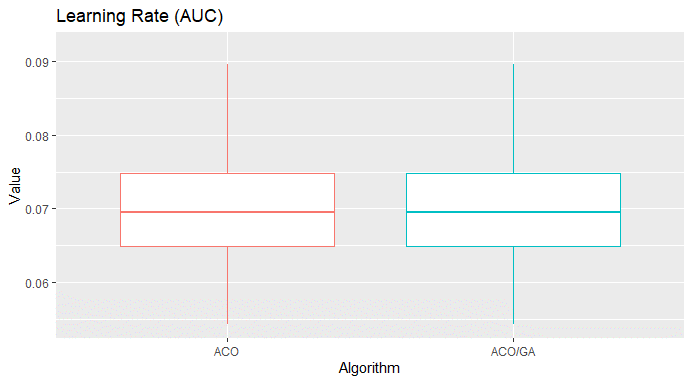


Figure 44: Hybrid vs Base Cropped Box Plot

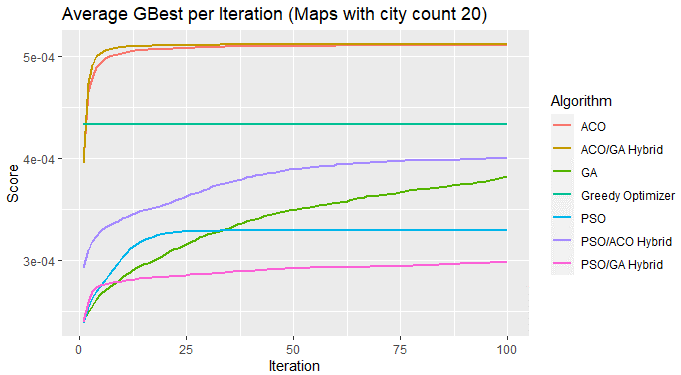


Figure 45: Hybrid vs Base Line Plot (city count 20)

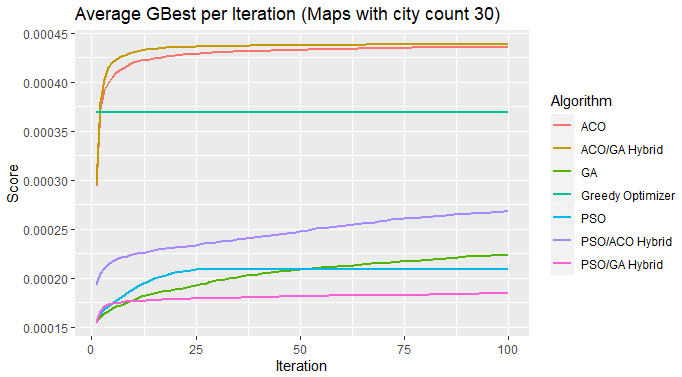


Figure 46: Hybrid vs Base Line Plot (city count 30)

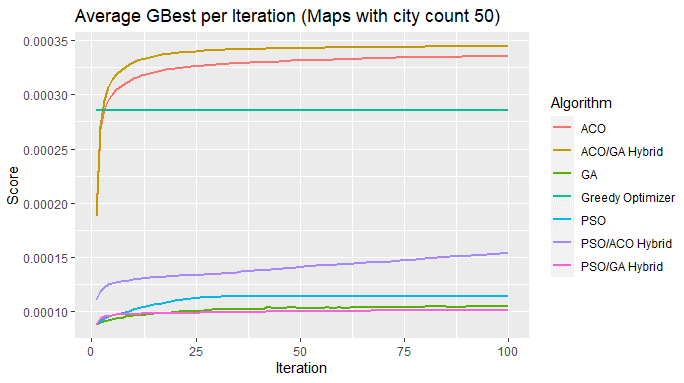


Figure 47: Hybrid vs Base Line Plot (city count 50)

## 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 using the hardware specifics provided in **Section 1.5**. 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 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.

