UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

DISCRETE FLOWER POLLINATION ALGORITHM FOR SOLVING THE SYMMETRIC TRAVELING SALESMAN PROBLEM

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfillment of the requirements for the degree of Masters of Science in Engineering (Electrical).

Ryan Strange - Student No. 537434
Research Supervisor: Prof. Ling Cheng

This research was carried out with the financial assistance of the National Research Foundation (NRF). Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Johannesburg, 2017

Declaration

I declare that this dissertation is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Masters of Science in Electrical Engineering to the University of the Witwatersrand, Johannesburg, South Africa. It has not been submitted before for any degree or examination at any other university.

Candidate Signature: .....

Name: .....

Date: (Day).....(Month).....(Year).....

Abstract

The Travelling Salesman Problem (TSP) is an important NP-hard combinatorial optimisation problem that forms the foundation of many modern-day, practical problems such as logistics or network route planning. It is often used to benchmark discrete optimisation algorithms since it is a fundamental problem that has been widely researched. The Flower Pollination Algorithm (FPA) is a continuous optimisation algorithm that demonstrates promising results in comparison to other well-known algorithms. This research proposes the design, implementation and testing of two new algorithms based on the FPA for solving discrete optimisation problems, more specifically the TSP, namely the Discrete Flower Pollination Algorithm (DFPA) and the iterative Discrete Flower Pollination Algorithm (iDFPA). The iDFPA uses two proposed update methods, namely the Best Tour Update (BTU) and the Rejection Update (RU), to perform the iterative update process. The two algorithms are compared to the Ant Colony Optimisation's (ACO) $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) as well as the Genetic Algorithm (GA) since they are well studied and developed. The DFPA and iDFPA results are significantly better than the GA and the iDFPA is able to outperform the ACO in all tested instances. The iDFPA with 300 iterations was able to achieve the optimal solution in the Berlin52 benchmark TSP problem as well as have improvements of up to 4.56% and 41.87% compared to the ACO and GA respectively. An analysis of how the RU and the annealing schedule used in the RU impacts on the overall results of the iDFPA is given. The RU analysis demonstrates how the annealing schedule can be manipulated to achieve certain results from the iDFPA such as faster convergence or better overall results. A parameter analysis is performed on both the DFPA and iDFPA for different TSP problem sizes and the suggested initial parameters for these algorithms are outlined.

In loving memory of David Strange

This dissertation is in recognition of his constant dedication to his family and a product of his support and belief in his children.

Acknowledgments

The author would firstly like to acknowledge his family for the constant support provided throughout the course of his academic career which has led to the completion of this master's dissertation. His mother, Vanessa, and his sisters, Ashleigh and Megan, have provided endless support throughout this difficult period after the passing of their father, David Strange. The friends and family that have stood with the author and his family throughout this process are also acknowledged for their support.

The author is also thankful for the guidance and supervision provided by Professor Ling Cheng throughout the research process as well as the facilities provided by the School of Electrical and Information Engineering at the University of the Witwatersrand. Furthermore, the aid of Kerren Ortlepp, Ellen de Mello Koch and Yves-Francois Rivard with proofreading of this dissertation is highly appreciated.

Finally, the author acknowledges the National Research Foundation (NRF) and the Center for Telecommunications Access and Services (CeTAS) as well as the University of the Witwatersrand scholarship office who have given their financial support in the last two years through bursaries and merit awards.

Table of Contents

Declaration i

Abstract ii

Dedication iii

Acknowledgments iv

List of Figures vii

List of Tables ix

List of Algorithms x

List of Abbreviations xi

List of Symbols xii

Chapter 1: Introduction 1

1.1 Research Question 2
1.2 Research Significance 2
1.3 Scope and Research Objectives 3
1.4 Research Achievements 3
1.5 Dissertation Organisation 4

Chapter 2: Literature Review **6**

2.1 Travelling Salesman Problem 6
2.2 Types of Algorithms for TSP 8
2.3 Nature-Inspired Heuristic Algorithms 9

Chapter 3: Preliminaries **12**

3.1 Travelling Salesman Problem 12
3.2 Flower Pollination Algorithm 13
3.3 Genetic Algorithm 15
3.4 Ant Colony Optimisation 17
3.5 Simulated Annealing 20

Chapter 4: Discrete Flower Pollination Algorithm for solving the Symmetric Travelling Salesman Problem **23**

4.1 Discrete Flower Pollination Algorithm 24
4.1.1 Algorithm Description 24
4.1.2 Local and Global Search Algorithms 26
4.2 Iterative Discrete Flower Pollination Algorithm (iDFPA) 28
4.2.1 Best Tour Update 30
4.2.2 Rejection Update 30
4.3 Results 32
4.3.1 Experiment Setup 32
4.3.2 DFPA Results 33
4.3.3 iDFPA Results 33
4.3.4 Parameter Selection 42

Chapter 5: Conclusion **47**

5.1 Research Summary 47
5.2 Contributions 47
5.3 Recommendations and Possible Future Work 48
5.4 Conclusion 49

References **50**

A. Complexity Analysis **1**

List of Figures

2.1 Five-node TSP complete graph and an example of a valid path. 7

3.1 Probability Density Function of the Levy Flight Distribution where $\lambda = 1.5$. 14

3.2 A set of diagrams showing the process of ants exploring for food (F) and updating pheromone levels along various paths until the shortest path is determined between the colony (C) and the food source. 19

4.1 System flowchart of the DFPA algorithm. 25

4.2 Radial distance threshold clustering used in the local search: black node is the previous node, shaded nodes are considered for the local search, others are excluded. 28

4.3 System flowchart for the iDFPA Algorithm. 29

4.4 Minimum tour distance vs. acceptance ratio plot with annotated phases. . . 38

4.5 Tour distance of different values for the annealing schedule in rejection update of iDFPA for 50 iterations on the Pr264 problem set. 40

4.6 iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Berlin52 problem set. 41

4.7 iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Eil76 problem set. 42

4.8 iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Pr264 problem set. 43

4.9 A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Berlin52 problem set. 44

4.10 A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Eil76 problem set 45

4.11 A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Pr264 problem set 46

0.1 Complexity of significant elements of DFPA and iDFPA algorithms 2

List of Tables

1 Selected simulation problem sets' properties. 33

2 Comparative tour distance results for three TSP problem sets. 34

3 Berlin52 comparative percentage results 35

4 Eil76 comparative percentage results 35

5 Pr264 comparative percentage results 36

6 Parameters selected for iterative simulations. 36

7 Generalised Parameter 46

List of Algorithms

1 2-Opt Search 22
2 Discrete Flower Pollination Algorithm 27
3 Rejection Update: Annealing Rejection 31

List of Abbreviations

ACO	**A**nt **C**olony **O**ptimisation
AS	**A**nt **S**ystem
bsf	*b*est-*s*o-*f*ar
BTU	**B**est **T**our **U**pdate
CTSP	**C**olored **T**raveling **S**alesman **P**roblem
DFPA	**D**iscrete **F**lower **P**ollination **A**lgorithm
FPA	**F**lower **P**ollination **A**lgorithm
GA	**G**enetic **A**lgorithm
iDFPA	**i**terative **D**iscrete **F**lower **P**ollination **A**lgorithm
$\mathcal{MMAS}$	$\mathcal{MAX}-\mathcal{MIN}$ $\mathcal{A}$nt $\mathcal{S}$ystem
mTSP	**m**ultiple **T**raveling **S**alesman **P**roblem
PSO	**P**article **S**warm **O**ptimisation
PTSP	**P**robabilistic **T**raveling **S**alesman **P**roblem
PMX	**P**artially **M**atched crossover
RU	**R**ejection **U**pdate
SA	**S**imulated **A**nnealing
TSP	**T**raveling **S**alesman **P**roblem
TSPM	**T**raveling **S**alesman **P**roblem with **M**ultiple visits

List of Symbols

Symbol	Description
A	Normalisation constant
α, β, γ, e	Control parameters
C	Cost matrix
c_{ij}	Cost from node i to node j
ΔC	Change in cost
D	Distance/weight of edges
d_{ij}	Single distance between node v_i and v_j
$\mathbf{d}$	Set of tour distances
$\mathbf{d}'$	A subset of $\mathbf{d}$ containing the Rejection Update accepted tour distances
d_{prev}	The previous iteration's best tour distance
$d_{\mathbf{s}_j}$	Tour distance of tour $\mathbf{s}_i$
$d_{\mathbf{s}^*}$	Current best tour distance
$\Delta dist$	$d_{\mathbf{s}_j} - d_{prev}$
E	Set of edges
ΔE	Change in energy
ϵ	Uniform distribution variable
η	Desirability level
F_L	Continuous Levy Distribution CDF
Γ	Standard Gamma Function

G	TSP graph
$\mathbf{g}^*$	Current best solution
i, j, k	Counting variables
k_b	Boltzmann constant
$\mathcal{L}$	Levy distribution variable
$\mathbf{L}(\bar{V})$	Discrete Levy Distribution
l^*	Node index chosen by the Categorical Distribution
λ	Levy constant
m	Number of agents
N	Number of iterations
N_k^i	Set of nodes not visited by ant k when at node i
N_{curr}	Current iteration number
n	Total number of nodes
p	Probability of distance being accepted by the Rejection Update
P	Probability
$P(\Delta E)$	Probability of ΔE
p_{ij}	Probability of choosing node j from node i
Φ	Maximum input value for the Discrete Levy Distribution
ρ	Switching probability
q	Specifies the exponential function shape for RU
r	Random real number
r_{dist}	Radial threshold distance
$\mathbf{S}$	Set of tours
$\mathbf{s}_i$	Ordered sequence (Tour)
s_i^j	A node on the tour $\mathbf{s}_i$
$\mathbf{s}^*$	Current best tour
$\mathbf{S}'$	A subset of $\mathbf{S}$ containing the Rejection Update accepted tour
T	Temperature
T_{curr}	Current iteration's annealing value
t_i	Individual temperature in the annealing schedule

τ	Pheromone level
$\Delta\tau$	Change in pheromone level
τ_{max}, τ_{min}	Maximum and minimum pheromone levels for $\mathcal{MMAS}$
U	Uniform distribution
V	Set of nodes
V'	Temporary set of nodes
$\|V'\|$	Cardinality of V'
$\bar{V}$	Ordered set of V' nodes
V''	Local node subset of V'
v_i	Single node in V
v_{prev}	Previous node
w	Specifies the utilised area of the exponential function for RU
$\{\mathbf{x}_i\}$	Set of solutions

Chapter 1: Introduction

The Travelling Salesman Problem (TSP) is a discrete combinatorial optimisation problem that has been proven to be NP-hard meaning that it cannot be solved in polynomial time. The TSP is significant as it is the fundamental underlying problem in many real-life applications such as vehicle routing [1], logistics and planning [2] as well as manufacturing of microchips and other circuitry [3]. Different variations of the basic TSP constraints allow for a large variety of applications to be optimised. The TSP is often used to benchmark new optimisation algorithms against other optimisation algorithms [4] since it is a fundamental problem that has been widely researched. The TSP is easily stated and understood which allows for the algorithm's behaviour to be observed without being obscured by too many technicalities [5]. The TSP is a standard benchmark that allows for efficient algorithms to be developed and benchmarked for a variety of other problems by altering and introducing new constraints to fit specific criteria [4, 5]. It is widely studied and commonly used as a benchmark problem, therefore, allowing for new algorithms to be comprehensively compared to existing, well-developed algorithms.

There are two main types of optimisation algorithms namely optimal and heuristic algorithms. Optimal algorithms find the best solution to a specific problem by exploring all the possible outcomes of the problem. Heuristic algorithms, unlike optimal algorithms, find a solution amongst all possible solutions without the guarantee that the optimal solution will be found. Nature-inspired algorithms mimic flexibility and robustness of the natural processes [6] in order to approach the optimal solutions. These algorithms are beneficial as they are able to adapt to a variety of problem sets without intervention. A variety of nature-inspired heuristic algorithms, including the Genetic Algorithm (GA), Ant Colony Optimisation (ACO) and Simulated Annealing (SA), have been successfully used to solve optimisation problems including the TSP.

The Flower Pollination Algorithm (FPA) is a continuous optimisation heuristic algorithm, implemented in 2012, that mimics the natural flower pollination process. It has shown good results compared to existing algorithms such as the GA and Particle Swarm Optimisation (PSO) [7] for a variety of continuous optimisation problems. The FPA was able to achieve results 82% and 61% better than the GA and PSO respectively for the Shubert test function with a 100% success rate on solving the function compared to the 89% and 92% for the GA and PSO respectively [7]. Similar results were seen for the nine other continuous test functions [7]. A few discrete versions of the FPA have been proposed for specialised discrete optimisation problems and initial results have supported the notion that the FPA, with the random nature of the underlying Levy Flight Distribution, provides a good basis for optimisation problems.

1.1 Research Question

"Can the Flower Pollination Algorithm (FPA) be implemented to generally determine a sub-optimal solution for discrete optimisation problems, but more specifically the Travelling Salesman Problem? If so, how does it perform compared to existing solutions as well as to the optimal solution?"

1.2 Research Significance

The significance of being able to create new heuristic algorithms that can successfully optimise combinatorial optimisation problems to a better degree of suboptimal solutions with less computational complexity allows for real-world applications of these problems, such as transportation and logistics systems, to be made more efficient. An example would be for public transportation, by providing a bus driver with shorter routes not only does the bus company save money through lower fuel consumption and shorter travel times but the consumer will benefit from having shorter travel times. In the manufacturing of circuit boards, by being able to program a machine to be milliseconds faster on an individual board leads to significant time savings overall. These are two real-world examples of the direct application of the TSP to demonstrate the significance improved algorithms can have, however, the TSP is just a benchmark problem and therefore applying the new algorithm to other discrete optimisation problems has the potential to lead to further applications in real-world scenarios.

1.3 Scope and Research Objectives

The main focus of this research is to develop a discrete version of the FPA to solve the TSP and, more generally, discrete optimisation problems.

The first objective is to identify possible areas of the TSP and nature-inspired heuristic algorithms that could be improved upon in order to create a new discrete optimisation algorithm.

The second objective of the research is to design and implement a discrete version of the FPA. This objective comprises of designing and implementing both a single iteration discrete FPA (DFPA) as well as an iterative DFPA (iDFPA).

The final objective of the research is to compare the algorithms against popular, well researched and benchmarked algorithms. This is done by running simulations of both the DFPA and iDFPA as well as the Genetic Algorithm (GA) and Ant Colony Optimisation (ACO) $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) algorithms on a variety of TSP problem sets and performing a comparison. These problem sets are different in size as well as node orientation in order to compare the performance of the two new algorithms with the two well studied and developed algorithms for a variety of types of TSPs.

Sufficient information is provided about the existing algorithms, implementation of the new algorithms as well as the simulation process for a reader proficient in the field to fully understand all components of the research. Results of the comparisons, analyses of the results, general parameter selection for the new algorithms as well as possible future work are presented.

1.4 Research Achievements

In the research two algorithms, namely the DFPA and iDFPA, are successfully designed, implemented and benchmarked. The DFPA is able to indicate the best performance that can be achieved for a single iteration. The iDFPA uses "knowledge" gained from previous iterations to iteratively improve the results and converge on a sub-optimal solution.

In order to evaluate the algorithms, they are compared to well-known and researched algorithms, namely, the GA and ACO. The DFPA and iDFPA are significantly better than the GA with respect to final sub-optimal TSP value as well as the speed of

convergence. The iDFPA was also able to outperform the ACO in all tested instances.

The iDFPA consists of two iterative methods, namely, the Best Tour Update (BTU) and Rejection Update (RU), an analysis of the RU's acceptance ratio and how it affects the overall result has been done and it was found that the annealing schedule can be adjusted in order to manipulate the results of the iDFPA. For example, the annealing schedule can be altered to give a faster-converging result with the trade-off being that the result is not necessarily the best sub-optimal route.

A parameter analysis was done for the new algorithms and shows the suggested initial parameter settings that should be used for various problem sizes in order to achieve the decent results. These are just initial suggestions where specific optimisation of parameters would need to be done for the specific problem.

1.5 Dissertation Organisation

The following section provides an overview of this masters dissertation in order to outline to the reader the structure of the research and what can be expected.

