

Solving the TSP using the DFO algorithm for logistical optimisation

Student name: Mahid Miah (ID: 001063915)

Supervisor: Dr. Mohammad Majid al-Rifaie

February 7, 2023

Part I

The Report

Contents

I	The Report	i
1	Introduction	1
1.1	Logistical optimisation based on solving the TSP using the DFO algorithm	1
1.2	Motivation for the use of the DFO	2
1.3	Research Questions	3
2	Background	4
2.1	The Travelling Salesman Problem	4
2.2	The Dispersive Flies Optimisation Algorithm (DFO)	4
2.2.1	What is the DFO	4
2.2.2	DFO basics	4
2.2.3	DFO pseudocode	5
2.3	Literature Review	6
2.3.1	Machine learning (ML) algorithms	7
2.3.2	Ant Colony Optimisation (ACO) algorithm	7
2.3.3	Improved Ant Colony Optimisation (IACO) algorithms	8
2.3.4	Particle Swarm Optimisation (PSO) algorithm	9
2.3.5	Bat based algorithm	10
2.3.6	Wolf based algorithm	12
2.4	Methodology	13
3	Project Description	15
3.1	Proposed DFO algorithm	15
3.2	Textual Description	16
3.2.1	Pseudocode	16
3.2.2	City generation feature	16
3.2.3	Fitness function	16
3.2.4	Updated update function	18
4	Project Evaluation	21
4.1	How the proposed DFO algorithm can be tested and evaluated	21
4.2	Experiments and results	22
4.2.1	Running the DFO algorithm on a TSP	22

4.2.2	Optimising the parameters	24
4.2.3	Further Optimisation	32
4.2.4	Resolving the local optimum trap	36
4.2.5	Testing the DFO on TSP datasets	40
4.2.6	Comparing the DFO against other algorithms	41
4.3	How the project can be further improved based on this evaluation	46
5	Summary and Conclusions	48
5.1	Summary of the Project	48
5.2	Conclusions Drawn from the Project	51

List of Figures

2.1	DFO pseudocode (Al-Rifaie 2020)	5
2.2	DFO update procedure figure	6
3.1	Distance matrix table example	16
3.2	Euclidean distance formula	17
3.3	Euclidean distance example	17
3.4	Fitness function pseudocode	18
3.5	Updated update function pseudocode	19
4.1	wi29 dataset point set and optimal solution	22
4.2	Running DFO on wi29 dataset - table 1	23
4.3	Running DFO on wi29 dataset - table 2	23
4.4	test 1 - results table	25
4.5	test 1 - graph 1	25
4.6	test 1 - graph 2	26
4.7	test 2 - results table	27
4.8	test 2 - graph 1	28
4.9	test 2 - graph 2	29
4.10	test 3 - results table	30
4.11	test 3 - graph 1	31
4.12	test 3 - graph 2	31
4.13	test 1 - results table	33
4.14	test 2 - results table	35
4.15	Local optimum trap - Example	37
4.16	Approach 1 - Oscillating Disturbance Threshold Approach results	38
4.17	Approach 2 - Multiple Swarms Approach results	39
4.18	Running DFO algorithm on datasets	40
4.19	Execution time based on number of nodes	42
4.20	Comparing the PSO, ACO, and GA algorithms against the DFO algorithm	43
4.21	Comparing the DFO, PSO, and ACO algorithms	44
4.22	Comparing the DFO against the GA	45

Chapter 1

Introduction

1.1 Logistical optimisation based on solving the TSP using the DFO algorithm

The logistics industry is an essential service that plays a great and significant role to the economy of any country, with many stating that ‘economies will fall into recession’, if any essential or non-essential logistics movements are affected (Shin et al. 2020).

Based on the research conducted in this report from various research articles it is found that the logistics industry currently has a booming annual growth rate with the package delivery market currently growing at an annual rate of 7% - 10%, whilst in developing markets such as that of India or China the annual growth rate is higher than 300% (Yuan et al. 2021). The main goal within the logistics industry is to achieve logistics movements using the most optimal paths and routes to save on time, costs, and fuel consumption, the optimal paths are often those which are the shortest and quickest.

In recent times, logistics has been a key point mentioned within the UK’s national news for example the recent fuel shortages which has significantly affected most of the UK with many areas including the capital. London still has very low levels of fuel, which is caused by shortage of HGV (heavy goods vehicle) drivers who are a key part and also the backbone of the logistics industry. They are the essential people who transport goods to their required locations, in addition due the shortage of HGV drivers it is also found that many are fearing that this supply chain crisis can lead to a possible hike in food prices due to possible food shortages (Graeme 2021, Lisa 2021).

Furthermore, many researchers have conducted research of the significance of the logistics in regards to economies, with some as mentioned above finding that major disruptions caused to the logistics industry can potentially cause economies to go into recession. The fuel shortage mentioned above is a perfect example of the how disruption of logistics can affect society, as the fuel shortage had caused many within the UK to be unable to carry out their jobs and professions

due to not being able to travel.