Table 11: Algorithm Runtimes

Table

Description automatically generated

# Conclusion

Given in this section is a discussion of the results drawn from the experiments concerning the objective of this research project.

## 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.

## 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. By answering the research question, this dissertation has successfully demonstrated the value that could be gained through the use of hybridization.

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 Wilcoxon Unsigned-Rank 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.

## Algorithm Methodology Findings

Sectioned here are some notable findings deduced from the experiment results with regards to the methodology and composition of the algorithms used.

### Particle Inertia

The results that were drawn from the exploration into particle inertia, in **Section 4.2**, support the fundamental necessity for including inertia as a part of the *Particle Swarm* methodology. Though a statistical basis could not be drawn for absolutely establishing the model lacking inertia as the worst-performing PSO method, in this study, it was still found that the model having no inertia offered the worst learning rate (AUC) of the group even at this problem level (maps of 10 cities). These results highlight the shortcomings of the algorithm design proposed by Wang et al. (2003).

Another finding, observed in **Figure 31**, was that the PSO models having higher inertia weights were not as susceptible to premature convergence as the ones with lower inertia weights, though offering a higher AUC. Although the highest average AUC recorded was with the PSO model having an inertia weight of 0.4, at the end of the algorithm run, the PSO model with a weight of 0.8 was observed on average to return the best solution (highest score recorded). In fact, unlike the former, the 0.8 model was found on average to have not yet converged on an optimum solution and possibly, if given more iterations for searching, could offer even better solutions. This finding was in line with and gives justification for, the popularity of using higher inertia weights like 0.8 (Shi & Eberhart, 1998).

### Tournament Sampling Selection Pressure

In the experiments done for the Fitness Function of the GA in **Section 4.1**, it was observed that *tournament sampling* was one of the best performing algorithms out of the group. This agrees with the findings of Razali & Geraghty (2011) who, however, 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 48**, 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 49**,stillfind 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.

**Chart, line chart

Description automatically generated**

Figure 48: Supplementary Delta Test

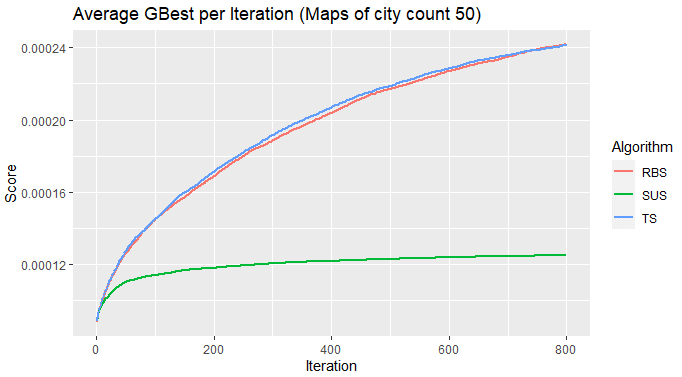


Figure 49: Supplementary Fitness Function Test

### Hybridization Strategy

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 50** show that, though slightly better, not much gain in performance was actually 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.

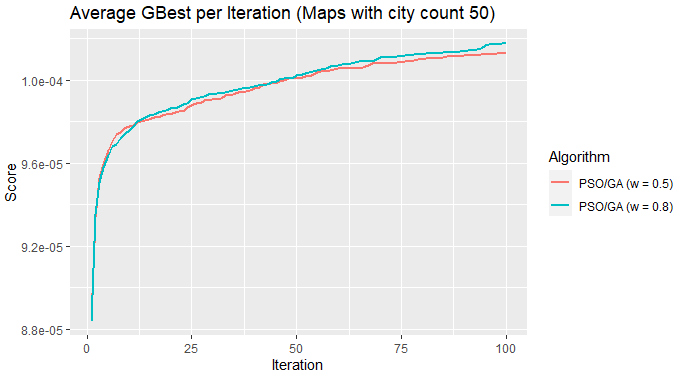


Figure 50: Supplementary Hybridization Strategy

## Optimization Methodology Findings

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.