Literature Review (Chapter 2)

This chapter provides a literature review for the TSP, the TSP variations and the types of algorithms that can be used to solve these problems. Optimal and heuristic algorithms are discussed with a special focus on nature-inspired algorithms. The FPA continuous optimisation algorithm, as well as recent research into the extensions and discrete applications, have also been investigated as it forms the basis for the rest of the research.

Background (Chapter 3)

The chapter presents detailed information about some existing methods and algorithms, and their implementations used to solve the TSP which were used to develop the DFPA and iDFPA. The GA and ACO's implementations that are used for the comparison with the DFPA and iDFPA have also been discussed.

Discrete Flower Pollination Algorithm for solving the Symmetric Travelling Salesman Problem (Chapter 4)

This chapter is based on a paper which has been submitted to the *IEEE Transactions on Cybernetics* is presented in this chapter. The paper details the full implementation of the two proposed algorithms, namely the DFPA and iDFPA. The results of the comparative study done on these two algorithms with the Genetic Algorithm and the ACO's $\mathcal{MMAS}$ through various simulations are also presented. Finally, the chapter gives a generalised parameter selection for using the algorithm and an analysis of the annealing schedule utilised for the iDFPA.

Conclusion (Chapter 5)

The final chapter provides a conclusion to the research which includes a summary of the research, the achievements made as well as possible future work that is identified for further research.

Chapter 2: Literature Review

2.1 Travelling Salesman Problem

The TSP was first defined and studied by mathematician Karl Menger in Vienna in 1930 [2], earlier mentions of the problem had been made in the 1800s but no formal mathematical definition had been formulated until Menger. Flood of Princeton University was the first mathematician to popularise the topic of TSP to his colleagues at the RAND Corporation in the 1940s [8]. Solutions to the TSP appeared from the 1950s onwards and in 1972 Kamp proved that the TSP was an NP-Hard problem since it could not be solved in polynomial time [8]. An NP-Hard or NP-Complete problem is classified as a problem which cannot be solved in polynomial time, they usually consist of combinatoric optimisation problems such as the TSP [9]. The TSP has become a classic combinatorial optimisation problem [10] with modern researchers continuously attempting to improve on existing results by creating new algorithms or modifying previous algorithms. The appeal of solving the TSP comes from the fact that the problem is easily formulated and understood but a general solution has not been found and is extremely difficult to obtain [2].

The Travelling Salesman Problem is a classic NP-Hard or NP-Complete problem [11] which requires a minimum distance between nodes to be determined [9, 12, 13]. The TSP consists of n nodes with a complete graph of distances between nodes, the objective is to find the minimum distance route from any node through all other nodes and back to the starting node where each node is visited exactly once. Fig. 2.1 shows a simple TSP example as well as a valid route in the TSP [14].

The TSP has a variety of applications in a vast number of fields such as logistics [2] and vehicle routing [1], network routing, manufacturing of microchips and circuitry [15] as well as job scheduling [3]. It is a fundamental TSP problem and is seen as a subproblem in many other combinatorial optimisation problems, for example, genome se-

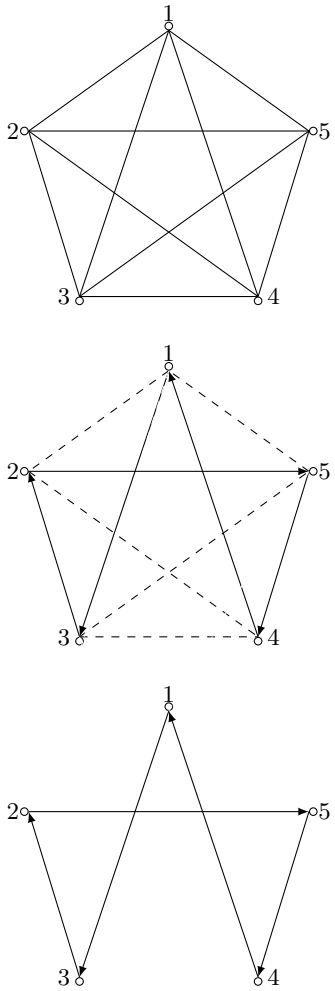


Figure 2.1: Five-node TSP complete graph and an example of a valid path.

quencing [2], and therefore by being able to solve it, even sub-optimally, faster and to better accuracy helps to improve the results of many real-life tasks.

There are several types of TSP problems including symmetric TSP, asymmetric TSP and TSP with multiple visits (TSPM) [9, 14]. The TSPM is the same fundamental problem except the salesman is not constrained into visiting each node only once but merely has to visit each node at least once. Solving the TSP is useful because a number of practically relevant problems can be shown to be equivalent to an altered version of it. Some other variants of the TSP include the multiple Travelling Salesman Problem (mTSP), the Colored Travelling Salesman Problem (CTSP), the Probabilistic Travelling Salesman Problem (PTSP) and Dubins' TSP. The mTSP assumes that there are several salesmen that start at different nodes, have to complete a route and return to their starting node with the requirement being that each node has to be travelled to exactly

once and the overall combined route distance needs to be minimised [16]. The mTSP has many real-life applications such as optimising school bus routes [16]. The CTSP is an extension of the mTSP where the salesmen have both individual tasks but shared tasks as well [17], this type of problem occurs when an overlap of the routes is required for example in multi-machine engineering systems [17]. The PTSP has applications in science and engineering when the performance is affected due to the uncertainty of events [18]. The PTSP helps with modelling situations such as job sequencing when there is changeover expense, e-commerce for home delivery and stochastic services that involve demand pickup and delivery [18]. Dubins' TSP is an extension of the TSP that is used for real-time surveillance applications in order to map routes to avoid line-of-sight surveillance (i.e. nodes with a radius) [4].

2.2 Types of Algorithms for TSP

There are two main types of algorithms that are used to solve the TSP, namely optimal algorithms and heuristic algorithms. Optimal algorithms such as brute force [3], branch-and-bound [3], dynamic programming method [19], cutting plane method [19] and integer programming method [20] have been implemented to find the optimal tour (solution) for the TSP.

The brute force method involves computing and evaluating all tour permutations of the TSP and determining which tour has the minimum distance. This method is guaranteed to give you the optimal solution but it is computationally expensive since every tour in the problem domain needs to be evaluated in order to determine the optimal solution. For example for a TSP with 10 nodes, there are $\frac{1}{2}(10-1)! = 181440$ possible tours and all of these tours need to be evaluated in order to determine the optimal solution.

The branch-and-bound method uses an estimated upper bound to determine which branches of the problem solution can be ignored since the optimal solution is guaranteed not to be included in those branches [12, 21]. The algorithm calculates the distance of the branch as it descends and once the bound has been reached the algorithm stops on the current branch at that point and then starts on a new branch [12]. The main issue with this method is that the upper bound is difficult to estimate for the TSP and either is too strict which results in the optimal solution being ignored or too lenient resulting in

a similar issue as the brute force method where the method is computationally expensive due to the number of solutions being computed.

All these optimal algorithms have a common downfall in that the increase in computation time as the number of nodes increases make them impractical for real-life applications. This downfall has led to an increase in the research into heuristic algorithms which can solve the TSP sub-optimally within a reasonable amount of time, a trade-off between computational expense and optimality of the result takes place when considering these types of algorithms. Commonly used heuristic methods include nearest insertion method, double minimum spanning tree [20] and nearest neighbour (greedy) algorithms [22].

The simplest type of heuristic algorithm is the greedy or nearest neighbour algorithm [22]. The algorithm determines a sub-optimal solution to the TSP problem by randomly selecting a starting node and then developing a tour by selecting the next node based on which of the remaining nodes is the closest to the current node, i.e. which node has the shortest distance from the current node [13].

All of these common heuristic methods have, however, failed to achieve TSP solutions that have a sufficient degree near to the optimal tour distance [20]. Therefore research focus moved to nature-inspired algorithms in order to overcome this problem with the common heuristics. They are of particular interest for optimisation problems, such as the TSP, since these nature-inspired algorithms adapt to changes in the environment automatically.

2.3 Nature-Inspired Heuristic Algorithms

Many modern heuristic algorithms have been inspired by natural processes or naturally occurring phenomenon [23]. Nature-inspired algorithms are appealing since the utilised processes have been optimised naturally through evolution which provides a good starting platform for solving difficult problems. This section presents a variety of nature-inspired algorithms and their benefits as well as the new Flower Pollination Algorithm (FPA) that becomes the main focus of this research.

Swarm intelligence algorithms attempt to exploit the collective problem-solving abilities exhibited in naturally occurring swarms [24], such as bees and ants, as an optimisation property. These algorithms mimic the self-organisation, flexibility and adaptability

of the swarms in order to optimise the specified problem [6, 24]. Swarm intelligence has led to the creation of algorithms such as the well-known Ant Colony Optimisation (ACO) [5] and Particle Swarm Optimisation (PSO) [23] to the lesser known Bee Colony [4, 25], Firefly [26], Wasp [27], Fruit Fly [28], Wolfpack [29, 30] and African Buffalo algorithms [31].

The ACO provides a common framework under which ant colony algorithms are able to be developed [24]. The algorithms use the concept of pheromone trails as a form of communication between ants to optimise the problems [32]. The optimisation occurs based on the experience gained through the altering of the pheromone trail levels as the life cycles progress [32].

The Bee Colony algorithms use behaviour of bees and their interactions in order to solve combinatorial optimisation problems. Some algorithms use the identification of nectar sources and the interactions of the bees and then the solution is obtained by analysing the intensity of the bee interactions at nectar sources [33]. Others use the reproduction process of the bees for the optimisation [33].

Other nature-inspired algorithms that have been developed for optimisation problems include Neural Networks [34], Simulated Annealing (SA) [35, 36], Genetic Algorithms (GA) [11, 37–39] and the River Formation Algorithm [23]. GA algorithms are one of the most popular nature-inspired algorithms that use genetic science and natural selection in order to simulate natural evolution in order to solve optimisation problems [38]. SA algorithms mimic the annealing process of metal in order to optimise problems [35]. They have been shown to achieve good optimisation results in numerous optimisation problems as they allow for a broad range of solutions to be explored before converging on a solution.

Yang has taken a special focus in developing nature-inspired heuristic algorithms with some notable ones including the firefly algorithm [26] as well as the cuckoo search [40]. In 2012, the FPA was developed for continuous optimisation problems [7], this algorithm is described further in Section 3.2. Extensions of the FPA include applying the FPA to multi-objective problems [41] as well as the Binary FPA which was applied to feature selection on a binary lattice search space [42]. Many modifications have also been made on the FPA such as adding chaos theory in the Improved FPA with Chaos algorithm which was used to solve definite integrals to evaluate distribution functions [43]. Other

modifications include multiple hybridisations with GA, PSO or K-Means versions of the algorithms [44]. Some research into discrete FPA algorithms has been done specifically for the Graph Coloring Problem [45] and Resource Constrained Problems [46]. In both discrete cases, the results demonstrated that for their specific applications the discrete algorithm was able to outperform the majority of comparison algorithms.

Chapter 3: Preliminaries

3.1 Travelling Salesman Problem

The TSP is a combinatorial optimisation problem that is deceptively easy to state and extremely difficult to solve. The TSP comprises of n nodes or cities where the objective is to find the shortest tour distance for a "salesman" to travel from any starting node through every other node exactly once and then return to the starting node. As the number of nodes increases so the number of possible permutations increases factorially. The problem has been proven to be an NP-Hard problem meaning that it cannot be solved in polynomial time.

The TSP can be defined mathematically using graph theory as $G = (V, E, D)$ where V is the set of nodes, E is the set of edges and D is the weight of the edges [9]. For the traditional TSP, all elements of E have a corresponding element in D since it is a complete weighted undirected graph. The object of TSP is to find a solution from the solution set such that the total cost is minimised as given in (1) [11, 14].

$$argmin[d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})] \tag{1}$$

Where:

n is the total number of nodes in the problem set;

v_i is the specific node at i;

$d(v_i, v_j)$ is the distance/weight from node i to j.

In the symmetric TSP, the distance from node X to Y is the same as from Y to X. This is the most fundamental form of the TSP and has $\frac{1}{2}(n-1)!$ possible solutions. In asymmetric TSP the distance from node X to Y is different to the distance from Y to X, this therefore has $(n-1)!$ possible solutions. In this research the symmetric TSP will be focused on and all results will be based on symmetric problem sets. The asymmetric

TSP will not be considered in this research and will be considered future work.

3.2 Flower Pollination Algorithm

The Flower Pollination Algorithm (FPA) is an optimisation heuristic algorithm developed by Yang in 2012 [7]. The FPA is designed for continuous optimisation problems such as the Ackley, de Jong and Easom's functions. The results presented by Yang showed that the FPA is more efficient than the GA as well as the PSO algorithm for a classic pressure vessel design problem [7].

The FPA is based on the natural process of flower pollination where there are biotic pollination (cross-pollinators) and abiotic pollination (self-pollinators) [7]. Biotic pollination is aided by pollinators such as bees and birds and can occur over long distances. Abiotic pollination occurs either within the flower itself or in the direct vicinity of the flower.

The following rules are applied in FPA in order to optimise the solution [7]:

1. Biotic pollination is a global process and therefore provides the possibility of the solution jumping from a converging state to a less favourable one with the intention of finding a better solution. This is done using the Levy Flight Distribution, as shown in Fig. 3.1. The Levy distribution is used since it characterises the natural flight of insects and birds. Close trips are therefore favoured but there is still a possibility that a long distance flight could be made, this is because of the heavy tailed nature of the distribution [47].

2. Abiotic pollination is the local process where the solution continues to converge based on its current trajectory, this is done using a uniform distribution.

3. A switching probability, $\rho \in (0, 1)$, controls the pollination process between global and local pollination. Local pollination will make up the significant portion of the pollination process because it naturally occurs more often.

For optimisation pollination, the process can simply be understood as two pollination operations, local and global, that are performed interchangeably [48]. Each individual flower is considered a solution to the problem. The constancy of each flower (solution), which is defined as the likelihood of a pollinator choosing the flower, is used as

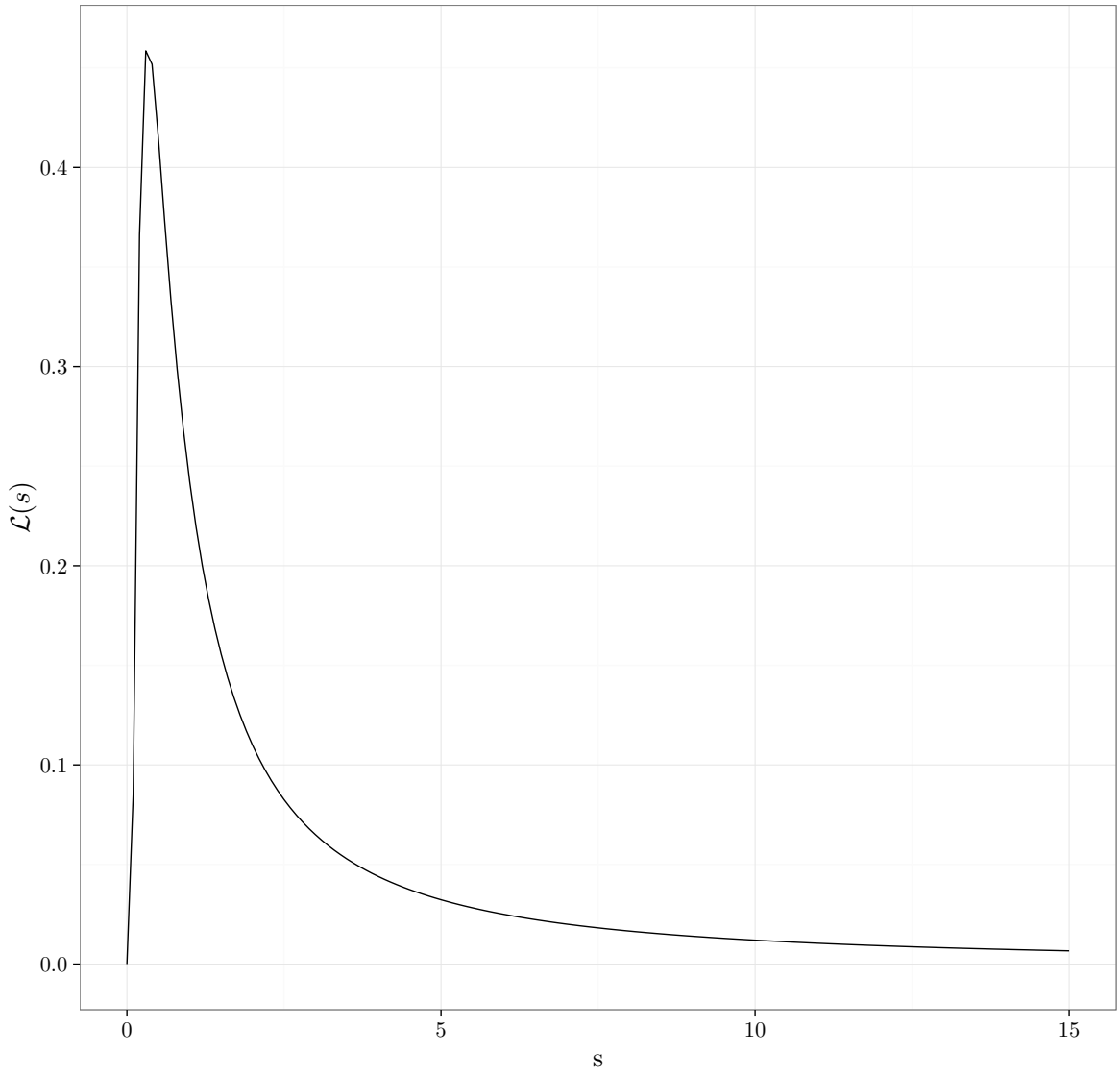


Figure 3.1: Probability Density Function of the Levy Flight Distribution where $\lambda = 1.5$.

the flower's fitness. The global pollination operation allows pollinators to travel long distances to pollinate flowers with higher constancies [41, 48]. The local pollination operation constrains the pollination within a limited range [41, 48].

