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: 26/04/2022**

ABSTRACT

AAAA

*(approx. 250-300 words)*

**Key words:** *list 5 to 8 words*

ACKNOWLEDGEMENTS

I would like to express my sincere thanks ……….

*(thank all the people how have assisted you in completing your dissertation. Start with your supervisor, all DIT staff that may have helped, other people can include family and friends, industrial and academic staff from other institution, etc.)*

TABLE OF CONTENTS

[ABSTRACT ii](#_Toc103114132)

[ACKNOWLEDGEMENTS iii](#_Toc103114133)

[TABLE OF CONTENTS iv](#_Toc103114134)

[TABLE OF FIGURES vi](#_Toc103114135)

[TABLE OF TABLES vii](#_Toc103114136)

[1. Introduction 1](#_Toc103114137)

[1.1 Background 1](#_Toc103114138)

[1.2 Research Project 3](#_Toc103114139)

[1.3 Research Objectives 4](#_Toc103114140)

[1.4 Research Methodologies 4](#_Toc103114141)

[1.5 Scope and Limitations 5](#_Toc103114142)

[1.6 Document Outline 5](#_Toc103114143)

[2. LITERATURE REVIEW 6](#_Toc103114144)

[2.1 Combinatorial Optimization and the TSP 6](#_Toc103114145)

[2.2 History of the chosen algorithms 6](#_Toc103114146)

[2.2.1 Genetic algorithm (GA) 6](#_Toc103114147)

[2.2.2 Particle Swarm Optimization (PSO) 7](#_Toc103114148)

[2.2.3 Ant Colony Optimization (ACO) 9](#_Toc103114149)

[2.3 Algorithm Methodology and Variants 11](#_Toc103114150)

[2.4 Statistical Analysis 11](#_Toc103114151)

[3. Design and Methodology 12](#_Toc103114152)

[4. Results, evaluation and discussion 13](#_Toc103114153)

[5. Conclusion 14](#_Toc103114154)

[6. BIBLIOGRAPHY 15](#_Toc103114155)

[Appendix A 20](#_Toc103114156)

TABLE OF FIGURES

[Figure 3: Example Search Space/Landscape (Mirjalili, 2019b) 9](#_Toc103347598)

[Figure 4: Local vs Global Optimums 9](#_Toc103347599)

[Figure 1: Double-Bridge Experiment 9](file:///C:\Users\C14460702\Dissertation\Documents\Dissertation.doc.docx#_Toc103347600)

[Figure 2: Different length bridges 9](file:///C:\Users\C14460702\Dissertation\Documents\Dissertation.doc.docx#_Toc103347601)

[Figure 5: GA Flowchart 9](#_Toc103347602)

[Figure 6: Roulette Wheel Sampling 12](#_Toc103347603)

[Figure 7: Stochastic Universal Sampling 13](#_Toc103347604)

[Figure 8: Rank Based Sampling 14](#_Toc103347605)

[Figure 9: GA Crossover Operator 15](#_Toc103347606)

[Figure 10: PSO applying a Movement Vector on a Position Vector 18](#_Toc103347607)

TABLE OF TABLES

# Introduction

## Background

Biologically Inspired Algorithms (BIAs) are a term used to denote a family of algorithms that each arose from an algorithm engineer’s understanding gained through analysis of nature’s solution to common problems. They usually are further subcategorized by their general methodologies like Evolutionary algorithms (using the concept of genetic crossovers) (Back et al., 1997) 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) (Kennedy & Eberhart, 1995), among many others.

Originally developed by the American engineer John Holland sometime in the 10960s (Coley, 1999; Holland, 1992), one of the earliest occurring members of these Biologically Inspired Algorithms (BIAs) in history is the Genetic Algorithm inspired by Charles Darwin’s theory of evolution through natural selection (Darwin, 1876; Mirjalili, 2019a). 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 (Kennedy et al., 2001). Two of the most popular algorithms that arose from implementations of swarm intelligence are the *Ant Colony Optimization* (ACO) and the *Particle Swarm Optimization* (PSO) algorithms (Blum & Li, 2008).

ACO, originally proposed by Dorigo et al. (2006), and PSO, originally proposed by Kennedy et al. (2001), have found great success in application to discrete and continuous domains respectively. ACO has been used as a rough set approach to feature selection (Y. Chen et al., 2010), heart disease prediction and classification (Khourdifi & Bahaj, 2019) and real-time routing problems (Samà et al., 2016). PSO 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 been done to bridge the gap in application domains between the two algorithms through some variations in their methodologies (Socha & Dorigo, 2008; Zhong et al., 2007).