Moreover, the shortage of fuel had led to panic-buying which has caused further disruptions with many major roads being stuck in grid-locked traffic due to huge queues at fuel stations. However, this potentially could have led to a catastrophic consequence for the economy as the traffic and long queues caused major havoc. In recent years many researchers have tried to solve the issues faced within the logistics industry by applying the traveling salesman problem (TSP) concept to the logistics industry. The TSP is a combinatorial NP-hard problem, which consists of a theoretical salesman and a set of locations (nodes) which must be visited by the theoretical salesman. More specifically, the theoretical traveling salesman is faced with the challenge of traversing all the cities with a designated start city using the shortest and most optimal route (Juneja et al. 2019). The TSP also has the constraint of only being able to visit each city once (excluding the start city which the salesman must return back to) (Juneja et al. 2019).

In order to resolve these logistics problems faced during the logistical crisis and to further optimise existing logistical systems currently in use, it is proposed in this paper to apply the TSP to existing and new logistical systems. Then to apply a Dispersive Flies Optimisation (DFO) based algorithmic solution to the given TSPs in order to solve them which will result in a much more optimised logistical system. This approach of applying the TSP and then using a DFO based solution is very likely to succeed in both improving existing logistics and to also ease the current logistical issues faced within the logistics industry. This is because existing logistical routes can be optimised to reduce the travel distances and time that HGV drivers require during their deliveries. This means that although there is a shortage of these HGV drivers, the number of existing HGV drivers can make more deliveries as the time and distance required for their previous deliveries would be reduced using an optimised route. Therefore, allowing them to be able to make more deliveries in the same amount of given time.

1.2 Motivation for the use of the DFO

Based on research conducted in this paper, it has been found that historically many different conventional optimisation algorithms have been used to solve and optimise the TSP, including the Ant Colony Optimisation (ACO), Simulated Annealing (SA) algorithm, and the Particle Swarm Optimisation (PSO) algorithm. Despite the fact that these algorithms have been used to solve the TSP frequently, the motivation behind the use of the DFO algorithm being used to optimise and solve the TSP is based on the findings that the DFO algorithm has not previously been used to solve or optimise the TSP. Thus making any and all findings unique, additionally, it has also been found that DFO algorithm has successfully outperform other notable optimisation algorithms that have been used to solve other various optimisation benchmark tests. This includes the PSO algorithm, with Mohammad Majid Al-Rifaie finding that the DFO algorithm "is more efficient in 84.62% and more reliable in 90% of the 28 standard optimization benchmarks used" during his experiments (Al-Rifaie 2014).

1.3 Research Questions

Through the research conducted it is clear that the TSP is a classic combinatorial problem which has attracted the attention of countless researchers, who have all tried to find an effective algorithmic approach to solve it with a high degree of accuracy and a short execution time throughout the decades. Therefore, the key research questions this report aims to successfully answer include, whether or not the DFO algorithm can successfully optimise and solve any given TSP. Moreover, the second research question that this paper will aim to answer, is whether or not the DFO algorithm can successfully optimise and solve any given TSPs to a high degree of accuracy whilst maintaining a low execution time. Furthermore, the final question that this paper will investigate, is if the DFO algorithm is able to outperform other prominent optimisation algorithms that have been used frequently to solve the TSP based on their outputted degree of solution accuracy and also execution time. These predominantly used algorithms include the ACO, PSO, and Genetic Algorithm (GA) which based on the findings of many researchers have been found to be effective approaches to solving the TSP.

Chapter 2

Background

2.1 The Travelling Salesman Problem

The TSP is a classic combinatorial problem that was first proposed during the 1800s by mathematicians Sir William Hamilton and Thomas Penyngton (Hamilton n.d.), and is simply the challenge where a theoretical travelling salesman is required to traverse a series of cities using the most optimal route, often based on having a short accumulated distance. Furthermore the TSP is a NP-hard problem, meaning that no quick and simple solution exists, as the TSP has a very large number of possible different solutions that exist in its solution search space, where the complexity of the problem increases as more nodes are added. Moreover, the solution space for the TSP is based on the factorial of the total number of nodes in the given TSP, so that a given TSP consisting of 10 nodes would result in a solution search space consisting of over 300000 different possible solutions.

2.2 The Dispersive Flies Optimisation Algorithm (DFO)

2.2.1 What is the DFO

The DFO algorithm is an optimisation algorithm first introduced by Mohammed Majid Al-Rifaie in 2014 (Al-Rifaie 2014), and is inspired by the natural behaviour of a swarm of flies that are in search of a food source with the presence of a threat which can lead to the disturbance of their convergence towards an optimal food source (Al-Rifaie 2014). Although, the theoretical threat can cause disturbance to the convergence of the swarm, it also has the positive impact of distributing some of the flies in the swarm to random positions in the given search space which can lead to the disturbed fly finding other potentially better optimal convergence points.

2.2.2 DFO basics

The DFO algorithm consists of a specified population of flies, where each fly in the given swarm represents a possible solution for the given problem the algorithm is being used to optimise

or solve, for example, if the DFO algorithm was being used to find an optimal point on a 3-dimensional space, each fly would contain a solution in the form of a position vector. The DFO algorithm makes use of a ring topology system where each fly has a left and right neighbouring fly, the flies then ‘communicate’ with one another and decide the best neighbouring fly for each given fly in the swarm, where each fly’s solutions fitness value is calculated using a specified fitness function, and the fly with the best fitness value will be chosen as the best neighbour. The algorithm then with each iteration will update the position of each individual fly in the swarm to converge towards an optimal point, which it does based on each fly’s best neighbours position along with the position of the overall best fly and also the disturbance threshold.

