

2017-9

The Use of Persistent Explorer Artificial Ants to Solve the Car Sequencing Problem

Kieran O'Sullivan
Technological University Dublin

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

 Part of the [Computer Engineering Commons](#)

Recommended Citation

O'Suillivan, K. (2017) The Use of Persistent Explorer Artificial Ants to Solve the Car Sequencing Problem, Masters Dissertation, Technological University Dublin.

This Dissertation is brought to you for free and open access by the School of Computing at ARROW@TU Dublin. It has been accepted for inclusion in Dissertations by an authorized administrator of ARROW@TU Dublin. For more information, please contact yvonne.desmond@tudublin.ie, arrow.admin@tudublin.ie, brian.widdis@tudublin.ie.



This work is licensed under a [Creative Commons Attribution-Noncommercial-Share Alike 3.0 License](#)

The Use of Persistent Explorer Artificial Ants to Solve the Car Sequencing Problem



Kieran O'Sullivan

D14127972

A dissertation submitted in partial fulfilment of the requirements of Dublin
Institute of Technology for the degree of
M.Sc. in Computing (Advanced Software Development)

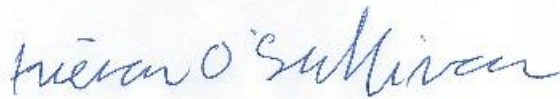
2017

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

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology 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: **2nd July 2017**

ABSTRACT

Ant Colony Optimisation is a widely researched meta-heuristic which uses the behaviour and pheromone laying activities of foraging ants to find paths through graphs. Since the early 1990's this approach has been applied to problems such as the Travelling Salesman Problem, Quadratic Assignment Problem and Car Sequencing Problem to name a few. The ACO is not without its problems it tends to find good local optima and not good global optima.

To solve this problem modifications have been made to the original ACO such as the Max Min ant system. Other solutions involve combining it with Evolutionary Algorithms to improve results. These improvements focused on the pheromone structures. Inspired by other swarm intelligence algorithms this work attempts to develop a new type of ant to explore different problem paths and thus improve the algorithm. The exploring ant would persist throughout the running time of the algorithm and explore unused paths.

The Car Sequencing problem was chosen as a method to test the Exploring Ants. An existing algorithm was modified to implement the explorers. The results show that for the car sequencing problem the exploring ants did not have any positive impact, as the paths they chose were always sub-optimal.

Key Words: Ant Colony Algorithms, Max Min Ant Systems, Explorer Ants, Metaheuristics, Particle Swarm Optimisation, Evolutionary Algorithms.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr Luca Longo for his guidance during this project and Dr John Gilligan who gave me advice and suggested the Car Sequencing Problem.

CONTENTS

1	Introduction	1
1.1	Background	1
1.2	Research Problem	2
1.3	Research Objectives	3
1.4	Research Methodologies	4
1.5	Scope and Limitations.....	5
1.6	Document Outline	5
2	Literature Review	7
2.1	Key Concepts Underpinning this Research	7
2.1.1	Evolutionary Approaches to Artificial Intelligence.....	8
2.1.2	The Ant Colony Metaheuristic	10
2.1.3	Types of Ant Colony Optimisation.....	12
2.1.4	Other Relevant Metaheuristics	14
2.1.5	Computational Complexity of Combinatorial Problems	15
2.1.6	Constraint Satisfaction Problems CSP.....	17
2.1.7	Graph Theory.....	18
2.1.8	Representing a Constraint Problem as a Graph – The N Queens Problem 19	
2.1.9	Greedy Randomised Search Algorithms	20
2.2	Typical Problems ACO are Used to Solve	21
2.2.1	Traveling Salesman Problem.....	21
2.2.2	Image Edge Detection.....	22
2.2.3	Inventory Routing.....	23
2.2.4	Network Traffic Management	23
2.2.5	Quadratic Assignment Problem QAP.....	25
2.2.6	The n-bit Trap Problem	25

2.2.7	The Grid Scheduling Problem	26
2.2.8	Assembly Line Car Sequencing	26
2.3	Common Solutions to the Inherent Problem with ACO	30
2.3.1	Multiple Pheromone Structures	31
2.3.2	Exclusivity / Elitist Solutions	33
2.3.3	Combining ACO and Evolutionary/Genetic Algorithms (EA/GA).....	35
2.3.4	Combining ACO with Other Metaheuristics	37
2.3.5	Back-Tracking Ants for Solving Search Bias.....	38
2.4	The Work of Christine Solnon Explained.....	39
2.5	Gaps and Limitation of Literature.....	41
2.6	Research Question	43
3	Design of Experiments and Methodology.....	45
3.1	Data Preparation.....	47
3.1.1	Car Sequencing Data Explained	47
3.1.2	The Car Sequencing Graph.....	49
3.2	The Ant Car Software Code.....	51
3.2.1	Supporting Software Tools	51
3.3	Experiment Setup.....	52
3.4	Repeat Original Ant Car Experiment.....	53
3.5	Implement Changes to Add Explorer Ants	53
3.6	HA Experiment – Reduce the Number of Cycles	54
3.7	HB Experiment – Unsolvable Car Sequencing Problem	55
3.8	Evaluation of Experimental Results.....	55
3.9	Strengths and Limitations of Solution Design	56
3.10	Triangulation of Findings with State of the Art Techniques	57
4	Experimental Results.....	58
4.1	Data Preparation.....	59

4.2	The Ant Car Software Code.....	60
4.3	Experiment Execution.....	61
4.4	Repeat Original Ant Car Experiment.....	61
4.5	Implement Changes to Add Explorer Ants	62
4.6	HA Experiment – Reduce the Number of Cycles	63
4.7	HB Experiment – Unsolvable Car Sequencing Problem	67
4.8	Evaluation of Experimental Results.....	67
4.9	Strengths and Solution Findings	69
5	Discussion and Conclusions	70
5.1	Research Overview and Problem Definition	70
5.2	Experiment Evaluation and Results	71
5.3	Contribution to the Body of Knowledge.....	72
5.4	Future Work	73
	Bibliography	75
	Appendix A: Violation Rates of PEAA on Ubuntu	82
	Appendix B: Car Sequence Data Example	87
	Appendix C: Shell Script for Executing Algorithm	88
	Appendix D: Makefile for Building Algorithm.....	92
	Appendix E: Modified Source Code.....	92

TABLE OF FIGURES

Figure 2.1 ACO in the Context of Evolutionary Approaches	9
Figure 2.2 Experimental setup for the double bridge experiment (Dorigo, Birattari, & Stutzle, 2006, p. 29).....	10
Figure 2.3 Basic Ant Colony Algorithm (Dorigo, Birattari, & Stutzle, 2006, p. 31) ...	11
Figure 2.4 Common Components of an ACO(Gupta, Arora, Singh, & Gupta, 2012, p. 148).....	11
Figure 2.5 Max Min Algorithm (Kötzing, Neumann, Sudholt, & Wagner, 2011, p. 210)	13
Figure 2.6 Max Min Variant (Kötzing, Neumann, Sudholt, & Wagner, 2011, p. 211)	13
Figure 2.7 Simple Social Network Graph (Robinson, Webber, & Eifrem, 2015, p. 2)	18
Figure 2.8 Graph of the N-Queens Problem (Solnon, 2010, p. 58).....	20
Figure 2.9 Comparative ACO vs. Canny vs. Sobel Edge Detection (Agrawal, Kaur, Kaur, & Dhiman, 2012).....	23
Figure 2.10 n-bit trap fitness function (Chen & Sun, 2008, p. 3).....	25
Figure 2.11 Comparison of the percentage of trials reaching the optimal (Chen, Bolun, Chen, Ling, & Sun, Haiying, 2014, p. 59).....	32
Figure 2.12 Percentage of vehicles which reached destination during simulation (Doolan & Muntean, 2014, p. 955).....	34
Figure 2.13 A Comparison of Optimal Process with GAACA & HGAACA (Yao, Pan, & Lai, 2009, p. 246).....	37
Figure 2.14 The flow of foreword and backword ant system (Hsin, Chang, & Wu, 2013, p. 48).....	39
Figure 2.15 Greedy Ransomised Car Sequencing Algorithm (Solnon, 2008, p. 1046)	40
Figure 3.1 Overview of the Experiment	46
Figure 3.2 Graph of Cars Before Processing	50
Figure 3.3 Graph After Processing	51
Figure 3.4 Example of PEAA Processing the Sequences.....	54
Figure 4.1 Overview of Experiment Execution.....	58
Figure 4.2 Time taken by Non-PEAA and PEAA (PEAA shown on right).....	63
Figure 4.3 Ubuntu Single Pheromone Structure Violation Rates for First Non-PEAA and First PEAA.....	64

Figure 4.4 Ubuntu Dual Pheromone Structure Violation Rates for First Non-PEAA and First PEAA	65
Figure 4.5 Single Pheromone Structure Violation Rate Rises for Every New PEAA .	65
Figure 4.6 Dual Pheromone Structure Violation Rate Rises for Every New PEAA....	66
Figure 5.1 Research Process	70

TABLE OF TABLES

Table 3.1 Sample CSPLab as it appears in the file.....	48
Table 3.2 Sample CSPLib Data in a more user friendly layout.....	48
Table 3.3 Valid solution	49
Table 3.4 Test Environments.....	53
Table 4.1 Test Environments.....	61
Table 4.2 Time Taken Single & Dual Pheromone Structure.....	62
Table 4.3 Explorer Ants Time Taken Single & Dual Pheromone Structure	62
Table 4.4 Ubuntu Single Pheromone PEAA Violation Rates	63
Table 4.5 Ubuntu Dual Pheromone PEAA Violation Rates.....	64
Table 4.6 Kali Single Pheromone PEAA Violation Rates	66
Table 4.7 Kali Dual Pheromone PEAA Violation Rates.....	66
Table 4.8 Test of Randomness Sequences Generated	69

1 INTRODUCTION

1.1 Background

The idea of using the natural processes of evolution to produce intelligent behaviour in artificial systems was proposed as far back as 1975 by J. H. Holland (Holland, 1975). He suggested that the incremental improvements over time achieved by natural selection could be mimicked by computer systems to achieve an unguided solution.

The key concepts of Evolutionary Systems are that there is re-production with random variation and that the most suitable off-spring get to re-produce (Simon, 2013). Each iteration of the evolutionary algorithm solves part of the problem and the best solution is propagated to the next stage. The algorithm continues until the problem is solved or until the number of iterations of the algorithm exceeds a determined value which causes it to terminate.

A subset of this type of research is “swarm intelligence” or Particle Swarm Optimisation PSO (Kennedy & Eberhart, 1995) where the behaviour of swarms is studied to produce un-guided results. In 1992 for a PhD thesis entitled “Optimization, learning and natural algorithms” (Marco Dorigo, 1992) applied the behaviour of foraging ants to the Travelling Salesman Problem TSP.

As ants forage for food they leave cent trails (pheromone) along the ground so that they can find their way back to the nest. The more a path is used the stronger these pheromones become. If a food supply is exhausted the path is abandon and the pheromone evaporates over time. Ants are more likely to follow paths with stronger cent trails however they may also randomly choose to follow non-cent-marked trails. This process results in ants having the ability to find the shortest paths to food as a shorter path will have more pheromone and it will not have time to evaporate before the next ant finds it.

Apart from ants the swarming behaviour of other insects have also been used to develop un-guided intelligence. Artificial Bee Colony (ABC) (Karaboga, 2005) and Mosquito Fly Optimisation (Alauddin, 2016) are two examples of this. As this work deals with ACO it will not go into depth on other approaches to swarm intelligence.

Ant Colony Optimisation ACO is particularly suited to Constraint Satisfaction Problems CSP's which can be expressed as choices of paths through a graph. As Dan Simon points out in Chapter 10 (Simon, 2013) ACO it remains a growing area of research and solutions for real world problems.

1.2 Research Problem

All ACO suffer from an inherent problem the negative search bias referred to hereafter as just search bias. The problem is inherent to the design of the algorithm itself. As an ant finds a useful path it lays pheromone on that path and this skews the probability that more and more ants will follow this path. This is what gives the algorithm its power and leads to its inherent flaw.

The best way to understand this flaw is the hill climbing problem. The objective of hill climbing is to get to the highest point and the best way to do that is to walk up-hill however this simplistic strategy may not succeed as the climber could follow a path to a lower point and have no way to get to the highest point. This is known as a local optimal point. ACO are vulnerable to this problem as the pheromone trails will guide more and more ants to the incorrect location. This problem does have a counterpart in nature called the “death spiral” where ants can become confused and will walk in circles until they die.

To prevent the search bias a number of solutions have been proposed these include:

1. Multiple Pheromone Structures. The use of negative pheromone to prevent bad paths from being chosen. The use of dual pheromone structures to identify the good paths and critical components.
2. Elitist Solutions. The Max Min system allows only the best ants to lay pheromone. The Rank system places more pheromone on shorter paths.

3. Combining ACO and EA. The use of Evolutionary Algorithms and ACO together has allowed for the finding of good global optima and the ACO takes over to find good local optima.
4. Combining ACO with Other Metaheuristics. Tabu search and Simulated Annealing have been used in conjunction with ACO to solve search bias.
5. Back-Tracking Ants for Solving Search Bias. This combines ACO with odd even turn for adaptive routing algorithm.

1.3 Research Objectives

The objective of this research is to contribute to the body of scientific knowledge in the area of Ant Colony Optimisation. More specifically to determine if a new type of Ant added to the ACO metaphor known as Permanent Explorer Artificial Ants PEAA can improve the performance of ACO by exploring unused paths.

From a list of problems which included Image Edge Detection, Traveling Salesman, Quadratic Assignment and Network Packet routing; the car sequencing problem was chosen as it is a benchmark for ACO and the work done by (Solnon, 2008) on which this project is based used the car sequencing problem.

The research question and hypothesis are outlined below.

Can the addition of Persistent Explorer Artificial Ants (PEAA) as opposed to setting Pheromone values alone reduce the number of cycles and detect unsolvable sequences in the Car Sequencing Problem?

Linked to the question the following hypothesis were defined.

Null Hypothesis HA₀: PEAA will not statistically Reduce the number of cycles when solving the Car Sequencing problem.

Alternative Hypothesis HA₁: PEAA will Reduce the number of cycles when solving the Car Sequencing problem.

Null Hypothesis HB₀: PEAA will not statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

Alternative Hypothesis HB₁: PEAA will statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

1.4 Research Methodologies

The work started with secondary research into the underlying concepts necessary to understand ACO. These concepts are:

1. Ant Colony Algorithms & Pheromone Trails
2. Evolutionary Approaches to Artificial Intelligence
3. Computational Complexity of Combinatorial Problems
4. Constraint Satisfaction Problems CSP
5. Types of Ant Colony Optimisation
6. Graph Theory
7. Representing a Problem as a Graph – The N Queens Problem
8. Greedy Randomised Search Algorithms

The CRISP-DM (CRoss Industry Standard Process for Data Mining) (Wirth & Hipp, 2000) is also used as a reference standard for the overall project methodology. These methodologies are iterative processes and somewhat evolutionary which is appropriate for the subject matter of this research.

The primary research involved building on the work of Christine Solnon (Solnon, 2008). New code was added to modify the original algorithm to implement Persistent Explorer Artificial Ants (PEAA). The results of the addition would be compared against the original algorithm to perform qualitative and deductive research.

1.5 Scope and Limitations

The strengths of the approach are the use of a comprehensive set of car sequencing data. The 109 datasets are more extensive than that used by (Solnon, 2008) which used 82 datasets. The experiments will also be validated against the best available solutions currently recorded in the literature.

The limitations on this approach are the relatively limited number of platforms on which it is run all of which are Unix or Linux. The hardware was also limited to only 3 processors two intel processors and one AMD processor.

While the design of this experiment is a good approximation of the Car Sequencing problem it is not as extensive as that proposed by the French car manufacturer Renault. The ROADEF (Solnon, Cung, Nguyen, & Artigues, 2008) car sequencing problem includes additional constraints for paint batching. Renault needed this constraint to cut down on the use of solvents.

Given the time constraints and the different dataset formats used for the ROADEF problem it would not be possible to incorporate it here. This however would be a useful next step in this research if the PEAA approach proves useful.

1.6 Document Outline

The remainder of this document is laid out with the following structure.

Chapter 2 Literature Review: - This chapter explains the key concepts behind ACO as described in the literature. These concepts include classes of combinatorial problems, constraint problems, pheromone trails and the types of ACO. Once the concepts are explained the types of problems that ACO are used to resolve are discussed with examples from the literature. The problems inherent to using the ACO method as highlighted from the available literature are then discussed. The work of Christine Solnan is described as it is essential to this project. Finally, the gaps and limitations of the research and the research question that arises from these gaps are outlined.

Chapter 3 Design of Experiments and Methodology: - This chapter describes the collection and structure of the dataset. The experiment setup and design is then explained. The process of altering the algorithm is explained and the limitations of the approach are discussed. Finally a triangulation of the results against state of the art solutions is proposed.

Chapter 4 Experimental Results: - This chapter is very similar in structure to chapter 3. The experimental results are discussed here and the methods for measuring them. The results are related to the research question and the appropriate hypothesis are accepted or rejected.

Chapter 5 Discussion and Conclusions: - The final chapter discusses the results in the context of the current literature and explains the contribution and impact as well as possible future research.

2 LITERATURE REVIEW

This chapter covers the relevant research into the area of Ant Colony Optimisation. The sections are briefly explained here:

1. **Key Concepts Underpinning this Research:** - This explains the concepts that need to be understood when discussing ACO.
2. **Typical Problems ACO are Used to Solve:** - This lists the types of problems most commonly found in the literature to which ACO are applied.
3. **Common Solutions to the Inherent Problem with ACO:** - Common solutions to the problem that all ACO can encounter are discussed here.
4. **The Work of Christine Solnon:** - This is explained as it is essential to the project.
5. **Gaps and Limitation of the Literature:** - The limitations of the current literature on ACO
6. **Research Question:** - The research question inspired by the gaps and the hypotheses to answer it.

2.1 Key Concepts Underpinning this Research

There are some relevant concepts which must be understood to fully understand this work these are outlined below. In brief, these are:

1. Evolutionary Approaches to Artificial Intelligence
2. The Ant Colony Metaheuristic
3. Computational Complexity of Combinatorial Problems
4. Constraint Satisfaction Problems CSP
5. Types of Ant Colony Optimisation
6. Graph Theory
7. Representing a Problem as a Graph – The N Queens Problem
8. Greedy Randomised Search Algorithms

2.1.1 Evolutionary Approaches to Artificial Intelligence

In 1975 J. H. Holland realised the potential for artificial intelligence to be modelled on biological systems. In a paper on the subject (Holland, 1975) he proposed that the unguided mechanism of evolution could be used as a form of Artificial Intelligence. An evolutionary algorithm works by generating a number of solutions to a problem the best of these solutions is then taken and used to refine the problem. As the algorithm progresses the solution progressively improves until the problem is solved. The key to this process is mutation where all the solutions have a random component. This process draws heavily on the Darwinian theory of natural selection. Evolutionary Algorithms EA also known as Genetic Algorithms GA are an ongoing topic of research. Dan Simon (Simon, 2013) describes the four key components to an EA/GA as:

1. Natural Selection the solutions which are the best at solving the problem are those which will get to propagate to the next stage.
2. Reproduction the two best solutions are combined to create new solutions which should be better than their parents. Some EA/GA do not use the crossover method and re-produce with a form of mitosis (parents sub-divide like bacteria).
3. Mutation the new solution is mutated to ensure that it is different from its parent(s). This ensures that the algorithm can progress and not simply re-produce the same solutions over and over again.
4. Termination Case to prevent the algorithm executing forever a termination case is built in these can be:
 - a. Time Limitations where the algorithm will finish after a set period.
 - b. Improvement Limitations where the children are no better than their parents for a set number of generations.
 - c. Generational Limitations the algorithm will only execute for a set number of generations and terminate.

A subset of EA/GA is swarm intelligence or Particle Swarm Optimisation PSO (Kennedy & Eberhart, 1995). A PSO iteratively tries to improve a solution to a given problem by re-ordering the population of candidate solutions. Each solution or

“particle” is moved to its best possible local position based on the position of its neighbouring particle. A subset of PSO is swarm intelligence which is based on the behaviour of swarming insects.

The collective behaviour or swarming insects has inspired the development of a number of approaches to artificial intelligence. The behaviour of swarming insects is an unguided process however it leads to successful outcomes for the insects. The intelligence is not to be found in the individual intelligence of the insect but in the behaviour of the swarm (Simon, 2013).

Ant Colony Optimisation (Marco Dorigo, 1992), Artificial Bee Colony (ABC) (Karaboga, 2005) and Mosquito Fly Optimisation (Alauddin, 2016) all use the collective foraging behaviour of insects to solve problems with each step progressively improving the solution. While ACO pre-dates PSO it is still classified as a subset of PSO.

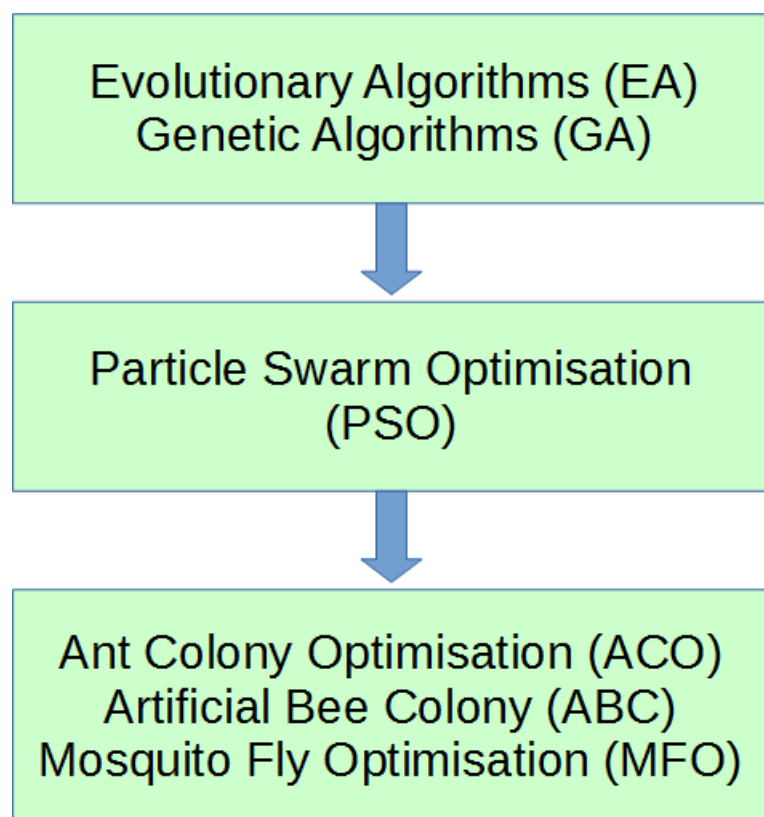


Figure 2.1 ACO in the Context of Evolutionary Approaches

It is worth mentioning that the development of metaphor or nature inspired algorithms is not without its critics. In an article entitled “Metaheuristics—the metaphor exposed” (Sörensen, 2015) argues that the concept is merely a way to hide a lack of innovation behind a fancy name. This research will not discuss whether this is a valid criticism or not in any detail other than to acknowledge that the controversy does exist.

2.1.2 The Ant Colony Metaheuristic

The concept of An Ant Colony Algorithm also referred to as Ant Colony Optimisation or Ant Colony Systems was first developed by (Marco Dorigo, 1992) for his PhD Thesis. As ants forage for food they leave cent rails along the ground so that they can find their way back to the nest. The more a path is used the stronger these cent trails become. If a food supply is exhausted the path is abandoned and the pheromone evaporates over time. Ants are more likely to follow paths with stronger cent trails however they may also randomly choose to follow non-cent-marked trails. This process results in ants having the ability to find shortest paths to food as a shorter path will have more pheromone and it will not have time to evaporate before the next ant finds it.

The original inspiration for the ACO metaphor came from experiments on Argentinian ants (Deneubourg, Aron, Goss, & Pasteels, 1990). The experiment set-up two bridges over an obstacle one was longer than the other. It was observed that ants preferred to use the short bridge to the food source. Figure 2.1 Illustrates the experiment.

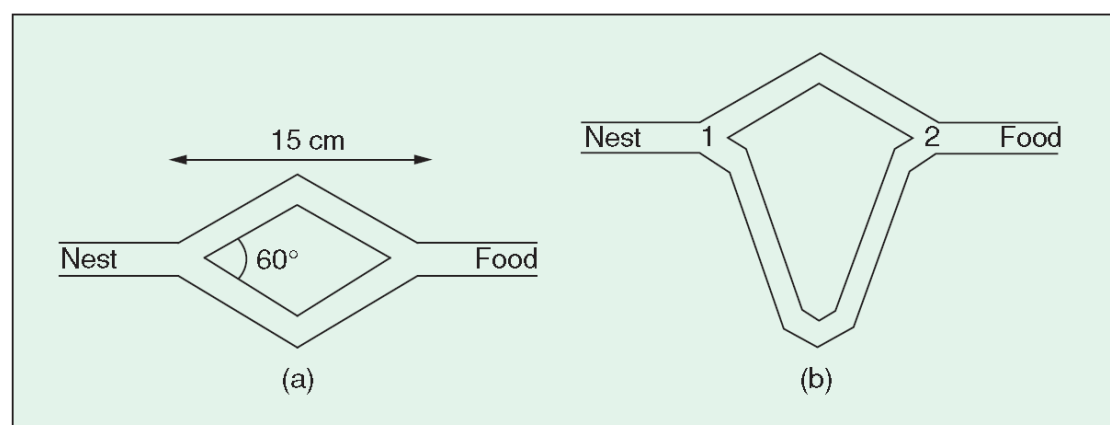


Figure 2.2 Experimental setup for the double bridge experiment (Dorigo, Birattari, & Stutzle, 2006, p. 29)

The figures below describe the components common to all ACO

Algorithm

Common Features

Set parameters
initialize pheromone trails

while termination condition not met do

 ConstructAntSolutions()

 ApplyLocalSearch

(optional)

 UpdatePheromones()

End while

- **Nodes** through which ants move at each iteration.
- **Paths** connecting the **Nodes**.
- **Pheromone Trails** (τ) τ is a matrix of the nodes in the graph.
- **Heuristic function** (η) updates the **pheromone** after each iteration.
- **Probability Function** to decide the probability of a **path** to be chosen by an **ant** when moving between **nodes**.
- **Evaporation Rate** (p) This is a number between 0 and 1 and is multiplied the value of τ_{ij} (each point in the matrix is multiplied by p)
- **Ants** data structures used pass through **nodes** and lay the **pheromone**.

Figure 2.3 Basic Ant Colony Algorithm (Dorigo, Birattari, & Stutzle, 2006, p. 31)

Figure 2.4 Common Components of an ACO (Gupta, Arora, Singh, & Gupta, 2012, p. 148)

In ACO pheromone trails are usually represented as a matrix. The matrix is the coordinates of each node in the problem graph. (Cordon, Viana, Herrera, & Moreno, 2000). The pheromone structure is updated throughout the life-time of the algorithm as different ants lay pheromone at different coordinates.

When the original ACO was applied to the Travelling Salesman problem (Flood, 1956) the algorithm was able to solve it. However it could only solve simpler versions of the

problem 75 cities or less as the problem grew the performance worsened. This led to modifications to the original design and new variants of the ACO were developed.

2.1.3 Types of Ant Colony Optimisation

To solve harder problems variants of the Ant colony systems were developed those listed here are relevant to this work however this is not a comprehensive list.