Algorithm hybridization is a technique through which the methodologies of the given algorithms are combined in an effort to merge their strengths. Through use and analysis, advantages and drawbacks have been highlighted over the years in many algorithms, like the GA, ACO and PSO (Rohini & Natarajan, 2016; Selvi & Umarani, 2010). As such, nowadays many computer programmers prefer to use the stronger hybrid models, pooling the advantages of their algorithms, for their applications (Luan et al., 2019; Mahi et al., 2015; Mandloi & Bhatia, 2016). 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 BIAs has popularity in literature, not much attention has been drawn to analysing their hybridization strategies.

In a lot of the studies mentioned above, comparative analysis for the algorithms has simply been completed by first choosing an application domain that the algorithms would be run on, and then comparing the results drawn from the study. An example would be combinatorial optimization problems, for which, the ACO algorithm has established great accomplishments (Huang et al., 2013; Papadimitriou & Steiglitz, 1998). With regards to combinatorial optimization problems, the Traveling Salesman Problem (TSP) is one of the most important and widely applicable issues that has arisen (Yousefikhoshbakht, 2021). That is because it sets the foundation for building efficient routing systems; systems that we are all constantly in connection with today. The TSP has been one of the many tools used over the years to benchmark the performance of and demonstrate the merits that the use of BIAs can bring (S.-M. Chen & Chien, 2011).

## 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 hybrid methodologies 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 for 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 by comparing the performance of hybrid versus base algorithms through the research question:

*“When a statistical analysis is done on the results between hybrid and base models, which of the chosen Biologically Inspired Optimization Algorithms is the best at finding the most optimal solution to the Traveling Salesman Problem, given standardized population size, number of maximum iterations, and a statistical significance threshold of 0.05?”*

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 main objective of this study is to establish the benefits of hybridization by showing that a hybrid algorithm performs better than the base algorithm. To do this, 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 in order to find and understand some of the most popular variations that have occurred in their design.
2. To define appropriate parameters and statistical tests to be used in this study, justifying any conclusions drawn from the final comparative analysis.
3. To perform the comparative analysis of the base algorithms run against their hybrid versions to extract the conclusion(s) of the study.

## 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 some of the most popular variations that have occurred 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 variants extracted from step 1, conduct empirical research on the efficacy of that variant within the chosen problem domain (TSP) by implementing them in Python and running them against the TSP maps generated to find the best representative 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(s) chosen from step 2 and representatives chosen from steps 3 and 4, conduct the final comparative analysis of the algorithms documenting any conclusion(s) drawn.

## Scope and 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 variants 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 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).

Also, as mentioned earlier in Section 1.2, the time and recourses allocatable for this project were a limitation that spurred the decision not to tackle directly the original gap highlighted found in the body of knowledge (*optimum hybridization strategies*).

## Document Outline

* Chapter 2:
* Chapter 3:
* Chapter 4:
* Chapter 5:
* Chapter 6:

# LITERATURE REVIEW

## Combinatorial Optimization Problems and the TSP

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 TSPs (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 using a combination of input variables *x* and *y* represented as the *length* and *width* dimensions, and the score given from the objective function for those possible inputs as the *height* dimension.

Chart

Description automatically generated

Figure 1: Example Search Space/Landscape (Mirjalili, 2019b)

In Section 2.2, 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 GAs, *Selection Pressure* (SP) 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 SP 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, (the SP is too high, i.e., is too focussed on a certain region of the search space), disregarding the potential gain from considering the other members, then the algorithm converges toward those member’s 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 SP 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 certain criteria are reached, and the best solution is given. 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 is reached.

## History of the chosen algorithms

### Genetic algorithm (GA)

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.

Genetic algorithms are a family of computational models that draw inspiration from Darwin’s evolutionary theory 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 problems often offering optimized potential solutions.

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. The ease in flexibility plays is a part of the 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 back-propagation (BP) neural networks (Ding et al., 2011), among many others.

### Particle Swarm Optimization (PSO)

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 (AI) (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 (ANN) (Jain et al., 1996). However, the social psychologist James Kennedy and his associates (Kennedy et al., 2001) observed a stereotype creeping into the general understanding of AI at the time that was limiting the understanding of AI at the time. They noted that early AI researchers understood the measure of intelligence as the ability to solve large, complex, and sometimes multipart, problems quickly. Due to the variety in methods to approach many problems, building an intelligent computer program that finds the best choice required and motivated them to think of a number of clever methods. 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 simpler things like conversation and face recognition. 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.