2.2.3 DFO pseudocode

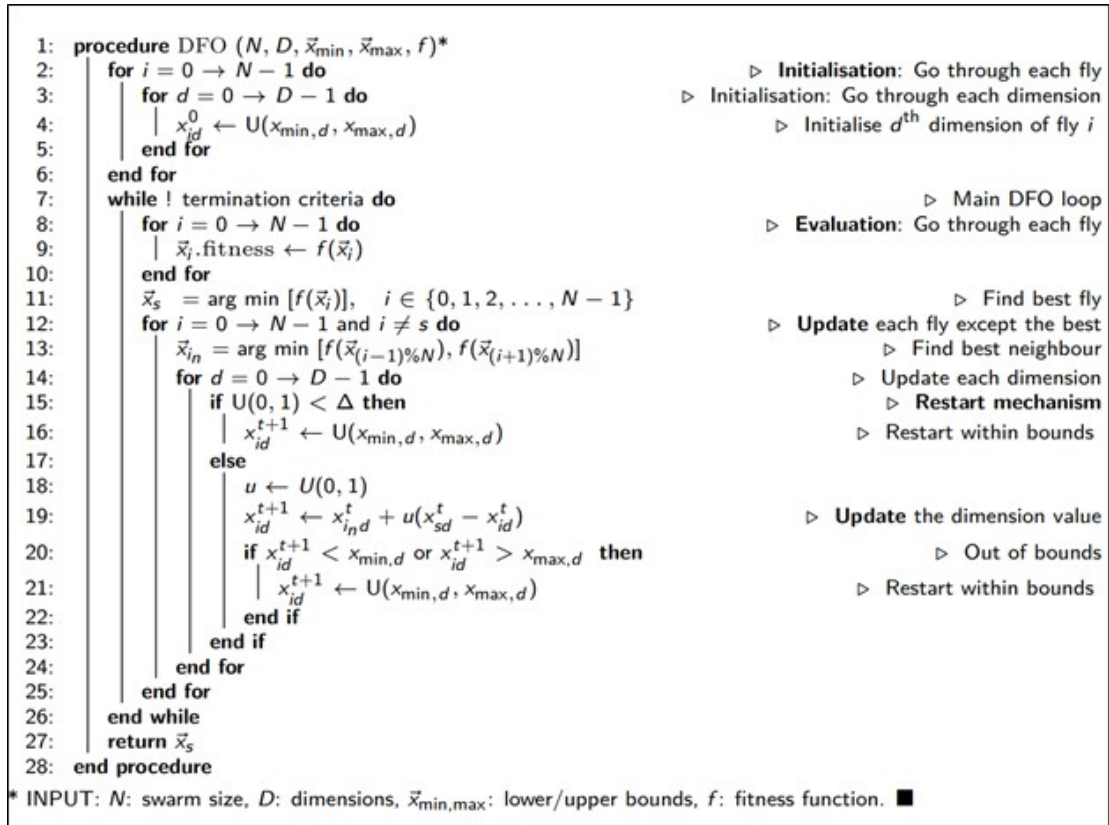


Figure 2.1: DFO pseudocode (Al-Rifaie 2020)

As seen from figure 2.1, the algorithm itself is based on 3 different phases which include an initialisation phase, an evaluation phase, and an update phase. In addition, the algorithm also takes several different input parameters which can affect the result obtained by the algorithm, which include the swarm size, the dimensions, the lower and upper bounds, and the fitness function which is user-defined and is often tailored towards the problem that is being solved or optimised using the DFO algorithm, so given the example where the DFO algorithm is being used to

optimise a time-based problem, the fitness function can be as simple as summing the total time required per solution each fly contains, with the smallest required time being the more desired solution.

Once the algorithm is called, the initialisation phase is started, where a ‘for loop’ is used to loop through each fly in the swarm. An embedded second ‘for loop’ then loops through each dimension in the given fly and initialises it with a randomly generated value within the provided upper and lower bounds of the solution search space.

Next, the main body of the algorithm runs, where the evaluation and update procedures are continuously executed for a specified maximum number of iterations. During the evaluation procedure, a ‘for loop’ is used to loop through each fly in the swarm and find the overall best fly which is determined by calling the fitness function on each fly.

Once the evaluation procedure has completed running, then the update procedure will run were a ‘for loop’ loops through each fly in the swarm and find its best neighbouring fly, an embedded ‘for loop’ is then used to loop through each dimension of the current fly in loop, and based on whether or not a randomly generated value (between 0 and 1) is less than the specified disturbance threshold, the dimension will either be updated or reset within the specified bounds of the solution search space. If the randomly generated value is found to be less than the disturbance threshold then the current dimension in loop will be reset randomly within the specified bounds. However, if found to be false, then the dimension will be updated using the same indexed dimension value of the predetermined best fly in the swarm, the best neighbouring fly and a random value generated between 0 and 1, using the following formula:

$$x_{id}^t + u(x_{sd}^t - x_{id}^t)$$

Figure 2.2: DFO update procedure figure

Where x_{id}^t is the dimension of the best neighbour, ‘u’ is a randomly generated value between 0 and 1 (generated with each dimension update), x_{sd}^t is the dimension of the best overall fly in the swarm (for the current iteration), and x_{id}^t is the dimension of the current fly in loop.