2.1.3.1 Max Min Ant Systems

The Max Min system introduced by (Stützle & Hoos, 2000) and known as MMAS added four new concepts these were:

- (i) After each iteration only one ant laid pheromone, this ant was the best for that iteration.
- (ii) Search stagnation was avoided by setting a minimum and maximum value for pheromone T_{\min} and T_{\max} . Only pheromone values between these two values were allowed.
- (iii) The pheromone values were initialised to the T_{\max} value which allowed for a better exploitation of solutions at the start of the algorithm.
- (iv) To prevent premature convergence, pheromone trails are bounded between T_{\max} and T_{\min} such that $0 < T_{\min} < T_{\max}$.

A variant to the Max Min system was introduced by (Neumann, Sudholt, & Witt, 2009) the only difference is that the MMAS* only chooses a new solution if the solution is better as can be seen by comparing line 6 of both algorithms. The algorithms are shown in Figures 2.5 and 2.6

Algorithm 1 MMAS	Algorithm 2 MMAS*
1: Set $\tau(u,v) = 1/2$ for all $(u, v) \in E$	1: Set $\tau(u,v) = 1/2$ for all $(u, v) \in E$
2: Construct a solution x^+	2: Construct a solution x^+
3: Update pheromones w. r. t. x^+	3: Update pheromones w. r. t. x^+
4: Repeat forever	4: Repeat forever
5: Construct a solution x .	5: Construct a solution x .

6: if $f(x) \geq f(x^+)$ then $x^+ := x$. 7: Update pheromones w. r. t. x^+ <i>Figure 2.5 Max Min Algorithm (Kötzing, Neumann, Sudholt, & Wagner, 2011, p. 210)</i>	6: if $f(x) > f(x^+)$ then $x^+ := x$. 7: Update pheromones w. r. t. x^+ <i>Figure 2.6 Max Min Variant (Kötzing, Neumann, Sudholt, & Wagner, 2011, p. 211)</i>
--	---

These changes showed improvements to the Ant Colony System proposed by (Dorigo & Gambardella, 1997) when applied to larger versions of the Traveling Salesman Problem. This project will use the Max Min approach.

2.1.3.2 Rank Based

Like the Max Min ant system Rank based system AS(rank) (Bullnheimer, Hartl, & Strauß, 1997) is an elitist system where the best ants get to lay the most pheromone. All solutions are weighted according to the path length of the solution they produce. The solutions with the shortest paths get the most pheromone. This was applied to the Traveling Salesman problem and like the Max Min system showed improvements over the original system. The approach has been applied to vehicle routing problems (Bullnheimer, Hartl, & Strauss, 1999) published a paper “Applying the ant system to the vehicle routing problem” the same authors published a similar paper entitled “An improved Ant System algorithm for the Vehicle Routing Problem” (Bullnheimer, Hartl, & Strauss, 1999). Both of these studies used AS(rank) for this problem. According to a comparative study done by (Adubi & Misra, 2014). The literature shows that the Rank based systems are seldom used and the Max Min system is more popular.

2.1.3.3 Continuous Orthogonal Ant Colony (COAC)

A COAC (Hu, Zhang, & Li, 2008) is similar to a Max Min system in that it is elitist and only the most efficient ants get to lay pheromone. These algorithms are useful for multi-factor problems. The problem is broken up into Orthogonal arrays and ants lay pheromone on these sub paths first. The best of these is selected to proceed to the next stage.

COAC defines its orthogonal array $OA(N,k,s)$ where N is the number of combinations to be tested, k is the number of factors and s is the number of levels for the factors. Orthogonal arrays have been used in many other areas of research and they are designed to do a “partial experiment” where the number of factors are too large to explore all the possible combinations.

2.1.3.4 Recursive Ant Colony Optimisation (RACO)

The term recursive ACO relates more to the method of implementing the algorithm in a recursive manner than modifications to the fundamentals of the ACO algorithm. In a survey of the various ACO algorithms (Adubi & Misra, 2014) date the development of RACO to work done in 2012 by (Gupta, Arora, Singh, & Gupta, 2012) who developed a new component to ACO “depth” which records the recursion point of the algorithm. Results of each recursive execution contribute to the parameter values of the next recursive step. The algorithm was first tested on geophysical data to locate elements inside the earth.

The concept of RACO has been applied to other areas since its inception (Amudhavel et al., 2015) applied it to a collision avoidance system for traffic. This use Vehicle Ad-Hoc network communication to allow vehicles to avoid congestion. Ad-Hoc networks are not reliable and RACO was used to break the network down into more manageable chunks. This paper did not add any innovation or changes to the original RACO.

2.1.4 Other Relevant Metaheuristics

While the Ant Colony metaheuristic is the focus of this research there are others which are relevant and they are briefly covered here.

2.1.4.1 Tabu Search

Tabu Search was developed by Fred Glover (Glover, 1986) and formalised three years later (Glover, 1989). It searches local nodes of the search space to attempt to find a better solution however to prevent the local optimal problem it allows for inferior solutions to be used if the algorithm is stuck and choosing an inferior solution will allow the algorithm to explore a greater area of the search space.

As a tabu search executes it maintains a fixed list of recent moves, not all the moves are remembered. To prevent the local optimal problem a move is measured against the previous move if the move is not superior a move from the list is selected to allow the algorithm to proceed.

2.1.4.2 Simulated Annealing

Simulated Annealing was developed in 1983 by (Kirkpatrick, Gelatt, Vecchi, 1983) as a method of optimisation which avoided the local optimal by retaining some poorer results which can be used to allow the algorithm to escape the problem which exists with ACO. SA is based on the process of heating metal and then slowly lowering the temperature to remove defects.

2.1.4.3 K-Means Clustering

This mechanism was used by (Niknam, Firouzi, & Nayeripour, 2008) in combination with ACO. The description here is based on that paper and a second paper (Niknam & Amiri, 2010). K-means clustering is a mechanism for grouping or clustering n discrete data points into k clusters. Each data point is grouped into the cluster with the closest mean value. In the same way as ACO k-means clustering can very quickly arrive at a local optimum. As with ACO methods have been applied to overcome this problem such as (Likas, Vlassis, & J. Verbeek, 2003) who added a global search function to search the clusters. Another mechanism to prevent this problem is random re-starts of the algorithm.

2.1.5 Computational Complexity of Combinatorial Problems

Combinatorial problems are those which can be resolved by a review of a finite set of combinations. The best example of this is a time-table containing a small number of meetings. If the number of meetings is small then the computational process is relatively easy. However, if only a small few more meetings are added the combination becomes much more complex (Solnon, 2010).

Problem classes have been introduced for those which ask a question and return a value. There are many classifications of problem complexity which will not be discussed here. According to the Complexity Zoo there are 533 problem

classifications and counting (Aaronson, n.d.). This section will only briefly explain the P, NP, NP-Hard and NP-Complete classes as these terms are used throughout the document.

2.1.5.1 P Class

These are problems which may be solved in polynomial time by a Turing machine. In practice, this means that the problem can be solved in $O(n^k)$ where n is the size of the input data and k is the constant independent of that input data. Some P class problems are:

- Searching for a number in an array.
- Searching for a character in an array.
- Deciding if an integer is a prime number or not.
- Searching for the shortest path between two nodes in a weighted graph.
- Finding a value in a table.

2.1.5.2 Polynomial Time Algorithms

This refers to the time it takes for an algorithm to execute (be solved). Polynomial time algorithms are said to be “fast” that is they will execute in a reasonable amount of time and they have an upper limit to that execution. For a given input $O(n^k)$ where k is a non-negative integer and n is the complexity of the input. ACO are non-polynomial as they can potentially execute for ever. (Solnon, 2010)

2.1.5.3 NP Class

These are problems which may be solved in polynomial time on a non-deterministic Turing machine. This could be thought of as a machine which runs a set of alternatives in parallel. NP problems usually require a large number of combinations and it may be exponential. Another way of thinking of an NP problem is using computer passwords. If a password can be any combination of letters, symbols and numbers it is very difficult to guess but if given a solution the computational machine can quickly verify it.

2.1.5.4 NP-Hard

The Car sequencing problem was shown to be NP Hard (Kis, 2004). In general an NP-Hard problem is any problem where the algorithm for solving it could be translated into solving any NP problem (Solnon, 2010).

2.1.5.5 NP-Complete

The most difficult set of NP problems to be solved are referred to as NP-Complete. The first problem to be shown to be NP-Complete was the SAT problem (Cook, 1971). Graph colouring, travelling salesman problems and clique problems have now also been shown to be NP-Complete (Papadimitriou, 1994). The car sequencing problem which will be discussed below was said to be NP-Complete by (Parrello, Kabat, & Wos, 1986) however (Kis, 2004) showed that it was only NP-Hard.

2.1.6 Constraint Satisfaction Problems CSP

Constraint Satisfaction Problems CSP are those where the states of the objects in the problems must satisfy a set of constraints. In most situations problems cannot have limitless resources thrown at them so constraints are placed upon them where it becomes too costly to solve the problem it can be abandon. This leads to optimal as opposed to perfect solutions. For simpler constraint satisfaction problems such as the n-queens problem a consistent solution can be found however the subject of this work the car sequencing problem is an NP-hard problem which cannot be solved as easily.

A formal definition of the general constraint problem given by (Khichane, Albert, & Solnon, 2008, p. 85),

"A Constraint Satisfaction Problem CSP is defined by a triple (X,D,C) such that X is a finite set of variables. D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$, that is, the finite set of values that can be mapped to x_i and C is a set of constraints, that is, relations between some variables which restrict the set of values that can be assigned simultaneously to these variables. "

To solve a CSP values must be assigned to variables so that constraints are satisfied. If all the variables are assigned in such a manner that no constraints are violated then the solution is said to be consistent.

To solve CSP the algorithm will start at the beginning and assign values to variables until the problem is resolved or until it starts violating constraints. The algorithm can then back-track and try other paths to solve the problem.

In CSP's with very large problems for example when the number of sites in the Quadratic Assignment Problem is greater than 26 the problem cannot be solved to consistency (Sahni & Gonzalez, 1976). The use of an ACO to get an optimal solution becomes more appropriate in situations like this. All the problems described in this research are CSP.

2.1.7 Graph Theory

(Robinson, Webber, & Eifrem, 2015) In graph theory a graph is a structure or collection of vertices and edges also called nodes and relationships. The nodes in a graph are related to each other using the edges. An example of this can be seen in Figure 2.7 showing a simplified social network graph.

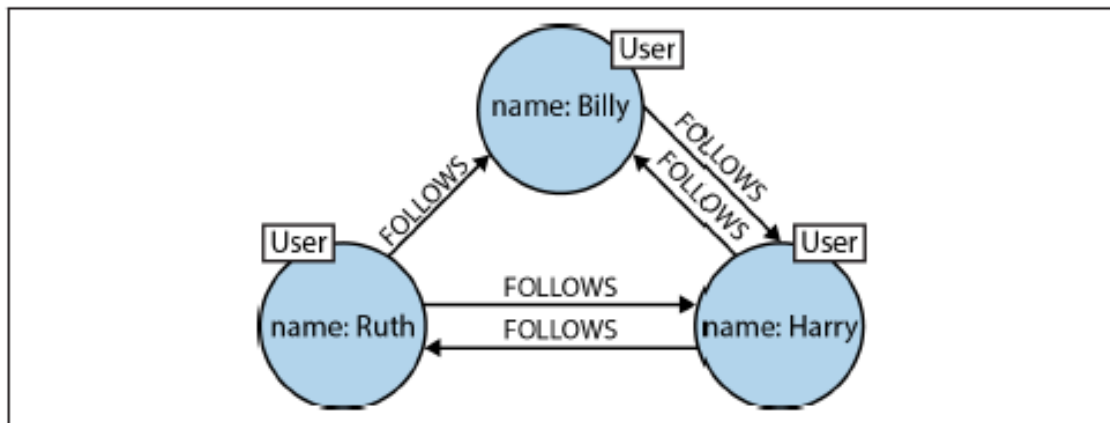


Figure 2.7 Simple Social Network Graph (Robinson, Webber, & Eifrem, 2015, p. 2)

The Swiss mathematician Leonhard Euler is regarded as the father of graph theory when in 1736 he showed that the Seven Bridges of Königsberg problem could not be solved. (Shields, 2012). The problem was to try and cross each of the bridges of the city of Königsberg once and only once. A variation on this problem is the Hamiltonian

circuit named after William Rowan Hamilton, in which the route taken through the graph must pass through each node only once and end where it started.

The problems that ACO solve are based on the concept of finding the best path through a graph. The solution can be seen as the best collection of nodes for the shortest possible journey. In relation to CSP this is the best collection of nodes with the least constraint violations.

2.1.8 Representing a Constraint Problem as a Graph – The N Queens Problem

The simple social network graph shown above is relatively easy to understand representing a constraint problem as a graph is a bit more abstract. To create a true graph of the car sequencing problem would create a very confusing graph a better example was chosen to illustrate this it is known as the N-Queens problem.

The N-Queens problem involves placing n queens on a $n \times n$ chessboard in such a configuration that a queen cannot capture another queen in one move. In chess queens can move any number of squares diagonal or on the same row or column until they find an occupied square.

For this example, there will be 4 queens on a four by four chessboard. The following steps describe the graph:

1. The first is the empty state where there are no queens placed on the board.
2. The first queen is placed at position (1,1). Because of the constraints this excludes the placing of the second queen on a number of positions.
3. The second queen is placed at position (2,3) this excludes further possibilities for the third placement.
4. The placement of the third queen will ensure that a fourth queen cannot be placed.
5. The algorithm will continue until a solution is found as can be seen.

The importance of the first correct move can be clearly seen in this example.

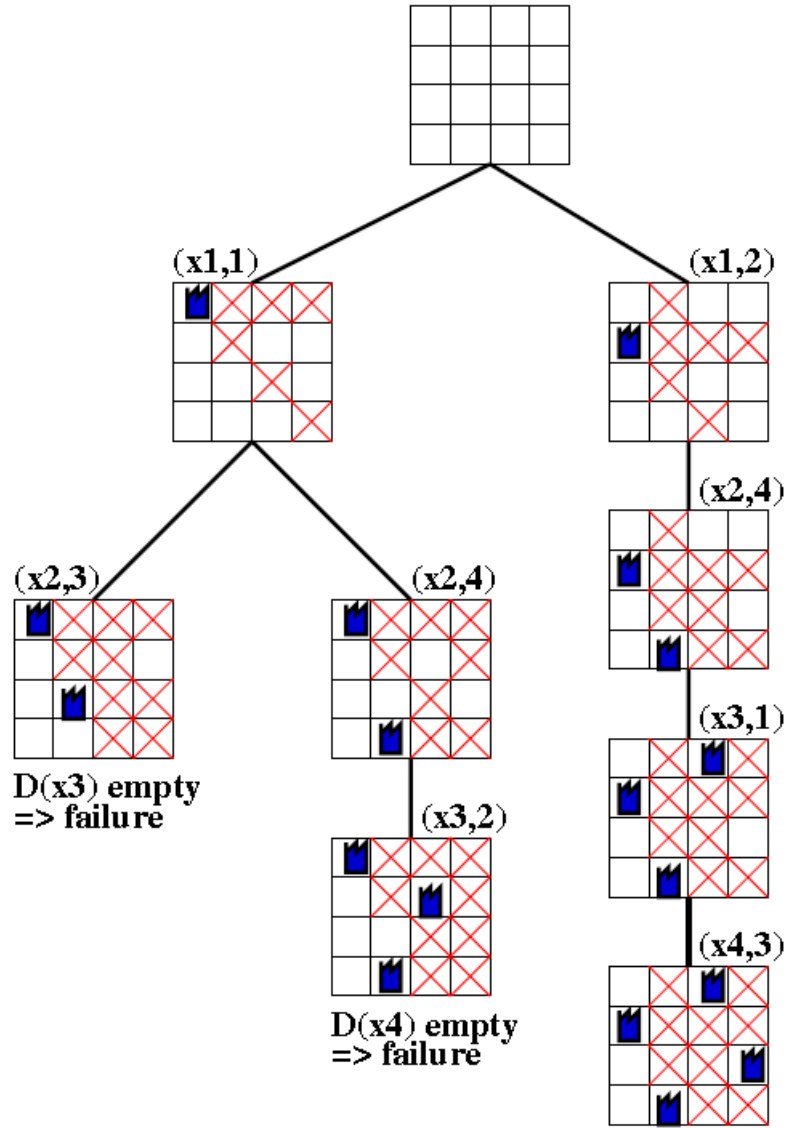


Figure 2.8 Graph of the N-Queens Problem (Solnon, 2010, p. 58)

2.1.9 Greedy Randomised Search Algorithms

Greedy randomised search algorithms also known as Greedy Randomized Adaptive Search Procedure (GRASP) first appeared in the late 1980's (Hart & Shogan, 1987) and (Gottlieb, Puchta, & Solnon, 2003). Starting with an empty list this process adds elements to the end of that list using a greedy function. In the context of the car sequencing problem starting with an empty list cars would be added onto the end of the list until a list had been constructed. Greedy functions choose the best options from the list which can make them vulnerable to the local optimal problem. An example of a greedy randomised search algorithm for the car sequencing problem can be seen in Figure 2.15.

2.2 Typical Problems ACO are Used to Solve

The types of problems that ACO's are used for are those that can be represented as finding paths through graphs. The problems listed below are a range of the most common found in the literature. These problems are:

1. Travelling Salesman Problem TSP
2. Image Edge Detection
3. Inventory Routing
4. Network Packet Routing
5. Quadratic Assignment Problem
6. Assembly Line Car Sequencing

2.2.1 Traveling Salesman Problem

The following description of the Travelling Salesman Problem is paraphrased from (Biggs, 1986). The travelling salesman problem is the most well-known of all the combinatorial problems. The principle is that a salesman wishes to travel to all of the cities on his route while covering the least distance possible. In the most common variant the salesman must visit all the cities however there are other variants of this problem which are more realistic. The multiple TSP accounts for the fact that in reality there is likely to be more than one salesman travelling for a particular company.

The TSP is the go to problem for ACO it is used to benchmark most new variants of the ACO metaphor. (Dorigo & Gambardella, 1997) tested the concept of ACO and found that it could solve this problem. (Stützle & Hoos, 2000) used TSP to verify that Max Min ant systems were superior to those proposed by (Dorigo & Gambardella, 1997).

(Junjie & Dingwei, 2006) worked on a variant of the TSP involving multiple salesmen MTSP. MTSP is a more realistic variant of the problem as it has to cope with multiple salesmen travelling to multiple locations. This has similarities to network packet routing. The work on the MTSP used a Max Min system they found that it had

competitive results when compared to the Modified Genetic Algorithm (MGA) developed as part of work done by (Tang, Liu, Rong, & Yang, 2000) to improve iron and steel production in China. However, the MGA algorithm performed better than the ACO when there were larger numbers of salesmen and cities to visit.

Another variant of the TSP is the GTSP. This breaks the cities up into sub-classes and the salesman must visit these sub classes. (Yang, Shi, Marchese, & Liang, 2008) developed an ACO to solve this problem it is a more realistic type of problem than the TSP.

2.2.2 Image Edge Detection

Image edge detection is fundamental to computer vision. ACO have been significantly useful in this area (Tian, Yu, & Xie, 2008), (Agrawal, Kaur, Kaur, & Dhiman, 2012) and (Rafsanjani, Marjan, Kuchaki, & Varzaneh, Zahra Asghari, 2015) have all used an ACO approach to detect image edges. The approaches were all similar. Edges were detected by finding divergences between areas of the image. If a divergence was detected pheromone was laid on these edges. Each of the papers showed that ACO could outperform other edged detection algorithms such as Canny and Sobel. The results of this can be seen in Figure 2.9.

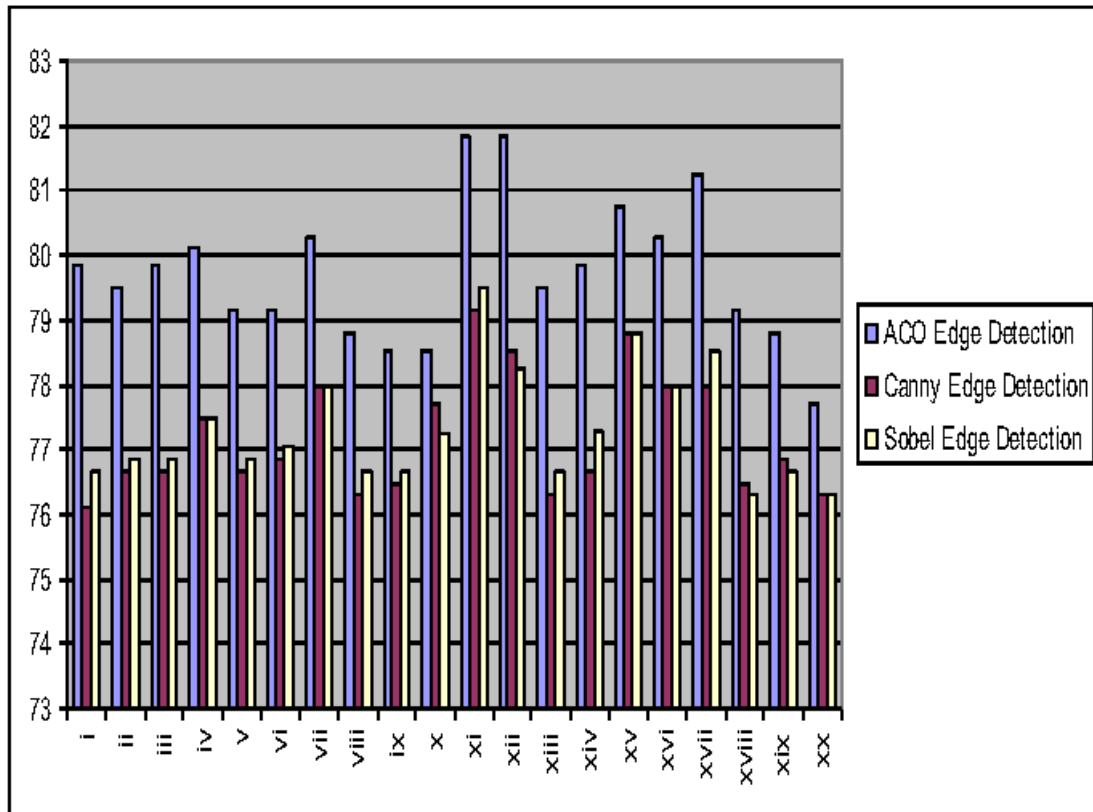


Figure 2.9 Comparative ACO vs. Canny vs. Sobel Edge Detection (Agrawal, Kaur, Kaur, & Dhiman, 2012)

The work by (Rafsanjani, Marjan, Kuchaki, & Varzaneh, Zahra Asghari, 2015) was an improvement on the previous methods which used ACO for edge detection. It was able to improve edge detection in noisier images and the edges were thinner. This work used a state detection function which was less sensitive to gaussian noise.

2.2.3 Inventory Routing

A feature of modern e-commerce is the return of non-damaged goods these can be re-sold to new customers. (Deng, Li, Guo, & Liu, 2016) developed an inventory routing system using ACO to cater for the return of non-damaged and damaged goods. The paper showed that the ACO system could reduce costs and improve turn-around times for return of non-damaged goods.

2.2.4 Network Traffic Management

Routing network packets is not much different than the MTSP. The structure of TCP/IP is ideal for mapping onto a graph with packets travelling from node to node in much the same way as salesmen travelling from city to city. Research into the power usage of sensors in a wireless network (Gnanasundari, 2015) showed that power

savings of up-to 10% could be achieved using an ACO approach. While ACO may be useful for wireless sensor networks the idea has been around since the late 1990's (Di Caro & Dorigo, 1998) proposed a distributed stigmergetic Control for Communications Networks this was an Ant Colony system which would work on wired networks. It did not catch on as TCP/IP is a robust mechanism and the cost of changing over would have been prohibitive. This was also just before the development of Max Min systems so it is not certain how the network would have responded as it grew. Given the relative cheapness of network hardware it is usually cheaper to simply throw hardware at any network than to fix the bottleneck.

Mobile networks on the other hand have benefited from an ACO approach. As GSM networks become busier the method for assigning frequencies has become more urgent. The Automatic Frequency Planning (AFP) problem encountered by GSM networks was researched by (Luna, Blum, Alba, & Nebro, 2007), they showed in a comparison between ACO and EA that an ACO could outperform the (10,1) EA in allocating signals using time limits as a stopping condition. The AFP had already been shown to be solvable using ACO by (Maniezzo & Carbonaro, 2000) in a paper entitled "An ANTS heuristic for the frequency assignment problem".

In the network multi-processing environment of Network-on-chip (NoC) systems (Hsin, Chang, & Wu, 2013) have shown that ACO can be used to increase the performance of routing between the processors. The results showed a 16% improvement over a simple ACO. This work implemented a backward mechanism which worked when network congestion was detected. This was achieved by use of a "pheromone routing table" when pheromone had built up too much on any route this was an indicator of congestion and the ants were back-tracked to other routes.

Ad-Hoc vehicle networks are another area where ACO have been used (Amudhavel et al., 2015) discussed the use of ACO to improve the working of vehicle networks. These are networks where vehicles talk to each other through road-side stations. This has many potentials for self driving vehicles which could communicate with each other.

From the examples given above ACO can be used for network management. However, this is more useful with mobile topologies as they are more fluid. A

standard desktop wired network is relatively stable and congestion problems are more easily tracked and resolved. Given that the relative cost of hardware ACO are unlikely to have much of an impact here as it is easier to simply throw hardware at the problem. Mobile networks will benefit more from an ACO approach as the band-width is limited and the network much more fluid.

2.2.5 Quadratic Assignment Problem QAP

The QAP is a problem for assigning facilities at given distances and maximising the flows between them. If there are a set of n facilities and n locations each pair of locations has a distance specified. Each facility has a flow between it and other facilities (the flow could be anything from liquid to goods and services). The problem is to optimise the flow between facilities while covering the minimum distance possible.

The QAP was one of the tests used by (Stützle & Hoos, 2000) to demonstrate the performance of the Max Min ant system. (Tavares & Pereira, 2011) used QAP to develop a prototype for a Self Ant System this is discussed further in section 2.3.2. In a survey of solutions to the QAP (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007) highlighted the effectiveness of ACO when working on this problem. (Sahni & Gonzalez, 1976) showed that when the number of sites in the QAP rises above 26 it cannot reach convergence and only an optimal solution is possible.

2.2.6 The n-bit Trap Problem

The n-bit Trap Problem is a first order deceptive problem used to test both ACO and EA/GA (Chen, Bolun, Chen, Ling, & Sun, Haiying, 2014). It is specifically useful for testing search bias. The n-bit trap problem is to find a binary number with the highest fitness in a set of binary numbers from 0 to $2^n - 1$. The fitness is given by the equation where $h(s)$ is the Hamming distance between s and 0. The answer is $s^* = 0$.

$$f(s) = \begin{cases} h(s) & s \neq 0 \\ n + 1 & s = 0 \end{cases},$$

Figure 2.10 n-bit trap fitness function (Chen & Sun, 2008, p. 3)

2.2.7 The Grid Scheduling Problem

This is similar to job-shop problems in that jobs must be executed in a reasonable time however this process relates to the submission of jobs to a computing grid or queue. As (MadadyarAdeh & Bagherzadeh, 2011) explain no one process controls the grid and jobs are added to it in a dynamic manner. The problem is to allocate the jobs to the available nodes and gain an optimal performance in terms of resources.

2.2.8 Assembly Line Car Sequencing

As the car sequencing problem is used as the test case it is explained more thoroughly here than any of the other problems. A quote often attributed to Henry Ford is “people can have a car in any colour as long as it is black”. In such a simplistic world the car sequencing problem does not arise. Modern manufacturing allows for mixed-model assembly lines where many variants of a model can be produced each variant has a set of features and these are grouped in to classes (Flidner & Boysen, 2008). (Parrello, Kabat, & Wos, 1986) described the car sequencing problem in a paper entitled “Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem”. It was introduced to the Constraint Programming community two years later by (Dincbas, Simonis, Van Hentenryck, 1988) who developed a new Prolog variant for constraint handling Constrain Handling In Prolog (CHIP). The car sequencing problem was used to demonstrate the effectiveness of CHIP. When the CSPLib (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent, 1999) project was first setup the car sequencing problem was the first to be documented. It was said to be an NP-Complete problem by (Parrello, Kabat, & Wos, 1986) however (Kis, 2004) showed that it was only NP-Hard.