A significant assumption made in the FPA is that each plant can only have one flower which simplifies the implementation. This assumption allows for a set of solutions, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, where each solution $\mathbf{x}_i = \{x_i^1, x_i^2, \ldots, x_i^N\}$ is a "plant" in the global sense and represents a set of solutions to the problem for the N iterations [7]. The solutions in $\mathbf{x}_i$ are set to random initial solutions and the current best $\mathbf{g}^*$ is identified.

The switching probability, ρ, is defined from a uniform distribution (0,1). For each of the solutions in the solution set a random real number, $r \sim U(0,1)$, is generated. If $r > \rho$ the solution is updated according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathcal{L}(\mathbf{g}^* - \mathbf{x}_i^t), \tag{2}$$

where $\mathbf{x}_i^{t+1}$ and $\mathbf{x}_i^t$ are the new and current solution in $\mathbf{x}_i$ respectively. Here $\mathbf{g}^*$ is the current best solution. The pollination strength $\mathcal{L}$, which is essentially a step size, is the Levy Flight Distribution variable defined as:

$$\mathcal{L}(s) \sim \frac{\lambda\Gamma(\lambda)\sin(\frac{\pi\lambda}{2})}{\pi s^{1+\lambda}},$$

where Γ is the standard Gamma function, s is a large step size such that $s \gg s_0 > 0$, and λ is a constant chosen to be 1.5 for the FPA [7]. The initial step size $s_0 = 0.1$ as specified in [41].

Otherwise, it is updated according to

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \epsilon(\mathbf{x}_j^t - \mathbf{x}_k^t), \tag{3}$$

where ϵ is a random step size drawn from a uniform distribution and $\mathbf{x}_j^t$ and $\mathbf{x}_k^t$ are two different solutions chosen at random from the current iteration's solutions.

Once all the new solutions from the current iteration have been determined, each solution x_i^{t+1} is compared to the previous solution x_i^t, if it is better then it is updated, otherwise, it stays the same. The $\mathbf{g}^*$ is updated to the new current best and the process is repeated for a predefined number of iterations.

The combination of the local and global pollination with the random switching between the two allows for a wide variety of the problem space to be explored, increasing the chance of achieving a good result and reducing the likelihood of falling into a local optimum.

3.3 Genetic Algorithm

Genetic Algorithms (GAs) are the most commonly used optimisation algorithms. They are based on the concept of natural evolution by employing a *survival of the fittest* concept where only the stronger members of the population survive into a new

generation [38]. Generally, GAs consist of the following steps in order to solve the optimisation problems [37, 38]:

1. Encoding: The process of determining a method of encoding the population in such a way as to represent the problem being solved. In the TSP, the "gene" of an individual comprises of a sequence of numbers where each number represents a node in the problem. The sequence, therefore, can only contain a number once and must contain every number in order to meet the constraints of the TSP and represent a valid solution. An example would be an individual with the gene $(1, 3, 5, 2, 4, 6)$ would mean that a solution to the TSP is to travel along the route: $1 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 1$.

2. Evaluation: An initial population, of size m, is selected by randomly generating their starting city and using a greedy algorithm to determine a valid solution to make up their gene. The fitness of each member of the population is then evaluated according to (4). Once each member of the population has been evaluated, they are ranked according to the fitness function from highest (shortest tour distance) to lowest (longest tour distance).

$$f_i = \frac{1}{\sum_{k=1}^{n} d_{k,k+1} + d_{1,k}} \tag{4}$$

   Where:

   f_i is the fitness of the i'th member of the population (m);

   n is the total number of nodes;

   $d_{k,k+1}$ is the distance from node k to $k+1$.

3. Crossover is where strong members of the population are combined to create potentially stronger new generations. Typically the two strongest members of the current population are used to create two new members which will replace the two weakest members of the same population. A variety of crossover methods which have been designed for TSP are presented in [38]. The main issue with the GA and TSP is that when the crossover is performed the new members have to be valid solutions to the TSP problem which increases the complexity of generating the new members.

Partially matched crossover (PMX) is the most common crossover method for the TSP which employs a two split method of creating the children. The parent genes are divided into three parts by randomly generating two positions, the chromosomes in the middle section of the division represent the chromosomes which need to be switched in order to create the children [49]. As shown in (5), the two positions are 2 and 5, the middle sections of the genes are swapped, as seen from the parent to intermediate stage, after the swap the genes do not represent valid TSP routes and therefore the duplicated nodes on the outside of the middle section are swapped out with missing nodes, as seen from the intermediate to the child stage.

$$\begin{aligned} \text{Parent 1} &: (1,3|5,2,4|6) \\ \text{Parent 2} &: (3,6|4,2,1|5) \\ &- - - - - - - - - - - - \\ \text{Intermediate 1} &: (1,3|4,2,1|6) \\ \text{Intermediate 2} &: (3,6|5,2,4|5) \\ &- - - - - - - - - - - - \\ \text{Child 1} &: (5,3,4,2,1,6) \\ \text{Child 2} &: (3,6,5,2,4,1) \end{aligned} \tag{5}$$

4. Mutation is where genes have a certain probability of being randomly altered to provide an arbitrary path with the intention that premature convergence (local optimum) is avoided.

Generally, in order to achieve decent results from the GA requires a large population size and many generations are required. This requirement leads to a large simulation which is typically computationally expensive and even then good results are not guaranteed.

3.4 Ant Colony Optimisation

The Ant Colony Optimisations (ACOs) are a set of algorithms which were proposed by Dorigo and colleagues in the early 1990s [5]. The ACO is a type of Swarm algorithm that is based on the pheromone based foraging of real ant colonies [50]. The ants explore the search space for "food" and leave pheromones on the travelled path in

order to indicate to other ants about the prospects of that path. As the "food" is found and the ants return to the colony they update the pheromone level along their path so that other ants can also find the "food". As more ants follow the higher pheromone level (shortest distance) paths, the evaporation of the pheromone on the less travelled paths further emphasises the good route, allowing for the optimal path to be explored further. It is through this process that the ACO algorithms can optimise discrete optimisation problems by applying the finding technique of using pheromones to determine the shortest path to the "food" source. This process is illustrated in Figure 3.2.

The Ant System (AS) was the first proposed ACO algorithm. AS uses the process of ants seeking the shortest path between the colony and a food source in order to solve the optimisation problem [5].

The ants make decisions on which route to follow based on pheromone levels as well as the distance of each connecting route between nodes. The probability of each route is determined according to (6).

$$P_{ij}^k = \frac{[\tau_{ij}]^\gamma [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\gamma [\eta_{il}]^\beta}, \text{ if } j \in N_i^k \tag{6}$$

Where:

P_{ij}^k is the probability of ant k choosing to take the route connecting nodes i and j;

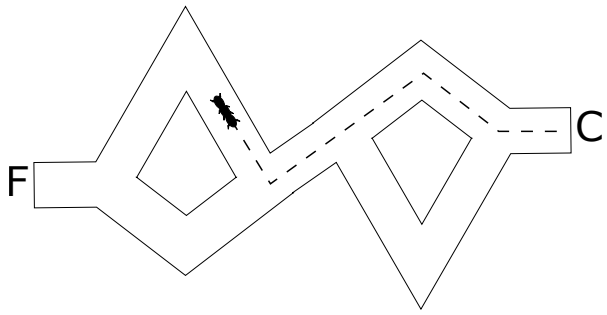
N_i^k is the set of nodes not visited by ant k when at node i;

τ_{ij} is the pheromone level of the route connecting nodes i and j;

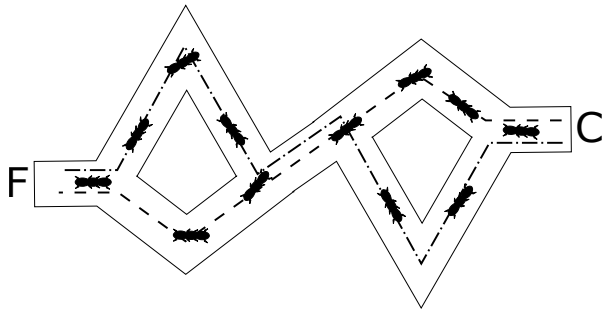
η_{ij} is the desirability level of the route connecting nodes i and j, typically $\frac{1}{d_{ij}}$ where d_{ij} is the distance or weight of the route;

γ, β are the parameters used to control the influence of τ and η respectively.

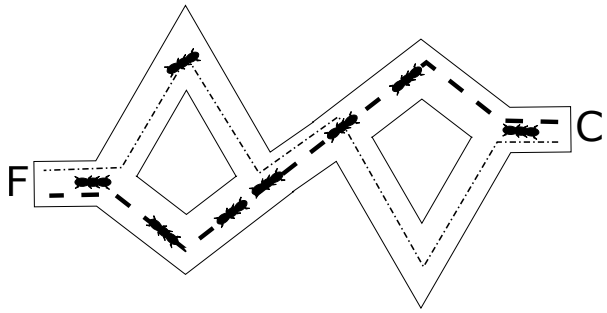
An initial population of ants, m, are created and randomly initialised to various starting nodes in the TSP graph. Each ant chooses the next node to travel to using the probability according to (6). This process continues until every ant has a complete tour and then returns to its initial node [5]. The pheromone levels on the tour of each of the m ants are updated according to (7). The pheromone update function includes an evaporation factor which is used to decrease the level of the pheromone to discourage



(a) Ants explore paths from the colony in order to locate food.



(b) Pheromone levels are updated along the individual paths taken from the colony to the food.



(c) As the exploration continues, paths with higher pheromone levels (the thick dashed line) are favoured by the majority of the ants but some of the lower pheromone paths (the dotted-dashed lines) are still utilised infrequently.

Figure 3.2: A set of diagrams showing the process of ants exploring for food (F) and updating pheromone levels along various paths until the shortest path is determined between the colony (C) and the food source.

popular routes from being chosen in subsequent iterations [5].

$$\tau_{ij} \to (1-\alpha)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \tag{7}$$

where α is the evaporation constant and $\Delta\tau_{ij}$ is the total distance of the ant's path if the $arc(i,j)$ is included otherwise it is 0.

This select-and-update process is repeated either until the specified maximum num-

ber of iterations is reached or the solution stagnates where there is no change in the solution for a specified number of consecutive iterations [5].

Dorigo proposed other ACO algorithms that improve the results of AS as well as the speed in which the solution is determined [5]. The Elitist Ant System places more emphasis on the *best-so-far* solution by updating the pheromone levels using an extra term $e\Delta\tau_{ij}^{bsf}$, where e is an arbitrary constant used to adjust the influence of the *best-so-far* solution.

In the Rank Ant System algorithm, ants with better tours will have a greater influence on the update process of the pheromone levels than those with worse solutions [5, 32].

The algorithm that was able to improve the most compared to the original AS was the $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) which makes four main modifications to the original AS algorithm in order to improve results which include [5, 32]:

1. Only the *best-so-far* route and the best current route are able to deposit pheromones.

2. To avoid stagnation from 1, the pheromone levels are bounded between a predefined τ_{max} and τ_{min}.

3. The pheromone levels are initialised to τ_{max} with a small evaporation constant to promote the exploration rate initially.

4. The pheromone levels are reset to the initial values whenever the solution stagnates.

3.5 Simulated Annealing

Simulated annealing (SA) is an algorithm which is based on the physical process of annealing which is used to find the low-temperature state of a material through experiments [35, 36]. In 1983, Kirkpatrick, Gelatt and Vecchi formed the hypothesis that large physical systems and combinatorial problems have similar behaviours and therefore results from classical statistical mechanics could be applied to combinatorial optimisation problems [36]. The Boltzmann distribution, $P(\Delta E) = e^{-\Delta E/k_b T}$, is used in physical systems to describe the probability of configurations occurring. The Metropolis Monte Carlo method is usually used to simulate the annealing process but Kirkpatrick, Gelatt and Vecchi argued that since the temperature in the algorithm controls the probability of accepting a longer tour length, the Boltzmann constant, k_b, can, therefore,

be ignored [35]. An annealing schedule is defined as a sequence of temperatures and the corresponding times it takes for each temperature's equilibrium to be reached, for the TSP, the time is counted in iterations of the algorithm. There are several variations which have been made to the SA algorithm for the TSP in terms of the search algorithms used, but the fundamental concept remains the same.

The original SA Algorithm, as defined by Kirkpatrick, Gelatt and Vecchi, uses the 2-Opt search algorithm and follows the following steps [35]:

1. Randomly define a valid TSP route as well as the total route cost, define the annealing schedule as $S = \{t_1, \ldots, t_l\}$ where $t_1 > t_2 > \ldots > t_l$, set $i = 1$.

2. One step of the 2-Opt algorithm is executed, as defined below, and the new route cost, ΔC, is calculated. Determine $P(\Delta C) \equiv e^{-\Delta C/t_i}$ and select a random real number $r \sim U(0, 1)$.

3. *If* $\Delta C < 0$ or $r \geq P(\Delta C)$ accept the 2-Opt new route and go to step 4, *otherwise* reject the new route and go back to step 2.

4. *If* the equilibrium for t_i has been reached, meaning that the maximum number of iterations for the current temperature has been achieved, then increment i. *If* $i > l$, end the algorithm.

The 2-Opt is a simple local search algorithm for TSP, proposed by Croes in 1958 [12], which reorders a route so that the new route is still valid. The basic 2-Opt swap algorithm is given in Algorithm 1 [13]. The route and two indices are given to the 2-Opt algorithm, the new route is comprised of the initial sequence (from the start to the first index) of the original route. The middle Section (from the first index to the second index) of the original route is then added to the new route in the reverse order and finally, the end sequence of the route (from the second index to the end) is added to the new route from the original route.

Algorithm 1 2-Opt Search

Require:

$\mathbf{s}$ ▷ A valid TSP tour.

i ▷ The position of the first index.

j ▷ The position of the second index.

1: **function** 2Opt($\mathbf{s}$, i, j)

2: $\mathbf{s}^{t+1} = \mathbf{s}^t[1] \rightarrow \mathbf{s}^t[i-1]$

3: $\mathbf{s}^{t+1} = \mathbf{s}^{t+1} + \text{ReverseOrder}(\mathbf{s}^t[i] \rightarrow \mathbf{s}^t[j-1])$ ▷ Reverse the order of the nodes between the two indices

4: $\mathbf{s}^{t+1} = \mathbf{s}^{t+1} + \mathbf{s}^t[j] \rightarrow \mathbf{s}^t[\text{end}]$

5: **return** $\mathbf{s}^{t+1}$

6: **end function**

Chapter 4: Discrete Flower Pollination Algorithm for solving the Symmetric Travelling Salesman Problem

The two algorithms, as well as the results and analysis, have been submitted as a journal article to the *IEEE Transactions on Cybernetics* and this article is presented in this chapter. The proposed Discrete Flower Pollination Algorithm (DFPA) and iterative Discrete Flower Pollination Algorithm (iDFPA) are described. The results and analysis of the comparison between the DFPA, iDFPA, ACO and GA are done using simulations on three problem sets. A single iteration in the form of the DFPA has comparable results to the other multiple iteration algorithms and the iDFPA outperformed both the GA and ACO in terms of average minimum tour distance, best tour distance and convergence speed. The reference to the paper is currently the following until acceptance by the peer-review process:

> **R.D. Strange** and L. Cheng, "Discrete Flower Pollination Algorithm for solving the Symmetric Travelling Salesman Problem", currently submitted to the *IEEE Transactions on Cybernetics* for the peer-review process as of 23/11/2016.