2.3 Literature Review

Based on the research conducted it is found that many researchers have solved the issues faced within the logistics industry by applying the TSP and then applying the various algorithms that exist to solve the TSP, which include machine learning (ML) algorithms such as deep learning,

reinforcement learning, GA, and swarm intelligence algorithms such as the ACO algorithm or the PSO algorithm, and much more.

2.3.1 Machine learning (ML) algorithms

Gianpaolo Ghiani and his colleagues have found that in recent years there have been multiple attempts to use ML algorithms in order to improve the performance of optimisation algorithms (Ghiani et al. 2020). In their research article, they went into great details about how supervised and unsupervised strategies can be used to improve the solutions generated to solve TSP with no increased computation. The researchers in their works had leveraged both supervised learning methods based on the nearest neighbour procedure and also unsupervised learning methods, for which they had used the K-means algorithm paired with Euclidean distance (Ghiani et al. 2020). The researchers had then carried out several experiments in order to show the effectiveness of their approaches in terms of the computation required to solve the TSP. They have tested their approaches in their experiments to solve the TSP using a dataset generated which is based on real travel times between London and Paris (Ghiani et al. 2020). Their results found that their heuristic approach over the traditional nearest neighbour algorithm is on average 4-5% better, however, the researchers find that although their results have improved quality, there are still limitations to this approach, one major limitation is that in order to find the best solution using this approach it could potentially take several up to several hours (Ghiani et al. 2020).

Furthermore, Umberto Junior Mele and his colleagues find that many recent systems which have applied ML to solve the TSP face issues when they are tested on large-scale real use case scenarios with hundreds of nodes which the theoretical salesman must visit (Mele et al. 2021). In order to solve the issues faced with large-scale TSP problems, the researchers propose the use of candidate lists (CLs), which in this case is a subset of all edges for a given node within the TSP which are believed to be part of the optimal solution. In their approach, the researchers state that ML was used to create CLs and their values, additionally the researchers find that the use of the CL restricts the search space during the search for the solution which leads to a significant reduction in computation (Mele et al. 2021). In turn, this allows the proposed approach to work much better with large-scale TSPs without losing any quality throughout the process compared to other systems which also use ML algorithms (Mele et al. 2021).

2.3.2 Ant Colony Optimisation (ACO) algorithm

The ACO algorithm is a classic swarm intelligence algorithm that is very suitable for combinatorial optimisation problems. More specifically the ACO is a meta-heuristic based on ants and their behaviour to identify the shortest and most optimal path between a food source and their colony (Xu et al. 2019). Therefore, the ACO algorithm is often implemented in order to find the most optimal path and has been widely applied to a huge range of applications over the years, these include the TSP, data mining, robotics routing, telecommunication routing and much more (Ning et al. 2018).

Ayoub Ali Mohammed Saeed finds that solving the TSP using classic methods requires a great amount of time and effort (Saeed 2019). Based on this, in his paper, Saeed implements both the ACO approach and a GA approach to solve the TSP (Saeed 2019). After conducting his experiment where he had used both approaches to solve the same TSP, based 100 real Iraqi cities with their respective latitude and longitude. The results were then compared, and Saeed concluded that the ACO algorithm was the superior approach used to find the most optimal path as it takes much less time to compute the most optimal path (Saeed 2019).

In contrast, Wei Gao finds that although the ACO algorithm is suitable to solve the TSP it does however still has many shortcomings which include slow convergence of finding an optimal solution and also low efficiency (Gao 2020). Gao, in turn proposes a new ACO algorithm, in this new ACO, Gao employs a strategy of combining pairs of searching ants, furthermore in order to reduce the number of meeting ants which have influence he has also implemented an influence threshold. Gao had applied his new ACO algorithm in an experiment consisting of 20 TSPs where he had compared his obtained results against 16 other state-of-the-art algorithms which were also used on the same 20 TSPs (Gao 2020). Based on the results, Gao concluded that his proposed algorithm is a highly suitable approach to solve the TSP and its performance is much better than the traditional ACO algorithm with the results showing that his proposed algorithm can obtain better more accurate solutions with less computation and effort (Gao 2020).

2.3.3 Improved Ant Colony Optimisation (IACO) algorithms

Haiou Xiong finds that ‘traditional optimisation methods spend too much time searching’ (Xiong 2021), making it difficult to find the globally optimal path which results in higher distribution costs and lower efficiency (Xiong 2021). The authors therefore proposed the use of the IACO algorithm, based on two different strategies to improve the way that pheromone (a chemical secreted by ants on a given path that other ants follow) is updated in the basic ACO approach, which includes real-time updating and global updating. The “former means that the ant renovates the pheromone of the scheduling path immediately after it gets from one node to another. The latter means that the ant updates pheromones along the path only after it has traversed all the nodes” (Xiong 2021). Based on the author’s research, it was found that the global updating strategy often results in having a good effect. However, in this case, it can lead to the ants converging too early resulting in other possible optimal paths not being found. Overall, the author finds from the experiments conducted the IACO is slightly better than the traditional ACO with the average path found by the ACO being 114.7km while the average found by the IACO being 111km, which is an improvement of 3.7km (Xiong 2021).