To paraphrase (Bruce D. Parrello, Waldo C. Kabat, & L. Wos, 1986) from their paper on job shop scheduling using automated reasoning. On a single assembly line of cars there are a number of work stations each of which install different components. For example, one work station installs sun-roofs, another installs alloy-wheels and another installs air-conditioning systems. Each work station has a limited capacity which means that they can only install a limited number of items.

A very simplistic solution to this problem would be to slow down the assembly line so that all cars get all options however this is extremely costly. If only 60% of cars will ever get air conditioning it is better to design the system so that the rest of the cars are not waiting to be processed.

The French car manufacturer Renault introduced further modifications to this problem (Solnon, Cung, Nguyen, & Artigues, 2008). Their solution also took into account the environmental impact of spray-painting cars and was an attempt to reduce solvent usage. This is a more real-world example of the Car sequencing problem but for the purposes of this research the car sequencing datasets being used are those available from CSP Library (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent, 1999).

ACO were applied to the Renault problem by (Gagné, Gravel, & Price, 2006). The work showed improvements over the previous work done on this by (Chew, David, Nguyen, & Tourbier, 1992). As this project does not use the Renault problem it is not further discussed here.

2.2.8.1 Formalising the Car Sequencing Problem

A formal definition of the car sequencing problem is given in (Solnon, Cung, Nguyen, & Artigues, 2008, p. 913)

“The car sequencing problem defined by a tuple (V, O, p, q, r) where:

- $V = \{v_1, v_2, \dots, v_n\}$ The set of vehicles to be produced.
- $O = \{o_1, o_2, \dots, o_m\}$ The set of options.
- $p : O \rightarrow \mathbb{N}$ and $q : O \rightarrow \mathbb{N}$ this defines the capacity constraint associated with each option $o_i \in O$ this capacity constraint imposes that, for any subsequence of q_i consecutive cars on the line, at most p_i of them may require o_i .
- $r : V \times O \rightarrow \{1, 0\}$ defines options requirements, i.e., for each vehicle $v_j \in V$ and for each option $o_i \in O$, $r_{ji} = 1$ if o_i must be installed on v_j , and $r_{ji} = 0$ otherwise. “

2.2.8.2 The Car Sequencing Problem as a CSP

In their review of state-of-the-art solutions to the car sequencing problem (Solnon, Cung, Nguyen, & Artigues, 2008) outlined that converting the car sequencing problem to CSP involved adding three different types of variables and two different types of constraints. The CSP they propose for the problem is:

The Variables

1. A Slot Variable X_i this is associated with each position i in the sequence of cars. This represents the class of the i^{th} car in the sequence and its domain is the set of classes for the cars. Using the data in Table 3.1 the domain would be $D = \{0,1,2,3,4,5\}$.
2. An Option Variable O_i^j this is associated with each position i in the sequence and each option j . If option j is required by the i^{th} car this variable is set to 1 otherwise it is set to 0. The domain for this value is $D = \{1,0\}$.

The Constraints

1. The Link Constraint is a link between the slot and option variables. $O_i^j=1$ if option j has to be installed on X_i .
2. The Capacity Constraints specifies that the work station capacity must not be exceeded. This means that for each option j and each subsequence q_i cars a linear inequality specifies that the sum of the corresponding option variables must be $\leq p_i$
3. The Demand Constraints specifies that for each car of this class the number of cars that need to be sequenced.

2.2.8.3 The Car Sequencing Problem and ACO

The Car Sequencing problem has been used as a benchmark for many ACO algorithms and for testing new variants of these. During the process of integrating an ACO with a constraint programming language (Khichane, Albert, & Solnon, 2008) used the Car Sequencing problem as a benchmark for this work.

In a study to determine the effectiveness of combining dual pheromone structures (Solnon, 2008) used the car sequencing problem to demonstrate the effectiveness of this approach showing that the algorithm outperformed its competitors.

The car sequencing problem was used to evaluate the performance of a Dual Layer ACO (DLACO) (Li-Ning Xing, Ying-Wu Chen, & Ke-Wei YANG, 2008). This approach broke the problem up into two steps the first was to assign operations to machines, the second step was to schedule the operations of the machines.

2.2.8.4 Other Approaches to the Car Sequencing Problem

As it is a benchmark for constraint satisfaction problems other approaches have been used to solve this problem. These are briefly discussed here.

(Fliedner & Boysen, 2008) used a Branch and Bound B&B approach to solve car sequencing. This is similar to an ACO in that it represents a problem space as a decision tree, however there is no pheromone laying and solutions are checked against lower and upper bounds of the optimal solution.

Simulated Annealing (SA) (Kolonko, 1999) applied this process to the car sequencing problem. They showed that the Car Sequencing Problem was not easily solved using SA then they used a EA to execute a number of SA to improve the results. When a SA algorithm had found a good local optimal this result would be combined to previous results using the EA over time a superior global optimal was found. This process is similar in many ways to the combining of ACO with EA for example the work done by (Zhao, Yao, Luan, & Song, 2016) for more information see section 2.3.3 Combining ACO and Evolutionary/Genetic Algorithms (EA/GA).

Tabu Search (TS) (Pezzella & Merelli, 2000) used the car sequencing problem to evaluate the performance of TS and a new technique for finding bottlenecks in the process.

A combination of ACO and TS (Huang & Liao, 2008) used the car sequencing problem to evaluate the algorithm. The work done by (Huang & Liao, 2008) is discussed in more detail in section 2.3.3.

2.3 Common Solutions to the Inherent Problem with ACO

All ACO suffer from an inherent problem the negative search bias referred to hereafter as just search bias. The problem is inherent to the design of the algorithm itself. As an ant finds a useful path it lays pheromone on that path and this skews the probability that more and more ants will follow this path. This is what gives the algorithm its power and leads to its inherent flaw.

The best way to understand this flaw is the hill climbing problem. The objective of hill climbing is to get to the highest point and the best way to do that is to walk up-hill however this simplistic strategy may not succeed as the climber could follow a path to a lower point and have no way to get to the highest point without back tracking down hill. This is known as a local optimal point. ACO are vulnerable to this problem as the pheromone trails will guide more and more ants to the incorrect location. This problem does have a counterpart in nature called the “death spiral” where ants can become confused and will walk in circles until they die.

Search bias is not unique to ACO it can be found in other similar algorithms. Tabu Search (Glover, 1986), (Glover, 1989) and Simulated Annealing (Kirkpatrick, Gelatt, Vecchi, 1983) both use a similar process of retaining sub optimal solutions to solve the search bias problem. Artificial Bee Colony (ABC) (Karaboga, 2005) uses a system of scout bees to overcome the local optimal problem.

In a comprehensive study of ACO (Adubi & Misra, 2014) listed the main variants of the algorithm and its differences based on this work the list below describes how the main ACO variants deal with the local optimal problem. The list contains a summary of these approaches and they are explained in detail in the subsequent sub sections.

1. Multiple Pheromone Structures. The use of negative pheromone to prevent bad paths from being chosen. The use of dual pheromone structures to identify the good paths and critical components.
2. Elitist Solutions. The Max Min system allows only the best ants to lay pheromone. The Rank system places more pheromone on shorter paths.

3. Combining ACO and EA. The use of Evolutionary Algorithms and ACO together has allowed for the finding of good global optima and the ACO takes over to find good local optima.
4. Combining ACO with Other Metaheuristics. Tabu search and Simulated Annealing have been used in conjunction with ACO to solve search bias.
5. Back-Tracking Ants for Solving Search Bias. This combines ACO with odd even turn for adaptive routing algorithm.

2.3.1 Multiple Pheromone Structures

Based on the behaviour of real ants, observed in a paper by (Robinson, Jackson, Holcombe, & Ratnieks, 2007) the idea of a no-entry signal or negative pheromone has been used to identify paths which ants should not follow. The papers listed below use this system of oppositional learning to improve the results of ACO, however most of them pre-date the publication of the paper so it cannot have been said to have influenced their work.

The idea of a negative pheromone surfaced at about the same time as Max-Min systems it was proposed by (Iredi, Merkle, & Middendorf, 2001). They used it in the Single Machine Tool Tardiness Problem SMTTP. The results were mixed but the approach showed some promise with smaller versions of the problem.

Another attempt to apply opposition based learning ideas to ACO was introduced by (Malisia & Tizhoosh, 2007) who used negative pheromone values. The use of a negative pheromone value was applied to the TSP. The research showed mixed results with improved performance on the smaller versions of the TSP but worse performance on the larger versions. While the results were mixed this approach was innovative.

The (Robinson, Jackson, Holcombe, & Ratnieks, 2007) research was exploited by (Rodrigues & Ramos, 2014) in a document categorisation system for news articles the negative pheromone structures showed improved performance. The process of finding news articles is becoming more difficult, as a result a word association graph was created where key words were associated with larger articles. The negative pheromone structure was able to discount non-related documents reducing the search results to more relevant topics.

In a more comprehensive paper on finding a method for avoiding the searching bias in ACO deceptive problem solving, (Chen, Bolun, Chen, Ling, & Sun, Haiying, 2014) used the n-bit trap problem to test the performance of an ACO which did not use negative pheromone against an ACO which did use negative pheromone. The negative pheromone over-came the search bias inherent to ACO. The bias avoiding ACO (BA-ACO) tested showed significant improvement over the classical ACO as can be seen in Figure 2.11.

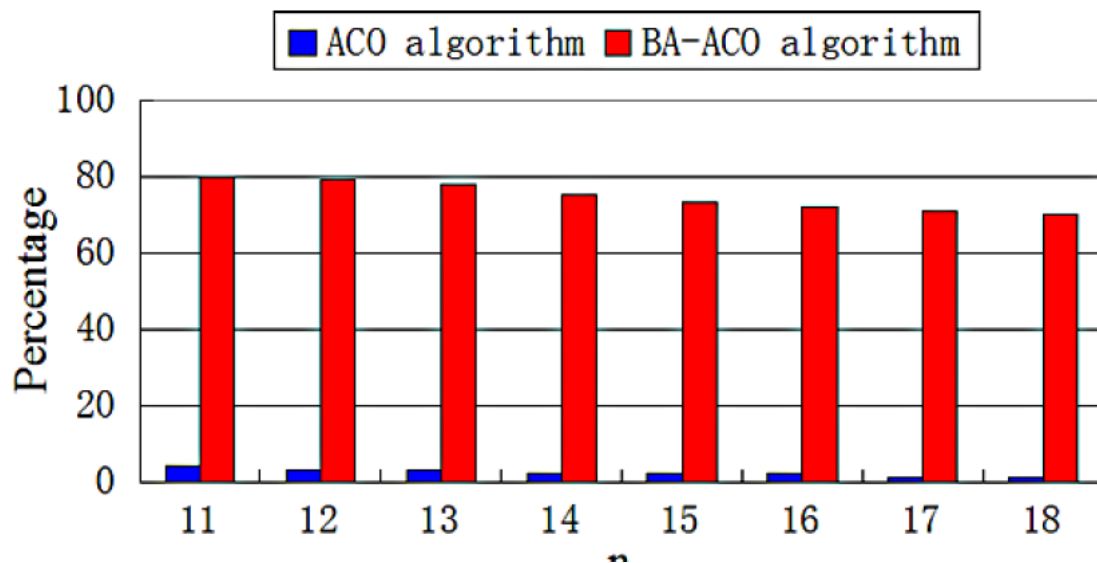


Figure 2.11 Comparison of the percentage of trials reaching the optimal (Chen, Bolun, Chen, Ling, & Sun, Haiying, 2014, p. 59)

In a paper entitled “Ants can Learn from the Opposite” (Rojas-Morales, Riff, & Montero, 2016) introduce negative pheromones at the start of the learning process of the ACO to identify paths which should not be visited. The second step in the process removed the elements marked with negative pheromone from the search space. This research used the Ant Solver Algorithm proposed in a 2002 paper “Ants can solve constraint satisfaction problems” (Solnon, 2002) which is a Max Min system. The addition of the negative pheromone improved the performance of Ant Solver on the CSP problems it was applied to. The problems were TSP and QAP.

The concept of crowding was discussed in relation to ACO by (Czaczkes, 2014). This system tracks the number of ants on a path and determines if there are too many. The excess are allocated to other paths. The study implemented two mechanisms

Crowding Negative Feedback CNF and Pheromone Negative Feedback PNF. The findings are at odds with most of the research in that it found Pheromone Negative Feedback did not improve performance but Crowding Feedback did improve results. The crowding mechanism is similar to the behaviour of Max Min systems which achieve the same results by setting maximum pheromones. It is also less elegant in that it records the number of ants on a path in a separate data structure as opposed to Min-Max systems which use the concepts of ACO to achieve the same effect.

The combining of pheromone structures to improve performance when solving the Car Sequencing problem was proposed by (Solnon, 2008). This combines two pheromone structures the first structure identifies good car sequences the second identifies critical cars. The results showed that this was a competitive result when compared to IDWalk and VLFS algorithms. This work differs from (Iredi, Merkle, & Middendorf, 2001) and the other examples discussed in that both the pheromone values here are positive pheromones which encourage courses of action as opposed to discouraging courses of action.

As a method of solving search bias inherent to ACO the use of negative pheromone is clearly a well-researched method for achieving this. The research in this area is ongoing and shows significant results. The use of positive pheromone structures also improves performance.

2.3.2 Exclusivity / Elitist Solutions

Another approach to the problem of search bias is the use of an exclusivity policy. This began with (Stützle & Hoos, 2000) when the Max Min ant system was developed. The principle is that each ant builds its solution and the ant that builds the best solution lays the pheromone. This system was tested on the TSP and the QAP. It showed better performance for the TSP than the original ant system. At the time, it was the best ACO for the QAP.

In their paper “Towards the Development of Self-ant Systems” (Tavares & Pereira, 2011) used an exclusive strategy to build a prototype generic ACO. Most ACO are limited to specific problem domains and require tweaking to work in other areas. The

self ant system proposed here used the QAP to test an exclusive strategy allowing ants to determine what the specifics of the QAP were. This showed promise in the development of a Generic ant system to solve QAP.

The routing of traffic using an ACO was explored by (Doolan & Muntean, 2014). This is certainly an unusual usage for ACO as the metaphor is based on ants following the same paths. The solution to this problem was to use time as a factor in the pheromone laying. Ants would only lay pheromone on particular roads at optimal times. Only the best ants laid pheromone for each cycle. This resulted in a 19% increase in performance when compared to the best variant of the Dynamic Navigation Algorithm DNA1 as can be seen in Figure 2.12.

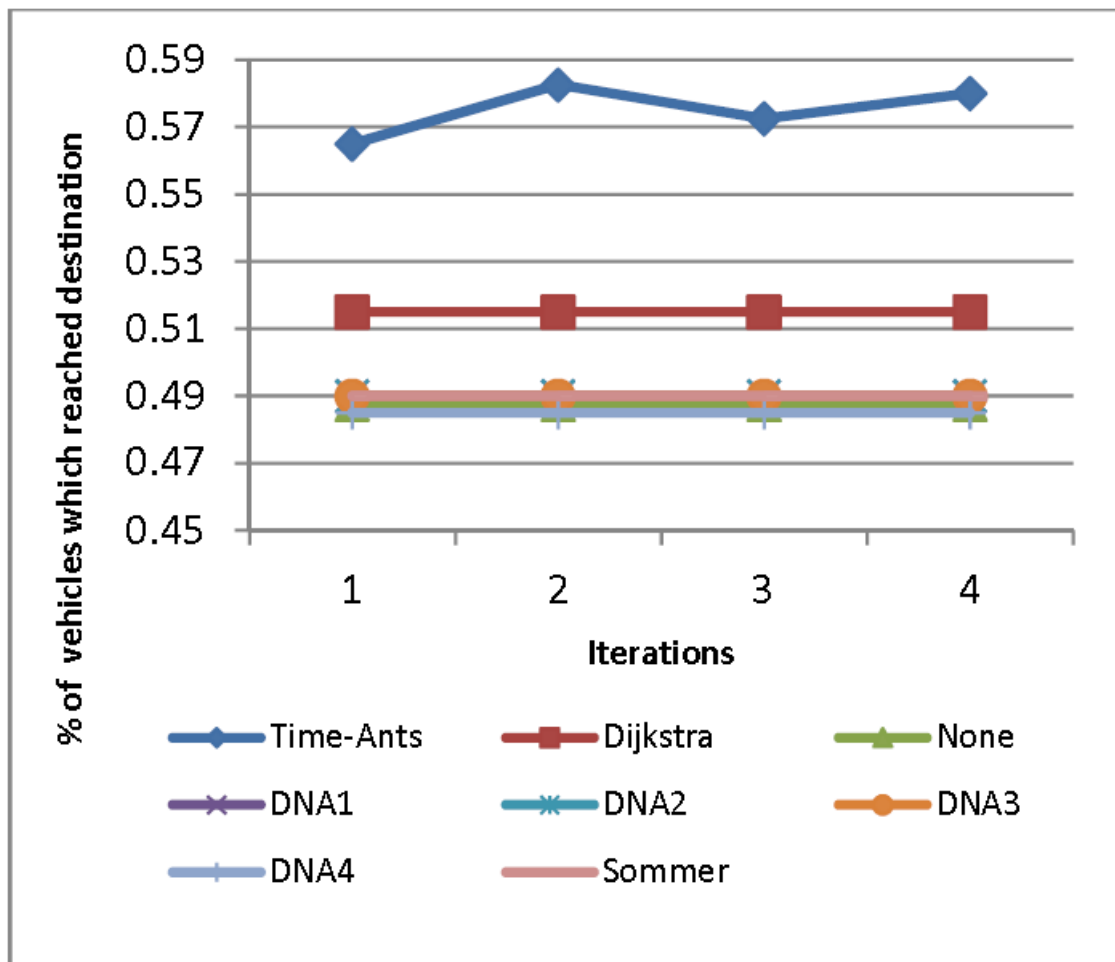


Figure 2.12 Percentage of vehicles which reached destination during simulation (Doolan & Muntean, 2014, p. 955)

An elitist coefficient was added by (MadadyarAdeh & Bagherzadeh, 2011) to an ACO to solve the Grid Scheduling Problem. The elitist strategy is similar to Max Min systems however the pheromone does not have a Max value but is initialised by a

deterministic algorithm which examines the grid to determine the initial values for the pheromone. The elitist coefficient is then used to determine which ant lays pheromone.

A combination of an elitist strategy and reduced candidate list was used by (Karmakar, Mitra, Dey, Chakraborty, & Nayak, 2016) to improve the performance of solutions to the TSP. The solution involved the breaking down of the number of cities into smaller candidate lists which ants would choose from once the list of candidates had been exhausted the ants would then move to the next set of candidates. Combined with an elitist strategy where the ant with the shortest path laid pheromone this improved the performance of the algorithm.

The key difference between this and other approaches is that the candidate lists are dynamic and will change as the algorithm progresses. In other elitist ACO the candidate list is constructed at the start and involves all the cities in the TSP.

As a method for improving ACO performance and preventing the search bias elitist systems have their supporters and have been shown to produce competitive results. The algorithm used in this project is an elitist Max Min system.

2.3.3 Combining ACO and Evolutionary/Genetic Algorithms (EA/GA)

Combining of ACO with other algorithms as a strategy for setting the initial pheromone values is not new. As mentioned in the previous section (MadadyarAdeh & Bagherzadeh, 2011) used this approach with a determinist algorithm. This section will focus on the combination of ACO and EA/GA.

In their proposed Best Worst Ant System (Cordon, Viana, Herrera, & Moreno, 2000) adopt an evolutionary approach to the ACO system. It incorporated three factors:

1. The global best and the current worst ant were used to create positive and negative updates.
2. If the pheromone value is too high for some nodes it is re-set to a lower value. This prevents all the ants going down the same route.

3. Mutation of the pheromone structure, this concept is unique to this particular type of ACO and is borrowed directly from Evolutionary algorithms.

This approach combines EA and ACO and pheromone re-set behaves similarly to the Min Max system. Given the approach it could be considered both a Min Max system and hybrid ACO/EA system.

Fusing of the Genetic Algorithm and a ACO to optimise partner selection for a virtual enterprise was proposed by (Yao, Liu, & Wang, 2008). The research was exploratory and used a small dataset as proof of concept. The algorithm worked in two stages. The GA would find good global solutions. Once it had found optimal solutions initial pheromone values were set on the most promising paths. The ACO would then take over to find the local optimal solution. The paper acknowledged that further research was needed and a larger dataset. Despite this the work was interesting in that it modelled a business problem which is not usually modelled using ACO business partner selection has not been subject to this form of analysis in the same way as other problems like TSP or manufacturing.

Following on from this (Yao, Pan, & Lai, 2009) improved this process they added a full Max Min system and used the GA/EA to set the Max and Min values of pheromone. This improved the solution compared to the GA, and ACA algorithms. The superior performance of HGAACA can be seen in Figure 2.13.

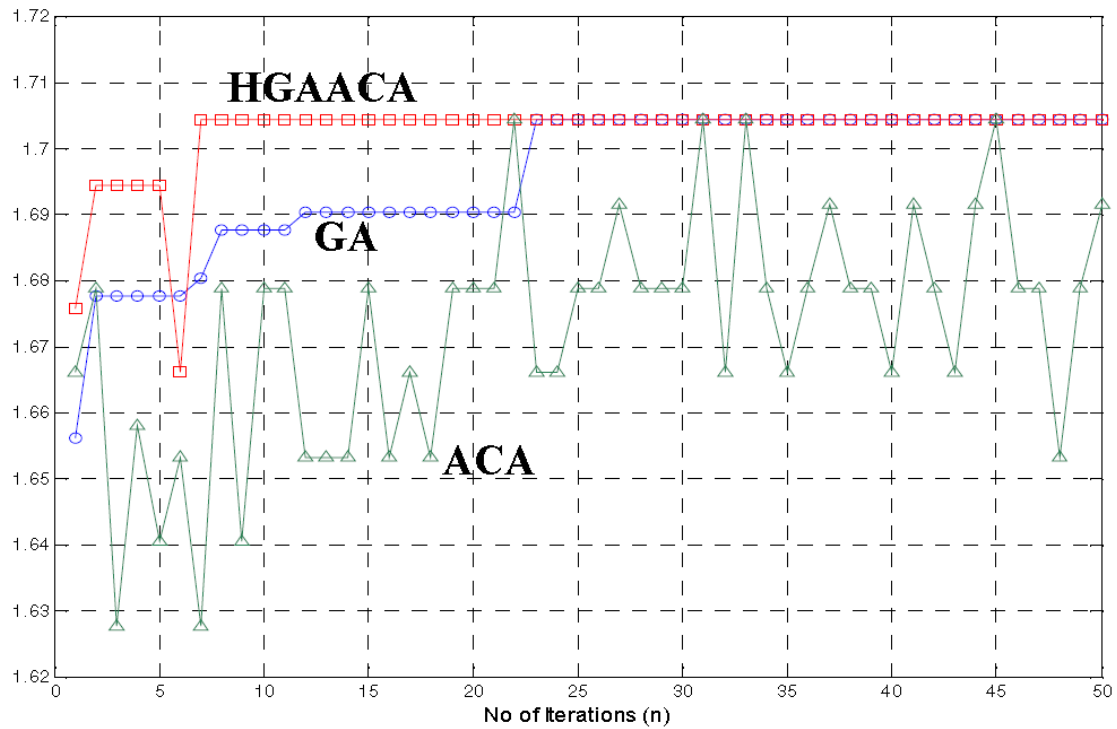


Figure 2.13 A Comparison of Optimal Process with GAACA & HGAACA (Yao, Pan, & Lai, 2009, p. 246)

A further fused ACO/GA algorithm was proposed by (Zhao, Yao, Luan, & Song, 2016) this worked on the same principles as outlined by (Yao, Liu, & Wang, 2008) and (Yao, Pan, & Lai, 2009). The GA would work on the global problem until it reached a good global solution. The global solution was the point at which the GA could not gain any improvements after a number of generations. The pheromone values would be then set and the ACO would take over to solve the local problems arriving at an optimal solution. When applied to the supplier selection problem this approach showed a time improvement over ACO or GA used separately.

2.3.4 Combining ACO with Other Metaheuristics

In a paper entitled “Ant colony optimization combined with tabu search for the job shop scheduling problem” (Huang & Liao, 2008) combined an ACO with tabu search mechanism in an attempt to improve performance. A global pheromone queue was created to guide the Tabu search and the research showed that results were competitive with a standard Max Min ACO.

The ACO carried out local search and updated its local search pheromone. Once this was completed the global pheromone was then updated. The global pheromone queue

then guided the tabu search which maintained the tabu list and updated local pheromone if the ACO was becoming trapped. This was detected by comparing the results of the ACO during its recent cycles and if it was not improving the Tabu search mechanism was employed.

A hybrid ACO and Simulated Annealing algorithm was developed by (Niknam, Firouzi, & Nayeripour, 2008) this outperformed the standard ACO, SA or k-means clustering algorithms (MacQueen, 1967) run separately. The combination worked by generating a series of small ant colonies and placing them on separate areas of the search graph. The SA is used to do this. The Ants find the best local optima and the SA moves them to new positions. This process continues until a convergence is reached.

Based on the work done by (Niknam, Firouzi, & Nayeripour, 2008) an ACO was combined with FAPSO (fuzzy adaptive particle swarm optimization) and k-means algorithm. It was called FAPSO-ACO-K (Niknam & Amiri, 2010). The k-means cluster mechanism was used to combine data in clusters before searching by the ACO/PSO. This algorithm was benchmarked against the Simulated Annealing ACO combination. The results showed clear improvements. Given the modifications made to this system it is arguable that this is no longer an ACO.

2.3.5 Back-Tracking Ants for Solving Search Bias

As discussed previously (Hsin, Chang, & Wu, 2013) developed an ACO for Network-on-chip (NoC) environment. The innovation with this system is that it added a backtracking component which they referred to as backward ants. The process worked in conjunction with standard network routing tables and it maintained a “rooting table” of pheromone values. If the value got too high this indicated congestion and the ants were back-tracked to a less congested path.

The mechanism for doing this is the network packet header which contains an ant index as an ant passes through a node the pheromone table is updated. This process has elements in-common with Max Min systems but there is no minimum pheromone value.

It is debateable whether this can be considered an ant system or not as it uses the odd even turn for adaptive routing algorithm (Chiu, 2000) to determine ant behaviour not just the pheromone values.