Kennedy (2001) notes that those “early AI researchers had made an important assumption, so fundamental that it was never stated explicitly nor consciously acknowledged”. 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. 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 model.

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). With regards to the TSP, studies were undertaken to adapt the algorithm to these discrete domains (Zhong et al., 2007). One interesting algorithm adaptation was the one proposed by Wang et al. (2003) and further improved by Yousefikhoshbakht (2021), where a series of swap sequences were used to represent vectors. This adaptation was chosen as the PSO focus for this study.

### Ant Colony Optimization (ACO)

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”, 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 “abruptly, some minutes later, one branch becomes visibly preferred”. The ants, at first choosing stochastically, the shortest bridge were the first to reach the nest, so on return the probability that they take again take the shorter path is higher as there is no pheromone trail attracting them to the longer path yet 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 (Y. 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). In fact, 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 (GA) 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. 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) which is then taken by the Fitness Function and transformed it 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, Selection Operator chooses the best individuals from the population and compiles them into a mating pool. It is the Breeding Operator’s task then to combine the genetic components of those chromosome members in that mating pool in order to make the next generation.

At this point, the earlier mentioned issue of Selection Pressure (SP) 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, restricting the algorithm’s search space; 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 (Coley, 1999).

In the final step of the GA, mutations are randomly assigned under an appropriately low percentage to allow more variability in the search space. As the algorithm iterates through the generations, members of the population increase in fitness until a concluding best solution is found after the stop criteria is/are reached (Whitley, 1994).

Diagram

Description automatically generated

Figure 5: GA Flowchart

### The GA Selection Operator: Fitness Function Variants

The main variation in GA composition techniques occurs within the Selection Operator. As mentioned earlier, this operator performs its selection using a combination of two techniques, one of which is the fitness function. Earlier it was also highlighted that the role of this function is convert the score given from population evaluations into an allocation of reproductive opportunities. In other words, to convert the evaluation score into measure of *fitness* (or attractiveness) as a new *fitness score*. This can be done in a number of ways:

#### Roulette Wheel Selection (RWS)

As the title suggests, the concept of natural selection is simulated using a roulette wheel type selection process. The *fitness score* in RWS 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 RWS can be viewed as the probability of the selection pointer landing on that member after a roulette wheel, with the size of the segments for each member proportional to their fitness score, is spun (Razali & Geraghty, 2011). With the list of fitness values for all members of the population the selection probability for any individual is:

Equation 3: GA – RWS 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 size segment 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 Formula

*Where is the score, min(x) is the smallest score in the group, and max(x) is the largest score in the group.*

Equation 5: Converting ranges

*Where 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 to use those remainders as a probability for offering a bonus slot to that member. In that case, the final equation needed for calculating the fitness score would be:

Equation 6: GA – Fitness Score Remainders

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

Chart, pie chart

Description automatically generated

Figure 6: GA – Roulette Wheel Sampling

#### Stochastic Universal Sampling (SUS)

Over time, weaknesses were highlighted in the workings of the RWS, and variations of that fitness function emerged to solve those problems. One of those was the problem of inefficiency, for which, the SUS function was developed to tackle. In the RWS, 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: GA – Stochastic Universal Sampling

#### Rank Based Sampling

Another problem noted with the RWS technique was its SP 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 the RBS (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 to convert it to a fitness score.

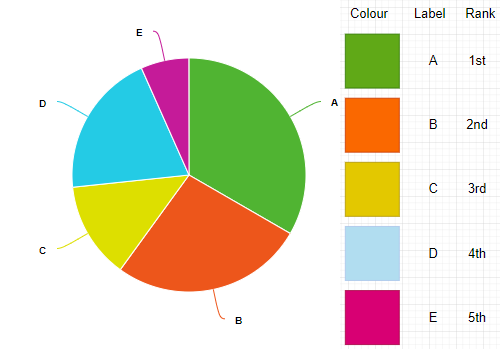


Figure 8: GA – Rank Based Sampling

#### Tournament Sampling

Unlike the variations mentioned above which followed the pattern of the RWS, the TS uses a completely different mechanism for selecting a mating pool for breeding. In the TS, 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 TS when used on small problem sets (Razali & Geraghty, 2011), it has been highlighted that TS also runs into a similar problem as the RWS: that its’ SP 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 TS applied to the scores of the members before each competition and found success with their proposed model.