Furthermore, Pengzhen Du and his colleagues in their research had implemented an IACO algorithm based on an adaptive heuristic factor (AHACO), in order to solve the TSP (Du et al. 2021). In the article the authors have proposed three main strategies in order to improve the performance of the ACO, these strategies include the use of the k-means algorithm (used to classify cities), a modified 2-opt local optimizer (used for fine-tuning the solution) and they had also implemented a mechanism in order to get out of a local optimal solution (Du et al. 2021). The authors had

conducted an experiment where they had used their proposed IACO (AHACO) algorithm on 39 TSP instances, the results show that the solutions found by the proposed algorithm are 83.33% higher than that of other compared algorithms (Du et al. 2021).

In contradiction, Wu Deng and his colleagues find in their research article that many iterations of the IACO have some inherent disadvantages when the complexity of a large-scale optimisation problem is increased which leads to a slow convergence speed, and a local optimum value being found (Deng et al. 2019). Hence, the authors propose an improved ACO known as the ICMPACO algorithm, which is based on multi-population and co-evolution mechanism, pheromone updating strategy. The authors had tested the ICMPACO algorithm on the TSP and also the gate assignment problem (GAP), the experiments conducted resulted in showing that "the proposed ICMPACO algorithm can effectively obtain the best optimization value in solving TSP and effectively solve the gate assignment problem" (Deng et al. 2019). Overall, it was found that the ICMPACO algorithm that Wu Deng and his colleagues had proposed in order to solve the TSP had resulted in better results compared to other IACO algorithms, thus ultimately making their ICMPACO algorithm a very effective approach to solving the TSP (Deng et al. 2019).

2.3.4 Particle Swarm Optimisation (PSO) algorithm

The particle swarm optimisation (PSO) algorithm was originally proposed by Eberhart and Kennedy in 1995 (Kennedy & Eberhart 1995). The PSO algorithm consists of a swarm of theoretical particles which have been designed to keep a record of their best position in a given search space. Furthermore, the swarm (the entirety of particles) itself is also designed to keep a record of the best particle in the overall swarm. Each particle's next position is then updated based on its current location and velocity and also the location and velocity of the best particle in the overall swarm which allows the particles to converge towards an optimal solution in the given solution search space (Guo & Sato 2020).

Bo Wei and his colleagues have found that the PSO algorithm has some shortcomings, which include premature convergence of a solution and also easily falling into sub-optimal solutions (Wei et al. 2021). In order to solve these issues faced the authors have therefore proposed an improved hybrid particle swarm optimisation (IHPSO) algorithm based on probability initialization and genetic operator strategies (Wei et al. 2021). Furthermore, the researchers find that the PSO algorithm is an effective approach to apply to the TSP, the researchers in their paper state that they have designed their approach based on previous PSO implementations used on the TSP with new features which will ultimately improve the performance of the algorithm compared to the traditional PSO algorithm (Wei et al. 2021). Wei and his colleagues had then benchmarked their algorithm on 9 different TSPs and the results were compared, based on their findings from their experiment it was concluded that their proposed IHPSO algorithm significantly outperforms the other optimisation algorithms which were tested against, in terms of performance and solutions obtained, these other algorithms tested against including a hybrid PSO (HPSO) and a genetic algorithm (GA) (Wei et al. 2021).

Lukasz Strkak and his colleagues have written a research paper where they have implemented a discrete particle swarm optimisation (DPSO) algorithm in order to solve the dynamic TSP (DTSP) (Strak et al. 2018). The researchers state that solving the dynamic TSP has many practical applications which include planning, logistics and chip manufacturing, they also find that it is particularly important in applications where new circumstances such as a traffic jam can forcibly change the problem specification (Strak et al. 2018). In this DPSO approach, the researchers have implemented the algorithm with a pheromone memory which is used to guide the particles during the search process similar to that of the ACO algorithm (Strak et al. 2018). The researchers first tested the performance of their DPSO algorithm on a set of generated DTSPs (each with a different number of nodes). Secondly, they tested their algorithm both with and without the pheromone memory strategy in order to investigate the difference of convergence speeds (Strak et al. 2018). Moreover, they had compared their results against the max-min ant system (MMAS) algorithm and also the population-based ACO (PACO) algorithm and based on their findings they concluded that their DPSO algorithm is suitable to output high quality solutions and its performance is competitive to that of the MMAS and PACO algorithms, they have also found the pheromone memory strategy has a positive impact for the convergence speed of the algorithm (Strak et al. 2018).