The main research idea was derived from a joint effort of **R.D. Strange** and Prof. L. Cheng. The software implementation of the two new algorithms and the simulations were fully coded by **R.D. Strange**. The journal article was written by **R.D. Strange** under the supervision of Prof. L. Cheng throughout the research as well as with the guidance through the journal article editing process.

4.1 Discrete Flower Pollination Algorithm

The Discrete Flower Pollination Algorithm (DFPA) is a proposed algorithm based on the natural pollination process as described in the FPA with the focus on discrete optimisation problems such as the TSP. The algorithm is designed using the main concept of a local and global search from the FPA with the multiple agent concept from the ACO. The solutions are developed one node at a time, unlike the genetic algorithm which determines a full solution and then attempts to improve the tour. The full DFPA system flowchart including the improved iterative DFPA, as described in Section 4.2, is given in Fig. 4.3.

Section 4.1.1 describes the full DFPA algorithm and Section 4.1.2 explains the implementation of the local and global search algorithms utilised in the DFPA.

4.1.1 Algorithm Description

The algorithm requires an $n \times n$ distance matrix, D, to be developed for each problem where n is the number of nodes in the problem. D has a diagonal of zeros and the remaining elements, d_{ij}, representing the distance between the i'th and j'th nodes as given in (8). For the symmetric TSP problem, the distance matrix will be symmetric across the diagonal, $d_{ij} = d_{ji}$.

$$d_{ij} = \begin{cases} d(v_i, v_j) & j \neq i, \\ 0 & j = i. \end{cases} \tag{8}$$

Similarly, an $n \times n$ cost matrix, C, is constructed using the inverse of the distance matrix in order to calculate a path probability, where:

$$c_{ij} = \begin{cases} \frac{1}{d_{ij}} & j \neq i, \\ 0 & j = i. \end{cases} \tag{9}$$

There are m agents simultaneously developing independent solutions with the intention of exploring all possible tours to determine the best tour. The algorithm uses a switching probability, $\rho \in (0, 1)$, to randomly switch between the local and global search in order to determine which node to move to next. The algorithm builds the tours using a Markov Chain like process, where the next node in a tour is globally or

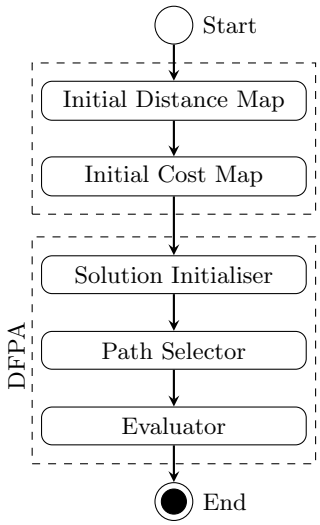


Figure 4.1: System flowchart of the DFPA algorithm.

locally attained using the probabilities from the current node.

Define a set of tours $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m\}$ where $\mathbf{s}_i = (s_i^1, s_i^2, \ldots, s_i^n)$ is an ordered sequence (tour), appearing as a permutation of the set $V = \{v_1, v_2, \ldots, v_n\}$. First, let $V' = V$ where V' is a temporary set. s_i^1 is then randomly selected from V' as the initial node for the solution and set the previous node $v_{prev} = s_i^1$ and $V' = V' - \{s_i^1\}$ as shown in lines 4-9 of Algorithm 2.

To determine the rest of the elements in $\mathbf{s}_i$, set $j = 2, \ldots, n$, and for each j, a process is used to determine the next node in the solution. Let $|V'|$ denote the cardinality of set V'. The normalisation constant $A = \sum_{\forall v_k \in V'} c_{ik}$, where v_i is the previous node and c_{ik} is determined from the remaining nodes in V'. This process is shown in lines 11-12 of Algorithm 2. The probability of each node in V' is determined according to:

$$p_{ij} = \frac{c_{ij}}{A}. \tag{10}$$

A random number $r \sim U(0, 1)$ is then generated. As can be seen in lines 15-22 of Algorithm 2, if r is greater than ρ then the global search algorithm is used to determine s_i^j, otherwise, the local search algorithm is used, these algorithms are described in Section 4.1.2. Finally, the temporary variables are updated according to $V' = V' - \{s_i^j\}$ and $v_{prev} = s_i^j$ before incrementing j. Once all the solutions are determined for $\mathbf{S}$, the tour distance is evaluated for each of the elements of $\mathbf{S}$ and stored in a resulting set $\mathbf{d} = \{d_{\mathbf{s}_1}, d_{\mathbf{s}_2}, \ldots, d_{\mathbf{s}_m}\}$. The shortest tour distance $d_{\mathbf{s}^*}$ in $\mathbf{d}$ and the corresponding tour

$\mathbf{s}^*$ in $\mathbf{S}$ are chosen as the TSP solution for the algorithm. The complete algorithm is given in Algorithm 2.

4.1.2 Local and Global Search Algorithms

Both the local and global search algorithms are based on the set V' introduced in the last subsection. All the nodes in V' are re-ordered to generate a new set $\bar{V} = \{v'_1, v'_2, \ldots, v'_{|V'|}\}$ such that $p_1 > p_2 > \ldots > p_{|V'|}$, where p_i is the probability of node v_i. Note it is assumed that $p_i \neq p_j$ to avoid some technical ambiguities. The local search uses a categorical distribution and the global search uses a discrete Levy distribution.

For the local search a subset, V'', of $\bar{V}$ is obtained according to:

$$V'' = \{v | d(v, v_{prev}) \leq r_{dist}, v \in \bar{V}\}. \tag{11}$$

Then let V'' be the new $\bar{V}$. Note that the index system retains the features of the old $\bar{V}$, in other words, the probability of nodes is still in descending order. Ordering $\bar{V}$ is also beneficial as it reduces the complexity of the multinomial distribution. Nodes that are within the radial threshold distance r_{dist} are considered "local" nodes and therefore are viable for selection through the local search and the remaining nodes are excluded. This selection process is demonstrated by Fig. 4.2. Once the remaining "local" nodes are determined the remaining costs are renormalised in the same manner as described by (10).

The selection of s_i^j is then done explicitly using the categorical distribution:

$$l^* = min\{l \in \{1, 2, \ldots, |\bar{V}|\} : (\sum_{i=1}^{l} p_i) - r \geq 0\}, \tag{12}$$

where s_i^j is v'_{l^*} in $\bar{V}$ and $r \sim U(0, 1)$ is a random variable. If there are no nodes remaining after the radial clustering process then the global search method is used to determine the next node, a procedure shown in lines 18-20 of Algorithm 2.

In order to formalise the global search algorithm, a definition for the discrete Levy Distribution is defined. The probability mass function of the discrete Levy Distribution

Algorithm 2 Discrete Flower Pollination Algorithm

Require:
n ▷ Number of nodes
$\mathbf{D}$ ▷ $n \times n$ distance map
m ▷ Number of solving agents

function DFPA(n, $\mathbf{D}$, m)
  $\mathbf{C} = \frac{1}{\mathbf{D}}$ ▷ Initial cost map is inverse distance map
  $\mathbf{S} \leftarrow m \times n$ empty array ▷ Solutions Vector
  $\mathbf{S}[1, :] \leftarrow$ random number $(0, 1)$ ▷ Initialise starting position of each solution
  $\rho \leftarrow$ random number $(0, 1)$ ▷ Switching probability
  for $i \leftarrow 1, m$ **do** ▷ $\mathbf{O(m)}$
    $V' \leftarrow$ remaining nodes for i^{th} solution
    for $j \leftarrow 2, n$ **do** ▷ $\mathbf{O(n)}$
      $v_{prev} \leftarrow \mathbf{S}(i, j-1)$ ▷ Current node
      $C' \leftarrow \mathbf{C}(v_{prev}, V')$ ▷ The cost from v_{prev} to each node in V'
      $A = \sum C'$ ▷ Normalisation constant is the sum of costs in C'
      $C'(k) \leftarrow C'(k)/A$ ▷ Normalise the remaining costs $\mathbf{O(n)}$
      $C' \leftarrow$ sorted in descending order ▷ $\mathbf{O(nlog(n))}$
      $r \leftarrow$ random real number $(0, 1)$
      if $r \geq \rho$ **then**
        v_{curr} = GSearch(V',C') ▷ $\mathbf{O(n)}$
      else
        [$success$,v_{curr}] = LSearch(V',C') ▷ $\mathbf{O(n)}$
        if $success$ is false **then**
          v_{curr} = GSearch(V',C') ▷ $\mathbf{O(n)}$
        end if
      end if
      $\mathbf{S}(i, j) \leftarrow v_{curr}$
      remove v_{curr} from V'
      $v_{prev} \leftarrow v_{curr}$
    end for
  end for
  $\mathbf{d} \leftarrow$ Evaluate each tour in $\mathbf{S}$ ▷ $\mathbf{O(m(n + log(m)))}$
  Determine $d_{\mathbf{s}^*}$ the minimum tour distance in $\mathbf{d}$
  return $d_{\mathbf{s}^*}$ and corresponding tour $\mathbf{s}^*$ in $\mathbf{S}$
end function

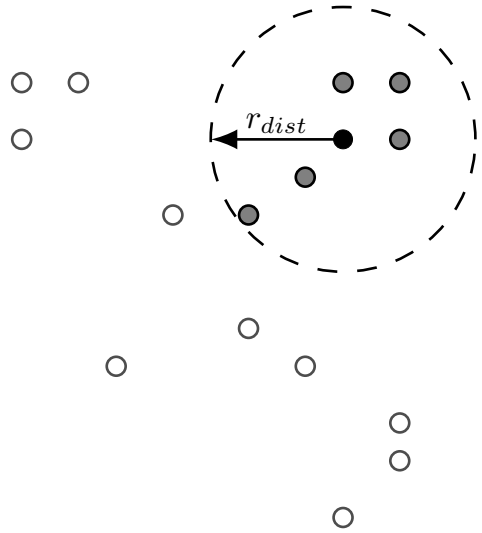


Figure 4.2: Radial distance threshold clustering used in the local search: black node is the previous node, shaded nodes are considered for the local search, others are excluded.

$\mathbf{L}(\bar{V})$ used in this paper is defined as:

$$\begin{aligned} f(v_i;\bar{V}) &= Pr(v_i \text{ node is chosen}) \\ &= F_L(\Phi \sum_{k=1}^{i} p_k) - F_L(\Phi \sum_{k=1}^{i-1} p_k), \end{aligned} \tag{13}$$

where F_L is the continuous Levy CDF and Φ is a constant. Since the continuous Levy Distribution takes inputs from zero to infinity, a maximum value is chosen as a practical consideration in order to limit the continuous distribution for the discretization process. Let Φ denote the maximum input value. Φ is selected to be sufficiently large so as to make $\mathcal{L}(x > \Phi)$ tend to zero due to the discretization process. The discrete Levy Distribution is applied to the node set $\mathbf{L}(\bar{V})$ using (13). The next node $s_i^j \sim \mathbf{L}(\bar{V})$ is then generated explicitly using (12).

4.2 Iterative Discrete Flower Pollination Algorithm (iDFPA)

The multiple agent and random walk nature of the DFPA mean that the problem space is thoroughly explored, however, this knowledge is lost at the end of the process. The iterative Discrete Flower Pollination Algorithm (iDFPA) uses two update processes, namely best tour update and rejection update, to take advantage of this systemic knowledge that is developed through the DFPA process. A combination of these processes is used to achieve the best, converged results for a variety of problem types.

The system flowchart for the complete iDFPA is illustrated in Fig. 4.3.

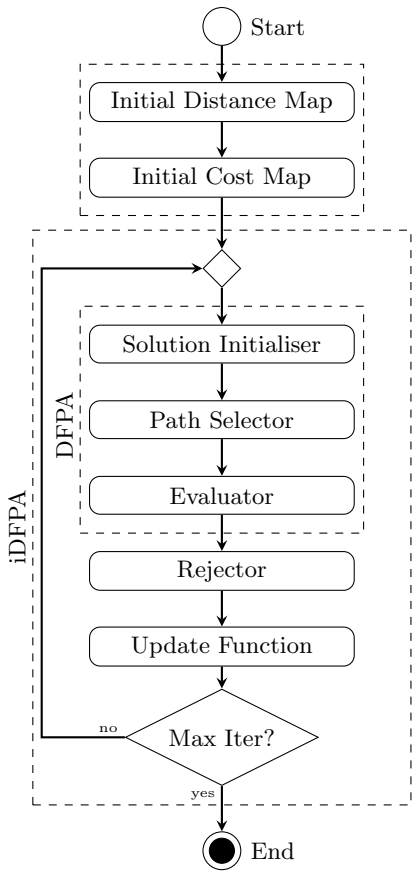


Figure 4.3: System flowchart for the iDFPA Algorithm.

The cost matrix C^t, where t represents the cost matrix of the t'th iteration, and c_{ij}^t, the cost from node v_i to v_j, is updated in order to reflect the new knowledge gained about the system through the iterative process. After each of the N iterations, an update process takes place which includes evaporation, best tour update and rejection update subprocesses. The evaporation process is implemented as:

$$c_{ij}^{t+1} = (1 - \alpha)c_{ij}^t, \tag{14}$$

where α is a control variable of the rate at which the cost is evaporated [5]. The evaporation mechanism reduces the memory of the system so that the "bad" knowledge is filtered out. The iDFPA then performs two update functions, namely, the Best Tour Update (BTU) and the Rejection Update (RU).

4.2.1 Best Tour Update

The Best Tour Update (BTU) process uses, as determined by the DFPA in Section 4.1.1, the minimum tour distance $d_{\mathbf{s}^*}$ and the corresponding tour $\mathbf{s}^*$ from the tour distance set $\mathbf{d} = \{d_{\mathbf{s}_1}, d_{\mathbf{s}_2}, \ldots, d_{\mathbf{s}_m}\}$ and tour set $\mathbf{S} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m\}$ respectively. $d_{\mathbf{s}^*}$ and $\mathbf{s}^*$ are used to update C^t on nodes that make up the tour according to:

$$c_{ij}^{t+1} = \begin{cases} c_{ij}^t + \gamma \frac{d_{ij}}{d_{\mathbf{s}^*}} & \text{if } arc(i,j) \text{ is part of } \mathbf{s}^*; \\ c_{ij}^t & \text{otherwise,} \end{cases} \tag{15}$$

where the constant γ controls the influence the best tour has on the system. The complexity of the BTU is $O(n)$ since there is a single path to be updated.

4.2.2 Rejection Update

The Rejection Update (RU) process utilises the tour distance set $\mathbf{d} = \{d_{\mathbf{s}_1}, d_{\mathbf{s}_2}, \ldots, d_{\mathbf{s}_m}\}$ from the DFPA evaluation process in order to reject results that are worse than the previous best, according to Algorithm 3, while still providing a small probability that they are accepted in order to explore new search spaces. An exponential annealing schedule is defined as:

$$T_{curr} = e^{-w \frac{1}{q} \frac{N_{curr}}{N}}, \tag{16}$$

where w is a constant that controls the utilised region of the exponential function, q specifies the shape of the exponential function, N_{curr} is the current iteration number, and T_{curr} is the annealing value for the current iteration. This exponential annealing function is used to determine the probability for a solution.