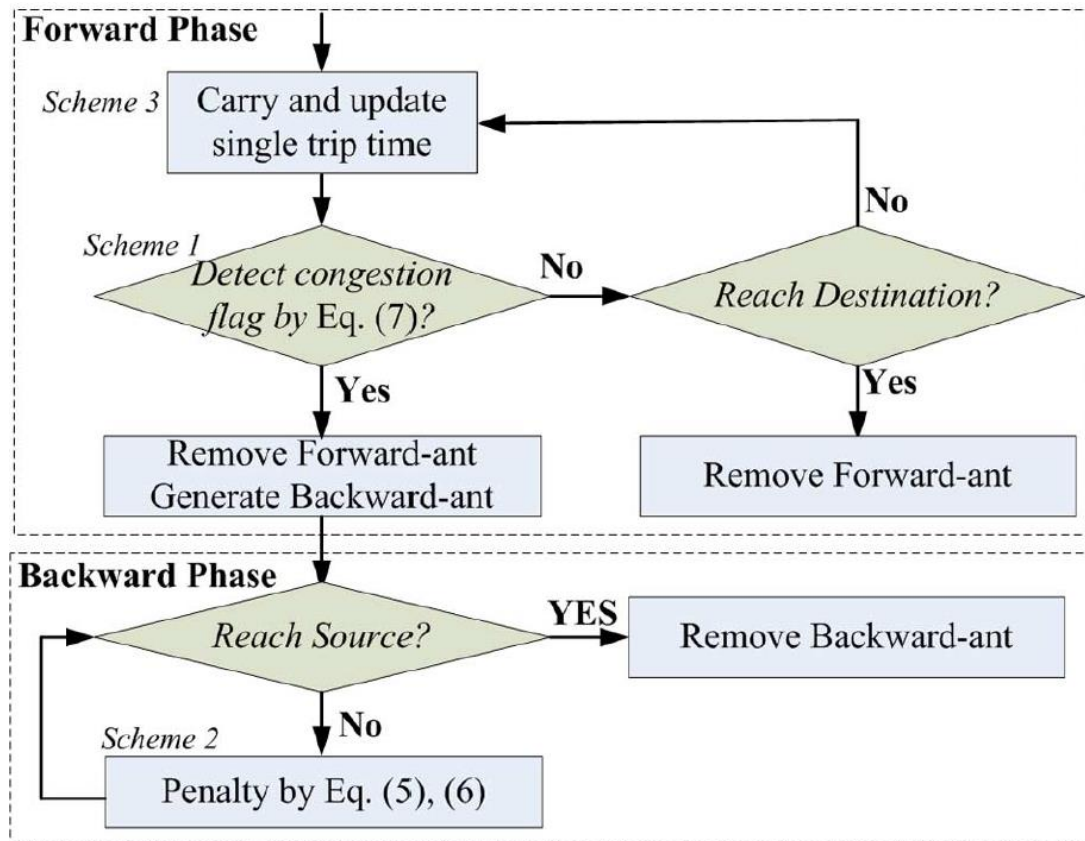


Figure 2.14 The flow of foreword and backword ant system (Hsin, Chang, & Wu, 2013, p. 48)

2.4 The Work of Christine Solnon Explained

The work of Christine Solnon is the work on which this project is based it originated as two papers one on Solving permutation constraint satisfaction problems with artificial ants, (Solnon, 2000) and the second on combining pheromone structures “Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization” (Solnon, 2008) and was further described in her book “Ant Colony Optimization and Constraint Programming” (Solnon, 2010). This work is explained here.

The paper proposed dual pheromone structures to solve the car sequencing problem using a greedy randomised local search (Gottlieb, Puchta, & Solnon, 2003). The

algorithm for the greedy randomised local search can be seen in Figure 2.15. Starting with an empty sequence the cars are added to the end of the sequence until all cars have been sequenced. Line 4 of the algorithm ensures that cars with the smallest number of violations are added at each step. The greedy approach was then combined with two pheromone structures.

The first pheromone structure identified good car sequences. This was achieved by finding permutations of the car sequencing problem (C, O, p, q, r) where no constraints were violated. A classical Max Min ant system (Stützle & Hoos, 2000). The pheromone trails were first set to the T_{\max} limit during a cycle each ant constructs a sequence of cars then the pheromone trails are updated. The algorithm continues until an ant has found a solution or the maximum number of cycles has been found. This is similar to the elitist coefficient structure proposed by (MadadyarAdeh & Bagherzadeh, 2011) when they were working on an improved ant algorithm for the grid scheduling problem using biased initial ants. Both of these systems used a T_{\min} value in the case of (MadadyarAdeh & Bagherzadeh, 2011) it was set by a deterministic function. Solnon does not use the term elitist coefficient however the behaviour of the first pheromone structure is similar.

Input: an instance (C, O, p, q, r) of the car sequencing problem
a transition probability function $p : C \times \mathcal{P}(C) \times \Pi_C \rightarrow]0; 1]$
Output: a sequence π that contains each car of C once

```

1-  $\pi \leftarrow \langle \rangle$ 
2- while  $|\pi| \leq |C|$  do
3-   let  $C-\pi$  denote the set of cars of  $C$  that are not yet sequenced in  $\pi$ 
4-    $cand \leftarrow \{c_k \in C-\pi \mid \forall c_j \in C-\pi, cost(\pi \cdot \langle c_k \rangle) \leq cost(\pi \cdot \langle c_j \rangle) \text{ and}$ 
5-      $(\forall o_i \in O, r_{ki} = r_{ji}) \Rightarrow (k \leq j) \}$ 
6-   choose  $c_i \in cand$  w.r.t. probability  $p(c_i, cand, \pi)$ 
7-    $\pi \leftarrow \pi \cdot \langle c_i \rangle$ 
8- end while
9- return  $\pi$ 

```

Figure 2.15 Greedy Ransomised Car Sequencing Algorithm (Solnon, 2008, p. 1046)

The second pheromone structure lays pheromone on critical cars these are cars which have the highest number of constraints and are hard to sequence. This structure is not managed by a Max Min system as it is necessary to find critical cars quickly and the

Max-Min system is designed to increase the search space which would slow the algorithm down. A T_{\min} value was used to prevent the probability of choosing a car from becoming null. As car sequencing has classes of cars which require the same options the second pheromone structure uses this mechanism to identify the cars more efficiently.

The algorithm can be run using the single pheromone structure to build cars or it can be run with both in which case the first pheromone will have identified the best sequences and the second will identify the critical cars and ensure that they are built first.

The results of this work have shown that the dual pheromone structure makes it possible to obtain competitive results on the car sequencing problem. The algorithm was able to solve many instances much more quickly than the VFSL algorithm. VFSL was the local search based algorithm that won the ROADEF 2005 challenge (Solnon, Cung, Nguyen, & Artigues, 2008). However, on the largest instances for the car sequencing problem and for the longer time limits the dual pheromone approach is outperformed by VFSL.

2.5 Gaps and Limitation of Literature

From the types of problem that can be solved with ACO and the on-going research into the area it is clear that this approach has merits and real-world application. The literature shows an on-going need to improve the ACO algorithm to avoid the search bias inherent to the algorithm and the performance problems when applied to larger datasets.

These approaches include:

1. Multiple Pheromone Structures
2. Exclusive / Elitist Strategies
3. Combining ACO with Evolutionary / Genetic Algorithms
4. Combining ACO with Other Metaheuristics
5. Back-Tracking Ants for Solving Search Bias

What they all have in common is that they focus exclusively on the pheromone structure and ignore the ants which lay it. There are also inherent problems with them. In an attempt to develop a non-hybrid ACO (Krynicky, Houle, & Jaen, 2015) pointed out that most of the combinations of ACO and EA/GA suffer from efficiency problems or are too specific and can only solve a single type of problem.

In relation to negative pheromone (Robinson, Jackson, Holcombe, & Ratnieks, 2007) show how too much of it can cause the search space to become too large and the algorithm to become inefficient.

From reviewing the literature there are no attempts to look at the ants themselves and determine if they can be given more autonomy to find solutions without simply following pheromone trails.

The closest that any of the approaches listed above has come to this idea is the Self Ant system proposed by (Tavares & Pereira, 2011). This contained the seed of an idea that the ants should be able to discover more about their world and adapt to it.

Continuous Orthogonal Ant Colony (Hu, Zhang, & Li, 2008) also have a step where ants explore sub-sets of the problem but it too relies on pheromone structures and not the behaviour of the ants themselves or the creation of any type of specialist ant to do the exploring.

In the area of Artificial Bee Colonies the idea of an explorer or scout has been implemented. While ABC do not lay pheromone in the same way as ACO they have certain similarities. (Karaboga & Kaya, 2016) uses scouts as part of their approach when training ANFIS a hybrid artificial intelligence algorithm created by combining the learning ability of neural networks and the inference feature of fuzzy logic.

ABC Algorithm (Karaboga, 2005) has had an influence on the concept of different types of ants as outlined in this work. In the ABC approach the artificial bees go through three stages when solving a problem these are:

1. Employee Bee: - The bee is exploiting some solution to the problem.

2. Onlooker Bee: - The bee in this stage is waiting its turn to solve the same problem
3. Scout Bee: - This bee abandons the current solution and tries to find a new path to solve the problem.

The use of Evolutionary Algorithms to set initial pheromone values of the ACO also suggests the question why not have a type of ant to do this continuously throughout the life of the algorithm?

2.6 Research Question

The review of the literature and the questions it poses inspired this research. It is proposed to explore the idea of a type of ant that could prevent the inherent problem of search bias and in so doing improve the performance of the algorithm.

Given the vast amount of problems that are available to test these ideas on it was then necessary to find one which matched the following criteria:

1. A readily available dataset.
2. Good coverage in the literature.
3. An algorithm that could be modified and compared to its unmodified state.
4. The ability to test this without too much computing power.

The car sequencing problem and the work done by Christine Solnon were chosen as it matched these criteria. If a type of Exploring ant was implemented it would have to improve the performance of the algorithm and improve or at least generate results that were as good as the current algorithm.

The question was refined to suit these conditions and the question that this research proposes to answer is.

Can the addition of Persistent Explorer Artificial Ants (PEAA) as opposed to setting Pheromone values alone reduce the number of cycles and detect unsolvable sequences in the Car Sequencing Problem?

Based on the gaps found in the literature and the objectives of this research the next chapter will describe the design of experiments designed to achieve these goals.

3 DESIGN OF EXPERIMENTS AND METHODOLOGY

To answer the question formulated in section 2.6 the following hypothesis were developed.

Null Hypothesis HA₀: PEAA will not statistically Reduce the number of cycles when solving the Car Sequencing problem.

Alternative Hypothesis HA₁: PEAA will Reduce the number of cycles when solving the Car Sequencing problem.

Null Hypothesis HB₀: PEAA will not statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

Alternative Hypothesis HB₁: PEAA will statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

The approach to this research was informed by the methodologies used in similar experiments. It also draws on experience gained from years of development. Test Driven Development has had a large influence on this work. TDD is the process of developing a set of tests before development to help define requirements¹. TDD is a more scientific approach to development and appropriate for this work.

(Derrac, García, Molina, & Herrera, 2011) provide a tutorial on assessing the performance of swarm intelligence algorithms using various metrics. While not all of these are appropriate the Sign test is a useful method for determining the winning algorithm.

The CRISP-DM (CRoss Industry Standard Process for Data Mining) (Wirth & Hipp, 2000) is also used as a reference standard for the overall project methodology. These methodologies are iterative processes and somewhat evolutionary which is appropriate for the subject matter of this research.

¹ <http://www.agilemodeling.com/essays/agileRequirements.htm>

The design of the experiments is explained in detail in the following sections an overview of the process can be seen in Figure 3.1 and is summarised here.

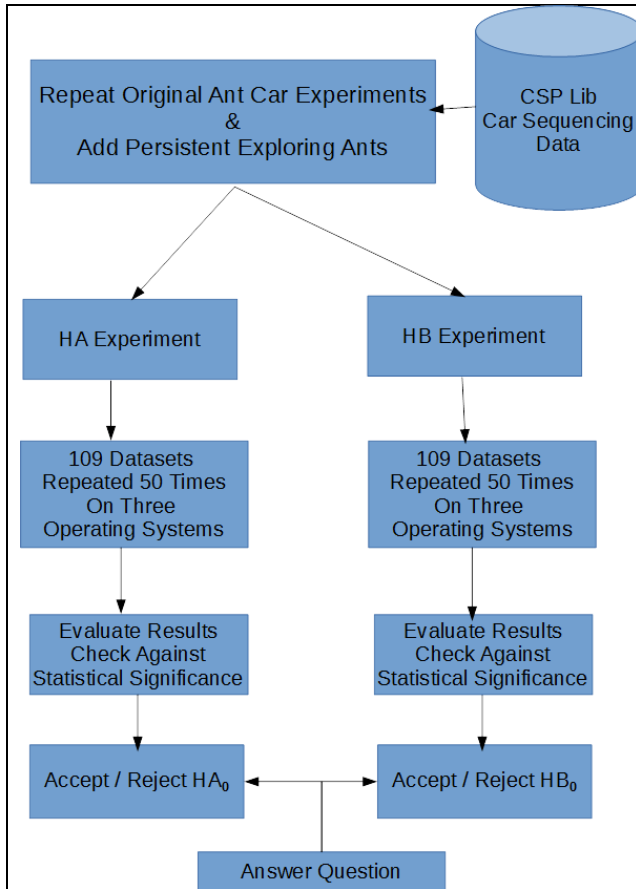


Figure 3.1 Overview of the Experiment

Data Preparation The data used in this experiment is available from the CSPLib resource this is explained in sections 3.1 and 3.2.

Repeat Original Experiment & Add Persistent Explorer Ants The first step will be to repeat the original experiment to ensure that the subsequent experiments are comparable. This is explained in sections 3.3 and 3.4. The addition of explorer ants to explore unexplored paths is explained in section 3.5

HA Experiment Reduce the number of Cycles using Explorer ants. This is explained in section 3.6

HB Experiment Detect the unsolvable Sequences and Terminate. This is explained in section 3.7

Evaluation of Results Based on the type of results this will determine the statistical methods used Section 3.8 explains this.

3.1 Data Preparation

CSPLib (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent, 1999) maintain a data-set which will be used for this problem. It is a series of space delimited text files. The structure of which is displayed in the next section. This data is available on the CSPLib website (Barbara Smith, 1999) and was originally proposed by (Dincbas, Simonis, and Van, 1988). There are 109 separate sets of car sequence data. Each contained in a separate file.

- 9 Files contain 100 cars
- 80 Files contain 200 cars
- 10 Files contain 300 cars
- 10 Files contain 400 cars

The experiments carried out by (Solnon, 2008), (Gottlieb, Puchta, & Solnon, 2003), and (Bruce D. Parrello, Waldo C. Kabat, & L. Wos, 1986) use car sequences of no less than 100 cars. As the number of cars increases the constraints on the problem making the tests more relevant. A set of 10 cars with 5 options is a relatively easy problem to solve. As the number grows the algorithm has to manage more constraints and prevent violations. For this reason data sets of 100 ,200, 300 and 400 cars are used.

3.1.1 Car Sequencing Data Explained

The data in this section is used to illustrate how the car sequencing data is structured it will not be used in the experiment as 10 cars is simply not useful for testing. The experiments carried out by (Solnon, 2008), (Gottlieb, Puchta, & Solnon, 2003), and (Bruce D. Parrello, Waldo C. Kabat, & L. Wos, 1986) use car sequences of no less than 100 cars. A full car sequencing data set can be seen in Appendix B.

	1	2	3	4	5	6	7
Line 1	10	5	6				
Line 2	1	2	1	2	1		
Line 3	2	3	3	5	5		
Line 4	0	1	1	0	1	1	0
Line 5	1	1	0	0	0	1	0
Line 6	2	2	0	1	0	0	1
Line 7	3	2	0	1	0	1	0
Line 8	4	2	1	0	1	0	0
Line 9	5	2	1	1	0	0	0

Table 3.1 Sample CSPLab as it appears in the file

	1	2	3	4	5	6	7
Line 1	10	5	6				
Line 2			1	2	1	2	1
Line 3			2	3	3	5	5
Line 4	0	1	1	0	1	1	0
Line 5	1	1	0	0	0	1	0
Line 6	2	2	0	1	0	0	1
Line 7	3	2	0	1	0	1	0
Line 8	4	2	1	0	1	0	0
Line 9	5	2	1	1	0	0	0

Table 3.2 Sample CSPLib Data in a more user friendly layout

Table 1 Line 1:- Explained

Col 1 Number of Cars = 10

Col 2 Number of Options = 5

Col 3 Number of Types/Classes = 6. They are numbered 0-5

Table 1 Line 2-3:- Explained

Col 1-5 This is the capacity of the work station to install an option

Capacity 1 of 2

Capacity 2 of 3

Capacity 1 of 3

Capacity 2 of 5

Capacity 1 of 5

For example, car class 3 (**on line 7**) requires installation of option 2 and option 4 (**Line 2 and 3 columns 2 and 4**), and two cars of this class are required (**Line 7, col 2**). The workstation for option 2 can process only two out of every sequence of three cars. The workstation for option 4 has less capacity—two out of every five cars.

Line 4-9:- Explained

Col 1 This is related to the sequencing data example as given above.

This is a valid solution to the car

Number of Types its values go from 0 to 5 (6 Types)

Col 2 This is related to the Number of Cars Row 1 Col 1 a total of the column is 10 the same as the number of cars.

Col 3-7 0 or 1 if a car class has an option the value is 1 otherwise it is 0

Cars	Class	Options					
1	0	1	0	1	1	0	
2	1	0	0	0	1	0	
3	5	0	1	0	0	1	
4	2	0	1	0	1	0	
5	4	1	0	1	0	0	
6	3	1	1	0	0	0	
7	3	1	1	0	0	0	
8	4	1	0	1	0	0	
9	2	0	1	0	1	0	
10	5	0	1	0	0	1	

Table 3.3 Valid solution

3.1.2 The Car Sequencing Graph

As illustrated by (Solnon, 2008) and (Solnon, 2010) the car sequencing problem can be modelled as searching for the best Hamiltonian path in a graph (a Hamiltonian path is a path where all the nodes in the graph are visited once). The nodes in the graph represent the different classes of cars. As there are 10 cars there will be 10 nodes. The order of these classes of cars is governed by the constraints as illustrated in table 3.1 in the previous section.

Figures 3.2 and 3.3 Illustrate a car sequence before and after it has been processed.

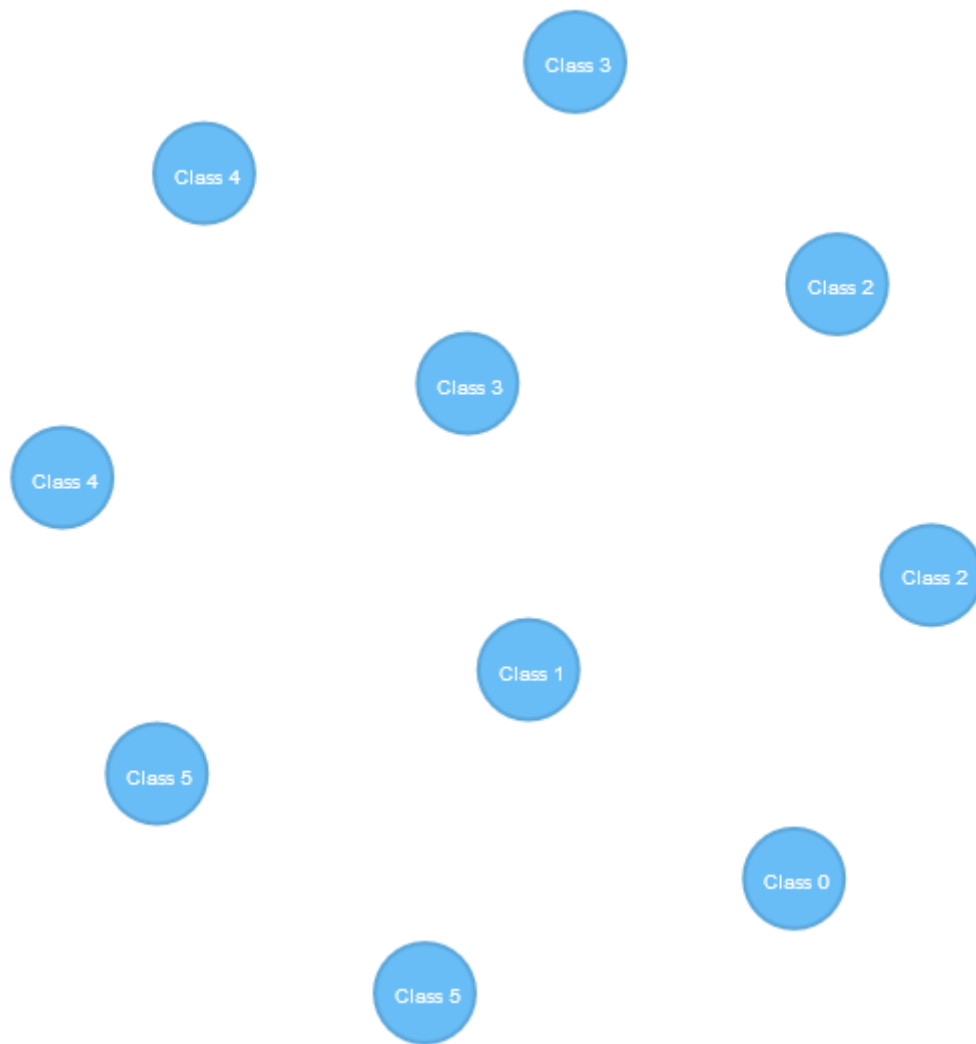


Figure 3.2 Graph of Cars Before Processing

- **gcc** – GNU Compiler Collection (v4.8.4). The Free Software Foundations C compiler.
- **bash** – Linux shell scripting.
- **valgrind** – This is used for memory management and detecting leaks. It will mainly be used for debugging purposes.
- **top** – This is used to monitor Linux/Unix processes and their resource usage.
- **make** – This is the standard GNU make utility and is used for compiling and testing changes to code.
- **Neo4j** – (v3.1.0) This is the graph database which is used to generate and validate the graphs generated as part of the research.

3.3 Experiment Setup

The data originally generated by the Ant Car algorithm was output as a series of name value pairs. This was modified to output the results as a set of CSV files which are more useful. The algorithm was also modified to take a test number to make the tracking of test output more accurate.

A shell script was developed to execute the algorithm and manage the test numbers. The tests were run on three versions of the Linux operating system to strengthen the conclusions derived from these experiments. This can be seen in Table 3.4. As with the original experiments each test was carried out 50 times both (Solnon, 2008) and (Derrac, García, Molina, & Herrera, 2011) also used 50 tests for their research.

Operating System	Kernel Version	CPU	Virtual Machine (Y/N)
Kali Linux	3.18.0_Kali3	Pentium 4 3.20 GHz 32-Bit	N
Ubuntu Linux	3.19.0-25_GENERIC	Intel® Core™ i7-5500U 2.40 GHz 64-Bit	Y
NetBSD Unix	7.0.1_PATCH	AMD64 Opteron 150	N

3.4 Repeat Original Ant Car Experiment

The original experiments carried out with the Ant Car solution will be re-run to measure the performance of the two-pheromone structures as a means of solving the car sequencing problem. This is to ensure that results can be re-produced. This also ensures that problems which might arise when using a different operating system are catered for.

Once it has been established that the Ant Car algorithm can perform as described in the original experiment the modifications to add the Persistent Exploring Ants will be carried out. The results of this experiment will be discussed in section 4.

The performance monitoring tools mentioned previously will be used to measure the performance and as a source of objective data which will be necessary to ensure that no inefficiencies were added when the Explorer Ants were added.

The efficiency of the code is not the primary focus of this research but it is necessary to know how it performs in terms of system resources to ensure that the algorithm does not use up too much resources to make it viable.

3.5 Implement Changes to Add Explorer Ants

The change to implement the Explorer ants will involve creating a data structure which will contain a list of the paths which have been explored by the normal ants. The explorers will be applied to paths which have not yet been processed. The explorers will randomly choose paths to explore from the list of unexplored paths.

If promising sequences are found the non-explorer ants will be re-directed to these nodes and the explorers will continue to explore new paths. The exploration and building of sequences will continue in tandem which will give this algorithm an advantage over the current algorithm. Figure 3.4 illustrates this process.

This algorithm will be implemented as an addition to the existing functions and will be controlled with a command line option.

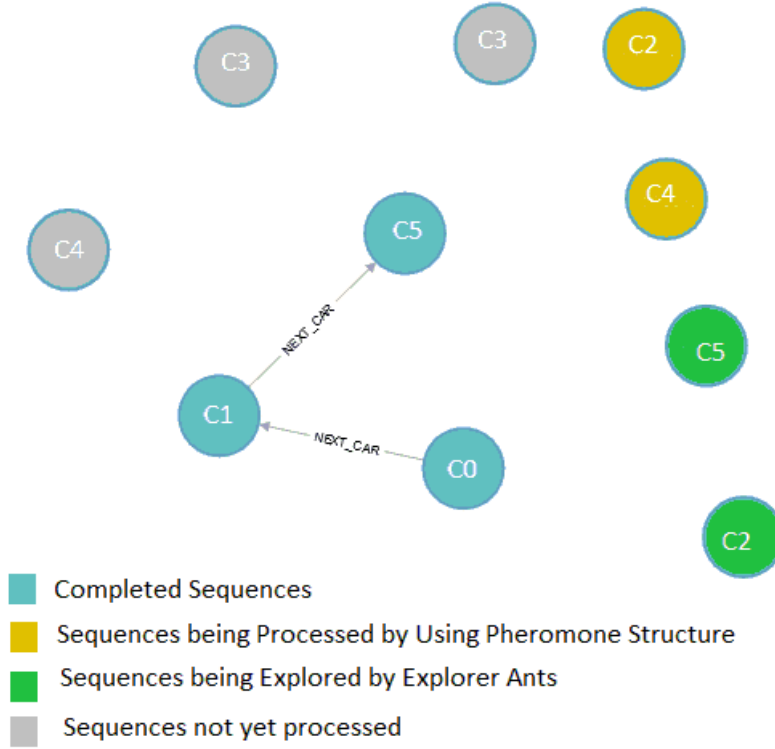


Figure 3.4 Example of PEAA Processing the Sequences

3.6 HA Experiment – Reduce the Number of Cycles

The number of cycles is the number of steps required to solve a sequence it will be first determined by repeating the original experiment as described in section 3.4. Each of the car sequences will be processed using both the unaltered algorithm and the explorer ants.

The results of the unaltered code will be compared to the previous results in section 3.4 to ensure that no errors or inefficiencies have been introduced during the changes. The previously mentioned tools will be used to monitor the code to ensure that the performance is objectively measured.

The experiment will be repeated on Ubuntu, Kali and NetBSD to ensure that the results are reliable and an accurate figure for the number of cycles has been obtained. The sequences will be checked against the results set maintained by the CSPLib to ensure that the sequences are valid. Section 3.9 explains this in more detail. Unsolvables sequences will not be included in this experiment.

3.7 HB Experiment – Unsolvables Car Sequencing Problem

There are unsolvable sequences of cars which cannot be built by any algorithm. The car sequences in block 10/93 and 16/81 are known to have no solution. The purpose of this experiment is to use explorer ants to discover this quickly and terminate the processing before the maximum number of cycles (5,000 by default) is reached. Only Unsolvables sequences will be included in this experiment.

The experiment will be repeated on Kali, Ubuntu and NetBSD to ensure that the results are reliable see Table 3.4.

3.8 Evaluation of Experimental Results

The results are most likely to be normally distributed however (Derrac, García, Molina, & Herrera, 2011) have shown that the comparison of swarm intelligence algorithms can produce non-parametric results. For these reasons, it is necessary to consider two approaches to evaluating the data.

If the data is normally distributed the T-Test will be used to evaluate the data. This is given by the formulas below.

T-Test

Where:

\bar{x}_1 = Mean of first set of values

\bar{x}_2 = Mean of second set of values

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

S_1 = Standard deviation of first set of values

S_2 = Standard deviation of second set of values

n_1 = Total number of values in first set

n_2 = Total number of values in second set.

Standard Deviation

$$S = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}}$$

Where:

x = Values given

\bar{x} = Mean

n = Total number of values.

If the data is not normally distributed then the Sign Test can be used. (Derrac, García, Molina, & Herrera, 2011) recommend this as a method for comparing results for swarm intelligence algorithms. The process involves:

1. Counting the case where each algorithm is the overall winner.
2. (Derrac, García, Molina, & Herrera, 2011, p. 6) “If the number of wins is at least

$$\frac{n}{2} + 1.96 \cdot \frac{\sqrt{n}}{2} \text{ then the algorithm is significantly better”}.$$

Table 4 and 5 of (Derrac, García, Molina, & Herrera, 2011) serves as an example of this process.

3.9 Strengths and Limitations of Solution Design

The strengths of the approach are the use of a comprehensive set of car sequencing data. The 109 datasets are more extensive than that used by (Solnon, 2008) which used 82 datasets. The experiments will also be validated against the best available solutions currently recorded in the literature.