Moreover, Nitesh M Sureja has written a research article in which he has solved the random travelling salesman problem (RTSP) with the use of a firefly algorithm (FA), the firefly algorithm is categorized under swarm optimisation algorithms (Sureja 2020). An RTSP is simply a TSP where every instance is randomly generated each time. The FA is a combinatorial optimisation algorithm inspired by nature and was originally proposed by Xin-She Yang in 2008 with the purpose of dealing with continuous optimization problems (Yang 2012). Fireflies in nature communicate with other firefly using its flashing behavior to send signals, attractiveness of a firefly is based on its level of brightness, meaning that fireflies will follow towards the brightest firefly in the swarm, a discrete version of the FA is used by Sureja in order to solve the RTSP. In his experiment, Sureja had tested his discrete FA against 3 other algorithms including a genetic algorithm (GA), on RTSPs ranging from 10 to 100 cities in intervals of 10 cities each iteration (10 tests were conducted), based on the results obtained from his experiments (Sureja 2020). It is concluded that the FA approach to solving the TSP and RTSP is highly suitable and significantly better than other optimisation algorithms and approaches as the results show that the FA is much superior in both computational performance and also in outputting more optimal paths (Sureja 2020).

2.3.5 Bat based algorithm

The Bat algorithm is a swarm intelligence-based algorithm and was first introduced in 2010 by Xin-She Yang (Yang 2010). The Bat algorithm is based on the echolocation capability, echolocation is a type of sonar, used by bats when in search of their prey. Yang finds that bats can use their echolocation capability in order to distinguish different types of animals in complete darkness, which they are able to do by using the echolocation to sense distances and also the size of different animals (Yang 2010).

Yassine Saji and Mohammed Barkatou have written a research article in which they have used a discrete version of the Bat algorithm to solve the travelling salesman problem (Saji & Barkatou 2021). Based on their research, the authors have found that the standard Bat algorithm is widely used due to its simplicity and easy handling allowing for it to be used in a huge range of applications, however, the authors have found that despite the algorithm being very practical to use, the algorithm often falls into the local optimum trap when used on large scale problems (Saji & Barkatou 2021). In order to resolve this major problem faced when using the Bat algorithm on large scale problems such as the TSP, the authors have proposed the use of a new discrete bat algorithm. In order to avoid getting trapped in a local optimum, the authors had enhanced the searching strategy within the algorithm by implementing random walks based on Levy's flights, furthermore they had also implemented a neural crossover operator into the algorithm in order to improve the diversity of the bat's convergence in the algorithm (Saji & Barkatou 2021). In their experiments, the authors had tested their proposed discrete Bat algorithm on 38 different datasets, and they had then compared the obtained results against 8 other algorithms including the standard Bat algorithm (Saji & Barkatou 2021). Overall, Saji and Barkatou have found that based on their experiments and the results obtained that the majority of tests conducted resulted in their proposed discrete Bat algorithm outputting significantly better results, therefore making it very suitable to be used to solve large scale TSPs (Saji & Barkatou 2021).

Furthermore, Eneko Osaba and his colleagues have written a research article in which they implement an improved version of the discrete Bat algorithm (IBA) in order to solve TSPs (Osaba et al. 2016). The authors explain that the improvement made in their IBA algorithm is related to the movement of the bats within the algorithm, all of the bats in the standard BA algorithm move around based on the same pattern regardless of each bats position in the search space of the problem. The authors in their IBA algorithm have implemented what they regard as a type of intelligence to the bats within the algorithm, each bats update function has been updated so that rather like the standard BA algorithm where each bat would move based on the same pattern, each bats position will now be updated based on the position of the best bat in the swarm (Osaba et al. 2016). More specifically when each bat in the swarm is updating its position, the update function will check if the bat's velocity is greater than $n/2$ where 'n' is the number of nodes in the given TSP, if found true then update function will update the bats position in the search space with what is regarded to as a larger move, however, if a bats velocity is less than $n/2$, it can be inferred that the bat has a promising position within the search space which could be potentially very close to the optimal position, and therefore the update function will perform a smaller move for the given bat (Osaba et al. 2016). The authors find that this simple improvement made in their proposed IBA algorithm allows for the bat swarm to better explore the solution search space and therefore enhances the quality of the results obtained from the algorithm. For their experiments, the authors had tested their proposed algorithm on 37 different TSP instances and had compared their results against five other algorithms which include evolutionary simulated annealing, genetic algorithm, an island based distributed genetic algorithm, an imperialist competitive algorithm, and a firefly algorithm (Osaba et al. 2016). Overall, from the experiments performed, the authors found that in the majority of cases their proposed IBA algorithm signif-

icantly outperforms the other aforementioned algorithms including the standard BA algorithm (Osaba et al. 2016).

2.3.6 Wolf based algorithm

The Grey Wolf optimisation algorithm (GWO) is a swarm intelligence-based algorithm and is inspired by wolves found in nature, the algorithm was first introduced in 2013 by Seyadali Mirjalili and his colleagues (Mirjalili et al. 2014). The GWO algorithm is based on the hunting process and leadership hierarchy that wolves follow in their packs. Furthermore, the algorithm makes use of 4 different categories of wolves when mimicking the leadership hierarchy which includes alpha, beta, delta, and omega. In addition, the main process of the algorithm is based on the wolf pack's hunting mechanism, the hunting mechanism is comprised of 3 different steps which include hunting, searching for their prey, and encircling their prey (Mirjalili et al. 2014).