If the tour distance $d_{\mathbf{s}_j}$ is less than the previous iteration's best tour distance d_{prev} then the solution is automatically accepted. If $d_{\mathbf{s}_j}$ is greater than d_{prev} then a probability for $d_{\mathbf{s}_j}$ being accepted is determined according to:

$$p = e^{-\Delta dist/(T_k \times d_{prev})}, \tag{17}$$

where $\Delta dist = d_{\mathbf{s}_j} - d_{prev}$. A random number $r \sim U(0,1)$ is then generated and if $r < p$ the solution is accepted otherwise it is rejected. $\mathbf{d}' = \{d'_1, d'_2, \ldots\}$ denotes the

subset of $\mathbf{d}$ that contains the distances of the accepted tours and the corresponding tours $\mathbf{S}' = \{\mathbf{s}'_1, \mathbf{s}'_2, \ldots\}$ denotes the subset of $\mathbf{S}$ that contains the corresponding tour paths for $\mathbf{d}'$. The update function on C^t for the nodes in $\mathbf{S}'$ is then performed as defined by:

$$c_{ij}^{t+1} = c_{ij}^{t} + \beta \sum_{\substack{\forall \mathbf{s}'_k \in \mathbf{S}' \\ \text{containing} \\ arc(i,j)}} \frac{1}{d'_k}, \tag{18}$$

where $d'_k \in \mathbf{d}'$, and β is a control variable used to adjust the influence of the rejection process on C^t. The RU has a complexity of $O(mn)$.

Algorithm 3 Rejection Update: Annealing Rejection

Require:
d_{prev} ▷ previous iteration's best distance
$\mathbf{d}$ ▷ all current agents' distances

function RU(d_{prev},$\mathbf{d}$)
  for each agent j in $\mathbf{d}$ **do**
    $\Delta dist = d_{\mathbf{s}_j} - d_{prev}$
    if $\Delta dist < 0$ **then**
      j^{th} agent's distance is *Accepted*
    else
      $p = e^{-\Delta dist/(T_{curr} d_{prev})}$ ▷ probability of the distance being accepted
      r = random real number (0,1)
      if $r < p$ **then**
        j^{th} agent's distance is *Accepted*
      else
        j^{th} agent's distance is *Rejected*
      end if
    end if
  end for
end function

A combination of the BTU and RU can be used by changing the β and γ constants. The best iDFPA results can be obtained for a variety of TSP problem sets by changing

the influence of the evaporation, BTU and RU processes on the iDFPA according to:

$$c_{ij}^{t+1} = \begin{cases} c_{ij}^{t} + \beta \sum\limits_{\substack{\forall \mathbf{s}'_k \in \mathbf{S}' \\ \text{containing} \\ arc(i,j)}} \frac{1}{d'_k} + \gamma \frac{d_{ij}}{d_{\mathbf{s}^*}} & \text{if } arc(i,j) \\ & \text{is part of } \mathbf{s}^*; \\ c_{ij}^{t} + \beta \sum\limits_{\substack{\forall \mathbf{s}'_k \in \mathbf{S}' \\ \text{containing} \\ arc(i,j)}} \frac{1}{d'_k} & \text{otherwise.} \end{cases} \tag{19}$$

4.3 Results

4.3.1 Experiment Setup

The DFPA and iDFPA algorithms were designed to be generic to the problem set size (number of nodes) and the configuration of the problem (sparse, dense, etc). Simulation experiments were therefore designed to encompass a variety of problem sets in order to determine how well the algorithm performs. Benchmark problem sets from *TSPLIB*[1] were used to perform a comparison of the DFPA, iDFPA, Ant Colony Optimisation's (ACO) $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$)[2] and Genetic Algorithm (GA)[3]. The ACO and GA were chosen for the comparison since they are well studied and developed algorithms that have been widely tested on the TSP.

Three Euclidean 2D symmetric TSP benchmark problem sets were selected, namely Berlin52, EIL76 and PR264. Table 1 shows the properties of the three problem sets. Small problem sizes (defined as problem sets with less than 30 nodes by the authors) were not selected as they are trivial to solve and comparative results do not illustrate the abilities of the algorithm. Medium and large sets, 30-100 nodes and 100+ nodes respectively, as well as the node density which is the layout of the nodes and how they are distributed (sparse or dense), were used as criteria to select the experimental problem sets.

[1]TSPLIB is a benchmark library of Travelling Salesman Problems where the optimal paths have been determined in order to compare algorithms `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

[2]The ACO algorithm used for comparison was developed by H. Wang `https://www.mathworks.com/matlabcentral/fileexchange/14822-solve-tsp-by-mmas`

[3]The TSP genetic algorithm used for comparison was developed by Joseph Kirk `https://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm`

Table 1: Selected simulation problem sets' properties.

Property	Berlin52	Eil76	PR264
Size	Medium	Medium	Large
Number of Nodes	52	76	264
Node Density	Sparse	Dense	Mixed

4.3.2 DFPA Results

The DFPA is used to give an indication of the best performance that can be achieved based on a single iteration. It can be used for a fast estimation of the initial result the DFPA can achieve compared to other algorithms. The simulation was run using the same parameters for each of the algorithms where the number of solving agents used for the problem sets were set equal to the number of nodes in the problem set as this provided the best results while keeping the computation time relatively low. The simulation was run independently 100 times to observe the average behaviour for each problem due to the random nature of the algorithm.

The DFPA, which is essentially a single iteration, significantly outperforms the GA in all three problem sets as shown in Table 2. As illustrated by the percentage comparisons in Tables 3, 4 and 5 the DFPA outperforms the GA but not the ACO in all the problem sets. The DFPA results illustrate the ability of the algorithm to comparatively compete with existing algorithms that run for multiple iterations.

4.3.3 iDFPA Results

The iDFPA outperforms the ACO and GA in both the 50 and 300 iteration simulations, as shown in Table 2. Tables 3, 4 and 5 show the percentage comparison between the DFPA, iDFPA 50 iteration and iDFPA 300 iteration algorithms with the optimal distance, ACO and GA on the Berlin52, Eil76 and Pr264 problem sets respectively. The tables illustrate how the iDFPA, in both the 50 and 300 iteration simulations, is able to outperform the ACO and GA over 300 iterations. The iDFPA 300 iteration also achieved the optimal distance for the Berlin52 problem set as shown in Table 3.

The iDFPA algorithm's performance was tested using a variety of simulations, not only to compare its performance against other algorithms but also to understand the influence of the parameters and what parameter values give the best results for the various problem sets. There are three main parameters for the iDFPA namely α, β and γ,

Table 2: Comparative tour distance results for three TSP problem sets.

Problem Set	Optimal	DFPA (1 Itr.)		iDFPA (50 Itr.)		iDFPA (300 Itr.)		ACO (300 Itr.)		GA (300 Itr.)	
		Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best
Berlin52	7542	8288.45	7913	7938.24	7571	7920.18	7542	8128.76	7681	8893.75	8526
Eil76	538	608.82	567	576.4	553	569.11	548	590.76	573	817.75	766
Pr264	49135	59239.31	55115	54736.91	53548	54542.62	53489	55343.40	53690	75970.88	70776

Table 3: Berlin52 comparative percentage results[a].

		Optimal	ACO (300 Itr.)	GA (300 Itr.)
DFPA	Avg(%)	-9.01	-1.93	7.30
DFPA	Best(%)	-4.69	-2.93	7.75
iDFPA (50 Itr.)	Avg(%)	-4.99	2.40	12.04
iDFPA (50 Itr.)	Best(%)	-0.38	1.46	12.61
iDFPA (300 Itr.)	Avg(%)	-4.77	2.63	12.29
iDFPA (300 Itr.)	Best(%)	0.00	1.85	13.05

Table 4: Eil76 comparative percentage results[a].

		Optimal	ACO (300 Itr.)	GA (300 Itr.)
DFPA	Avg(%)	-11.63	-2.97	34.32
DFPA	Best(%)	-5.11	1.06	35.1
iDFPA (50 Itr.)	Avg(%)	-6.66	2.49	41.87
iDFPA (50 Itr.)	Best(%)	-2.71	3.62	38.52
iDFPA (300 Itr.)	Avg(%)	-5.47	3.80	43.69
iDFPA (300 Itr.)	Best(%)	-1.82	4.56	39.78

which are used to adjust the influence of the evaporation, RU and BTU methods on the iterative process. These three parameters are adjusted to determine which combination gives the best results for the problem set. Other parameters such as the number of iterations and number of solutions can be adjusted to get the best results for specific circumstances such as within a small time frame or the best overall result.

The algorithm was run on the three problem sets by incrementally increasing the values of α, β and γ independently in order to determine what combination of values produced the best results for the different problem sets. The best parameter values, as determined by this process, are shown in Table 6 along with the parameters used for the comparative study presented in Section 4.3.3 B. Note that the ACO parameters are selected in the same way as the iDFPA parameters to ensure that the algorithm is optimised to each of the problem sets and the results are comparative. The ranges used for tuning the parameters were determined from literature, each of the parameters in Table 6 were iteratively increased or decrease from the recommended values given in literature to determine the best for the tested problem sets. A comprehensive analysis of the iterative process of the iDFPA has been done, as explained in Section 4.3.3 A.

Table 5: Pr264 comparative percentage results[a].

		Optimal	ACO (300 Itr.)	GA (300 Itr.)
DFPA	Avg(%)	-17.06	-6.58	28.24
DFPA	Best(%)	-10.85	-2.59	28.42
iDFPA (50 Itr.)	Avg(%)	-10.23	1.11	38.79
iDFPA (50 Itr.)	Best(%)	-8.24	0.27	32.17
iDFPA (300 Itr.)	Avg(%)	-9.91	1.47	39.29
iDFPA (300 Itr.)	Best(%)	-8.14	0.37	32.32

[a]Negative percentages imply that row elements are worse than the column elements and vice versa for the positive percentages.

Table 6: Parameters selected for iterative simulations.

Problem	iDFPA	ACO	GA
Berlin52	$m = 52$ $\rho = 0.1$ $\alpha = 0.1$ $\beta = 0$ $\gamma = 0.1$	$m = 52$ $\gamma = 1$ $\beta = 5$ $\alpha = 0.65$	$m = 52$
Eil76	$m = 76$ $\rho = 0.1$ $\alpha = 0.2$ $\beta = 0.1$ $\gamma = 0.4$	$m = 76$ $\gamma = 1$ $\beta = 5$ $\alpha = 0.65$	$m = 76$
Pr264	$m = 264$ $\rho = 0$ $\alpha = 0.4$ $\beta = 0.5$ $\gamma = 0$	$m = 264$ $\gamma = 0.2$ $\beta = 9$ $\alpha = 0.35$	$m = 264$

A. Iterative Process Analysis

An analysis of the acceptance ratio provides insight into how the algorithm works and how to manipulate the results according to what type of results are required for the problem, i.e. the trade-off of getting a minimum tour distance result for a faster convergence or having a slower convergence at a lower final result. By plotting the acceptance ratio over the set of iterations, as shown in Fig. 4.4, it can be seen that there are three main regions. The first is the initial decline, the second is the lower plateau

in the middle and the last is the incline and upper plateau. By analysing these three regions, how they change with parameter changes as well as how the overall result is altered according to the change in the behaviour of the rejection update process can be determined.

Fig. 4.4 shows the three phases of the acceptance ratio plot. The first region, known as the "discovery phase", is used to collect information about the problem and which routes give good and bad results. The system begins with a high acceptance ratio and decreases this ratio using the annealing schedule, as the acceptance ratio decreases (until it is roughly zero), less "bad" information is added to the system and only "good" information is accepted. The algorithm then moves into the second region, or "filtering phase", where minimal new information is added (usually only one or two paths per iteration), but the evaporation still takes place while the few new solutions are used to update the cost map. This stage acts to filter out bad information which the system added during the discovery phase. Once the filtering stage is complete the final region, or "convergence phase", is entered into. This phase allows the algorithm to converge to the best solution based on the filtered information from the preceding phase. This final stage appears to defy the concept of a decreasing acceptance ratio as the annealing process proceeds, however this is a characteristic due to the fact that after the filtering phase the probability of the algorithm choosing a bad path is small since the bad paths' costs have been depleted and the majority of the new solutions are either the *best-so-far* solution or a better new solution.

During the testing phase of the algorithm, it was determined that the acceptance ratio provided critical insight into the workings of the algorithm and its overall results. The annealing schedule affects the discovery and filtering phases directly as it determines the rate at which the discovery process occurs and how long the system stays in the filtering phase. Fig. 4.5 shows results for multiple annealing schedules where the value of (16) is changed in order to manipulate the three phases.

As shown in Fig. 4.5, if q is too large ($q = 5$ for example) the annealing schedule is slow and, therefore, the discovery phase is slow, the solution reached a minimum quickly but the filtering phase is also short. As a result, the overall result either bubbled and then converged at a bad result or diverged because the process accepted too many bad results in the initial phase. Conversely, if the annealing schedule is fast then the discovery phase

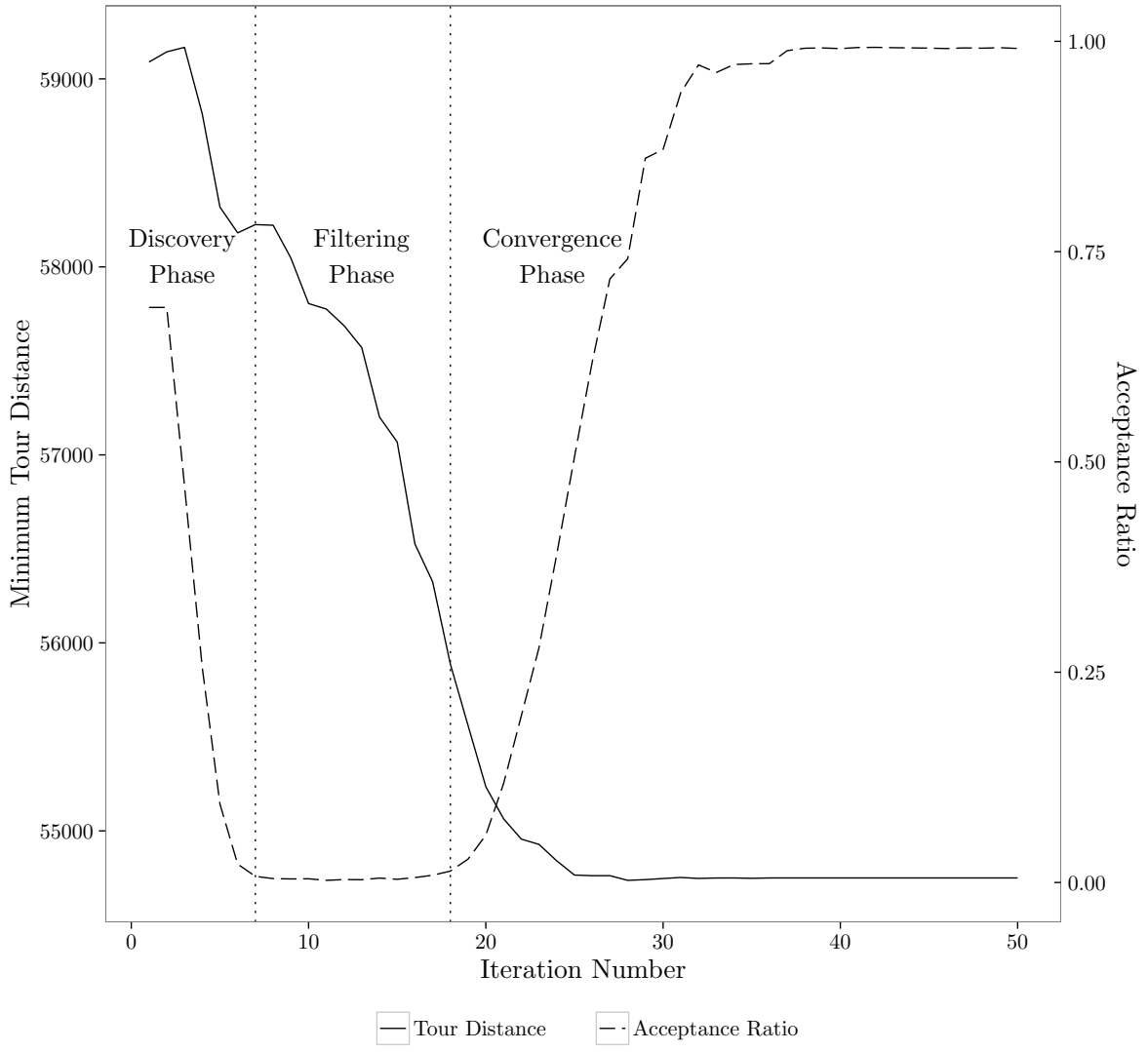


Figure 4.4: Minimum tour distance vs. acceptance ratio plot with annotated phases.

is also fast and it takes longer to reach a minimum but the overall converging result is the minimum converging value, for example, when $q = 0.8$ in Fig. 4.5. Therefore, a balance needs to be determined depending on what is required from the algorithm. If a fast minimum tour distance is required then a slow annealing schedule should be implemented and if a minimum converging result is required then an annealing schedule that drops to the converging minimum in the shortest number of iterations needs to be determined.