### GA Breeding Operation: 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 variants found in **Section 2.4.2**, there exists 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 single point crossover, 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 the genetic code between two points are swapped. Other example variations include Uniform crossover, 3 point crossover, and cycle crossover (Mirjalili, 2019a).

Diagram

Description automatically generated

Figure 9: GA – Crossover Operator

### GA 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 a 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 (SS)

This function can be thought of as the overlapping of generations takes the methodology of elitism even further. In SS, 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 SS. One method is by storing the new child generation in a separate list and then overriding selected parents with those in the new, fitter individuals. Another method is by appending the children to the end of the parent's list, temporarily creating an enlarged population size. Then, using chosen criteria, members are removed until the population size returns to its origin (Johnson & Rahmat-Samii, 1997). It should also be noted that since the SS aims to retain the best of each generation for the next, combined use of SS and Elitism brings redundancy.

### Particle Swarm Optimization (PSO) 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*.

The *pbest* is best solution found by the particle throughout its’ 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 combined use of this variable and application of the particle’s own inertia. The *gbest* on the other hand, is the collective best solution found globally in the algorithm’s history across all particles. Use of this variable is 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.

In the PSO, each member of the swarm is composed of 3 D-dimensional vectors (D stands for 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 according to its current velocity, 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.

Another important concept to the PSO is *inertia*. Das et al. (2008) argues that the concept of velocity, used in the *Particle Swarm*, 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 fount effective (Shi & Eberhart, 2001). With all of this in mind, the classical velocity calculation equation for the PSO (Das et al., 2008) is:

Equation 7: PSO – Velocity Calculation

where is the inertia weights, and are weights respectively managing the balance between exploration, known as “self-confidence”, and exploitation, also known as “swarm-confidence”. and are both random numbers between [0,1], generated each iteration, introducing a stochastic element to the search.

### PSO for the TSP

Unfortunately, the PSO was designed for continuous domains, so this base form ass-is cannot be used for this project. For the TSP with a finite discrete search space, the understanding of positional and movement vectors had to change. Wang et al. (2003) proposed the representation of position vectors, 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 10** 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 is needed until it becomes the *pbest* sequence?

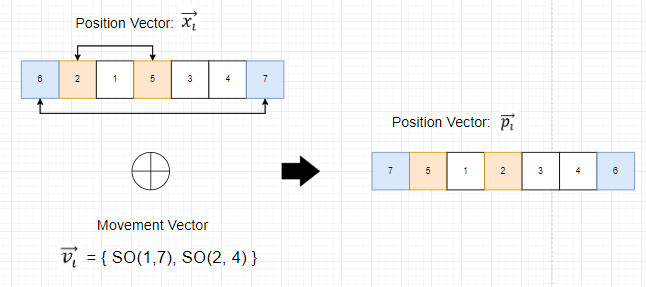


Figure 10: PSO – Applying a Movement Vector to a Position Vector

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 on 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.

Equation 8: PSO – weight applied on a Swap Sequence

*Where is an operator that returns with a probability of , otherwise, it returns nothing.*

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 movements vectors encapsulate the complete transition 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 was:

Equation 9: PSO adapted Particle Movement Calculation for the TSP

It was noted that no justification was given for the removal of the inertia from the calculation.

### MPSO variant

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 the MPSO. There, they introduced another important variable called *gcbest* which tracks the global best location found *for that iteration only*. To track the use of *gcbest* a variable , bound between a min () and max (), was used along with an accompanying inverse . is the probability that *gbest* will be used for this iteration’s movement vector calculation step, while its inverse is the probability of using *gcbest* instead. At the start of the algorithm, , representing a probability of for using gbest this iteration, and as the iterations increase, linearly progresses towards .Design and Methodology

### Ant Colony Optimization (ACO) 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 (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. Whichever the choice, some very minor adjustments would accordingly need to be made to the calculations used in the algorithm to accommodate this. Consider, for example, the second method (solutions stored as a sequence of vertices).

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 *Traveling Salesman Problem* (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 was allowed.

For each iteration of the algorithm, the set of *m* ants initially start 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 current solution list of the ant , which is a subset of the complete list of vertexes on the map V, and of which, 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 performing (Dorigo & Stützle, 2019). Fortunately, *Traveling Salesman* objective function chosen in **Section 2.1** in **Equation 2** does exactly this, and so, its value returned can be used as the pheromone strength for each ant.