The limitations on this approach are the relatively limited number of platforms on which it is run all of which are Unix or Linux. The hardware was also limited to only 3 processors two intel processors and one AMD.

While the design of this experiment is a good approximation of the Car Sequencing problem it is not as extensive as that proposed by the French car manufacturer Renault. The ROADEF (Solnon, Cung, Nguyen, & Artigues, 2008) car sequencing problem includes additional constraints for paint batching. Renault needed this constraint to cut down on the use of solvents.

Given the time constraints and the different dataset formats used for the ROADEF problem it would not be possible to incorporate it here. This however would be a useful next step in this research if the PEAA approach proves useful.

3.10 Triangulation of Findings with State of the Art Techniques

The CSPLib maintains a list of best solutions for the car sequencing data. These results will be used to determine if the solutions are valid. The best results are those obtained with the least number of violations of the constraints. The sequences have been downloaded and will be automatically compared against the results obtained during the experiments.

4 EXPERIMENTAL RESULTS

This chapter explains the experiment execution. It follows the heading structure of the previous chapter. The execution of the experiments is explained in detail in the following sections an overview of the process can be seen in Figure 4.1 and is summarised here.

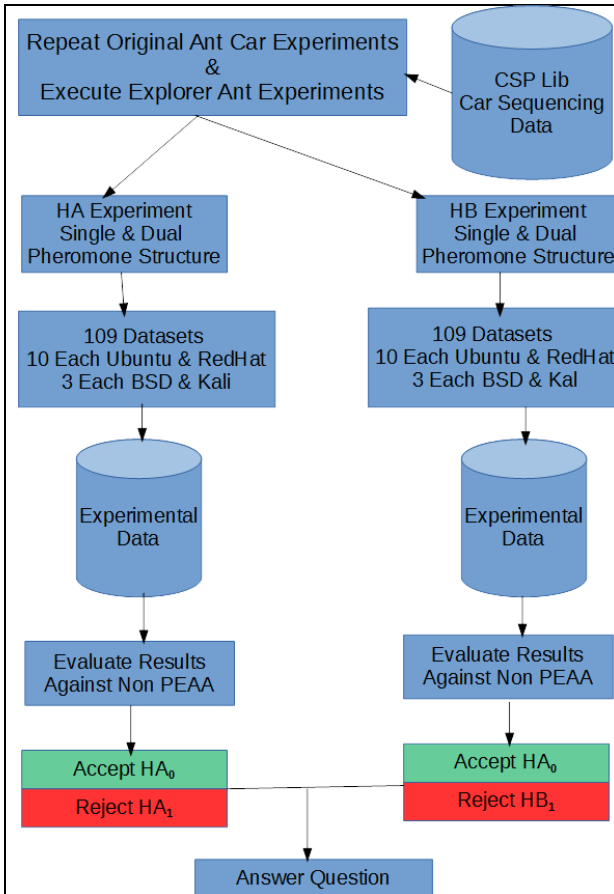


Figure 4.1 Overview of Experiment Execution

Data Preparation The data used in this experiment and the data processing undertaken are explained in sections 4.1

Code Modifications The code was modified to implement the PEAA and to provide reporting facilities using CSV files this is explained in sections 4.2 and 4.3

Repeat Original Experiment & Add Persistent Explorer Ants The Original Ant Car experiment was repeated due to time constraints a 4th test environment was added. This is explained in section 4.4

HA Experiment The PEAA did not reduce the number of cycles they produced car sequences with more violations and these could not be used as the non-explorers were better. This is explained in section 4.6

HB Experiment PEAA did not detect the unsolvable Sequences and Terminate. This is explained in section 4.7

Evaluation of Results Based on the results PEAA do not improve the performance of the algorithm. This is explained in sections 4.8 – 4.9.

4.1 Data Preparation

The Car sequencing datasets were downloaded from the CSP Library as described in the previous section. They were then locally stored on each of the test environments to ensure the fastest possible access to the data.

Each execution of the algorithm generated six data files (the files marked with * were for error checking purposes). The TEST_NO refers to the test number. The PID refers to the Process ID number. The six file types are listed below:

1. **<TEST NO>_<PID>_car_file_data.csv*** – This contained the number of cars, number of options and the classes of the cars.
2. **<TEST NO>_<PID>_cycles.csv** – This contained the number of cycles the algorithm ran for.
3. **<TEST NO>_<PID>_solution.csv** – This contained the final sequence of cars.
4. **<TEST NO>_<PID>_cycles_avg.csv*** – This contained the average cycles.
5. **<TEST NO>_<PID>_parameters.csv*** – This file contained the parameters with which the program was executed.
6. **<TEST NO>_<PID>_violations.csv** – this contained the number of violations for the execution of an algorithm.

A single execution of a test would generate 654 files and five tests would generate 3,270 files. Each test involves testing both the single and dual pheromone structures this will generate a total of 6,540 files. To process these files a second shell script was executed to read the contents and put them into cumulative CSV files.

Four log files were also generated to allow for validation of the test results. Each test execution wrote a single line to the log file. These files were:

- **strategy_1_sequence.csv** : - This contained the results for the single pheromone test with no explorers (a repeat of the first part of the original experiment).
- **strategy_1_sequence_exp.csv** : - This was a test of the PEAA against the first pheromone structure.
- **strategy_2_sequence.csv** : - This contained the results for the dual pheromone test with no explorers (a repeat of the second part of the original experiment)
- **strategy_2_sequence_exp.csv** : - This was a test of the PEAA against the dual pheromone structure.

Once a set of tests was completed the number of files created was verified by the logs to ensure that tests had run correctly. The parameters in the log-files were checked against the parameters for each execution of the algorithm. The ***_car_file_data.csv** and the ***_cycles_avg.csv** were not used as they were not necessary. There were no problems with this part of the experiment and all tests run were valid.

4.2 The Ant Car Software Code

The code was modified to report its results as CSV files rather than simply writing them to the console. This change affected the reporting of all results for tests using the PEAA and those not using the PEAA.

Two new command line arguments were added to allow for more control:

-Z <TEST NO>

-E <NO OF EXPLORER ANTS>

The PEAA were implemented using a standard C data structure and functions were added to process the list of unused options by the non-Explorers. Each non-explorer chooses a sequence based on the pheromone laid by previous non-explorers and there is a random component to this process. The PEAA use the options not chosen by the non-explorers to build alternative sequences of cars.

4.3 Experiment Execution

The tests were executed using a single shell script and run in batches of 5 or 3. Each test involved the 109 datasets. Each test was run for both single and dual pheromone structures. Due to time constraints, a fourth test environment was setup. The test environments are listed in the table below. To ensure that the tests executed as quickly as possible and had the highest priority the GNU nice³ command was used.

Operating System	Kernel Version	CPU	Virtual Machine (Y/N)
Kali Linux	3.18.0_Kali3	Pentium 4 3.20 GHz 32-Bit	N
Ubuntu Linux	3.19.0-25_GENERIC	Intel® Core™ i7-5500U 2.40 GHz 64-Bit	Y
NetBSD Unix	7.0.1_PATCH	AMD64 Opteron 150	N
RedHat Linux 4.4.6-4	2.6.32-042stab123.2	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	N

Table 4.1 Test Environments

4.4 Repeat Original Ant Car Experiment

Repeating the original ant car experiment was necessary to evaluate the effectiveness of the Persistent Explorer Artificial Ants PEAA. This gave a base-line for the performance of the PEAA. After the addition of the reporting functionality (CSV files) this would have slowed down the algorithm slightly as it was writing to files. The sequences of cars built would have been unaffected.

³ https://www.gnu.org/software/coreutils/manual/html_node/nice-invocation.html GNU nice command

Environment	No Times 109 Data Sets Processed	Average Time Single Pheromone	Average Time Dual Pheromone
Kali	3	00:02:32	00:01:17
Ubuntu	10	00:00:03	00:00:03
BSD	3	00:00:54	00:01:02
RedHat Linux	10	00:00:06	00:00:06

Table 4.2 Time Taken Single & Dual Pheromone Structure

4.5 Implement Changes to Add Explorer Ants

The PEAA were then activated using the command line switch. The number of PEAA was set to 10. This was a compromise between efficiency and thoroughly testing the PEAA concept. The tuning of this parameter was guided by testing the 9 datasets containing 100 cars and determining if all the PEAA would build sequences for cars. In all cases each of the PEAA built a sequence. This ensured that in the case of the larger datasets the PEAA would be used.

While 10 PEAA were created by the algorithm at start-up they were only used if they had alternative paths to explore. Otherwise they had no-where to go and nothing to do. As can be seen from Table 4.3 and Figure 4.2 performance was significantly reduced by the addition of PEAA.

Environment	No Times 109 Data Sets Processed	Average Time Single Pheromone	Average Time Dual Pheromone
Kali	3	00:08:01	00:08:07
Ubuntu	10	00:00:51	00:03:11
BSD	3	00:06:26	00:06:58
RedHat Linux	10	00:02:10	00:05:10

Table 4.3 Explorer Ants Time Taken Single & Dual Pheromone Structure

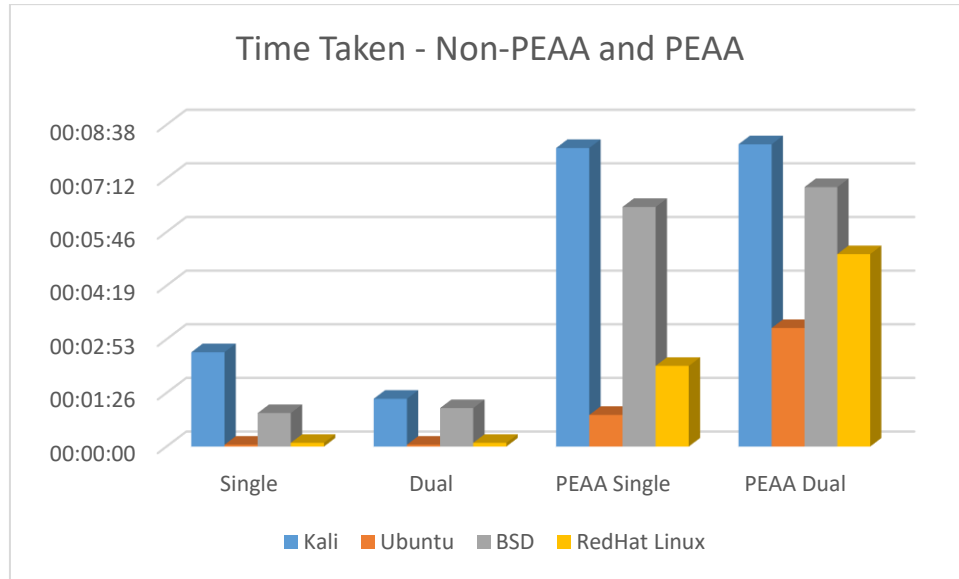


Figure 4.2 Time taken by Non-PEAA and PEAA (PEAA shown on right)

4.6 HA Experiment – Reduce the Number of Cycles

The Violation rates PEAA were consistently greater than those for the first non-explorer ant. The tables below are samples of the violation rates as taken from the Ubuntu test. The full tables are in Appendix A. The significance of these violation rates is that in a greedy Max Min ACO the best option is always selected (the option with the lowest violation rates) this means that the car sequences built by PEAA will not be chosen as the first non-PEAA builds a better option. This occurs for both the single and dual pheromone structures.

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
0	14	17	15	18	21	29	36	32	32	35
0	16	19	21	21	20	21	29	24	22	32
4	14	11	19	24	28	26	22	33	34	35
0	6	9	14	19	20	23	24	18	29	31
2	7	11	19	18	18	18	17	19	25	25
4	7	7	15	20	29	19	20	21	28	32
1	1	7	9	9	9	21	17	33	30	31
0	3	4	4	12	10	11	16	17	25	26
0	5	16	16	18	21	36	39	45	54	63
0	5	11	16	24	29	34	34	41	39	42

Table 4.4 Ubuntu Single Pheromone PEAA Violation Rates

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
4	44	53	94	94	98	107	115	114	119	127
21	47	66	76	73	75	83	89	88	91	94
15	94	123	131	136	156	148	159	155	165	162
25	79	104	121	131	114	140	132	133	139	140
1	65	80	102	98	132	134	127	144	156	169
25	88	135	139	157	166	173	176	183	185	190
1	69	104	111	130	131	142	141	152	165	175
7	92	115	131	139	169	179	174	178	162	176
29	109	155	162	176	180	194	199	206	204	212
17	60	69	80	91	116	111	114	119	123	138

Table 4.5 Ubuntu Dual Pheromone PEAA Violation Rates

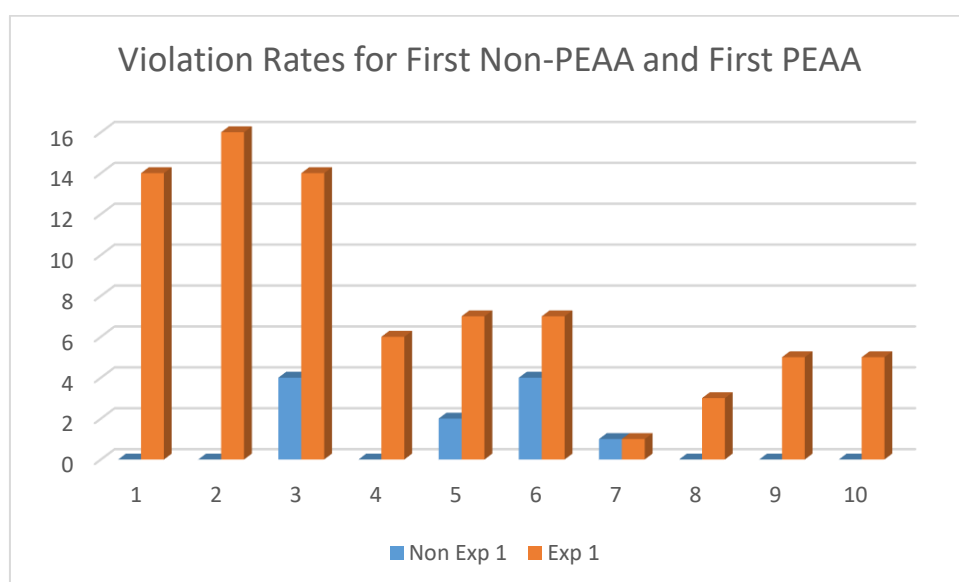


Figure 4.3 Ubuntu Single Pheromone Structure Violation Rates for First Non-PEAA and First PEAA

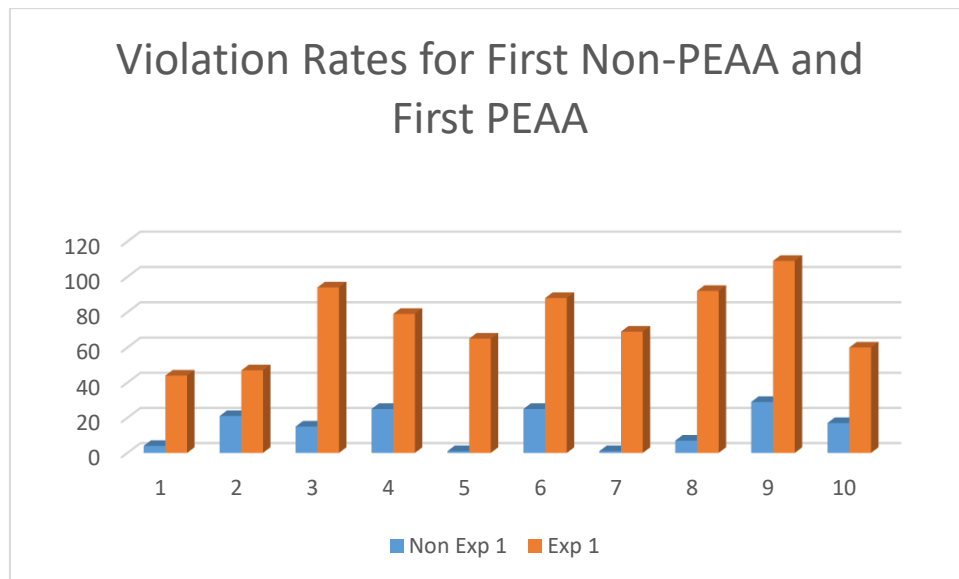


Figure 4.4 Ubuntu Dual Pheromone Structure Violation Rates for First Non-PEAA and First PEAA

Another clear trend in this data is that the violation rate of PEAA rises with each explorer.

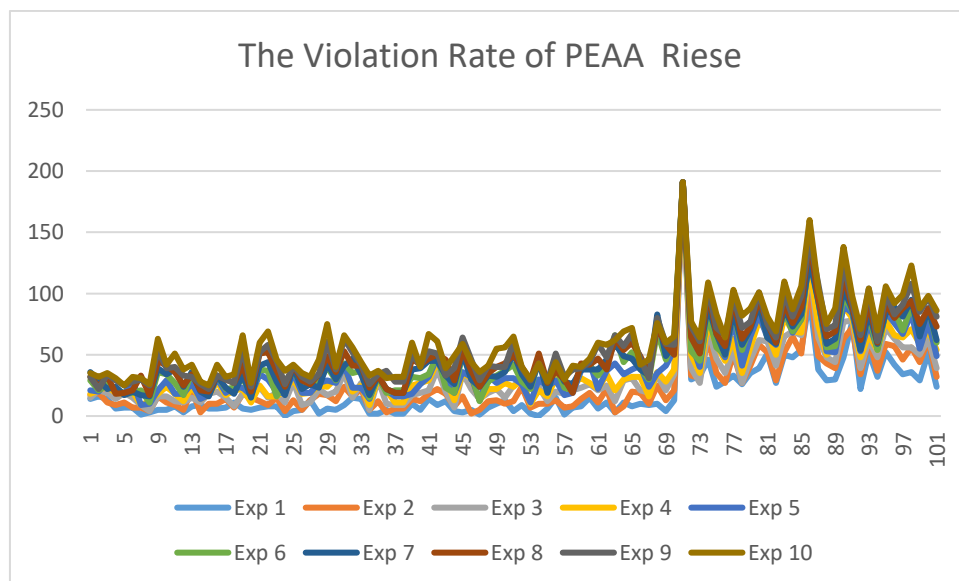


Figure 4.5 Single Pheromone Structure Violation Rate Rises for Every New PEAA

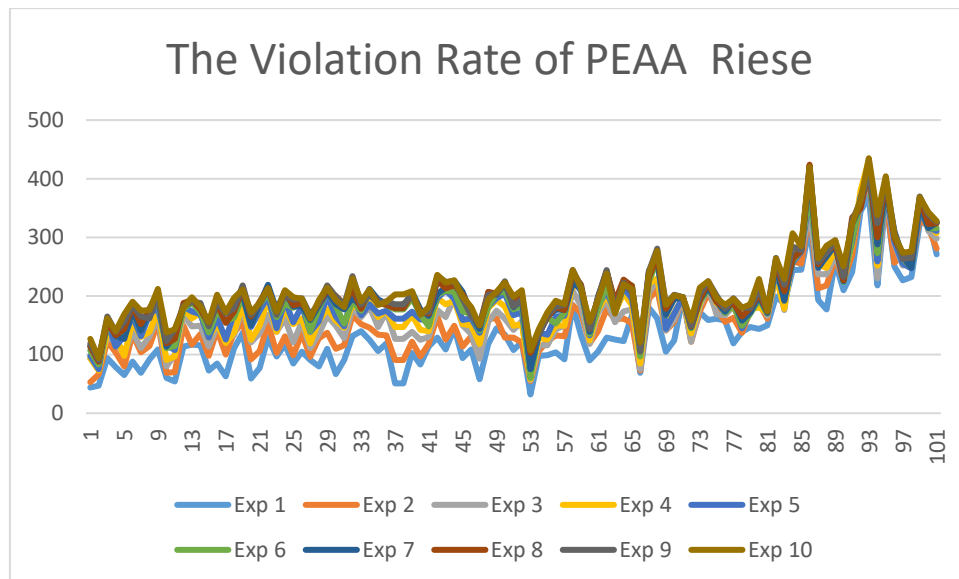


Figure 4.6 Dual Pheromone Structure Violation Rate Rises for Every New PEAA

Once the trend was identified on the Ubuntu environment the results for ten random datasets were selected from the Kali tests (five from the single pheromone test and five from the dual pheromone test) environment and these showed the same trend.

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
4	44	53	94	94	98	107	115	114	119	127
21	47	66	76	73	75	83	89	88	91	94
15	94	123	131	136	156	148	159	155	165	162
25	79	104	121	131	114	140	132	133	139	140
1	65	80	102	98	132	134	127	144	156	169

Table 4.6 Kali Single Pheromone PEAA Violation Rates

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
0	14	17	15	18	21	29	36	32	32	35
0	16	19	21	21	20	21	29	24	22	32
4	14	11	19	24	28	26	22	33	34	35
0	6	9	14	19	20	23	24	18	29	31
2	7	11	19	18	18	18	17	19	25	25

Table 4.7 Kali Dual Pheromone PEAA Violation Rates

This process was repeated for the RedHat and NetBSD test environments and the results showed that the PEAA consistently built worse sequences for these environments too.

4.7 HB Experiment – Unsolvability Car Sequencing Problem

The unsolvable car sequence as represented by datasets pb10-93.txt and pb16-81.txt are known to be unsolvable (Smith, n.d.). As illustrated by the data PEAA build increasingly sub optimal car sequences and as such are not a reliable way of detecting the unsolvable car sequences.

4.8 Evaluation of Experimental Results

In their paper on methods for comparing swarm intelligence algorithms (Derrac, García, Molina, & Herrera, 2011) recommend comparing the number of wins as a method for comparing algorithms quickly. This method is a useful first step as it can clearly show what further analysis is needed.

To determine if the PEAA had outperformed non-PEAA algorithm it had to do four tasks these were:

1. The PEAA had to build car sequences.
2. These car sequences had to have lower violation rates than the non-explorers.
3. The algorithm had to perform with a reduced number of cycles.
4. The final sequences had to have less or equal violations than those built when PEAA were not used.

The data clearly shows that the violation rate of PEAA was consistently higher than that for non-PEAA which ensured that while they built sequences the algorithm would not use them because of their poorer performance. The number of wins for the PEAA was zero due to the higher violation rates of the first PEAA compared to the non-PEAA.

The results of these experiments clearly show:

- The time taken by the algorithm is significantly increased when the PEAA are added.

- PEAA have consistently higher violation rates than the first non-explorer ant. As this is a Min Max ant system (Stützle & Hoos, 2000) the setting of the pheromone values at the start of the algorithm clearly worked for the Car Sequencing problem in allowing it to pick very good candidates at the start.
- PEAA violation rates increase with each new PEAA added.
- They cannot be used to reduce the number of cycles in solving the car sequencing problem.
- They cannot be used in detecting and abandoning the unsolvable problems.

The reason PEAA are out-performed by the Non-PEAA is the selection criteria applied by (Solnon, 2008) when choosing the first car in a sequence is extremely effective whether using the single or dual pheromone structure. As is repeated throughout the literature the ACO are specific problem solvers. When they are designed they are designed to solve a specific problem.

The (Solnon, 2008) algorithm put much greater weighting on the options required for each car and was able to choose very good options. The random element was far less significant than it might have been. The result was that as a non-Explorer ant was building a sequence it left worse options for the PEAA to use and as a result the PEAA results were always going to be sub-optimal.

A subset of the data was tested to demonstrate this. Ten problems were selected and tested five times. The GNU command diff was used to determine if there was any difference between the sequences generated by the first non-PEAA. Table 4.8 shows the results for these tests. It is clear that the random element is not as important as the problem specific elements of the ACO metaheuristic.

Single Pheromone			Dual Pheromone		
Problem	Cars	Differences	Problem	Cars	Differences
pb_16_81.txt	100	0	pb_16_81.txt	100	0
Pb_200_01.txt	200	0	Pb_200_01.txt	200	0
pb_300_01.txt	300	0	pb_300_01.txt	300	0
pb_400_01.txt	400	0	pb_400_01.txt	400	0
pb_400_02.txt	400	0	pb_400_02.txt	400	0

Table 4.8 Test of Randomness Sequences Generated

The tests were then re-run a day later on the BSD environment and were compared with each other using the diff command. They showed the same results. Finally, the tests run on ubuntu were compared with those run on BSD and these tests also showed no difference. This clearly demonstrates that the dual pheromone structure is superior at selecting car sequences and the random element is far less important.

4.9 Strengths and Solution Findings

The strengths of the approach are the use of a comprehensive set of car sequencing data. The experimentation process was shown to be easily adaptable when a further test environment was easily added. The method chosen to rank the algorithms in terms of winners and losers proved effective in determining that the PEAA were not impacting the sequences being built. The reporting facility added to the algorithm allowed for the generation of results which were easily analysed.

The limitation of the approach is that it cannot completely exclude the use of PEAA for some problems however it does show that they are unlikely to be of much use.

It was proposed that the results be triangulated with the list of best known solutions maintained by the CSPLib however, given that the PEAA did not improve the building of any of the car sequences this step is not necessary.

5 DISCUSSION AND CONCLUSIONS

This section summarises the structure and findings of the project. The contributions to the body of knowledge is addressed and recommendations for further research. Figure 5.1 Illustrates this process.

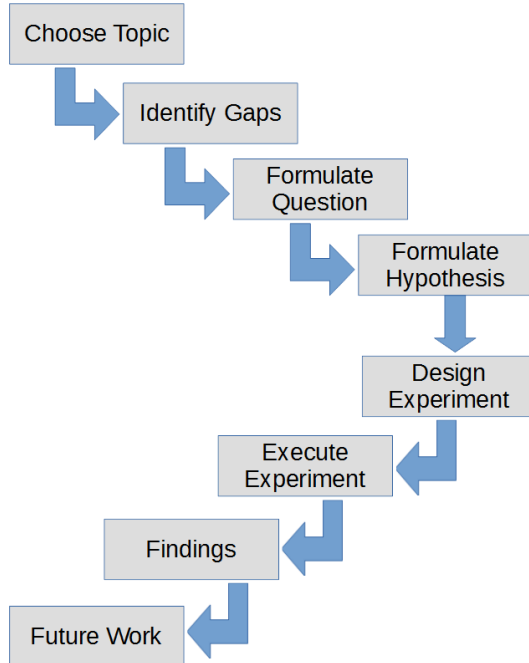


Figure 5.1 Research Process

5.1 Research Overview and Problem Definition

This research was undertaken to determine if a more efficient Ant Colony Optimisation Algorithm could be developed by using Permanent Explorer Artificial Ants PEAA. It was based on work done by (Solnon, 2008) who used a dual pheromone structure to more efficiently solve the Car Sequencing problem as defined by (Parrello, Kabat, & Wos, 1986).

It is clear from the literature that most of the work to improve ACO involved altering or adding pheromone structures. This research investigated if it might be better to use a different type of ant to explore un-explored paths and thus find a better solution. The idea of Permanent Explorer Ants was conceived to implement this solution.

The idea for PEAA was inspired by (Tavares & Pereira, 2011) who developed a model for what they termed a “self-ant system”, where the ants could explore the problem space before deciding how to set pheromone values. Further inspiration came from a

related area of swarm intelligence Bee Colony Algorithms (Karaboga & Kaya, 2016) which use “scouts” to find new solutions.

Once the concept had been formulated a problem area was then necessary to test this on. After many areas such as Image Edge Detection (Agrawal, Kaur, Kaur, & Dhiman, 2012) and closed loop inventory problems as described by (Deng, Li, Guo, & Liu, 2016), were reviewed. It was decided to use the Car Sequencing problem.

The car sequencing problem was chosen as it could be worked on within the time constraints of the project and there was an available data source maintained by the Constraint Programming Library project (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent, 1999). As well as maintaining a data set this project also maintains a list of best solutions which allowed for the further strengthening of the findings of the project as it could be determined not only that the PEAA performed as expected but also that they produced good solutions.