## 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.

## 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 design were voiced when this adaptation was further analysed. These conclusions suggest that more work is needed to refine this methodology or to develop a completely new technique for discrete PSO adaptation.

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. As more work into this untapped field is completed, it is hoped that enough data will eventually be amassed to sponsor a more qualitative research methodology. When grounded justifications can be highlighted for why certain algorithm techniques work best when composed a certain way, this promotes a more collective understanding of the field of AI development and could pioneer new theories in AI engineering. Perhaps, similar to what was motioned in the book by Kennedy et al. (2001), AI theory could break past the barriers of biological bias toward something completely unprecedented but far better.

# 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

Arqub, O. A., & Abo-Hammour, Z. (2014). Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. *Information Sciences*, *279*, 396–415. https://doi.org/10.1016/j.ins.2014.03.128

Assareh, E., Behrang, M. A., Assari, M. R., & Ghanbarzadeh, A. (2010). Application of PSO (particle swarm optimization) and GA (genetic algorithm) techniques on demand estimation of oil in Iran. *Energy*, *35*(12), 5223–5229. https://doi.org/10.1016/j.energy.2010.07.043

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

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.

Darwin, C. (1876). *The Origin of Species: By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* (6th ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511694295

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.

Deneubourg, J.-L., Aron, S., Goss, S., & Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, *3*(2), 159–168. https://doi.org/10.1007/BF01417909

Deng, W., Xu, J., & Zhao, H. (2019). An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem. *IEEE Access*, *7*, 20281–20292. https://doi.org/10.1109/ACCESS.2019.2897580

Ding, S., Su, C., & Yu, J. (2011). An optimizing BP neural network algorithm based on genetic algorithm. *Artificial Intelligence Review*, *36*(2), 153–162. https://doi.org/10.1007/s10462-011-9208-z

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., & 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

Eberhart, & Shi, Y. (2001). Particle swarm optimization: Developments, applications and resources. *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, *1*, 81–86 vol. 1. https://doi.org/10.1109/CEC.2001.934374

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

Gordon, M. S. (1999). The Concept of Monophyly: A Speculative Essay. *Biology and Philosophy*, *14*(3), 331–348. https://doi.org/10.1023/A:1006535524246

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

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press.

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

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

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

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

Mazur, S. (2008, October 11). *The Origin of Form Was Abrupt Not Gradual—Archaeology Magazine Archive* [Interview Article]. Archaeology Archive - A Publication of the Archaeological Institute of America. https://archive.archaeology.org/online/interviews/newman.html

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

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

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

Razali, N. M., & Geraghty, J. (2011). *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. 6.

Salman, M., Garzón Ramos, D., Hasselmann, K., & Birattari, M. (2020). Phormica: Photochromic Pheromone Release and Detection System for Stigmergic Coordination in Robot Swarms. *Frontiers in Robotics and AI*, *7*. https://www.frontiersin.org/article/10.3389/frobt.2020.591402

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

Sun, Y., Xue, B., Zhang, M., Yen, G. G., & Lv, J. (2020). Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification. *IEEE Transactions on Cybernetics*, *50*(9), 3840–3854. https://doi.org/10.1109/TCYB.2020.2983860

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

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

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

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

# APPENDIX

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

Chart, radar chart

Description automatically generated

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

Chart, radar chart

Description automatically generated

Figure 52: PSO Solution for TSP map (50 cities)

Chart

Description automatically generated

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

Chart

Description automatically generated

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

Chart, radar chart

Description automatically generated

Figure 55: PSO/ACO solution for TSP map (50 cities)

Chart, radar chart

Description automatically generated

Figure 56: PSO/GA solution for TSP map (50 cities)

Chart, radar chart

Description automatically generated

Figure 57: Greedy-Optimizer solution for TSP map (50 cities)

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> [↑](#footnote-ref-1)
2. For information on how to read a box plot, please see: <https://www.statisticshowto.com/probability-and-statistics/descriptive-statistics/box-plot/> [↑](#footnote-ref-2)