Equation 10: ACO – Pheromone Deposit

*Where is the pheromone level of the edge from vertex to vertex and is the pheromone level deposited by the ant on that edge, taken from* ***Equation 2***

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 as follows:

Equation 11: ACO – Pheromone Update

*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 vertices to .*

Note that in **Equation 10** evaporation is applied on the pheromone levels of the edges *before* the new pheromones are deposited. This is in line with the purpose for evaporation, which is to gradually forget older solutions.

### ACO variant: Ant System (AS)

There are a few ways in which a choice from the list of valid solution components . 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).

In AS, two mechanisms come into play that influence 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 that choice’s . The second is the heuristic information about that choice direction, by which the individual ant can make an independent assessment of the choice. The heuristic commonly used is a score for the length of the choice path demonstrated through **Equation 11**. 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.

Equation 12: ACO – Ant System Heuristic Calculation

*where*  *is the distance between vertices and , and is the heuristic score given to for that solution component choice.*

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. The calculation for this stochastic decision rule can be defined as:

Equation 13: ACO – Stochastic Ant Decision Rule

*where is the probability of choosing vertex given a current position .*

### ACO variant: Max-Min Ant System (MMAS)

Though still effective, further research as shown that the performance of the classic *Ant System* could be further improved through a stronger exploitation of the best solutions found during the search. By allowing all ants to update pheromone levels, better solutions were not as clearly 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 are also bound between a min and max value. The MMAS formula for pheromone update calculation is as follows (Dorigo et al., 2006):

Equation 14: ACO – Min-Max Pheramone Update

where is the solution returned by the best performing ant, and respectively are the upper and lower bounds imposed on the pheromones, and the operator is defined as:

Equation 15: ACO – Min-Max Clamp Operator

The paper by Stützle & Hoos (2000) offers guidelines, through which, the values used for and can be empirically configured.

### ACO variant: Ant Colony System (ACS)

The ACS algorithm, introduced by Gambardella & Dorigo (1996), blends the concepts posed in the *Ant System* and *Max-Min*, by having all ants update the pheromone through a *local pheromone update*, while keeping the main pheromone update done at the end for only the best 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 as:

Equation 16: ACO – AS Local Pheromone Update

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 like the *Max-Min* except the clamps and are not needed (**Equation 13**).

Another notable difference that the ACS brings is to the ant decision rule (**Equation 12**). 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 in **Equation 12** as their decision rule, otherwise, they would just deterministically choose the most attractive path:

Equation 17: ACO – Deterministic Ant Decision Rule

Though either global best or iteration best can be used as the best ant representative for pheromone calculations, for ACS, the *gbest* ant is typically used, while MMAS focuses on the use of *ibest*. Of course, a mixed strategy procedurally alternating between the *gbest* and *ibest* ants can also be employed (similar to the particle swarm’s MPSO variant) (Stützle & Hoos, 2000).

# Design and Methodology

In this section, the datasets, coding languages, and configurations used for the comparative analysis of the algorithms, as well as the structure of the analysis are described.

## Data Generation

Due to the simplicity of the data set used, list of *n* vectors, it is commonplace to self-generate the data used for *Traveling Salesman Problem* tests. Data generation was supported in the body of literature, and it was the method employed for data collection in this study.

For the study, 4 datasets were generated having maps of 10, 20, 30, and 50 cities. Each dataset contained 100 maps of its respective city count and each city is stored as a randomly generated (*x, y*) vector 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. The final analysis was run against all datasets

Text, letter

Description automatically generated

Figure 11: Example Dataset for map of 10 cities

## Languages Used

For data generation, the Java-based language Processing (P3D) was used because it is a simplistic, visual based language. Processing was also used to develop another small program to display a given solution for visual inspection. **Figure 12** shows a display for an example solution returned from an Optimization Algorithm. On the left displays a solution to a map of city size 10, and on the right is a map of city size 50. Finally, a 3rd miniature program was developed in Processing for formatting; to clean up the datasets storing the results returned from the Optimization Algorithms.

Chart, line chart

Description automatically generated A picture containing diagram

Description automatically generated

Figure 12: Solution Display Program (city count 10 and 50)

The second programming language used was Python which is a high-level but also general-purpose programming language that emphasizes code readability. All Optimization Algorithms used were developed using python. Specifically, the Anaconda Navigator’s Jupyter Notebook was used to develop these programs. **Figure 13** demonstrates how the *Traveling Salesman* Objective Function was coded in python.