By creating a hybrid annealing schedule it was determined that the acceptance ratio behaviour can be tailored to have specific trends and get a result that is a combination of

other annealing schedules. As shown in Fig. 4.5, the hybrid solution has characteristics of the $q = 5$ result in terms of its initial discovery result (a fast fall) but then instead of bubbling up, it appears to follow the $q = 1$ results for the remainder of the iterations. This is a result of the value of q being made up of a piecewise function. The function is created in order to have the discovery phase of the $q = 5$ schedule and the filtering and convergence phases of the $q = 1$ schedule. The hybrid annealing schedule is described by:

$$T_{curr} = \begin{cases} e^{-w\frac{1}{0.8}\frac{N_{curr}}{N}} & N_{curr} > 8, \\ e^{-w\frac{1}{-0.525N_{curr}+5}\frac{N_{curr}}{N}} & \text{otherwise}, \end{cases} \tag{20}$$

where $-0.525N_{curr} + 5$ is a function that reduces q from 5 to 0.8 over the iteration interval 0 to 8 after which it remains at 0.8 for the remaining iterations in order to get the hybrid tour distance curve as seen in Fig. 4.5.

B. Comparative Results

In order to get comparative results between the iDFPA, ACO and GA algorithms, the algorithms were run for the same number of iterations under the same situations. 50 and 300 iteration simulations on the three problem sets were used to compare the algorithms. The two simulations were run on each problem set using the parameter values defined in Table 6. These were each run independently and the results averaged in order to remove the noise introduced by the random nature of the algorithm. The comparative results are shown in Table 2.

The DFPA and iDFPA both have a complexity of $O(mn^2 \log n)$ which is determined by cascading the complexities of the individual components of the algorithms, as given in the right-hand comments of Algorithm 2. Only the significant component complexities have been included and the iDFPA has the same complexity as the DFPA since the number of iterations is a multiplying factor of the DFPA complexity. The ACO and GA both have a complexity of $O(n^3 \log n)$ [51, 52] under the assumption that a sophisticated parent selection is done for the GA. Since m and n have the same order of magnitude this simplifies the overall complexity for the DFPA and iDFPA to $O(n^3 \log n)$ which is the same as the two comparative algorithms. The full complexity analysis for the DFPA and iDFPA can be found in Appendix A.

The aggregated results for the iDFPA 50 iteration simulations are shown in Fig. 4.6, 4.7 and 4.8. The iDFPA not only starts at a lower initial result but also converges to a lower value at a faster rate than the ACO and GA while comparing the algorithms iteration by iteration. The iDFPA 50 iteration results are below that of the ACO for the entire duration of the simulation with the exception of the Pr264 where the ACO has a lower tour distance between the tenth and twentieth iterations.

The box plots of the simulation results for the problem sets with the iDFPA and ACO

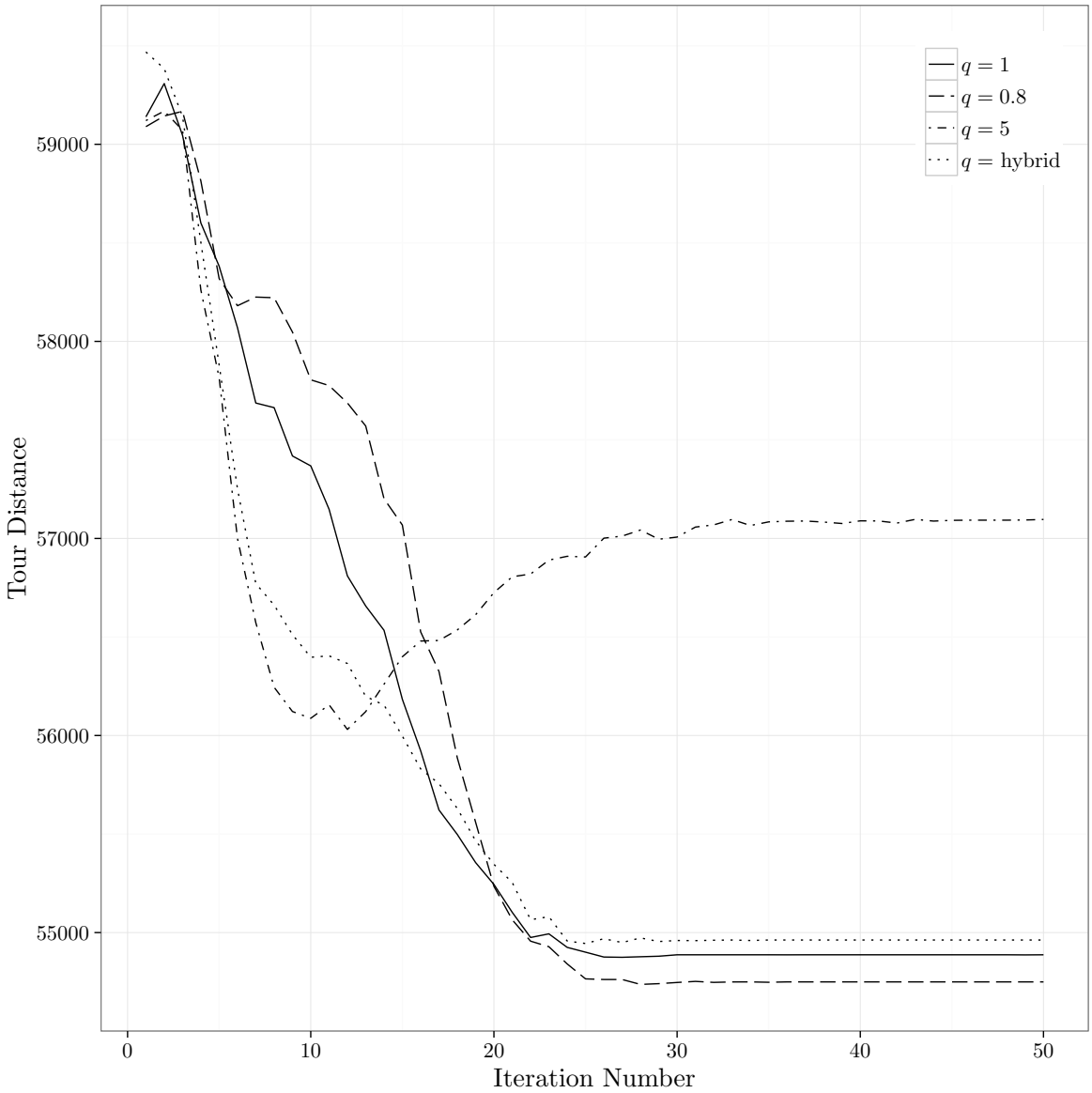


Figure 4.5: Tour distance of different values for the annealing schedule in rejection update of iDFPA for 50 iterations on the Pr264 problem set.

for 50 iterations as shown in Fig. 4.9-4.11 illustrate that there is a better convergence and smaller variance of the 100 independent results for the iDFPA than the ACO. Therefore not only does the iDFPA converge to a lower result on average but the tour distances in the independent results have a closer correlation than the ACO. This trend was seen in the other two problem sets as well.

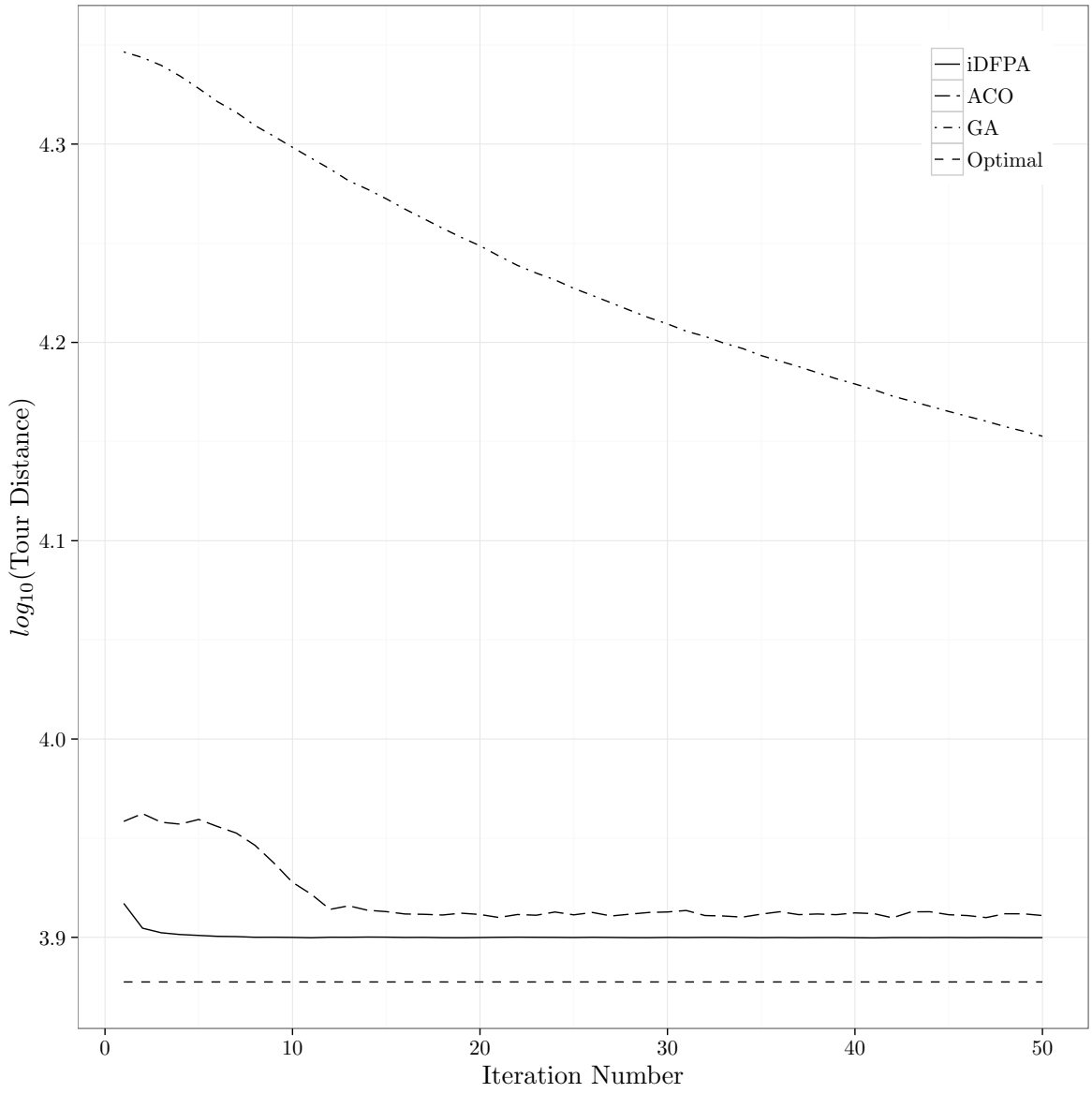


Figure 4.6: iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Berlin52 problem set.

4.3.4 Parameter Selection

As with most nature-inspired heuristic algorithms, the performance of the iDFPA depends on the values selected for the setup parameters. Intuitively it makes sense that, as the number of agents increase, so does the time required to provide a sub-optimal solution. But with this, so does the computation time, therefore, for a problem with n nodes the recommended minimum number of agents, m, is n as this produced good results within a reasonable amount of computation time. Table 7 provides the recommended

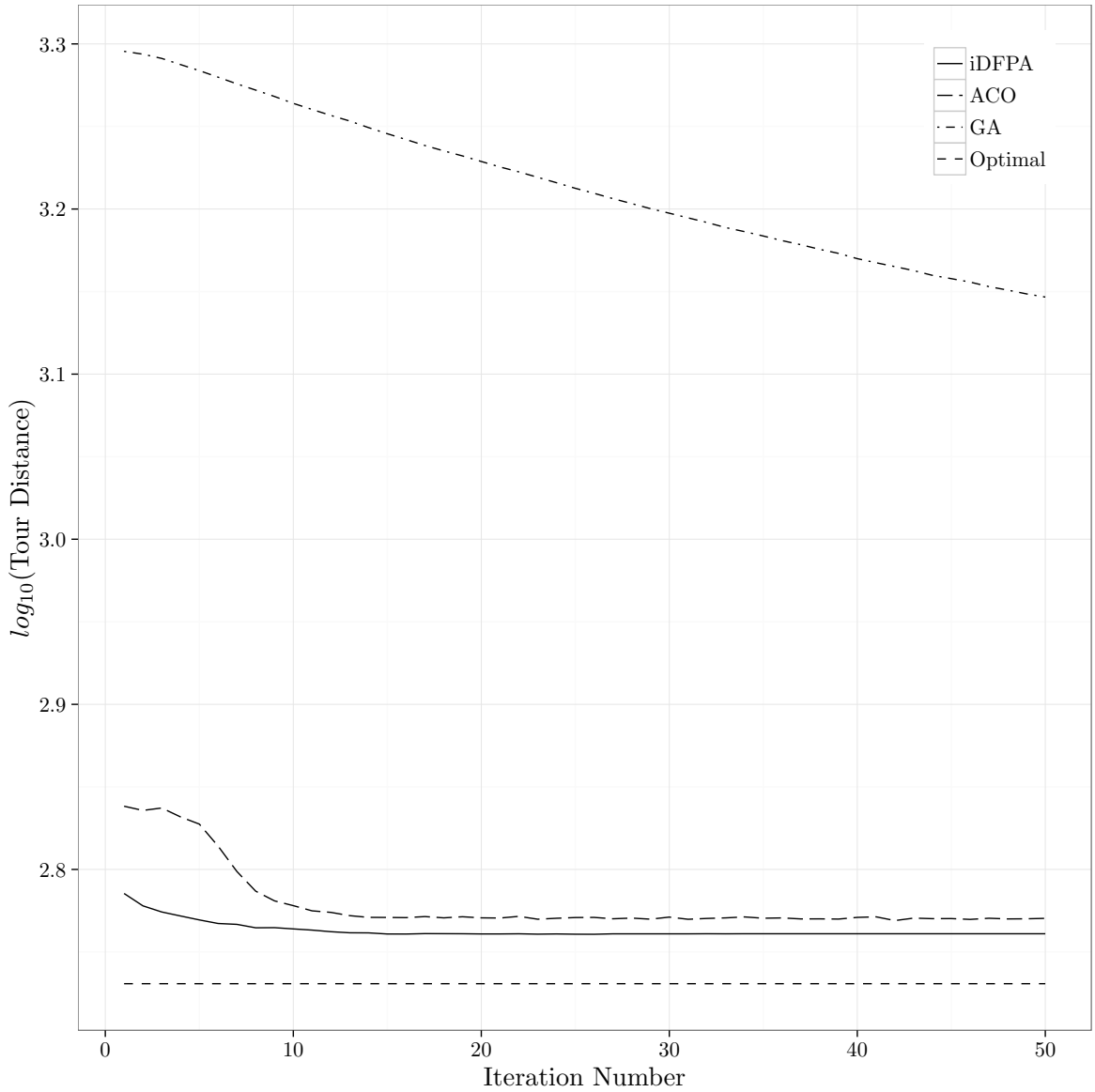


Figure 4.7: iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Eil76 problem set.