Dibbendu Singha Sopto and his colleagues have proposed the use of a modified version of the Grey Wolf Optimisation algorithm to solve the TSP (Sopto et al. 2018). In their research article, the authors find that the standard Grey Wolf Optimisation algorithm is designed to be for numerical optimisation problems and meaning the standard GWO algorithm will not be compatible to be used on the TSP (Sopto et al. 2018). In order to make the GWO algorithm work with the TSP the authors have proposed the use of a Modified Grey Wolf Optimisation (MGWO) algorithm, in their algorithm, they have implemented a swap operator and also a swap sequence (Sopto et al. 2018). The authors explain that a single swap sequence consists of several swap operators, with each swap operator containing two different values which can be used by the TSP in order to generate new solutions, more specifically, each theoretical grey wolf in the MGWO algorithm is considered as a solution and the implemented swap sequence feature is used within the updating functionality of the algorithm to regenerate each grey wolf with a new solution for each iteration the algorithm runs (Sopto et al. 2018). The authors in the experiments have tested their modified version of the GWO algorithm on several different TSP instances and had compared their results against the results outputted from other popular optimisation algorithms which include GA, ACO, and PSO, and overall, the authors had found that their proposed MGWO algorithm had performed adequately with the authors describing their findings as “a decent technique to resolve TSP” (Sopto et al. 2018).

Rumeng Wang and his colleagues have written a research article in which they have used an implementation of the GWO algorithm in conjunction with multi-strategy optimisation in order to solve the TSP (Wang et al. n.d.). Similar to the works of Sopto and his colleagues (Sopto et al. 2018), Wang and his colleagues have also found that the standard GWO algorithm suffers from the shortcomings of falling into the local minima trap and also slow convergence speeds when solving a TSP. Therefore, the authors propose the implementation of an altered GWO algorithm which incorporates multi-strategy optimisation (MSOGWO) (Wang et al. n.d.). The authors explain that the multi-strategy optimisation is used during the intialisation phase of the wolf population. The weighted distances of the locations for the given TSP is used during the searching stage in the algorithm in order to update each wolves position, by implementing this

optimisation strategy into the algorithm it ensures that wolf population in the algorithm is diverse and that none of the wolves fall into the local minima trap (Wang et al. n.d.). In their experiments, the authors have tested their implementation of the MSOGWO algorithm on several benchmark functions which include the Sphere function, Quadric function, Rosenbrock function, Rastrigin function, Griewank function, and the Ackley functions (Wang et al. n.d.). Based on the results obtained from testing the proposed algorithm in the aforementioned benchmark functions against the standard GWO, O-GWO, and WD-GWO (weighted distance-based GWO) algorithms it is found that the MSOGWO is able to obtain the optimal solution after an average of 15 iterations with better convergence speeds when tested on the Rastrigin, Ackley and Griewank functions (multimodal functions) against the compared algorithms (Wang et al. n.d.). In addition, when the proposed algorithm was tested on the Sphere, Rosenbrock, and Quadric functions (unimodal functions) it was found that MSOGWO algorithm is able to find the optimal solutions on an average of 10 iterations with slightly faster convergence speeds against the compared algorithms which were unable to find the optimal solutions within an average of 10 iterations (Wang et al. n.d.). Moreover, the proposed the MSOGWO algorithm was also tested on 3 different instances of the TSP ranging from city sizes of 20, 40, and 60 (Wang et al. n.d.). The 3 TSP instances were then used to test all of the aforementioned algorithms and based on the results it was found that the proposed MSOGWO algorithm had outputted the most optimal solutions with the fastest convergence speed and performance while doing it with the least number of iterations for all 3 TSP instances compared against the other three algorithms (Wang et al. n.d.).

2.4 Methodology

From the research conducted it was found that in recent years many researchers have applied a wide range of different algorithms to solve the TSP such as Ghiani and his colleagues who had implemented ML algorithms (Ghiani et al. 2020), Ayoub Ali Mohammed Saeed had implemented the ACO algorithm (Saeed 2019), Haiou Xiong had implemented an IACO algorithm (Xiong 2021), and Bo Wei and his colleague researchers have implemented a PSO algorithm (Wei et al. 2021). However, none of these approaches to solving the TSP are unique as many researchers over the years have implemented various variations of the aforementioned approaches and have shared their findings through their research articles.

The DFO algorithm will be used to solve the TSP which will allow for the optimisation of logistical routes in order to provide users with the most optimal route for deliveries with multiple delivery locations. The DFO algorithm will be implemented to solve the TSP as it has not yet been implemented to solve the TSP, hence will result in the findings being unique. Furthermore, Michael King and Mohammad Majid Al-Rifaie have found that the "DFO is more efficient in 84.62% and more reliable in 90% of the 28 standard optimization benchmarks used" (Al-Rifaie 2014). In addition, they also found that the DFO algorithm outperforms the well-known PSO, GA, and differential evolution (DE) algorithms and that it also "converges to better solutions in 71.05% of problem set" (King & Al-Rifaie 2017).

Based on this, the DFO algorithm appears to be highly suitable to solve the TSP. Additionally, in order to develop the proposed algorithmic solution for the TSP, it will be necessary to recreate the DFO algorithm to work with the TSP and its required constraints where a location can only be visited once, and the theoretical salesman must return to the start point at the end of the tour, this will be achieved by developing a TSP compatible DFO algorithm based on the originally proposed pseudocode for the DFO algorithm which is readily available on Mohammad Majid Al-Rifaie's public GitHub repository (Al-Rifaie 2020).