Graphical user interface, text

Description automatically generated

Figure 13: 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. Also, the colours automatically selected for the generated graphs in R, are quite pleasant to the eye as demonstrated in **Figure 14**.

Chart, scatter chart

Description automatically generated

Figure 14: Example Scatter Plot generated in R

Communication between languages was done through lightweight text files as all files generated and read from in this study are “.txt”.

## Algorithm Implementation and Configuration

Detail in this section is how the knowledge collected in the literature review **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 to offer only the project specific details of the algorithms developed. The implementation gaps found in the extracted body of knowledge from the literature review that needed personal review, often posed as questions in this section, were the foundation for structuring preliminary tests for algorithm configuration.

### Genetic Algorithm (GA)

The GA’s application requirement was that the problem domain be presentable as list of chromosomes and an evaluation function. In the GA implemented with regards the *Traveling Salesman Problem* (TSP), a gene was used to symbolize a city and a chromosome, which is a sequence of genes, correspondingly symbolized a sequence of cities or a route/solution to the TSP. Continually, a population, meaning a group of chromosomes, represents can be represented as a list of cities as shown in **Figure 15**.

Diagram

Description automatically generated

Figure 15: TSP setup for GA

By enforcing the gene sequence to comply with the TSP condition 1 detailed **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)

Also, in showing consideration to TSP condition 1, the methodology for breeding and mutation 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 using that information, only the missing cities in the child’s chromosome was taken from the alternate parent’s chromosome to fill out its missing section. This process is demonstrated in **Figure 16**. Mutation on the other hand, were treated as swaps between two cities chosen randomly along the chromosome sequence occurring at a rate denoted by the mutation-rate variable as illustrated in **Figure 17**.

Diagram

Description automatically generated

Figure 16: GA Breeding for the TSP

Diagram

Description automatically generated

Figure 17: GA Mutation for the TSP

The main, non-variant specific, variable used in the GA was the mutation probability which was set to 0.6% as recommended by Mirjalili (2019a). Use of the variations in GA highlighted in this study came with further configurations that needed to be investigated.

* Which Fitness function performs the best?
* If Elitist 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 vs Elitism vs Steady-State)

Along with this was the small concern found with the *Tournament Sampling* (TS) methodology highlighted in **Section 2.4.2.4**, that its SP was potentially too high. To attempt a solution to this, an approach was devised, borrowing inspiration from the Ant Colony’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 this technique back to the original TS mechanism can be done by setting the delta value . The introduction of this variable lowered the selection pressure of the TS but also brought along the question:

* What delta setting is best for the TS?

Another consideration was brought up when examining the Steady-State (SS) function and its seemingly greedy mechanism detailed in **Section 2.4.4.2**. To attempt a solution, a method was tried to localize the effect of the Steady-State while at the same time striving to preserve its essence. This *Local Steady-State* (LSS) function limits the power of high-performing children found by allowing them only to replace their direct parents if better, rather than any other, possibly lower, members of the population. Of course, this also brought along the question:

* Does this new LSS function bring any merit over the original SS?

### Particle Swarm Optimization (PSO)

The studies done by Yousefikhoshbakht (2021) and Wang et al. (2003) detailed an intriguing method for adapting the PSO to the *Traveling Salesman Problem*, 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 mistake when considering other sources. So:

* 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 MPSO model improved from the one proposed by Wang et al. (2003). He carried out test on 15 possible combinations in order to determine an optimal configuration.

* But what configuration suits this study?
* Does the MPSO suit this project more than the PSO?

### Ant Colony Optimization (ACO)

Due to the fact that the ACO was designed for tests like the *Traveling Salesman*, not much work was needed to adapt it for this study. The only things introduced to its methodology 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 that edge being stored in its corresponding matrix cells. **Figure 18** 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 18: 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 my Ant Colony algorithms was 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* (MMAS) and *Ant Colony System* (ACS) algorithms detailed in **Sections 2.4.10** and **2.4.11**, **Equations 17** and **18** using the *global* best ant solution given rather than the iteration’s best , was used:

Equation 18: ACO T-max Calculation

Equation 19: ACO T-min Calculation

For **Equation 18**, *n* is used to represent the number of components used to create a solution which, regarding the *Traveling Salesman Problem*, corresponds to the number of cities a complete route or map. Stützle & Hoos (2000) detailed an experiment process through which the appropriate configuration for the *Pbest* variable could found. So:

* What *Pbest* value is the most appropriate for this study?