5.2 Experiment Evaluation and Results

The hypotheses were then formed to test the performance of PEAA these were:

Null Hypothesis HA₀: PEAA will not statistically Reduce the number of cycles when solving the Car Sequencing problem.

Alternative Hypothesis HA₁: PEAA will Reduce the number of cycles when solving the Car Sequencing problem.

Null Hypothesis HB₀: PEAA will not statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

Alternative Hypothesis HB₁: PEAA will statistically Detect the unsolvable Car Sequencing Problems and abandon it before the maximum amount of iterations.

Once the hypotheses were formulated two tests were required to determine whether to accept or reject the Alternative hypothesis in both cases. These tests were carried out

on three Linux machines. As the experiments developed it became clear that more processing power was needed and a fourth test environment was added.

The experiments were run twenty six times using 109 datasets the results were analysed and it was clear that the PEAA were not and could not ever out-perform the single or dual pheromone structures as a method of building car sequences. The PEAA consistently build sub-optimal sequences from the start and only worsened as further explorers were added. Continuing with the original plan to run the experiments 50 times would not have changed the results. The testing of the results against the best-known solutions for the Car Sequencing problem was also unnecessary as the results were no different than the original algorithm.

5.3 Contribution to the Body of Knowledge

This research has shown that PEAA are not a good solution for the Car Sequencing problem and it is reasonable to extrapolate that they are not a good solution for similar types of problems such as Quadratic Assignment and the Travelling Salesman. They do not perform in an efficient or effective manner. The time taken to execute the PEAA significantly slows down the algorithm and the reason for this is that the PEAA build suboptimal solutions from the start and worsen as more PEAA are added. Once the non-PEAA have executed there simply are not enough good options left to build a car sequence.

Given the comprehensive dataset and the testing carried out the rejection of **HA₁** and **HB₁** is significant as It shows that the use of a dual pheromone structure is an efficient way of solving the car sequencing problem and that an approach along these lines will be useful in the future.

However, PEAA clearly show that they can consistently detect sub-optimal paths and this is useful for oppositional learning (learning from mistakes). Any future work could consider using PEAA if they wish to find sub-optimal results.

5.4 Future Work

While the PEAA will not improve the performance of ACO when finding optimal solutions there is still room for testing this concept in other areas. Like most approaches to ACO's this project concentrated on limiting the search space for the algorithm to the best solutions. In their paper "Ants can Learn from the Opposite" (Rojas-Morales, Riff, & Montero, 2016) propose a mechanism for using what they term anti-pheromone (pheromone with a negative value) to mark sub-optimal paths. Given that the PEAA cannot produce optimal results they may be useful for identifying sub-optimal results. Currently this oppositional approach to finding solutions is only employed at the start of the process PEAA might be a solution for continuously doing this throughout the search.

A second area of interest for further PEAA research would be to combine them with other Evolutionary Algorithms. Differential Evolutionary algorithms for example (Tang & Zhao, 2010) developed an oppositional Differential Evolutionary algorithm which used oppositional learning. As has been shown above the combination of ACO and EA has led to an improved performance for both algorithms.

In a paper entitled "Ant colony optimization combined with tabu search for the job shop scheduling problem" (Huang & Liao, 2008) combined an ACO with tabu search mechanism to improve performance. As tabu searches allow the choosing of a less favourable solution to escape the local optimal trap PEAA may be a way of conducting a global search to find the least worst global optima.

A final suggestion for future research using PEAA would be to combine the principles of PEAA with Artificial Bee Colony (Karaboga, 2005) and (Karaboga & Kaya, 2016). ABC use a system of explorers and a combination of these with ACO would be an area of many exciting possibilities. At the time of this research there is no known hybrid between an ACO and ABC.

Outside the area of ACO and PEAA the problems of the local optimal is common for all swarm intelligence algorithms and other search algorithms such as k-menas classification. This is a well known problem and the methods for solving it have

mostly involved combining algorithms. These approaches do suggest a number of questions:

1. Is there a general purpose algorithm which could detect when any type of PSO algorithm is stuck at a local optimal for a given problem for example the car sequencing problem?
2. Is there a general purpose algorithm which could detect when any type of PSO algorithm is stuck for any constraint problem regardless of what it is?
3. When a PSO algorithm is stuck at a local optimal is there a universal rule for choosing a sub-optimal solution to escape this for a specific problem for example the car sequencing problem?
4. When a PSO algorithm is stuck at a local optimal is there a universal rule for choosing a sub-optimal solution to escape this for all constraint problems?

BIBLIOGRAPHY

- Aaronson, S. (n.d.). Complexity Zoo. Retrieved 21 June 2017, from https://complexityzoo.uwaterloo.ca/Complexity_Zoo
- Adubi, S. A., & Misra, S. (2014). A comparative study on the ant colony optimization algorithms. In *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)* (pp. 1–4). <https://doi.org/10.1109/ICECCO.2014.6997567>
- Agrawal, P., Kaur, S., Kaur, H., & Dhiman, A. (2012). Analysis and Synthesis of an Ant Colony Optimization Technique for Image Edge Detection. In *2012 International Conference on Computing Sciences* (pp. 127–131). <https://doi.org/10.1109/ICCS.2012.14>
- Alauddin, M. (2016). Mosquito flying optimization (MFO). In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)* (pp. 79–84). <https://doi.org/10.1109/ICEEOT.2016.7754783>
- Amudhavel, J., Kumar, K. P., Jayachandrameena, C., Abinaya, Shanmugapriya, Jaiganesh, S., ... Vengattaraman, T. (2015). An robust recursive ant colony optimization strategy in VANET for accident avoidance (RACO-VANET). In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (pp. 1–6). <https://doi.org/10.1109/ICCPCT.2015.7159383>
- Biggs, N. (1986). THE TRAVELING SALESMAN PROBLEM A Guided Tour of Combinatorial Optimization. *Bulletin of the London Mathematical Society*, 18(5), 514–515. <https://doi.org/10.1112/blms/18.5.514>
- Bruce D. Parrello, Waldo C. Kabat, & L. Wos. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1), 1–42. <https://doi.org/10.1007/BF00246021>
- Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). An improved Ant System algorithm for the Vehicle Routing Problem. *Annals of Operations Research*, 89(0), 319–328. <https://doi.org/10.1023/A:1018940026670>
- Bullnheimer, Bernd, Hartl, R. F., & Strauß, C. (1997). A new rank based version of the Ant System. A computational study. [Paper]. Retrieved 9 June 2017, from <http://epub.wu.ac.at/616/>
- Bullnheimer, Bernd, Hartl, R. F., & Strauss, C. (1999). Applying the ant system to the vehicle routing problem. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, 109–120.
- Chen, Bolun, Chen, Ling, & Sun, Haiying. (2014). A method for avoiding the searching bias in ACO deceptive problem solving. *Web Intelligence & Agent Systems*, 12(1), 51–62. <https://doi.org/10.3233/WIA-140285>
- Chen, Y., & Sun, H. (2008). Convergence of ant colony optimization on first-order deceptive systems. In *2008 IEEE International Conference on Granular Computing* (pp. 158–163). <https://doi.org/10.1109/GRC.2008.4664719>
- Chew, T., David, J., Nguyen, A., & Tourbier, Y. (1992). Solving constraint satisfaction problems with simulated annealing: The car sequencing problem revisited. In

- Chiu, G.-M. (2000). The odd-even turn model for adaptive routing. *IEEE Transactions on Parallel and Distributed Systems*, 11(7), 729–738.
<https://doi.org/10.1109/71.877831>
- Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent. (1999, January 1). CSPLib: A problem library for constraints. Retrieved 11 September 2016, from <http://www.csplib.org/>
- Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151–158). New York, NY, USA: ACM. <https://doi.org/10.1145/800157.805047>
- Cordon, O., Viana, I. F. de, Herrera, F., & Moreno, L. (2000). *A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System*.
- Czaczkes, T. J. (2014). How to not get stuck—Negative feedback due to crowding maintains flexibility in ant foraging. *Journal of Theoretical Biology*, 360, 172–180. <https://doi.org/10.1016/j.jtbi.2014.07.005>
- 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, S., Li, Y., Guo, H., & Liu, B. (2016). Solving a Closed-Loop Location-Inventory-Routing Problem with Mixed Quality Defects Returns in E-Commerce by Hybrid Ant Colony Optimization Algorithm. *Discrete Dynamics in Nature and Society*, 2016, e6467812. <https://doi.org/10.1155/2016/6467812>
- Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1), 3–18. <https://doi.org/10.1016/j.swevo.2011.02.002>
- Di Caro, G., & Dorigo, M. (1998). AntNet: Distributed Stigmergetic Control for Communications Networks, 9, 317–365. <https://doi.org/10.1613/jair.530>
- Dincbas, M., Simonis, H., & Hentenryck, P. V. (1988). Solving the Car Sequencing Problem in Constraint Logic Programming (pp. 290–295). Presented at the In European Conference on Artificial Intelligence (ECAI-88. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.177.1141>
- Doolan, R., & Muntean, G. M. (2014). Time-Ants: An innovative temporal and spatial ant-based vehicular Routing Mechanism. In *2014 IEEE Intelligent Vehicles Symposium Proceedings* (pp. 951–956).
<https://doi.org/10.1109/IVS.2014.6856444>
- 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., Birattari, M., & Stutzle, T. (2006). Ant colony optimization-Artificial ants as a computational intelligence Technique. *IEEE Computational Intelligence Magazine*, 11, 28–39.
- Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53–66. <https://doi.org/10.1109/4235.585892>

- Dorigo, Marco, & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *Biosystems*, 43(2), 73–81. [https://doi.org/10.1016/S0303-2647\(97\)01708-5](https://doi.org/10.1016/S0303-2647(97)01708-5)
- Fliedner, M., & Boysen, N. (2008). Solving the car sequencing problem via Branch & Bound. *European Journal of Operational Research*, 191(3), 1023–1042. <https://doi.org/10.1016/j.ejor.2007.04.045>
- Flood, M. M. (1956). The Traveling-Salesman Problem. *Operations Research*, 4(1), 61–75. <https://doi.org/10.1287/opre.4.1.61>
- Gagné, C., Gravel, M., & Price, W. L. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3), 1427–1448. <https://doi.org/10.1016/j.ejor.2005.02.063>
- Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5), 533–549. [https://doi.org/10.1016/0305-0548\(86\)90048-1](https://doi.org/10.1016/0305-0548(86)90048-1)
- Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, 1(3), 190–206. <https://doi.org/10.1287/ijoc.1.3.190>
- Gnanasundari, P. (2015). Scheduling and routing algorithm for co-operative wireless networks. In *2015 International Conference on Soft-Computing and Networks Security (ICSNS)* (pp. 1–7). <https://doi.org/10.1109/ICSNS.2015.7292437>
- Gottlieb, J., Puchta, M., & Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In *Workshops on Applications of Evolutionary Computation* (pp. 246–257). Springer.
- Gupta, D. K., Arora, Y., Singh, U. K., & Gupta, J. P. (2012). Recursive Ant Colony Optimization for estimation of parameters of a function. In *2012 1st International Conference on Recent Advances in Information Technology (RAIT)* (pp. 448–454). <https://doi.org/10.1109/RAIT.2012.6194620>
- Hart, J. P., & Shogan, A. W. (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3), 107–114. [https://doi.org/10.1016/0167-6377\(87\)90021-6](https://doi.org/10.1016/0167-6377(87)90021-6)
- Holland, J. H. (1975). Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*.
- Hsin, H. K., Chang, E. J., & Wu, A. Y. (2013). Implementation of ACO-Based Selection with Backward-Ant Mechanism for Adaptive Routing in Network-on-Chip Systems. *IEEE Embedded Systems Letters*, 5(3), 46–49. <https://doi.org/10.1109/LES.2013.2276211>
- Hu, X.-M., Zhang, J., & Li, Y. (2008). Orthogonal Methods Based Ant Colony Search for Solving Continuous Optimization Problems. *Journal of Computer Science and Technology*, 23(1), 2–18. <https://doi.org/10.1007/s11390-008-9111-5>
- Huang, K.-L., & Liao, C.-J. (2008). Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, 35(4), 1030–1046. <https://doi.org/10.1016/j.cor.2006.07.003>
- IEEE Xplore Abstract Record. (n.d.). Retrieved from <http://ieeexplore.ieee.org/abstract/document/5764154/>
- IEEE Xplore Full Text PDF. (n.d.). Retrieved from <http://ieeexplore.ieee.org/ielx5/5756602/5764069/05764154.pdf?tp=&arnumber=5764154&isnumber=5764069>

- Iredi, S., Merkle, D., & Middendorf, M. (2001). Bi-Criterion Optimization with Multi Colony Ant Algorithms. In *Evolutionary Multi-Criterion Optimization* (pp. 359–372). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-44719-9_25
- Junjie, P., & Dingwei, W. (2006). An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem. In *First International Conference on Innovative Computing, Information and Control - Volume I (ICICIC'06)* (Vol. 1, pp. 210–213). <https://doi.org/10.1109/ICICIC.2006.40>
- Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization*. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.
- Karaboga, D., & Kaya, E. (2016). An adaptive and hybrid artificial bee colony algorithm (aABC) for ANFIS training. *Applied Soft Computing*, 49, 423–436. <https://doi.org/10.1016/j.asoc.2016.07.039>
- Karmakar, R., Mitra, R., Dey, A., Chakraborty, V., & Nayak, A. (2016). Solving TSP Using Improved Elitist Ant System Based on Improved Pheromone Strategy and Dynamic Candidate List. *MAYFEB Journal of Computer Science*, 1(0). Retrieved from <http://mayfeb.com/OJS/index.php/COM/article/view/145>
- Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *IEEE International Conference on Neural Networks, 1995. Proceedings* (Vol. 4, pp. 1942–1948 vol.4). <https://doi.org/10.1109/ICNN.1995.488968>
- Khichane, M., Albert, P., & Solnon, C. (2008). Integration of ACO in a Constraint Programming Language. In *Ant Colony Optimization and Swarm Intelligence* (pp. 84–95). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-87527-7_8
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.
- Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4), 331–335. <https://doi.org/10.1016/j.orl.2003.09.003>
- Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113(1), 123–136. [https://doi.org/10.1016/S0377-2217\(97\)00420-7](https://doi.org/10.1016/S0377-2217(97)00420-7)
- Kötzing, T., Neumann, F., Sudholt, D., & Wagner, M. (2011). Simple Max-min Ant Systems and the Optimization of Linear Pseudo-boolean Functions. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms* (pp. 209–218). New York, NY, USA: ACM. <https://doi.org/10.1145/1967654.1967673>
- Krynicky, K., Houle, M. E., & Jaen, J. (2015). A Non-hybrid Ant Colony Optimization Heuristic for Convergence Quality. In *2015 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 1706–1713). <https://doi.org/10.1109/SMC.2015.300>
- Likas, A., Vlassis, N., & J. Verbeek, J. (2003). The global k-means clustering algorithm. *Pattern Recognition*, 36(2), 451–461. [https://doi.org/10.1016/S0031-3203\(02\)00060-2](https://doi.org/10.1016/S0031-3203(02)00060-2)
- Li-Ning Xing, Ying-Wu Chen, & Ke-Wei YANG. (2008). Double Layer ACO Algorithm for the Multi-Objective FJSSP. *New Generation Computing*, 26(4), 313–327.
- Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of*

- Operational Research*, 176(2), 657–690.
<https://doi.org/10.1016/j.ejor.2005.09.032>
- Luna, F., Blum, C., Alba, E., & Nebro, A. J. (2007). ACO vs EAs for Solving a Real-world Frequency Assignment Problem in GSM Networks. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (pp. 94–101). New York, NY, USA: ACM. <https://doi.org/10.1145/1276958.1276972>
- MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). Oakland, CA, USA.
- MadadyarAdeh, M., & Bagherzadeh, J. (2011). An improved ant algorithm for grid scheduling problem using biased initial ants. In *2011 3rd International Conference on Computer Research and Development* (Vol. 2, pp. 373–378). <https://doi.org/10.1109/ICCRD.2011.5764154>
- Malisia, A. R., & Tizhoosh, H. R. (2007). Applying Opposition-Based Ideas to the Ant Colony System. In *2007 IEEE Swarm Intelligence Symposium* (pp. 182–189). <https://doi.org/10.1109/SIS.2007.368044>
- Maniezzo, V., & Carbonaro, A. (2000). An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8), 927–935. [https://doi.org/10.1016/S0167-739X\(00\)00046-7](https://doi.org/10.1016/S0167-739X(00)00046-7)
- Marco Dorigo. (1992). *Optimization, learning and natural algorithms* (PHD). Politecnico di Milano, Italy.
- Neumann, F., Sudholt, D., & Witt, C. (2009). Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. *Swarm Intelligence*, 3(1), 35–68. <https://doi.org/10.1007/s11721-008-0023-3>
- Niknam, T., & Amiri, B. (2010). An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis. *Applied Soft Computing*, 10(1), 183–197. <https://doi.org/10.1016/j.asoc.2009.07.001>
- Niknam, T., Firouzi, B. B., & Nayeripour, M. (2008). An efficient hybrid evolutionary algorithm for cluster analysis. In *World Applied Sciences Journal*. Citeseer.
- Papadimitriou, C. (1994). *Computational Complexity* Addison-Wesley Reading. *Massachusetts Google Scholar*.
- Pezzella, F., & Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2), 297–310. [https://doi.org/10.1016/S0377-2217\(99\)00158-7](https://doi.org/10.1016/S0377-2217(99)00158-7)
- Rafsanjani, Marjan, Kuchaki, & Varzaneh, Zahra Asghari. (2015). Edge detection in digital images using Ant Colony Optimization. *Computer Science Journal of Moldova*, 23(3), 343–359.
- Robinson, E. J., Jackson, D. E., Holcombe, M., & Ratnieks, F. L. (2007). No entry signal in ant foraging (hymenoptera: Formicidae): new insights from an agent-based model. *Myrmecological News*, 10.
- Robinson, I., Webber, J., & Eifrem, E. (2015). Chapter 1 Graph Databases. In *Ant Colony Optimization and Constraint Programming* (pp. 1–5). 1005 Gravenstein Highway North: O'Reilly Media Inc.
- Rodrigues, D. M. S., & Ramos, V. (2014). Traversing News with Ant Colony Optimisation and Negative Pheromones. *arXiv:1405.6285 [Cs]*. Retrieved from <http://arxiv.org/abs/1405.6285>

- Rojas-Morales, N., Riff, M.-C., & Montero, E. (2016). Ants Can Learn from the Opposite. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016* (pp. 389–396). New York, NY, USA: ACM.
<https://doi.org/10.1145/2908812.2908927>
- Sahni, S., & Gonzalez, T. (1976). P-Complete Approximation Problems. *J. ACM*, 23(3), 555–565. <https://doi.org/10.1145/321958.321975>
- Shields, R. (2012). Cultural Topology: The Seven Bridges of Königsburg, 1736. *Theory, Culture & Society*, 29(4–5), 43–57.
<https://doi.org/10.1177/0263276412451161>
- Simon, D. (2013a). Chapter 2 Classical Evolutionary Algorithms. In *Evolutionary Optimization Algorithms* (pp. 35–63). New Jersey: John Wiley & Sons Inc.
- Simon, D. (2013b). Chapter 10 Ant Colony Optimisation. In *Evolutionary Optimization Algorithms* (pp. 35–63). New Jersey: John Wiley & Sons Inc.
- Smith, B. (n.d.). *CSPLib Problem 001: Car Sequencing*. (C. Jefferson, I. Miguel, B. Hnich, T. Walsh, & I. P. Gent, Eds.). Retrieved from
<http://www.csplib.org/Problems/prob001>
- Solnon, C. (2002). Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4), 347–357.
<https://doi.org/10.1109/TEVC.2002.802449>
- Solnon, Christine. (2000). Solving permutation constraint satisfaction problems with artificial ants. In *Proceedings of the 14th European Conference on Artificial Intelligence* (pp. 118–122). IOS Press.
- Solnon, Christine. (2008). Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research*, 191(3), 1043–1055.
<https://doi.org/10.1016/j.ejor.2007.04.037>
- Solnon, Christine. (2010a). Chapter 2 Computational Complexity. In *Ant Colony Optimization and Constraint Programming* (pp. 7–29).
- Solnon, Christine. (2010b). Chapter 4 Exact Approaches. In *Ant Colony Optimization and Constraint Programming* (pp. 53–65).
- Solnon, Christine. (2010c). Chapter 12 Sequencing Cars with ACO. In *Ant Colony Optimization and Constraint Programming* (pp. 165–183).
- Solnon, Christine, Cung, V. D., Nguyen, A., & Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191(3), 912–927. <https://doi.org/10.1016/j.ejor.2007.04.033>
- Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1), 3–18.
<https://doi.org/10.1111/itor.12001>
- Stützle, T., & Hoos, H. H. (2000). MAX–MIN Ant System. *Future Generation Computer Systems*, 16(8), 889–914. [https://doi.org/10.1016/S0167-739X\(00\)00043-1](https://doi.org/10.1016/S0167-739X(00)00043-1)
- Tang, J., & Zhao, X. (2010). On the improvement of opposition-based differential evolution. In *2010 Sixth International Conference on Natural Computation* (Vol. 5, pp. 2407–2411). <https://doi.org/10.1109/ICNC.2010.5583517>
- Tang, L., Liu, J., Rong, A., & Yang, Z. (2000). A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex.

- European Journal of Operational Research*, 124(2), 267–282.
[https://doi.org/10.1016/S0377-2217\(99\)00380-X](https://doi.org/10.1016/S0377-2217(99)00380-X)
- Tavares, J., & Pereira, F. B. (2011). Towards the Development of Self-ant Systems. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (pp. 1947–1954). New York, NY, USA: ACM.
<https://doi.org/10.1145/2001576.2001838>
- Tian, J., Yu, W., & Xie, S. (2008). An ant colony optimization algorithm for image edge detection. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (pp. 751–756).
<https://doi.org/10.1109/CEC.2008.4630880>
- Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (pp. 29–39).
- Yang, J., Shi, X., Marchese, M., & Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, 18(11), 1417–1422.
<https://doi.org/10.1016/j.pnsc.2008.03.028>
- Yao, Z., Liu, J., & Wang, Y. G. (2008). Fusing genetic algorithm and Ant Colony Algorithm to optimize virtual enterprise partner selection problem. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (pp. 3614–3620).
<https://doi.org/10.1109/CEC.2008.4631287>
- Yao, Z., Pan, R., & Lai, F. (2009). Improvement of the Fusing Genetic Algorithm and Ant Colony Algorithm in Virtual Enterprise Partner Selection Problem. In *2009 WRI World Congress on Computer Science and Information Engineering* (Vol. 1, pp. 242–246). <https://doi.org/10.1109/CSIE.2009.220>
- Zhao, F., Yao, Z., Luan, J., & Song, X. (2016). A Novel Fused Optimization Algorithm of Genetic Algorithm and Ant Colony Optimization. *Mathematical Problems in Engineering*, 2016, e2167413. <https://doi.org/10.1155/2016/2167413>

APPENDIX A: VIOLATION RATES OF PEAA ON UBUNTU

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
0	14	17	15	18	21	29	36	32	32	35
0	16	19	21	21	20	21	29	24	22	32
4	14	11	19	24	28	26	22	33	34	35
0	6	9	14	19	20	23	24	18	29	31
2	7	11	19	18	18	18	17	19	25	25
4	7	7	15	20	29	19	20	21	28	32
1	1	7	9	9	9	21	17	33	30	31
0	3	4	4	12	10	11	16	17	25	26
0	5	16	16	18	21	36	39	45	54	63
0	5	11	16	24	29	34	34	41	39	42
1	8	9	12	18	18	27	38	36	40	51
0	3	5	11	12	18	17	24	25	33	38
0	8	21	15	24	27	36	36	34	32	42
8	9	3	11	14	14	24	20	21	26	28
2	6	10	18	17	17	17	16	20	22	25
4	6	10	20	28	31	33	33	33	33	42
2	7	14	15	17	19	22	23	30	30	32
1	11	7	8	21	24	23	18	27	27	34
0	6	18	18	22	23	31	34	42	55	66
0	5	15	16	11	17	18	15	20	24	27
2	7	12	24	25	34	38	41	51	52	60
0	8	9	15	16	30	37	44	52	58	69
6	8	15	19	16	20	16	28	37	45	46
13	0	4	11	21	18	29	17	24	26	37
2	4	13	28	30	35	36	40	41	41	42
3	5	5	9	18	19	26	22	29	32	35
1	13	13	11	19	19	26	26	26	28	32
3	2	18	22	25	27	28	23	34	34	46
0	6	17	17	24	29	40	43	49	61	75
2	5	12	20	31	27	29	30	36	38	40
6	9	24	35	42	41	43	41	53	63	66
0	15	16	15	20	23	35	47	41	52	56
2	14	22	27	25	23	37	34	41	42	44
6	2	5	5	9	14	15	17	23	26	33
1	2	12	30	31	33	32	33	33	34	37
0	5	3	10	21	21	19	22	22	37	31
2	2	6	9	11	16	21	19	19	28	32
2	2	6	9	11	16	21	19	19	28	32

0	10	14	17	25	18	32	38	47	50	60
1	5	12	19	27	26	31	39	42	41	40
5	14	18	21	29	32	34	45	47	53	67
3	9	22	38	44	48	50	45	50	51	61
1	12	18	20	30	27	23	33	46	33	39
12	4	8	7	13	19	19	26	29	39	49
4	3	16	35	35	36	51	51	54	64	59
0	5	2	22	31	32	35	30	37	46	44
1	1	5	14	13	13	12	24	24	29	36
5	7	12	19	22	33	28	33	35	41	41
2	10	13	22	22	27	33	32	40	40	55
2	13	10	15	27	31	37	36	43	40	56
5	4	12	25	24	31	42	51	50	61	65
0	9	24	28	30	26	28	34	40	39	40
0	2	7	10	10	11	22	24	29	29	30
0	0	10	24	21	30	45	45	51	43	43
4	6	10	12	16	24	25	19	27	28	28
2	15	13	20	31	29	40	42	37	51	44
3	1	7	18	18	17	23	28	30	32	30
8	7	8	21	26	19	21	26	19	38	41
5	8	14	23	31	37	40	38	43	39	40
0	15	19	26	27	44	41	38	42	44	47
4	6	11	21	22	22	33	38	47	59	60
5	11	22	25	34	37	43	46	38	48	58
8	3	3	12	21	43	66	59	63	66	62
0	11	8	29	30	34	44	49	54	57	69
1	8	20	31	32	38	52	47	60	67	72
0	10	19	21	32	41	39	38	49	47	42
4	9	10	19	16	24	30	33	34	31	46
1	10	25	33	36	35	69	83	73	80	76
0	4	13	21	28	41	46	49	58	59	60
2	13	21	34	38	54	63	59	50	58	65
2	191	191	191	191	191	191	191	191	191	191
28	30	33	39	50	60	53	66	64	73	77
34	33	31	27	35	40	45	52	51	61	64
36	46	63	70	76	86	83	90	102	104	109
32	24	35	47	58	59	56	67	67	72	83
23	28	27	35	45	48	50	50	57	66	64
21	33	51	54	63	79	86	86	96	97	103
26	26	27	27	35	41	54	58	63	72	82
19	35	41	46	56	65	74	71	73	77	88
25	39	58	62	78	83	95	91	95	101	101