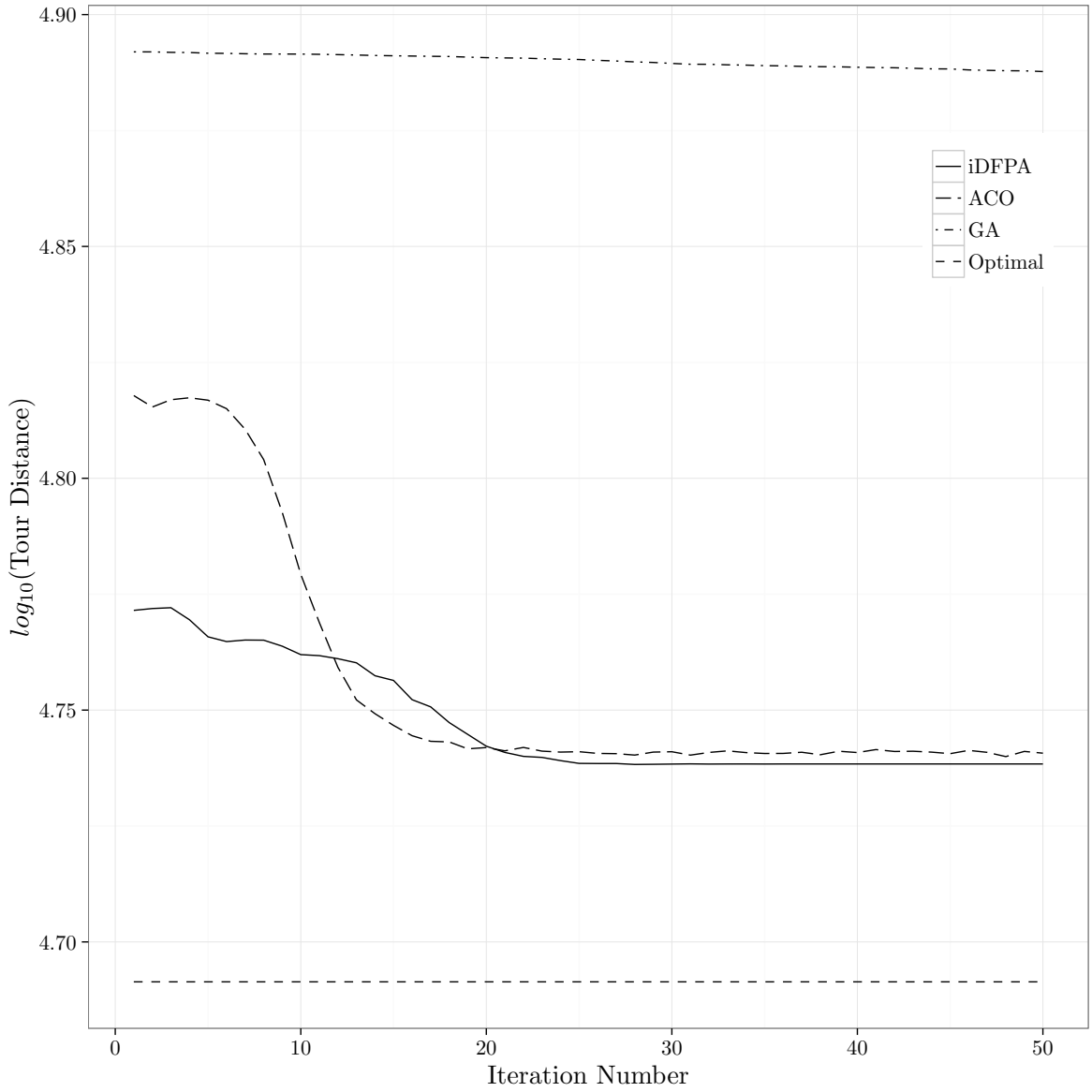


Figure 4.8: iDFPA vs ACO vs GA vs Optimal Solution average results for 50 iteration simulation on the Pr264 problem set.

initial parameters for different sized problems. The performance of the algorithm is correlated highly with a specific problem so users have to change the parameters in order to get the best performance from the algorithm. It can be seen in Table 7 that in large problem sets (where $n > 100$) the best results are achieved when $\rho = 0$ (the local search is disabled) and $\gamma = 0$ (the best tour update has no influence). This is because on the larger problem sets, the more freedom that the algorithm has for exploring all possibilities without being forced into local regions and further emphasising the best solution,

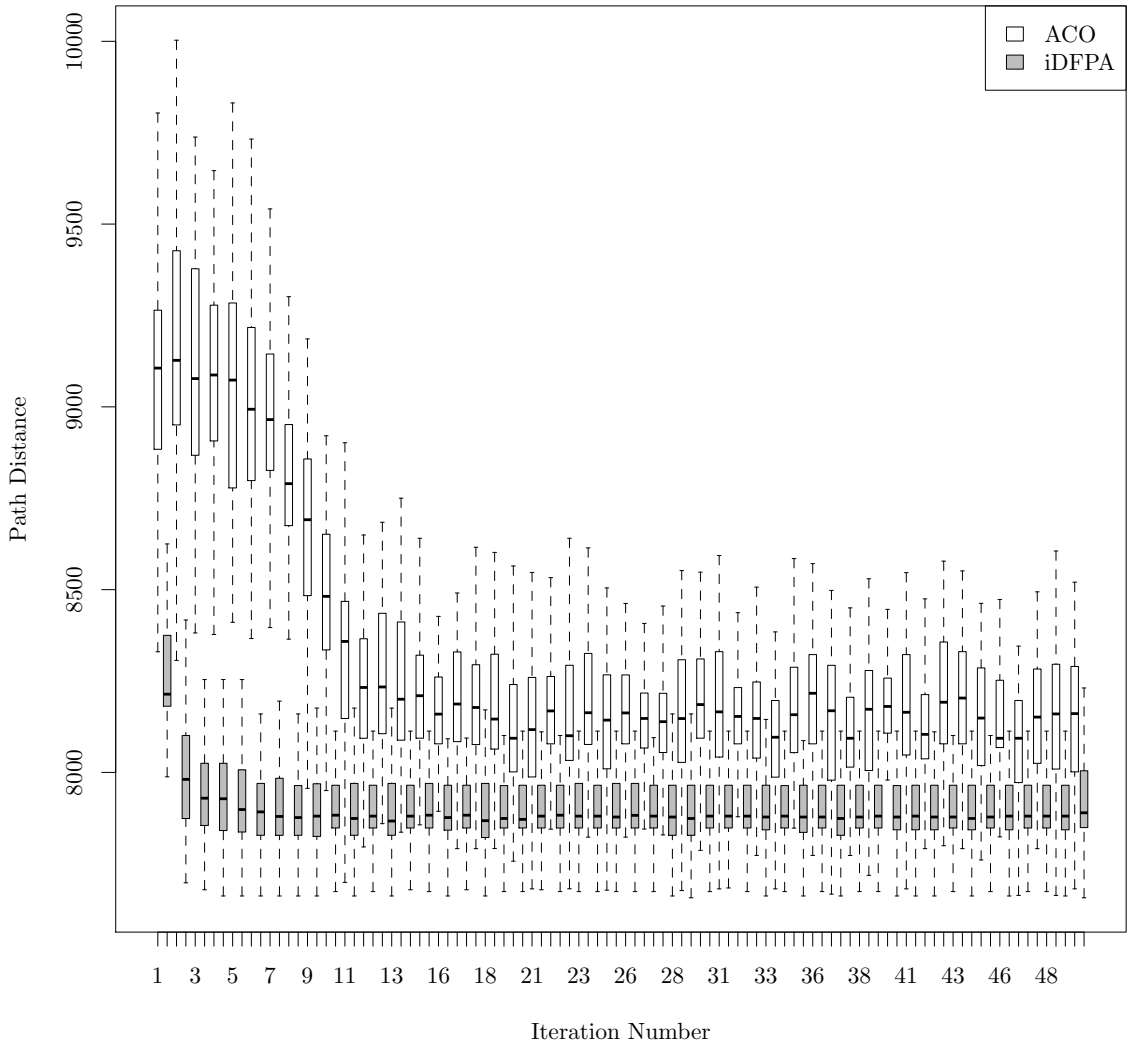


Figure 4.9: A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Berlin52 problem set.

allows for a better overall result. There is a possibility that a hybrid system could be implemented that would allow for these 2 parameters to be activated at some point in the iterative process in order to aid with converging but this is not included with the current implementation. The smaller the problem set, the less influence α, β, γ have on the results, and, therefore, it does not require much customisation on the user's behalf in order to achieve good results. However, as the problem size is increased, altering the parameters becomes more important.

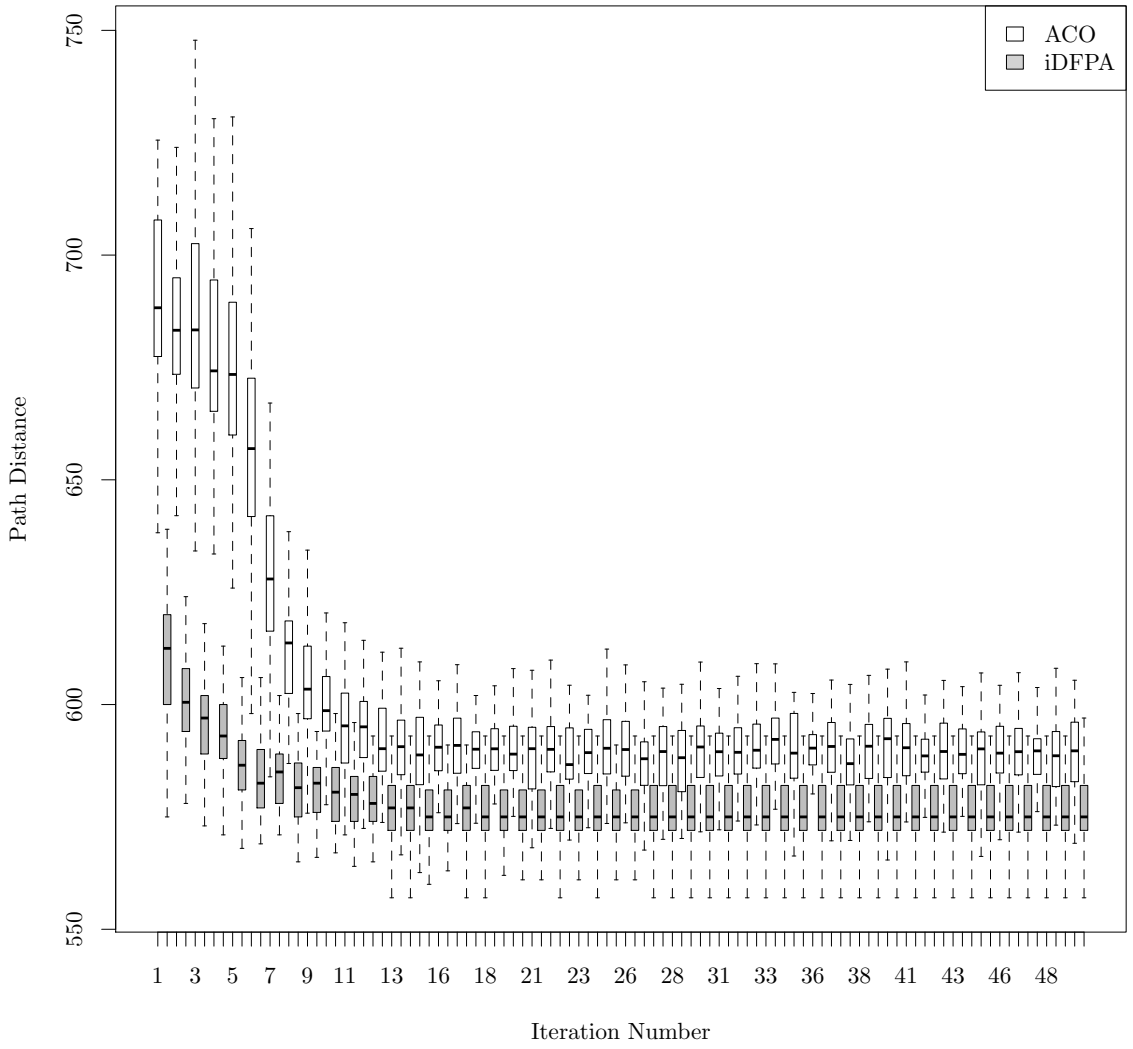


Figure 4.10: A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Eil76 problem set

The annealing schedule and search radius (for the local search) are two parameters which are set to default values within the algorithm as a more advanced understanding of the algorithm and how these parameters affect the results is required before they should be altered. The search radius is set to $\frac{1}{2d_{avg}}$ where d_{avg} is the average distance of all the arcs in the problem set, as this produced the best results in simulations. The default annealing schedule is set as specified as $q = 0.8$, this schedule can be altered and is described in Section 4.3.3 B.

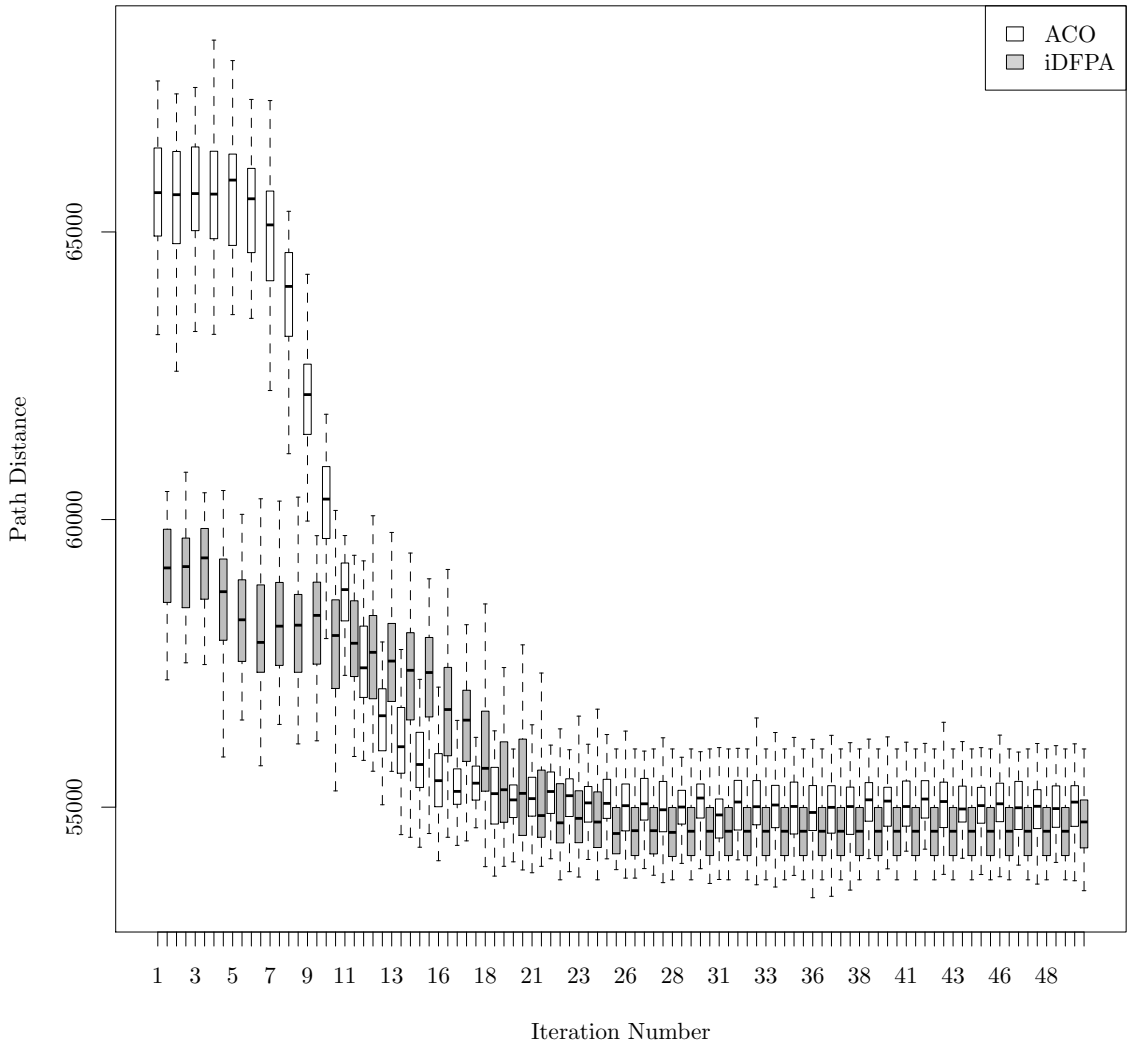


Figure 4.11: A Box Plot curve of the iDFPA vs ACO for a 50 iteration simulation on the Pr264 problem set

Table 7: Generalised Parameter

Problem Size	Small	Medium	Large
ρ	0.5	0.1	0
α	0.1	0.2	0.4
β	0	0.1	0.5
γ	0.1	0.4	0

Chapter 5: Conclusion

5.1 Research Summary

The research presented here aims to answer the question *"Can the Flower Pollination Algorithm (FPA) be implemented to generally solve for a sub-optimal solution for discrete optimisation problems, but more specifically the Travelling Salesman Problem? If so, how does it perform compared to existing solutions as well as to the optimal solution?"*. A literature review examines the various components that make up the basis of the research including the TSP, algorithms used to solve the TSP problems including optimal and heuristic algorithms, and finally, existing nature-inspired algorithms are presented. The modifications and extensions to the FPA as well as the few discrete implementations of the FPA are also discussed. Chapter 3 provides a background to the fundamental concepts and algorithms used in the formulation of the two new algorithms, namely the DFPA and iDFPA. The TSP is explained in detail and the mathematical, graph theory definition is given. The original FPA is discussed in detail to explain all the vital concepts, such as the local and global searches, that were used in the creation of the new discrete algorithms. Detailed explanations of the GA, ACO and SA algorithms are discussed as well. Finally, Chapter 4 presents the two new algorithms and their implementations along with the results and analysis of the comparative study as submitted to the *IEEE Transactions on Cybernetics*.