Initialization of the pheromone tables for the 3 Ant Colony variants (AS, MMAS and ACS), each operated differently. For the AS, simply initializing the table to trivial, non-zero values worked. However, for the MMAS, as instructed, specialized pheromone initialization was done *after* iteration 1 was complete, as pheromone levels for each edge was initialized to the calculated value. Similarly, the specialized ACS pheromone update was calculated *after* iteration 1 was complete, where the pheromone levels for each edge was initialized to the value calculated for the ants’ local pheromone update strength for the first iteration (Gambardella & Dorigo, 1996). Note that **Equation 1** forcalculating the tour *length* is used rather than **Equation 2** for the score. All of his left the final question:

* Which of the ACO variants performs the best?

## Hybridization Methodology

As a requirement for tackling the primary, 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. 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 another) produced better results that parallel hybridization (running algorithms together) when mixing the *Ant Colony* and *Particle Swarm* optimization 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 sections of this study.

The nature of the GA seems to also lend itself to the sequential approach 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 just swap out the words “a population” from its definition, with “ants” or “particles”. Figures … respectively show the methodology through which the ACO/GA, PSO/GA and PSO/ACO algorithms were hybridized. The names for the hybrid models built were assigned using the sequential order, by which, the algorithm methodologies occurring in the hybrid operate.

## Statistical Analysis

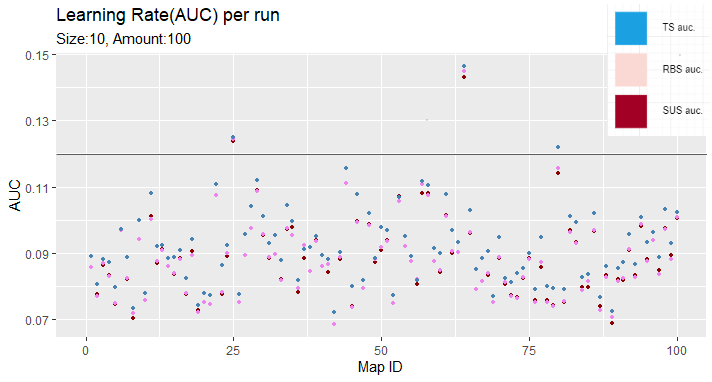
The research question of this study aimed to find the *best performing* algorithm. A statement, which unfortunately could be interpreted in a number of ways (e.g., best final solution given, optimization speed, algorithm runtime…). When it comes to Optimization, there are two main factors that come into play whenever a critique is made: the score of the final solution given, and the number of iterations take to achieve that score. In this study, it was decided to use a blend of the two by using each algorithm’s *learning rate* as its performance metric. The learning rate denotes how fast and efficiently the algorithm searches its problem space to find an answer. On a graph mapping the best scoring solutions found by the algorithm over its iterations run, the learning rate would be the area under the curve (AUC) as shown in **Figure 19** and it can be calculated using the trapezoidal function.

Diagram

Description automatically generated

Figure 19: Area Under the Curve (AUC)

For statistical analysis of a test between multiple participants, Demšar (2006) recommends the use of the Wilcoxon signed-rank tests over the widely used t-test because it is less susceptible to outliers and performs better that the t-test when they are present. 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, early experiments done drew some outliers. **Figure 20** displays the AUC data drawn from one of the early tests done comparing the fitness functions of the Genetic Algorithm. The top image displays a scatter plot showing outliers found in all algorithms, and the 3 images bellow are histograms displaying the approximate 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 20: AUC test showing outliers

For these reasons, Wilcoxon was chosen as the statistical test used for the comparative analysis and the significance threshold of 0.05 was chosen because it is the most common threshold used in statistical analysis.