50	51	52	61	68	64	64	64	76	75	81
20	27	29	41	50	58	61	59	60	64	69
29	50	52	65	81	87	86	88	92	100	110
26	48	65	69	69	68	69	73	76	81	87
35	54	51	66	68	79	79	83	89	98	106
76	96	110	121	123	131	131	127	143	145	160
24	38	49	64	74	95	94	94	96	108	104
17	29	43	48	50	53	56	59	65	71	75
30	30	39	44	56	52	57	63	69	74	88
25	48	63	77	90	90	98	104	115	125	138
68	77	72	78	80	84	86	87	89	93	98
13	22	34	39	48	53	55	54	62	66	71
50	52	61	62	66	76	83	85	93	104	104
26	32	37	48	54	55	54	59	60	61	70
52	52	59	72	78	85	85	84	92	100	106
24	42	57	63	66	76	86	89	80	83	92
21	34	46	56	64	67	70	81	88	91	99
20	36	56	56	71	84	90	95	94	108	123
38	29	44	50	59	55	68	65	75	85	88
49	52	62	63	70	77	88	88	88	97	98
21	24	32	39	54	49	60	62	73	81	86

Appendix 1 Single Pheromone PEAA Violation Rates

Non Exp 1	EX1	EX2	EX3	EX4	EX5	EX6	EX7	EX8	EX9	EX10
4	44	53	94	94	98	107	115	114	119	127
21	47	66	76	73	75	83	89	88	91	94
15	94	123	131	136	156	148	159	155	165	162
25	79	104	121	131	114	140	132	133	139	140
1	65	80	102	98	132	134	127	144	156	169
25	88	135	139	157	166	173	176	183	185	190
1	69	104	111	130	131	142	141	152	165	175
7	92	115	131	139	169	179	174	178	162	176
29	109	155	162	176	180	194	199	206	204	212
17	60	69	80	91	116	111	114	119	123	138
14	54	70	100	97	108	115	133	127	143	143
9	114	149	167	178	182	178	182	188	183	180
14	117	117	148	162	174	189	192	193	190	198
16	116	135	149	171	169	172	185	177	188	182
0	73	98	114	129	129	138	151	152	148	155
6	85	142	148	153	164	175	186	186	190	202
5	63	100	128	120	126	157	173	154	170	172

2	111	137	133	137	171	163	184	173	180	197
25	137	155	180	187	201	200	203	213	218	211
7	59	92	121	125	147	152	150	165	168	172
10	77	106	136	150	175	186	187	190	187	188
11	129	153	168	194	199	205	219	213	211	213
2	97	103	139	139	145	164	176	172	168	173
6	117	131	162	180	192	187	208	208	202	210
4	85	99	121	152	155	180	182	184	194	198
2	105	136	156	162	183	193	183	188	196	195
1	91	96	111	119	139	140	159	167	167	161
7	80	128	135	157	157	178	184	187	188	193
16	110	137	166	178	200	208	207	211	218	214
6	67	110	149	158	165	183	186	200	199	193
2	91	116	128	146	148	153	177	186	185	181
6	132	167	179	187	203	191	197	213	234	230
2	140	152	164	173	167	175	183	178	186	191
11	125	146	182	194	186	207	212	209	202	211
19	106	134	147	164	170	188	194	189	188	185
4	119	133	173	175	175	185	187	185	191	190
17	51	91	127	147	161	177	181	179	186	203
17	51	91	127	147	161	177	181	179	186	203
2	103	122	139	168	173	196	207	204	202	208
4	83	97	125	142	160	169	177	172	170	168
4	114	118	130	140	158	148	166	170	173	181
16	129	165	176	197	196	202	204	226	234	236
9	109	126	164	186	210	209	214	213	224	223
10	145	149	197	192	195	207	226	218	225	227
13	94	114	150	152	159	174	207	203	195	199
9	109	131	135	150	162	169	169	175	166	181
3	58	96	93	118	137	141	144	150	151	150
5	118	162	156	179	182	182	203	207	202	194
6	145	162	175	193	196	204	202	204	204	207
3	134	129	161	182	204	208	222	224	225	223
3	108	129	142	149	167	177	181	184	182	199
5	122	121	157	152	172	185	194	201	194	210
4	32	55	56	58	67	60	75	103	111	107
21	98	106	116	129	138	140	134	136	134	143
9	99	123	116	126	134	163	165	164	168	172
36	104	133	143	150	165	153	177	181	182	192
19	92	131	149	149	166	171	176	176	185	186
20	185	184	209	221	229	233	221	239	244	244
10	133	171	183	211	208	214	215	219	216	213

23	90	119	118	124	133	135	139	144	144	156
4	105	140	171	177	183	178	199	197	197	193
7	129	178	193	201	209	219	241	242	244	238
9	126	160	155	172	172	172	187	190	175	170
28	123	162	174	205	215	218	215	228	223	220
0	167	153	177	184	195	218	215	217	206	212
28	69	73	75	85	114	97	115	110	105	121
48	180	195	206	221	228	225	243	231	242	239
34	162	211	228	230	267	260	260	269	281	277
28	105	141	141	148	144	167	168	178	186	185
58	124	156	160	171	170	197	195	201	202	201
2	198	198	198	198	198	198	198	198	198	198
28	126	122	122	136	146	145	147	148	149	158
14	172	174	182	187	195	190	194	193	204	214
26	159	205	216	223	224	211	215	224	225	225
26	161	177	169	188	193	194	191	190	201	197
22	158	156	167	176	171	171	173	179	178	185
26	119	163	183	183	188	185	192	188	196	195
24	139	137	151	150	145	149	158	165	180	180
24	147	176	178	177	175	170	172	179	180	186
25	144	188	191	181	210	194	200	204	212	229
44	150	161	166	171	172	170	171	174	176	174
28	199	237	239	235	242	249	253	259	265	264
28	187	186	176	178	197	197	192	208	219	229
34	244	259	271	269	257	260	264	264	284	307
32	245	255	275	295	276	280	280	277	278	285
87	319	346	342	383	370	377	412	424	418	420
28	193	213	237	253	261	254	248	254	258	264
34	177	218	237	248	271	271	267	275	274	286
35	258	264	268	279	285	289	287	289	295	295
22	210	227	232	234	235	224	225	227	233	251
53	241	270	295	287	292	297	323	334	328	323
18	344	365	364	379	368	362	355	349	362	363
47	370	401	413	432	411	409	417	414	411	435
34	218	252	230	252	259	273	288	300	324	338
52	366	368	372	395	379	385	384	399	400	404
23	250	257	289	298	293	302	310	299	309	303
26	227	271	252	263	255	264	267	268	263	274
28	232	251	255	250	247	249	248	265	266	276
32	323	333	366	354	347	359	358	366	370	368
48	334	317	311	321	317	315	316	323	339	343
34	271	281	298	307	311	316	325	324	326	328

APPENDIX B: CAR SEQUENCE DATA EXAMPLE

#####

#

Problem 16-81 CSP Lib

(Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, & Ian P. Gent,
1999)

#####

#

100 5 26

1 2 1 2 1

2 3 3 5 5

0 10 1 0 0 0 0

1 2 0 0 0 0 1

2 8 0 1 0 1 0

3 8 0 0 0 1 0

4 6 0 1 1 0 0

5 11 0 1 0 0 0

6 3 0 0 1 0 0

7 2 0 0 1 1 0

8 7 1 1 0 0 0

9 2 1 0 0 1 1

10 4 1 0 1 0 0

11 7 1 0 0 1 0

12 1 1 1 1 0 1

13 3 0 1 1 1 0

14 4 0 1 0 0 1

15 5 1 1 1 0 0

16 2 1 1 0 0 1

17 1 1 0 1 1 1

```

18 2 1 0 1 1 0
19 3 1 0 0 0 1
20 2 0 1 1 0 1
21 1 0 1 0 1 1
22 3 1 1 0 1 0
23 1 0 0 1 1 1
24 1 1 1 1 1 1
25 1 1 1 1 1 0

```

APPENDIX C: SHELL SCRIPT FOR EXECUTING ALGORITHM

```

#!/bin/bash
PROG_DIR=/home/blindpng/dissertation/AntCar
DATA_DIR=$PROG_DIR/data
DATA_DIR2=$PROG_DIR/data/ProblemDataSet200to400
TEST_DIR=$PROG_DIR/TestResults

# top -b -u root -d 1 >$TEST_DIR/top.out &

if ! [ -d "$TEST_DIR" ]
then
    mkdir $TEST_DIR
fi

MAX_COUNT=5
TEST_No=0

# Strategy 1
# Run all tests in sequence 50 times
COUNTER=0

TEST_FILE=$TEST_DIR/strategy_1_sequence.txt
echo 'TestNo,Start Time,Test File,Command' > $TEST_FILE

while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -Z $TEST_No -f $f >> $TEST_FILE
        $PROG_DIR/main -Z $TEST_No -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done
# The second set of test files
COUNTER=0
while [ $COUNTER -lt $MAX_COUNT ]; do

```

```

for f in $DATA_DIR2/*.txt
do
    Strat_Date=`date`
    echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -Z $TEST_No -f $f >> $TEST_FILE
    $PROG_DIR/main -Z $TEST_No -f $f
    let TEST_No+=1
done

let COUNTER+=1
done

# Strategy 2
# Run all tests in sequence $MAX_COUNT times

COUNTER=0
TEST_FILE=$TEST_DIR/strategy_2_sequence.txt
echo 'TestNo,Start Time,Test File,Command' > $TEST_FILE

while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -Z $TEST_No -t 2 -f $f >> $TEST_FILE
        $PROG_DIR/main -Z $TEST_No -t 2 -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done

# second set of data
COUNTER=0
while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR2/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -Z -t 2 $TEST_No -f $f >> $TEST_FILE
        $PROG_DIR/main -Z $TEST_No -t 2 -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done

#####
### Explorer Ants ###
#####

# Strategy 1
# Run all tests in sequence 50 times
COUNTER=0

TEST_FILE=$TEST_DIR/strategy_1_sequence_exp.txt
echo 'TestNo,Start Time,Test File,Command' > $TEST_FILE

while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR/*.txt

```

```

do
    Strat_Date=`date`
    echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -E 10 -Z $TEST_No -f $f >> $TEST_FILE
    $PROG_DIR/main -E 10 -Z $TEST_No -f $f
    let TEST_No+=1
done

    let COUNTER+=1
done
# The second set of test files
COUNTER=0
while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR2/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -E 10 -Z $TEST_No -f $f >> $TEST_FILE
        $PROG_DIR/main -E 10 -Z $TEST_No -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done

# Strategy 2
# Run all tests in sequence $MAX_COUNT times

COUNTER=0
TEST_FILE=$TEST_DIR/strategy_2_sequence_exp.txt
echo 'TestNo,Start Time,Test File,Command' > $TEST_FILE

while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -E 10 -Z $TEST_No -t 2 -f $f >> $TEST_FILE
        $PROG_DIR/main -E 10 -Z $TEST_No -t 2 -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done

# second set of data
COUNTER=0
while [ $COUNTER -lt $MAX_COUNT ]; do

    for f in $DATA_DIR2/*.txt
    do
        Strat_Date=`date`
        echo $TEST_No,$Strat_Date,$f,$PROG_DIR/main -E 10 -Z -t 2 $TEST_No -f $f >> $TEST_FILE
        $PROG_DIR/main -E 10 -Z $TEST_No -t 2 -f $f
        let TEST_No+=1
    done

    let COUNTER+=1
done

cp $PROG_DIR/*.csv $TEST_DIR

```

```
rm $PROG_DIR/*.csv
```

```
pkill top
```

APPENDIX D: MAKEFILE FOR BUILDING ALGORITHM

```
CFLAGS = -O3 -g -Wall -fstrict-aliasing -std=c99
# TESTFILE = pbtest
TESTFILE = pb16-81
TESTFILE = pb_400_01.txt

test:main
# rm *.csv
# ./main -E 1 -Z 2 -t 2 -f $(TESTFILE)
./main -E 0 -Z 1 -f $(TESTFILE) 2> out1.txt
./main -E 0 -Z 2 -f $(TESTFILE) 2> out2.txt
./main -E 0 -Z 3 -f $(TESTFILE) 2> out3.txt
./main -E 0 -Z 4 -f $(TESTFILE) 2> out4.txt
./main -E 0 -Z 5 -f $(TESTFILE) 2> out5.txt

main:main.c
$(CC) $(CFLAGS) main.c -o $@

debug:main.c
$(CC) $(CFLAGS) -D DEBUG=1 main.c -o main
./main -Z 1 -f $(TESTFILE)
./main -Z 1 -f $(TESTFILE)

debugExp:main.c
$(CC) $(CFLAGS) -D DEBUG=1 main.c -o main
./main -E 1 -Z 1 -f $(TESTFILE)
./main -E 1 -Z 1 -t 2 -f $(TESTFILE)
```

APPENDIX E: MODIFIED SOURCE CODE

```
#define _XOPEN_SOURCE
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>

typedef struct {
    int nbCars, nbClasses, nbOptions;
    int *nbCarsInClass, **requires, *classOf, *firstValue, *capacity, *frequency;
    int *totReq;
    int **reqOpt;
} carSeqInstance;

typedef struct {
    int nbExpAnts; //The number of explorer ants set using command line argument.
    int **seq; // The car sequence built by the explorer
    int *nbViolations;
    int *evalSeq; // evalSeq[i] = nb of violated constraints in seq[i]
    int *bestSeq; // best computed sequence
    int evalBestSeq; // nb of violated constraints in bestSeq
    int bestCycle; // nb of violated constraints in the best sequence of the cycle
    int TotalNonExpAnts;
    int NonExpCurAnt; // the number of the current non explorer ant
    int *NonExpEvalSeq;
```

```

    int curAnt; // Current explorer ant used with seqPos
    int *seqPos; //Sequences to this pos have been filled.
    int *VisitedNodes; // Already visited nodes
    int i; // Counter added to prevent confusion with other counters.
    int j; // Counter added to prevent confusion with other counters.
    int *k; // K value is the randomly chosen path but with the explorers it is unexplored path
    // the value of k is set for each explorer ant.
} explorerAnt;

//int BldExpAntSeq2(explorerAnt* exAnt, int* candidates, int nbCars, int CurCarNo, int
nbCandidates);
int BldExpAntSeq2(explorerAnt* exAnt, int* candidates, int nbCars, int CurCarNo, int
nbCandidates, int PrevChosen);
int MakeUnique(int *arr, int size, int srch);
int SelOthCan(int* PrevChosenArr, int* candidates, int nbCandidates);
int ExpHasCand(int* PrevChosenExpArr,int PrevChosen,int CurCarNo);
int findAvailCar(int* PrevChosenExpArr,int maxPos, int nbCars,int HasBeenUsed,int LoHi);
// @author Kieran OSullivan
int PrintTau(int prnt,int Ant,int t1S, int t12S, int t2S, float** tau1, float* tau2){
    int i,j,k;
    fprintf(stderr, "Ant=%d",Ant);
    if (prnt>=1){
        fprintf(stderr,"tau1\n ");
        for (i=0; i<t1S; i++){fprintf(stderr,"%d ",i);}
        fprintf(stderr,"\n");
        for (i=0; i<t1S; i++){
            fprintf(stderr,"%d",i);
            for(j=0; j<t12S; j++){fprintf(stderr," %.0f",tau1[i][j]);}
            fprintf(stderr,"\n");
        }
    }
    if (prnt>=2){
        fprintf(stderr,"tau2\n");
        for (k=0; k<t2S; k++){fprintf(stderr,"%d ",k);}
        fprintf(stderr,"\n");
        for (k=0; k<t2S; k++){fprintf(stderr,"%.2f ",tau2[k]);}
        fprintf(stderr,"\n");
    }
    return 0;
}
/**
 * @author Kieran OSullivan
 */
int PrintPValue(float* p){
    int n,i;
    n = sizeof(p)/sizeof(p[0]);
    if (n == 0){fprintf(stderr,"\np[0]=%f",p[0]);}
    else {for (i=0;i<n;i++){fprintf(stderr,"\np[%d]=%f",i,p[i]);}}
    return 0;
}
// @author Kieran OSullivan
int PrintSequences(int* seq, explorerAnt* exAnt, carSeqInstance* c){
    for (int kie=0; kie < c->nbCars; kie++){
        fprintf(stderr,"\nseq[%d]=%d",kie,seq[kie]);
        if (exAnt->nbExpAnts > 0 ){
            for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){
                fprintf(stderr," Xseq[%d][%d]=%d",exAnt->i,kie, exAnt->seq[exAnt-
>i][kie]);
            }
        }
    }
}

```

```

    }
    fprintf(stderr, "\n");
    return 0;
}
/**
 * @author Kieran OSullivan
 */
int PrintCandidates(int a, int nbCandidates, int *candidates, int size){
    fprintf(stderr, "Calling PrintCandidates=%d nbCandidates=%d\n", a, nbCandidates);
    for (int i=0; i<size; i++){
        fprintf(stderr, "candidates[%d]=%d\n", i, candidates[i]);
    }
    return 0;
}
/**
 * @author Kieran OSullivan
 */
int MakeUnique(int *arr, int size, int srch){
    /**
     * This function searches for a particular number in an unsorted array.
     * If that number exists it increments it and re-starts the search by setting i to -1
     * The array is initialised to -1 so that searching for 0 does not cause an infinite loop.
     * The search term will be incremented until the loop finishes or a number is found that
     * is unique.
     * E.g. if 2 is passed down and 2 already exists then 3 will be returned.
     */

    int i, process;
    process=0;
    for (i=0; i < size; i++) {if (arr[i]==-1){process=1; i=size;}}
    if (process==1){
        for (i=0; i < size; i++){
            if (arr[i]==srch){srch++; if(i+1 < size){i=-1;}}
            if (srch > size-1){srch=0;}
        }
    }
    return srch;
}
/**
 * @author Christine Solnon
 */
void getData(char* file_name, carSeqInstance* c){
    // Read car sequencing instance contained in file_name and initialize c
    int unused __attribute__((unused));

    FILE* fd;
    int i, j, k, nbReq;
    k=0;
    if ((fd=fopen(file_name, "r"))==NULL){fprintf(stderr, "ERROR: Cannot open ascii input file
%s", file_name); return;};
    unused = fscanf(fd, "%d%d%d", &(c->nbCars), &(c->nbOptions), &(c->nbClasses));

    c->capacity = (int*)calloc(c->nbOptions, sizeof(int));
    c->frequency = (int*)calloc(c->nbOptions, sizeof(int));
    c->totReq = (int*)calloc(c->nbOptions, sizeof(int));
    c->nbCarsInClass = (int*)calloc(c->nbClasses, sizeof(int));
    c->firstValue = (int*)calloc(c->nbClasses, sizeof(int));
    c->requires = (int**)calloc(c->nbClasses, sizeof(int*));
    c->reqOpt = (int**)calloc(c->nbClasses, sizeof(int*));
    c->classOf = (int*)calloc(c->nbCars, sizeof(int));

```



```

for (i=0; i<c->nbOptions; i++){
    unused = fscanf(fd,"%d",&(c->capacity[i]));
}
for (i=0; i<c->nbOptions; i++){
    unused = fscanf(fd,"%d",&(c->frequency[i]));
    c->totReq[i] = 0;
}
for (i=0; i<c->nbClasses; i++) {
    c->requires[i] = (int*)calloc(c->nbOptions,sizeof(int));
    c->reqOpt[i] = (int*)calloc(c->nbOptions+1,sizeof(int));
    unused = fscanf(fd,"%d%d",&j,&(c->nbCarsInClass[i]));
    nbReq = 0;
    for (j=0; j<c->nbOptions; j++){
        unused = fscanf(fd,"%d",&(c->requires[i][j]));
        if (c->requires[i][j] > 0){
            c->reqOpt[i][nbReq] = j;
            nbReq++;
            c->totReq[j] += c->nbCarsInClass[i];
        }
    }
    c->reqOpt[i][nbReq] = -1;

    c->firstValue[i]=k;
    for (j=0; j<c->nbCarsInClass[i]; j++){
        c->classOf[k]=i;
        k++;
    }
}
fclose(fd);
}

/**
 * @author Christine Solnon
 */
int usage (char *exec) {
    #ifdef DEBUG
        fprintf(stderr, "\nEnterin Function: usage (char *exec)");
    #endif
    fprintf(stderr, "\nUsage : \n\n\t%s -a (alpha: int) -b (beta: int) -r (rho1: float) -R (rho2: float) -c
(max nb cycle: int) -n (nb ants: int) -m (tau1 min: float) -M (tau1 max: float) -t (strategy: 1 if
ACO(tau1,utilRate); 2 if ACO(tau1,tau2)) -Z (int test number) -E (int number of Explorer Ants) -f
(filename: string) -v display-frequency -s (seed: positive int)\n\n",exec);
    return(0);
}

/**
 * @author Christine Solnon
 */
int parse(int* alpha, int* beta, float* rho1, float* rho2, int* maxCycles, int* nbAnts, float* tau1Min,
float* tau1Max, int* verbose, int* displayFreq, char* fileName, int* seed, int* strategy, int* TestNo,
int* ExpAnt, char* argv[], int argc){
    // read arguments to initialize parameters
    #ifdef DEBUG
        fprintf(stderr, "\nEnterin Function: parse(int* alpha, int* beta, float* rho1, float* rho2,
int* maxCycles, int* nbAnts, float* tau1Min, float* tau1Max, int* verbose, int* displayFreq, char*
fileName, int* seed, int* strategy, int* TestNo, char* argv[], int argc)\n");
    #endif
    char ch;
    extern char* optarg;
    while ( (ch = getopt(argc, argv, "a:b:r:R:c:n:m:M:v:f?:h:s:t:Z:E:"))!= -1 ) {

```

```

#ifdef DEBUG
    fprintf(stderr, "\n Characters to process on command line %c", ch);
#endif

switch(ch) {
    case 'a': *alpha=atoi(optarg); break;
    case 'b': *beta=atoi(optarg); break;
    case 'r': *rho1=atof(optarg); break;
    case 'R': *rho2=atof(optarg); break;
    case 's': *seed=atoi(optarg); break;
    case 't': *strategy=atoi(optarg); break;
    case 'c': *maxCycles=atoi(optarg); break;
    case 'n': *nbAnts=atoi(optarg); break;
    case 'm': *tau1Min=atof(optarg); break;
    case 'M': *tau1Max=atof(optarg); break;
    case 'v': *verbose=1; *displayFreq=atoi(optarg); break;
    case 'Z': *TestNo=atoi(optarg); break;
    case 'E': *ExpAnt=atoi(optarg); break;
    case 'f': strncpy(fileName, optarg, 254); break;
    case '?':
    case 'h':
    default: usage(argv[0]); return(1);
}
}
return(0);
}
/**
 * @author Christine Solnon
 */
void displayCar(int car, carSeqInstance *c){
    int k = c->classOf[car];
    fprintf(stdout, "%d ", k);
}
/**
 * @author Christine Solnon
 */
float myPow(float x, int y){
    float res = 1.00;
    int i;
    if (y==1){res = x;}
    for(i=1; i < y; i++){
        if (i==1){res=x*x;}
        else {res=res*x;}
    }
    return res;
}
/**
 * @author Christine Solnon
 */
int choose(float *p, int nbCand, float f){
    /* for i in 0..nbCand-1, p[i] = sum_{j<i} tau1[j]^alpha */
    /* returns k with probability (p[k]-p[k-1])/p[nbCand-1] */
    // fprintf(stderr, "\nEntering Function: choose(float *p, int nbCand=%d, float
f=%f)", nbCand, f);
    // PrintPValue(p);

    int left=0;
    int right=nbCand-1;
    int k;
    float total=p[nbCand-1];

```

```

while (left<right){
    k=(left+right+1)/2;
    if (f<p[k-1]/total) {right=k-1;}
    else if (f>p[k]/total) {left=k+1;}
    else {return k;}
}
return left;
}
/**
 * @author Kieran OSullivan
 */
int findAvailCar(int* PrevChosenExpArr,int maxPos, int nbCars,int HasBeenUsed,int LoHi){
    int i,j;
    if (LoHi==0){
        for (i=0;i<nbCars; i++){
            if (i != HasBeenUsed){
                for (j=0; j <= maxPos; j++){
                    if (ExpHasCand(PrevChosenExpArr,j,maxPos)==0) {
                        return j;
                    }
                } // for j
            } // if i != HasBeenUsed
        } // for i
    } else {
        for (i=nbCars; i >= 0; i--){
            if (i != HasBeenUsed){
                for (j=0; j <= maxPos; j++){
                    if (ExpHasCand(PrevChosenExpArr,j,maxPos)==0) {
                        return j;
                    }
                } // for j
            } // if i != HasBeenUsed
        } // for i
    }
    return -200;
}
/**
 * @author Kieran OSullivan
 */
int BldExpAntSeq2(explorerAnt* exAnt, int* candidates, int nbCars, int CurCarNo, int
nbCandidates, int PrevChosen){
    int PrevChosenArr[nbCandidates];
    int PrevChosenExpArr[CurCarNo];
    int i,j,k;

    for (j=0;j< CurCarNo; j++){PrevChosenExpArr[j]=-1;}

    if (nbCandidates==1){
        // There is only one option so the explorers have to use it.
        // if it is already used then the lowest possible car is chosen.
        for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){

            for (j=0;j< CurCarNo; j++){
                PrevChosenExpArr[j]=exAnt->seq[exAnt->i][j];
                if (PrevChosenExpArr[j]==PrevChosen){PrevChosen=-1;}
            }
            if (PrevChosen==-1){
                for (k=0;k<nbCars;k++){
                    if (ExpHasCand(PrevChosenExpArr,k,CurCarNo)==0){

```

```

PrevChosen=k;
break; //k=nbCars; // No need to go further.
    }
}
}

exAnt->seq[exAnt->i][CurCarNo] = PrevChosen;
// fprintf(stderr, " exAnt->seq[%d][%d]=%d",exAnt->i,CurCarNo,exAnt-
>seq[exAnt->i][CurCarNo]);
    }
} else {
    PrevChosenArr[0]=PrevChosen;
    for (i=1;i< nbCandidates; i++){PrevChosenArr[i]=-1;}
    i=1;
// re-set is so that it can be used to track when the number of explorers exceeds the number of
candidates.

    for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){

        if (i < nbCandidates){
            for (j=0;j< CurCarNo; j++){PrevChosenExpArr[j]=exAnt->seq[exAnt-
>i][j];}
            exAnt->seq[exAnt->i][CurCarNo] =
SelOthCan(PrevChosenArr,candidates,nbCandidates);
            if (ExpHasCand(PrevChosenExpArr,exAnt->seq[exAnt-
>i][CurCarNo],CurCarNo)==1){
                // fprintf(stderr,"\n%d already used by this ant so using %d
instead.",exAnt->seq[exAnt->i][CurCarNo],PrevChosen);
                // remove this code it may be wrong.
                for (j=0;j< CurCarNo; j++){
                    PrevChosenExpArr[j]=exAnt->seq[exAnt->i][j];
                    if (PrevChosenExpArr[j]==PrevChosen){PrevChosen=-
1;}
                } // for j
                if (PrevChosen==-1){
                    for (k=0;k<nbCars;k++){
                        if
(ExpHasCand(PrevChosenExpArr,k,CurCarNo)==0){
                            PrevChosen=k;
                            break; k=nbCars; // No need to go further.
                        }
                    } // for k
                } // if PrevChosen==-1
                exAnt->seq[exAnt->i][CurCarNo] = PrevChosen;
            }

            PrevChosenArr[i]=exAnt->seq[exAnt->i][CurCarNo];

        } else {
            // There are too many explorers to be of any use.
            for (j=0;j< CurCarNo; j++){
                PrevChosenExpArr[j]=exAnt->seq[exAnt->i][j];
                if (PrevChosenExpArr[j]==PrevChosen){PrevChosen=-1;}
            }
            if (PrevChosen==-1){
                for (k=0;k<nbCars;k++){
                    if (ExpHasCand(PrevChosenExpArr,k,CurCarNo)==0){
                        PrevChosen=k;
                        break; //k=nbCars; // No need to go further.
                    }
                }
            }
        }
    }
}