5.2 Contributions

Listed below are the main contributions of the research:

- Two discrete optimisation algorithms, namely the DFPA and iDFPA, are designed, implemented, and tested. The DFPA algorithm is a single iteration algorithm that is used to give a sub-optimal TSP solution. iDFPA algorithm uses two iterative

methods, namely BTU and RU, to improve on previous iteration's results in order to converge on a sub-optimal TSP solution.

- A comparison between the DFPA, iDFPA, ACO and GA is completed for 3 diverse test TSPs. On all the test problem sets the DFPA is able to outperform the GA with an improvement of up to 35.1%. The iDFPA is able to outperform both the GA and ACO in all problem sets. In the Berlin52 problem, which is a TSP benchmark problem set, the iDFPA with 300 iterations is able to achieve the optimal solution. The iDFPA also achieved improvements of up to 4.56% and 41.87% compared to the ACO and GA respectively during the simulations.

- Suggested generalised parameters for various problems sizes for the two algorithms are determined as an initial guide for achieving good results for the problem size that the algorithm is being run on.

- A comprehensive analysis of the RU method's annealing schedule and its relationship to the overall performance of the iDFPA is done. The acceptance ratio from the annealing process and the overall results are analysed and the relationship between the annealing schedule and the performance is determined. This demonstrates how the annealing schedule can be selected depending on the type of performance required, for example, an annealing schedule can be selected for a faster convergence with a worse result or a slower convergence with a better result, or a combination of the two.

5.3 Recommendations and Possible Future Work

The results and analysis of the DFPA and iDFPA open up areas for further investigation and expansion of this research. The search radius used in the local search of both the DFPA and iDFPA could be investigated further to determine the optimal radius that should be used for various problem sizes so it is not a static value, half of the average distance between nodes, as in the current implementation. The annealing schedule used in the RU for the iDFPA could be investigated further to understand more comprehensively how it affects the results obtained by the algorithm as well as the relationship between the results and the schedule. The future analysis of the annealing schedule could also be expanded to other types of functions and not just the exponen-

tial functions currently implemented. Lastly, the algorithms should be implemented on other TSPs, such as the asymmetric TSP, as well as discrete optimisation problems and compared against the existing algorithms in order to validate the performance across a variety of discrete optimisation problems.

5.4 Conclusion

The results confirm that yes, the FPA can be implemented for discrete optimisation problems such as the TSP as developed in the DFPA and iDFPA algorithms. The new algorithms utilise the local and global search concept from the original FPA with the multiple agent concept of the ACO in order to find sub-optimal solutions to generic TSPs. The iDFPA uses the BTU and RU methods as the base iterative update process, in order to utilise knowledge gained from previous iterations, to achieve the best results. A comparative study between the two algorithms, the ACO's $\mathcal{MMAS}$ and the GA, was performed over three TSP problem sets which encompass a variety of problem sizes and orientations. The study shows that the iDFPA outperforms both the GA and ACO across the board of the test cases run in the simulations of this research. An analysis of the annealing function used in the RU method of the iDFPA demonstrates the function settings that are used to achieve different results from the algorithm. The DFPA and iDFPA control parameter suggestions, for various size TSPs, have been determined for future use of the algorithms.

References

[1] C. Blum, R. Chiong, M. Clerc, K. De Jong, Z. Michalewicz, F. Neri, and T. Weise, "Evolutionary Optimization," in *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–29. [Online]. Available: http://link.springer.com/10.1007/978-3-642-23424-8_1

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[3] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[4] T. A. S. Masutti and L. N. de Castro, "Tspoptbees: A bee-inspired algorithm to solve the traveling salesman problem," in *5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, Kumamoto, Japan, July 2016, pp. 593–598.

[5] M. Dorigo and T. Stützle, *Ant colony optimization*, illustrated ed. MIT Press, 2004, vol. 2, no. 3.

[6] J. Kennedy, J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. Morgan Kaufmann, 2001.

[7] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Springer*

International Conference on Unconventional Computing and Natural Computation. Springer, 2012, pp. 240–249.

[8] A. Maredia and R. Pepper, "History, Analysis, and Implementation of Traveling Salesman Problem (TSP) and Related Problems," University of Houston-Downtown, Houston, Tech. Rep., 2010.

[9] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and Its Variations*, 1st ed., G. Gutin and A. Punnen, Eds. Springer Science & Business Media, 2006, vol. 12.

[10] X.-F. Xie and J. Liu, "Multiagent optimization system for solving the traveling salesman problem (TSP)," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 489–502, 2009.

[11] O. Matei and P. Pop, "An efficient genetic algorithm for solving the generalized traveling salesman problem," in *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. Cluj-Napoca, Romania: IEEE, 2010, pp. 87–92.

[12] A. R. Saiyed, "The Traveling Salesman problem History of The TSP," Indiana State University, Tech. Rep., 2012.

[13] D. Johnson, "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, vol. 1, pp. 215–310, 1997.

[14] Y. Jiankun and G. Jun, "Formal Derivation of Traveling Salesman Problem," in *IEEE International Conference on Management of e-Commerce and e-Government (ICMeCG)*. Shanghai, China: IEEE, Oct 2014, pp. 329–332.

[15] G. Reinelt, "TSPLIBA traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[16] R. M. F. Alves and C. R. Lopes, "Using genetic algorithms to minimize the distance and balance the routes for the multiple traveling salesman problem," in *IEEE Congress on Evolutionary Computation (CEC)*, Sendai, Japan, May 2015, pp. 3171–3178.

[17] J. Li, M. Zhou, Q. Sun, X. Dai, and X. Yu, "Colored traveling salesman problem," *IEEE Transactions on Cybernetics*, vol. 45, no. 11, pp. 2390–2401, Nov. 2015.

[18] M. M. Smith and Y. S. Chen, "A novel evolutionary algorithm for the homogeneous probabilistic traveling salesman problem," in *IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, Okayama, Japan, June 2016, pp. 1–6.

[19] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *Journal of the ACM (JACM)*, vol. 9, no. 1, pp. 61–63, 1962.

[20] X. Yang and J. S. Wang, "Application of improved ant colony optimization algorithm on traveling salesman problem," in *Chinese Control and Decision Conference (CCDC)*, May 2016, pp. 2156–2160.

[21] G. Reinelt, "The traveling salesman: computational solutions for TSP applications," Jan. 1994. [Online]. Available: http://dl.acm.org/citation.cfm?id=1744275

[22] A. M. Frieze, G. Galbiati, and F. Maffioli, "On the worst-case performance of some algorithms for the asymmetric traveling salesman problem," *Networks*, vol. 12, no. 1, pp. 23–39, 1982. [Online]. Available: http://doi.wiley.com/10.1002/net.3230120103

[23] H. Afaq and S. Saini, "Swarm Intelligence based Soft Computing Techniques for the Solutions to Multiobjective Optimization Problems," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 326–334, 2011.

[24] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999, no. 1.

[25] L. Li, Y. Cheng, L. Tan, and B. Niu, "A discrete artificial bee colony algorithm for TSP problem," in *Springer International Conference on Intelligent Computing, Bangalore, India*. Bangalore, India: Springer, 2011, pp. 566–573.

[26] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Springer International Symposium on Stochastic Algorithms*. Springer, 2009, pp. 169–178.

[27] J. A. Ruiz-vanoye, O. Díaz-parra, F. Cocón, A. Soto, M. D. L. Ángeles, B. Arias, G. Verduzco-reyes, and R. Alberto-lira, "Meta-Heuristics Algorithms based on the Grouping of Animals by Social Behavior for the Traveling Salesman Problem," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 3, no. 3, pp. 104–123, 2012.

[28] L. Yin, X. Li, L. Gao, and C. Lu, "A new improved fruit fly optimization algorithm for traveling salesman problem," in *Eighth International Conference on Advanced Computational Intelligence (ICACI)*, Chiang Mai, Thailand, Feb. 2016, pp. 21–28.

[29] P. Zahadat and T. Schmickl, "Wolfpack-inspired evolutionary algorithm and a reaction-diffusion-based controller are used for pattern formation," in *Annual Conference on Genetic and Evolutionary Computation*. Vancouver, Canada: ACM, 2014, pp. 241–248.

[30] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Wolf search algorithm with ephemeral memory," in *IEEE Seventh International Conference on Digital Information Management (ICDIM)*. Macau, Macao: IEEE, 2012, pp. 165–172.

[31] J. B. Odili and M. N. Mohmad Kahar, "Solving the Traveling Salesman's Problem Using the African Buffalo Optimization," *Computational intelligence and neuroscience*, vol. 2016, p. 3, 2016.

[32] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life reviews*, vol. 2, no. 4, pp. 353–373, 2005.

[33] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[34] L. Yang and H. Zhou, "Research on path planning and TSP based on genetic algorithm and Hopfield neural network," in *IEEE International Conference on Computer Science and Service System (CSSS)*. Nanjing, China: IEEE, 2011, pp. 657–659.

[35] C. C. Skiścim and B. L. Golden, "Optimization by simulated annealing: A preliminary computational study for the TSP," in *IEEE 15th conference on Winter Simulation-Volume 2*. Piscataway, NJ, USA: IEEE Press, 1983, pp. 523–535.

[36] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by SA," *Science*, vol. 220, no. 4598, pp. 671–680, 2007.

[37] V. Dwivedi, T. Chauhan, S. Saxena, and P. Agrawal, "Travelling salesman problem using genetic algorithm," *IJCA Proceedings on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI 2012)*, no. 1, pp. 25–30, 2012.

[38] K. Bryant and A. Benjamin, "Genetic algorithms and the traveling salesman problem," *Department of Mathematics, Harvey Mudd College*, pp. 10–12, 2000.

[39] Y. Yu, Y. Chen, and T. Li, "A new design of genetic algorithm for solving tsp," in *IEEE Fourth International Joint Conference on Computational Sciences and Optimization (CSO)*. Washington DC, USA: IEEE, 2011, pp. 309–313.

[40] X.-S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, no. 4, pp. 330–343, 2010.

[41] X.-S. Yang, M. Karamanoglu, and X. He, "Flower pollination algorithm: a novel approach for multiobjective optimization," *Engineering Optimization*, vol. 46, no. 9, pp. 1222–1237, 2014.

[42] D. Rodrigues, X.-S. Yang, A. N. De Souza, and J. P. Papa, "Binary flower pollination algorithm and its application to feature selection," in *Recent Advances in Swarm Intelligence and Evolutionary Computation*. Springer, 2015, pp. 85–100.

[43] O. Abdel-Raouf, I. El-Henawy, and M. Abdel-Baset, "A novel hybrid flower pollination algorithm with chaotic harmony search for solving sudoku puzzles," *International Journal of Modern Education and Computer Science*, vol. 6, no. 3, p. 38, 2014.

[44] N. Diab, "Recent Advances in Flower Pollination Algorithm," *International Journal of Computer Applications Technology and Research*, vol. 5, no. 6, pp. 338–346, 2016.

[45] M. Bensouyad and D. Saidouni, "A discrete flower pollination algorithm for graph coloring problem," in *IEEE 2nd International Conference on Cybernetics (CYBCONF)*. Gdyina, Poland: IEEE, 2015, pp. 151–155.

[46] K. Bibiks, J.-P. Li, and F. Hu, "Discrete flower pollination algorithm for resource constrained project scheduling problem," *International Journal of Computer Science and Information Security*, vol. 13, no. 7, pp. 8–19, 2015.

[47] J. P. Nolan, *Stable Distributions: Models for Heavy-Tailed Data*, illustrated ed. New York, New York, USA: Springer New York, 2012.

[48] S. Łukasik and P. A. Kowalski, "Study of flower pollination algorithm for continuous optimization," in *Intelligent Systems' 2014*. Springer, 2015, pp. 451–459.

[49] J. Li, Q. Sun, M. Zhou, and X. Dai, "A New Multiple Traveling Salesman Problem and Its Genetic Algorithm-Based Solution," in *IEEE International Conference on Systems, Man, and Cybernetics*, Shanghai, China, 2013, pp. 627–632.

[50] B. C. Mohan and R. Baskaran, "A survey: Ant colony optimization based recent research and implementation on several engineering domain," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4618–4627, 2012.

[51] D. Sudholt and C. Thyssen, "Running time analysis of ant colony optimization for shortest path problems," *Journal of Discrete Algorithms*, vol. 10, pp. 165–180, 2012, string Masters 2009.

[52] W. J. Gutjahr, "Mathematical runtime analysis of aco algorithms: Survey on an emerging issue," *Swarm Intelligence*, vol. 1, no. 1, pp. 59–79, 2007.

Appendices

Complexity Analysis

The complexity analysis of the DFPA and iDFPA algorithms is done by analysing the individual components that make up the implementation of the algorithm. The components with significant complexities are shown in Figure 0.1.

The complexity is determined by cascading the individual components' complexities. The complexity of the DFPA is therefore:

$$O(m(n(n + n\log n + n + n)) + mn + m\log m) = O(mn^2 \log n). \tag{A.1}$$

The difference between the DFPA and iDFPA complexities is that the iDFPA has the two update functions as well as a number of iteration multiplier. Complexity of iDFPA per iteration is:

$$O(m(n(n + n\log n + n + n)) + mn + m\log m + mn + n) = O(mn^2 \log n). \tag{A.2}$$

As can be seen from the above equations, the DFPA and iDFPA algorithms have the same complexities even though the iDFPA performs additional methods.

The complexity of both the DFPA and iDFPA is equivalent to $O(n^3 \log n)$ since m is in the same order as n.

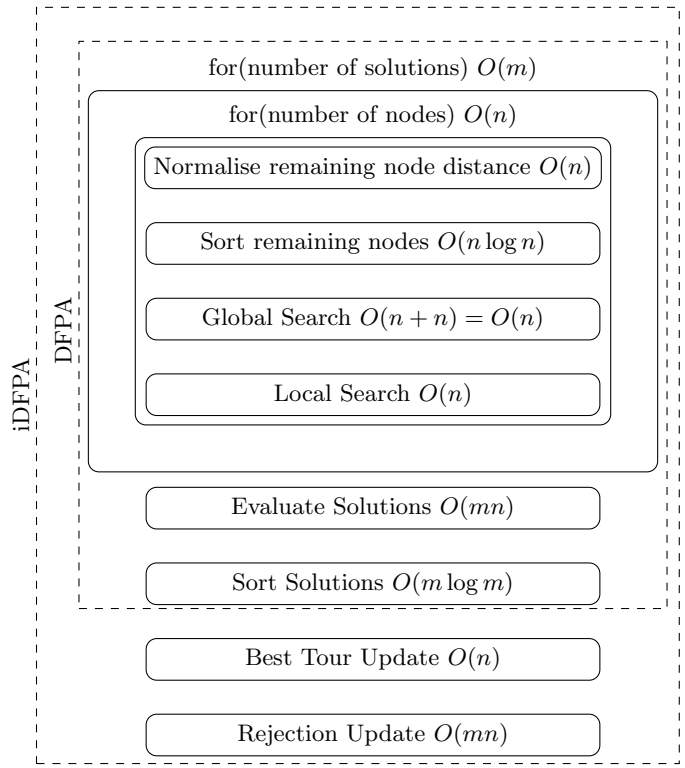


Figure 0.1: Complexity of significant elements of DFPA and iDFPA algorithms