# Results, evaluation and discussion

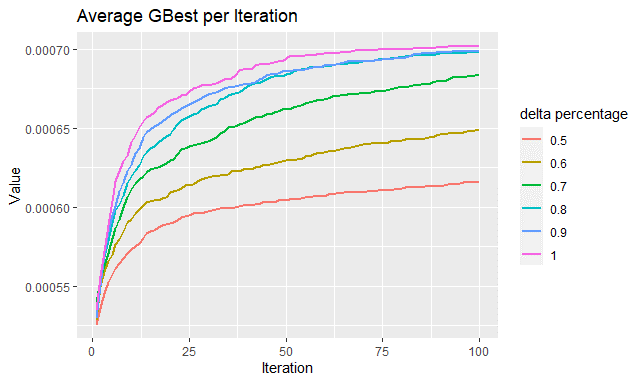
4 main experiments were performed in this study including 3 preliminary experiments, aiming to determine the best representative for each algorithm, and the final main hybrid vs base comparison of the study was the 4th. For all experiments done, unless specified otherwise, it can be assumed that the population size used for all Optimization Algorithms used was 50 and the maximum number of iterations allowed per problem was 100.

## Experiment 1: GA

Condensing the GA composition questions drawn from this study posed in **Section 3.3.1**, 2 main concerns were drawn. What Fitness function to use, and what enhancer (if any) was appropriate. This splits the experiment into two parts:

### Part 1 – The fitness function

As a first step, the effect of the introduced delta variable to the Tournament Sampling technique (TS) was examined and it was found that the original TS () performed the best. **Figure 21** displays a plot of the average global best score per iteration on the left, showing that the original TS as the top performing algorithm. Examining the AUC using a box plot shown in the image on the right, records a normal distribution for the learning rate (AUC) data, distinguishable by their approximate evenly spaced box and whiskers relative to their mean line, and it also confirms the win of the original TS. Only comparisons of the original TS against the TS with beat the Wilcoxon test with a p-value of 0.014. Deltas 1, 0.9 and 0.8 were too close in AUC for there to be declared a statistical winner.

Chart, box and whisker chart

Description automatically generated

Figure 21: Tournament Selection plots

Using this finding, 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 close rivalry between them as shown in **Figure 23**. A plot of their AUC shows that TS just barely having the highest AUC mean and range when compared to the other. There was no statistical winner declared for this comparison. Nevertheless, TS was chosen as the optimum sampling technique for this project.

Chart, line chart

Description automatically generatedChart, box and whisker chart

Description automatically generated

Figure 23: Fitness function plots

### Part 2: Enhancers

Elitism required an elite percentage specified before use, so as a first step, this an optimum setting for this needed to be found. The mapping of average scores on the left in **Figure 25**, showed very close competition but still found an elite percentage of 10% the best. However, tracking the AUC using the box plot on the left, showed that the highest mean AUC was actually the algorithm with an elite percentage of 20% and the base algorithm with an elite size of 0% came second. Statistical analysis using the Wilcoxon test was inconclusive about a winner. It seemed not much gain could be drawn from elitism.

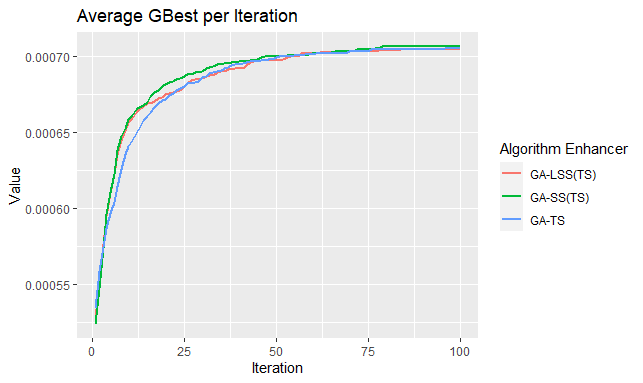
Chart, line chart

Description automatically generated Chart, box and whisker chart

Description automatically generated

Figure 25: Elitist Average plot

With elitism out of the picture,



# Conclusion

# 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, S.-M., & Chien, C.-Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications*, *38*(12), 14439–14450. https://doi.org/10.1016/j.eswa.2011.04.163

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

Mahi, M., Baykan, Ö. K., & Kodaz, H. (2015). A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing*, *30*, 484–490. https://doi.org/10.1016/j.asoc.2015.01.068

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

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation.

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.

Rohini, V., & Natarajan, A. M. (2016). Comparison of Genetic Algorithm with Particle Swarm Optimisation, Ant Colony Optimisation and Tabu Search based on University Course Scheduling System. *Indian Journal of Science and Technology*, *9*(21). https://doi.org/10.17485/ijst/2016/v9i21/85379

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

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

V.Selvi & Dr.R.Umarani. (2010). Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. *International Journal of Computer Applications*, *5*. https://doi.org/10.5120/908-1286

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

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 A