```

```

    }
}

exAnt->seq[exAnt->i][CurCarNo] = PrevChosen;
PrevChosenArr[i]=exAnt->seq[exAnt->i][CurCarNo];
}
// fprintf(stderr," nbCandidates=%d exAnt-
>seq[%d][%d]=%d",nbCandidates,exAnt->i,CurCarNo,exAnt->seq[exAnt->i][CurCarNo]);
i++;
} // for exAnt->i
} // end if else block
// fprintf(stderr,"\n");
return 0;
}
/**
 * @author Kieran OSullivan
 */
int SelOthCan(int* PrevChosenArr, int* candidates, int nbCandidates){
    int returnval;
    int i,j,k;
    returnval=-1;

    for(i=0; i<nbCandidates; i++){
        for (j=0; j<nbCandidates; j++){
            // fprintf(stderr,"\nPreviousCandidate[%d]=%d
candidate[%d]=%d\n",j,PrevChosenArr[j],i,candidates[i]);
            if (PrevChosenArr[j] == -1){
                j=nbCandidates; // the rest of this array does not need to be processed.
            } else {
                if (PrevChosenArr[j] != candidates[i]) {returnval=candidates[i];}
            }
            //if there is a return value then it cannot be in the list of previously used ones.
            if (returnval > -1){
                for (k=0; k < nbCandidates;k++){
                    if (PrevChosenArr[k]==-1){k=nbCandidates;}
                    else if (PrevChosenArr[k]==returnval){
                        returnval=-1;
                        k=nbCandidates;
                    }
                }
            } // if returnval > -1
            if (returnval > -1){return returnval;}
        } // for j
    } // for i
    return returnval;
}
/**
 * @author Kieran OSullivan
 */
int ExpHasCand(int* PrevChosenExpArr,int PrevChosen,int CurCarNo){
    for (int i=0; i < CurCarNo; i++){
        if (PrevChosenExpArr[i] == PrevChosen){return 1;}
    }
    return 0;
}
/**
 * @author Christine Solnon
 */
int walkDoublePhero(carSeqInstance* c, float** tau1, float* tau2, int alpha, int beta, float rho2, int*
seq, explorerAnt* exAnt){

```

```

// input: c = a car sequencing instance
//   tau1 = first pheromone structure
//   tau2 = second pheromone structure
//   alpha and beta = parameters
// output: seq = sequence of cars built by an ant w.r.t. ACO(tau1,tau2)
// returns the number of constraint violations

float total;
int nbAssignedCars, j, k, l, cc, min, nbViolations;
int k_tmp; // added as part of explorer ants to store values for each choice after the first choice
and before the last.
int nbCandidates = 0;
float p[c->nbClasses];
int candidates[c->nbClasses], nbSelected[c->nbClasses], cpt[c->nbOptions];
int first, next[c->nbClasses], nbViolationsClass[c->nbClasses];

nbViolations = 0;

// choice of the first car
total = 0;
first = 0;
for (j=0; j<c->nbClasses; j++){
    nbSelected[j] = 0;
    next[j] = j+1;
    p[j] = myPow(tau2[j],beta) + total;
    total = p[j];
}
next[j-1] = -1; //This is to mark the last classs if there are 26 classes the pos 25 = -1

k = choose(p,j,drand48());

seq[0] = c->firstValue[k];
// Explorer ants will start with different sequences
// Not the one already chosen.
// each explorer ant has its own sequence and does not just copy its previous ant.
// The first explorer ant is based on the value of k for the non explorer ant.
// the secont explorer ant is basd on the first and the third is based on the second.
// the possible candidates this time are all the cars in future calls to this function
// they will be limited to the candidates short-listed buy the noen explorer ants.
// k is passed down twice as the function uses it the second time to limit the possible choices.

if (exAnt->nbExpAnts > 0 ){
    // BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, 0, c->nbClasses);
    BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, 0, c->nbClasses, seq[0]);
}

nbSelected[k]++; //The selected option will be 1 all the others will be 0

if (c->nbCarsInClass[k]==1){
    if (k==0) {first = 1;}
    else {next[k-1] = next[k];}
}
for (l=0; l<c->nbOptions; l++) {
    cpt[l] = c->requires[k][l];
}

// choice of the next cars

for (nbAssignedCars=1; nbAssignedCars<c->nbCars-1; nbAssignedCars++){
    min = c->nbOptions+1;

```

```

for (j=first; j>=0; j=next[j]){
    nbViolationsClass[j] = 0;
    for (l=0; c->reqOpt[j][l]>=0; l++){
        if (cpt[c->reqOpt[j][l]]+1-c->capacity[c->reqOpt[j][l]]>0){
            nbViolationsClass[j]++;
        }
    }
    if (nbViolationsClass[j] < min){
        min=nbViolationsClass[j];
        candidates[0]=c->firstValue[j]+nbSelected[j];
        p[0]=myPow(tau1[seq[nbAssignedCars-
1]][candidates[0]],alpha)*myPow(tau2[j],beta);
        total = p[0];
        nbCandidates=1;
        // if (exAnt->nbExpAnts > 0 ){exAnt->nbCandidates = 1;}
    }
    else if (nbViolationsClass[j] == min){
        candidates[nbCandidates] = c->firstValue[j]+nbSelected[j];
        p[nbCandidates]=myPow(tau1[seq[nbAssignedCars-
1]][candidates[nbCandidates]],alpha)
        *myPow(tau2[j],beta) + total;
        total = p[nbCandidates];
        nbCandidates++;
    }
}
k_tmp = choose(p,nbCandidates,drand48());
// PrintCandidates(nbAssignedCars,nbCandidates,candidates,nbCandidates);
seq[nbAssignedCars] = candidates[k_tmp];

// Each Explorer ant now builds their sequence based on what other ants have done
// No point just doing it all from scratch explorers should go where normal ants have not gone
// fprintf(stderr, "\nwalkDual candidates[%d]=%d
k_tmp=%d",nbCandidates,candidates[nbCandidates],k_tmp);

if (exAnt->nbExpAnts > 0 ){
    // BldExpAntSeq2(exAnt, candidates, c->nbCars, nbAssignedCars, nbCandidates);
    BldExpAntSeq2(exAnt, candidates, c->nbCars, nbAssignedCars, nbCandidates,
seq[nbAssignedCars]);
}

k = c->classOf[seq[nbAssignedCars]];
if (min>0) // all classes violate constraints => add pheromone on tau2
    for (cc=first; cc>=0; cc=next[cc]) tau2[cc] += nbViolationsClass[cc];
for (l=0; l<c->nbOptions; l++){
    if (c->requires[k][l]==1){
        cpt[l]++;
        if (nbAssignedCars >= c->frequency[l]-1){
            if (cpt[l] > c->capacity[l]) nbViolations++;
            cpt[l] -= c->requires[c->classOf[seq[nbAssignedCars-c->frequency[l]+1]]][l];
        }
    }
    else if (nbAssignedCars>=c->frequency[l]-1){
        if (cpt[l] > c->capacity[l]) nbViolations++;
        cpt[l] -= c->requires[c->classOf[seq[nbAssignedCars-c->frequency[l]+1]]][l];
    }
}
nbSelected[k]++;
if (nbSelected[k]==c->nbCarsInClass[k]){
    if (first==k) first = next[first];
    else{

```

```

        for (j=first; next[j] != k; j=next[j]);
        next[j] = next[k];
    }
}
// choice of the last car
k_tmp = c->firstValue[first]+nbSelected[first];
seq[nbAssignedCars] = k_tmp;

if (exAnt->nbExpAnts > 0 ){
    // BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates);
    BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates,
seq[nbAssignedCars]);
}

for (l=0; l<c->nbOptions; l++){
    if (c->requires[first][l]==1){
        if (cpt[l]+1 > c->capacity[l]){
            nbViolations++;
            tau2[first]++;
        }
    }
    else if (cpt[l] > c->capacity[l]) nbViolations++;
}
/* evaporation of pheromone for the second pheromone structure */

for (j=0; j<c->nbClasses; j++)
    if (tau2[j]>1) tau2[j] *= 1-rho2;

// PrintSequences(seq,exAnt,c);
return nbViolations;
}
/**
 * @author Christine Solnon
 */
int walkUtilRate(carSeqInstance* c, float** tau, int alpha, int beta, int* seq, explorerAnt* exAnt){
    // input: c = a car sequencing instance
    //      tau = first pheromone structure
    //      alpha and beta = parameters
    // output: seq = sequence of cars built by an ant w.r.t. ACO(tau1,utilRate)
    // returns the number of constraint violations
    float total;
    int nbAssignedCars, i, j, k, l, min, nbViolatedSeq;
    int k_tmp;
    int nbCandidates = 0;
    float p[c->nbClasses];
    int candidates[c->nbClasses], nbSelected[c->nbClasses], cpt[c->nbOptions];
    int first, next[c->nbClasses];
    int currentReq[c->nbOptions];
    float utilRate[c->nbOptions];
    float u;
    for (j=0; j<c->nbOptions; j++){
        cpt[j] = 0;
        currentReq[j] = c->totReq[j];
        utilRate[j] = (float)(currentReq[j] * c->frequency[j]) /
(float)(c->capacity[j] * c->nbCars);
    }
    first = 0;
    nbViolatedSeq = 0;
    // choice of the first car

```



```

total = 0;
for (j=0; j<c->nbClasses; j++){
    nbSelected[j] = 0;
    next[j] = j+1;
    u = 0;
    for (i=0; c->reqOpt[j][i]>=0; i++) u += utilRate[c->reqOpt[j][i]];
    p[j] = myPow(u,beta) + total;

    total = p[j];
}
next[j-1] = -1;
k = choose(p,j,drand48());
// PrintCandidates(0, c->nbCars, c->firstValue,c->nbCars);
seq[0] = c->firstValue[k];

if (exAnt->nbExpAnts > 0 ){
    //BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, 0, c->nbClasses);
    BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, 0, c->nbClasses, seq[0]);
}

nbSelected[k]++;
if (c->nbCarsInClass[k]==1){
    if (k==0) {first = 1;}
    else {next[k-1] = next[k];}
}
for (l=0; c->reqOpt[k][l]>=0; l++){
    cpt[c->reqOpt[k][l]]++;
    currentReq[c->reqOpt[k][l]]--;
    utilRate[c->reqOpt[k][l]] = (float)(currentReq[c->reqOpt[k][l]] * c->frequency[c-
>reqOpt[k][l]]) / (c->capacity[c->reqOpt[k][l]] * (c->nbCars - 1));
}
// choice of the next cars
for (nbAssignedCars=1; nbAssignedCars<c->nbCars-1; nbAssignedCars++){
    min = c->nbOptions+1;
    for (j=first; j>=0; j=next[j]){
        k = 0;
        u = 0;
        for (l=0; c->reqOpt[j][l]>=0; l++){
            u += utilRate[c->reqOpt[j][l]];
            if (cpt[c->reqOpt[j][l]]+1-c->capacity[c->reqOpt[j][l]]>0) k++;
            /*
            if (exAnt->nbExpAnts > 0 ){
                for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){
                    if(exAnt->cpt[exAnt->i][c->reqOpt[j][l]]+1-c->capacity[c-
>reqOpt[j][l]]>0){
                        exAnt->k++;
                        // ***** COME BACK TO THIS *****
                    }
                }
            }*/
        }
        if (k<min){
            min=k;
            candidates[0]=c->firstValue[j]+nbSelected[j];
            p[0]=myPow(tau[seq[nbAssignedCars-1]][candidates[0]],alpha)*myPow(u,beta);
            total = p[0];
            nbCandidates=1;
            // if (exAnt->nbExpAnts > 0 ){exAnt->nbCandidates = 1;}
        }
    }
}

```

```

        else if (k==min){
            candidates[nbCandidates] = c->firstValue[j]+nbSelected[j];
            p[nbCandidates]=myPow(tau[seq[nbAssignedCars-
1]][candidates[nbCandidates]],alpha)
            *myPow(u,beta) + total;
            total = p[nbCandidates];
            nbCandidates++;
            // if (exAnt->nbExpAnts > 0 ){exAnt->nbCandidates++;}
        }
    }
    k_tmp = choose(p,nbCandidates,drand48());
    seq[nbAssignedCars] = candidates[k_tmp];
    if (exAnt->nbExpAnts > 0 ){
        // BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates);
        BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates,
seq[nbAssignedCars]);
    }

    // Each Explorer ant now builds their sequence based on what other ants have done
    // No point just doing it all from scratch explorers should go where normal ants have not gone

    k = c->classOf[seq[nbAssignedCars]];
    nbSelected[k]++;
    if (nbSelected[k]==c->nbCarsInClass[k]){
        if (first==k) { first = next[first];}
        else{
            for (j=first; next[j] != k; j=next[j]);
            next[j] = next[k];
        }
    }

    for (l=0; l<c->nbOptions; l++){
        if (c->requires[k][l]==1){
            currentReq[l]--;
            cpt[l]++;
            /*
            if (exAnt->nbExpAnts > 0 ){
                for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){
                    exAnt->cpt[exAnt->i][l]++;
                }
            }
            */
            if (nbAssignedCars >= c->frequency[l]-1){
                if (cpt[l] > c->capacity[l]) nbViolatedSeq++;
                cpt[l] -= c->requires[c->classOf[seq[nbAssignedCars-c->frequency[l]+1]]][l];
                /*
                if (exAnt->nbExpAnts > 0 ){
                    for (exAnt->i = 0; exAnt->i < exAnt->nbExpAnts; exAnt->i++){
                        if (exAnt->cpt[exAnt->i][l] > c->capacity[l]){exAnt-
>nbViolations[exAnt->i]++;}
                        exAnt->cpt[exAnt->i][l] -= c->requires[c->classOf[exAnt-
>seq[exAnt->i][nbAssignedCars-c->frequency[l]+1]]][l];
                    }
                }
            */
            }
        }
    }
    else if (nbAssignedCars>=c->frequency[l]-1){
        if (cpt[l] > c->capacity[l]) nbViolatedSeq++;
        cpt[l] -= c->requires[c->classOf[seq[nbAssignedCars-c->frequency[l]+1]]][l];
    }
}

```

```

        }
        utilRate[l] = (float)(currentReq[l]*c->frequency[l]) /
        (float)(c->capacity[l]*(c->nbCars - nbAssignedCars - 1));
    }
}

// choice of the last car
seq[nbAssignedCars] = c->firstValue[first]+nbSelected[first];
if (exAnt->nbExpAnts > 0 ){
    // BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates);
    BldExpAntSeq2(exAnt, c->firstValue, c->nbCars, nbAssignedCars, nbCandidates,
seq[nbAssignedCars]);
}

for (l=0; l<c->nbOptions; l++){
    if (c->requires[first][l]==1){
        if (cpt[l]+1 > c->capacity[l]){
            nbViolatedSeq++;
        }
    }
    else if (cpt[l] > c->capacity[l]){
        nbViolatedSeq++;
    }
}
//PrintSequences(seq,exAnt,c);
return nbViolatedSeq;
}
/**
 * @author Christine Solnon
 */
int checkEvalSeq(carSeqInstance* c, int* seq){
    // input: c = a car sequencing instance
    //      seq = a sequence of cars
    // returns the number of constraint violations in seq
    int i, j, k, l, nbViolations;
    k = 0;
    k = k + 1;
    k = k - 1;
    int cpt;
    nbViolations = 0;
    for (i=0; i<c->nbCars; i++){
        k = c->classOf[seq[i]];
        for (l=0; l<c->nbOptions; l++){
            if (i >= c->frequency[l]-1){
                cpt = 0;
                for (j=0; j<c->frequency[l]; j++) cpt += c->requires[c->classOf[seq[i-j]][l];
                if (cpt > c->capacity[l]) nbViolations++;
            }
        }
    }
    return nbViolations;
}
/**
 * @author Christine Solnon
 */
int main (int argc, char *argv[]) {
    // Parameters
    char fileName[1024];
    char buff[1024];

```

```

int verbose = 1;
int displayFreq = 200;
int alpha = 2;
int beta = 6;
float rho1 = 0.01;
int maxCycles = 5000;
int nbAnts = 30;
float tau1Min = 0.01;
float tau1Max = 4;
float rho2 = 0.03;
int TestNo = -1;
int ExpAnt = 0;

int seed = 1;
int strategy = 1;
pid_t pid = getpid();

carSeqInstance c;
int i, j, nbCycles, total;
int **seq; // seq[i] = sequence computed by ant i
int *evalSeq; // evalSeq[i] = nb of violated constraints in seq[i]
int *bestSeq; // best computed sequence
int evalBestSeq; // nb of violated constraints in bestSeq
int bestCycle; // nb of violated constraints in the best sequence of the cycle
float** tau1; // first pheromone structure
float* tau2; // second pheromone structure

float qtyLaid;
int max;
clock_t c0=clock();
if( parse(&alpha, &beta, &rho1, &rho2, &maxCycles, &nbAnts, &tau1Min, &tau1Max,
        &verbose, &displayFreq, fileName, &seed, &strategy, &TestNo, &ExpAnt, argv, argc)
== 1) return(1);

if (verbose==1){

/* 2017-03-18 Kieran O'Sullivan Start */
    sprintf(buff, "%d_%d_parameters.csv", TestNo,pid);
    if(freopen(buff, "w", stdout) !=NULL ){

        fprintf(stdout,
"TestNo,PID,alpha,beta,rho1,rho2,tau1Min,tau1Max,nbCycles,nbAnts,verbose,Display
Frequency,input,seed,strategy,ExpAnt\n");
        fprintf(stdout, "%d,%d,%d,%d,%f,%f,%f,%f,%d,%d,%d,%d,%s,%d,%d,%d\n",
            TestNo,pid,alpha, beta, rho1, rho2, tau1Min, tau1Max, maxCycles, nbAnts, verbose,
displayFreq, fileName, seed, strategy,ExpAnt);

/* 2017-03-18 Kieran O'Sullivan End */

/* Kieran 2017-03-16 fprintf(stdout, "Parameters: alpha=%d beta=%d rho1=%f rho2=%f
tau1Min=%f tau1Max=%f nbCycles=%d nbAnts=%d verbose=%d(%d) input=%s seed=%d
strategy=%d\n",
    alpha, beta, rho1, rho2, tau1Min, tau1Max, maxCycles, nbAnts, verbose, displayFreq,
fileName, seed, strategy);*/
    }

    getData(fileName,&c);

/*Kieran O'Sullivan Addition of Explorer Ant Data Structure */

```

```

explorerAnt exAnt;

exAnt.nbExpAnts = ExpAnt;

if (ExpAnt > 0 ){
    exAnt.TotalNonExpAnts = nbAnts;
    exAnt.NonExpEvalSeq = (int*)calloc(nbAnts,sizeof(int*));
    exAnt.seq = (int**)calloc(exAnt.nbExpAnts,sizeof(int*));
    exAnt.seqPos = (int*)calloc(exAnt.nbExpAnts,sizeof(int*));
    exAnt.curAnt =0;
    exAnt.nbViolations = (int*)calloc(exAnt.nbExpAnts,sizeof(int*));
    exAnt.k = (int*)calloc(exAnt.nbExpAnts,sizeof(int*));
    exAnt.evalSeq = (int*)calloc(exAnt.nbExpAnts,sizeof(int));
    exAnt.VisitedNodes =(int*)calloc(c.nbCars,sizeof(int));

    for (exAnt.i=0; exAnt.i < exAnt.nbExpAnts; exAnt.i++){
        exAnt.seq[exAnt.i] =(int*)calloc(c.nbCars,sizeof(int));
        // Set the explorer ant sequences to -1
        exAnt.seqPos[exAnt.i] = 0;
        for (exAnt.j=0; exAnt.j < c.nbCars; exAnt.j++){
            exAnt.seq[exAnt.i][exAnt.j]=-1;
            exAnt.seqPos[exAnt.i] = 0;
        }
        // exAnt.cpt[exAnt.i] =(int*)calloc(c.nbOptions,sizeof(int*));
        exAnt.k[exAnt.i] = 0;
    }
}

sprintf(buff, "%d_%d_car_file_data.csv",TestNo,pid);
if(freopen(buff, "w", stdout) !=NULL ){
    fprintf(stdout, "Test No,PID,File Name,No Cars,No Opts,No Classes\n");
    fprintf(stdout,
"%d,%d,%d,%s,%d,%d,%d\n",TestNo,pid,fileName,c.nbCars,c.nbOptions,c.nbClasses);

    srand48(seed);

    seq = (int**)calloc(nbAnts,sizeof(int*));

    for (i=0; i<nbAnts; i++){
        seq[i]=(int*)calloc(c.nbCars,sizeof(int));
    }

    bestSeq = (int*)calloc(c.nbCars,sizeof(int));
    evalSeq = (int*)calloc(nbAnts,sizeof(int));
    max = c.nbCars * c.nbOptions;

    tau1 = (float**)calloc(c.nbCars,sizeof(float));
    for (i=0; i<c.nbCars; i++){
        // Addition of Explorer Ants Set visited node to -1
        tau1[i] = (float*)calloc(c.nbCars,sizeof(float));
        for(j=0; j<c.nbCars; j++) tau1[i][j]=tau1Max;
    }
    tau2 = (float*)calloc(c.nbClasses,sizeof(float));
    for (i=0; i<c.nbClasses; i++) tau2[i] = 1;

    evalBestSeq = max;
    total = 0;
    /* 2017-03-18 Kieran O'Sullivan Start */
    sprintf(buff, "%d_%d_cycles.csv", TestNo,pid);

```

```

if (freopen(buff, "w", stdout)!= NULL){ }

    fprintf(stdout, "Test No,PID,Cycle,Walk,Cycle / Walk,New Best\n");
    /* 2017-03-18 Kieran O'Sullivan End */

for (nbCycles=0; ((nbCycles < maxCycles) && (evalBestSeq>0)); nbCycles++) {
    bestCycle = max;
    /* Each ant computes a path */
    for (i=0; ((i<nbAnts) && (evalBestSeq>0)); i++){
        //PrintTau(3,i,c.nbCars,c.nbClasses,tau1,tau2);
        // Start Addition of Explorer Ants Kieran O'Sullivan
        if (ExpAnt > 0){exAnt.NonExpCurAnt=i;}
        // End Addition of Explorer Ants Kieran O'Sullivan
        if (strategy==1){
            evalSeq[i]=walkUtilRate(&c,tau1,alpha,beta,seq[i],&exAnt);
        } else {
            evalSeq[i]=walkDoublePhero(&c,tau1,tau2,alpha,beta,rho2,seq[i],&exAnt);
        }

        if (ExpAnt > 0){
            for (exAnt.i = 0; exAnt.i < exAnt.nbExpAnts; exAnt.i++){
                exAnt.evalSeq[exAnt.i] = checkEvalSeq(&c,exAnt.seq[exAnt.i]);
            }
        }

        total += evalSeq[i];

        if (evalSeq[i]<bestCycle){
            bestCycle = evalSeq[i];
            if (bestCycle<evalBestSeq){
                evalBestSeq = bestCycle;
                if (verbose == 1){
                    // fprintf(stdout, "Cycle,Walk,Cycle / Walk,New Best\n");
                    fprintf(stdout, "%d,%d,%d,%d,%f,%d\n",
                        TestNo,pid,nbCycles+1,nbCycles*nbAnts+i+1,(float)(clock()-
c0)/CLOCKS_PER_SEC,evalBestSeq);
                    /* Kieran 2017-03-16 fprintf(stdout, "Cycle %d / Walk %d / %fs: new best =
%d\n", nbCycles+1,nbCycles*nbAnts+i+1,(float)(clock()-c0)/CLOCKS_PER_SEC,evalBestSeq);*/
                }
                for (j=0; j<c.nbCars; j++){ bestSeq[j] = seq[i][j];}
            }
        }
        // Clear Explorer
        if (ExpAnt > 0 ){
            exAnt.curAnt=0;
            for (exAnt.i = 0; exAnt.i < exAnt.nbExpAnts; exAnt.i++){
                exAnt.seqPos[exAnt.i] = 0;
            }
        }
    }
    if (evalBestSeq==0) {
        break;
    }
    /* evaporation of pheromone for the first pheromone structure */
    for (i=0; i<c.nbCars; i++)
        for (j=0; j<c.nbCars; j++)
            if (tau1[i][j]>tau1Min) tau1[i][j] *= 1-rho1;

    /* pheromone laying on the first pheromone structure */

```

```

    for (i=0; i<nbAnts; i++){
        if (ExpAnt >0){
            if (i < exAnt.nbExpAnts){ // SHIT
                if (evalSeq[i]<exAnt.evalSeq[i]){
                    if (exAnt.evalSeq[i]==bestCycle){
                        qtyLaid = 1.0/(float)(1+exAnt.evalSeq[i]-evalBestSeq);
                        for (j=1; j<c.nbCars; j++){
                            if (tau1[exAnt.seq[i][j-1]][exAnt.seq[i][j]]<tau1Max)
tau1[exAnt.seq[i][j-1]][exAnt.seq[i][j]] += qtyLaid;
                        } // for j
                    } // if evalSeq[i]
                } else {
                    if (evalSeq[i]==bestCycle){
                        qtyLaid = 1.0/(float)(1+evalSeq[i]-evalBestSeq);
                        for (j=1; j<c.nbCars; j++){
                            if (tau1[seq[i][j-1]][seq[i][j]]<tau1Max) tau1[seq[i][j-1]][seq[i][j]] +=
qtyLaid;
                        } // for j
                    } // if evalSeq[i]
                } // end else
            } // if (i < exAnt.nbExpAnts)
        } else {
            if (evalSeq[i]==bestCycle){
                qtyLaid = 1.0/(float)(1+evalSeq[i]-evalBestSeq);
                for (j=1; j<c.nbCars; j++){
                    if (tau1[seq[i][j-1]][seq[i][j]]<tau1Max) tau1[seq[i][j-1]][seq[i][j]] +=
qtyLaid;
                } // for j
            } // if evalSeq[i]
        } // end else
    } // for i

    if (((nbCycles+1)%displayFreq==0) && (verbose==1)){
        sprintf(buff, "%d_%d_cycles_avg.csv", TestNo,pid);
        if (freopen(buff, "w+", stdout)!= NULL){ }

        fprintf(stdout, "TestNo,PID,Cycle,Average Eval\n");
        printf(stdout, "%d,%d,%d / %f,%f\n",
            TestNo,pid,nbCycles+1,(float)(clock()-
c0)/CLOCKS_PER_SEC,(float)(total)/(float)(displayFreq*nbAnts));
        /* Kieran 2017-03-16 fprintf(stdout, "Cycle %d / %fs : average eval = %f\n",
nbCycles+1,(float)(clock()-c0)/CLOCKS_PER_SEC,(float)(total)/(float)(displayFreq*nbAnts));*/
        total = 0;
    }
};

sprintf(buff, "%d_%d_solution.csv", TestNo,pid);
if(freopen(buff, "w+", stdout) !=NULL ){ }

fprintf(stdout,"TestNo,PID,No Violations,File Name,Solution\n");
fprintf(stdout, "%d,%d,%d,%s,",TestNo,pid,checkEvalSeq(&c,bestSeq),fileName);
for (i=0; i<c.nbCars; i++) { displayCar(bestSeq[i],&c); }

sprintf(buff, "%d_%d_violations.csv", TestNo,pid);
if(freopen(buff, "w+", stdout) !=NULL ){ }

fprintf(stdout,"TestNo,PID,File Name,Ant No,No Violations,");
if (ExpAnt > 0){
    for (exAnt.i = 0; exAnt.i < exAnt.nbExpAnts; exAnt.i++){

```

```

        fprintf(stdout,"XAnt No,No Violations,");
    }
}
for (i=0; i<nbAnts; i++){
    fprintf(stdout,"%d,%d,%s,%d,%d","TestNo,pid,fileName,i,evalSeq[i]);
    if (ExpAnt > 0){
        for (exAnt.i = 0; exAnt.i < exAnt.nbExpAnts; exAnt.i++){
            fprintf(stdout,"%d,%d",exAnt.i,exAnt.evalSeq[exAnt.i]);

        }
        fprintf(stdout,"\n");
    }
}
;
return(0);
}

```