Furthermore, the DFO algorithm will be tested on TSP datasets which are provided with their known optimal solutions, in order to optimise the DFO algorithms parameters with the goal of achieving a high degree of accuracy and a short execution time in order to make it a feasible approach to solve any given TSP, the findings will then be used to successfully answer the research questions of whether or not the DFO algorithm is able to effectively solve any given TSP to a satisfactory degree of accuracy and execution time. In addition, the DFO algorithm will be compared against other prominent algorithms that have been used repeatedly to solve the TSP, such as the PSO, GA, and ACO with the intent of answering the research question of whether or not the DFO algorithm can outperform the aforementioned algorithms based on its degree of accuracy and execution times.

Chapter 3

Project Description

3.1 Proposed DFO algorithm

The original DFO algorithm depicted in figure 2.2 is not suitable to be used for solving the TSP due to the nature of how TSP solutions are formulated. TSP solutions have specific requirements that must be met in order to make it a valid TSP solution, which include the solution to consist only of unique values meaning there can be no duplicate values in the solution. and the solution is required to only contain nodes within the solutions bounds which means there is often a very limited number of nodes from which an updated solutions dimensional values can successfully be formed from.

Due to these requirements, the update procedure from the original DFO algorithm (figure 2.1), cannot be used to solve the TSP, as the update formula can more than often result in a dimension value which is either not in the solution search space or in the form of a decimal value, which cannot be used as TSP solution search spaces do not include nodes with a decimal value.

Therefore, this paper proposes an altered version of the DFO algorithm that is better suited towards solving the TSP based on the original pseudocode but with key changes made which allows the algorithm to support solving the TSP, which include a city generation feature allowing for TSPs to be processed and also an altered update function which allows for individual fly solution dimensions to be successfully updated within the bounds of the given TSPs solution search space.

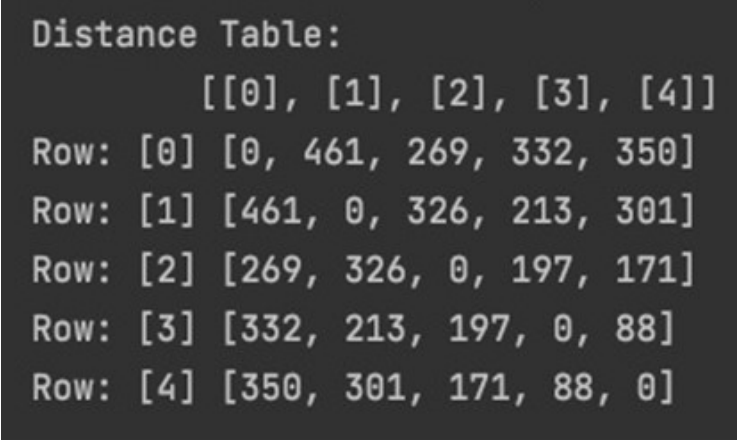
3.2 Textual Description

3.2.1 Pseudocode

3.2.2 City generation feature

The DFO algorithm includes a city generation feature which processes any given TSP city locations into TSP nodes with a distance matrix table which will be used for the fitness function and ultimately the updated update procedure of the algorithm.

The city generator function reads in TSP problems and processes all of the TSP city locations using their node number and their coordinates. For each given city read from the TSP, a ‘city’ object is generated, which contains a node number, an X and Y position value, and also a list of edges between the node itself and all other nodes in the given TSP.



Distance Table:					
	[0]	[1]	[2]	[3]	[4]
Row: [0]	[0]	461	269	332	350
Row: [1]	461	0	326	213	301
Row: [2]	269	326	0	197	171
Row: [3]	332	213	197	0	88
Row: [4]	350	301	171	88	0

Figure 3.1: Distance matrix table example

The city objects are then used to generate the distance matrix table by using the linked nodes (edges) in each city object’s list of edges. A ‘for loop’ is used loop through each generated city object, were another embedded ‘for loop’ is also executed which loops through each edge for the given node and stores the distances between the node and each edge in the distance matrix table. The distance matrix table can simply be used by providing two different node numbers and the distance between the two given nodes will be outputted.

3.2.3 Fitness function

The fitness function is a function which simply takes a solution for a given problem and processes it and outputs its fitness value which in simple terms is the cost or score of how good the given solution is, wherein a minimisation problem, a smaller fitness value would be desired, and in a maximisation problem a larger fitness value would be desired.

For the purpose of solving the TSP the proposed DFO algorithms fitness function will be based on the Euclidean distance where the cost of travelling between two given TSP nodes will be calculated using the Euclidean distance formula.

The Euclidean distance is a mathematical function which outputs the length of a line segment between two given points based on their coordinate values, the Euclidean distance makes use of the Cartesian coordinates of the given points in conjunction with the Pythagorean theorem in order to calculate the distance. The formula for the Euclidean distance function is depicted in figure 3.2.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figure 3.2: Euclidean distance formula

Where x_1 and y_1 is the coordinates for one given point, x_2 and y_2 are the coordinates for the other given point, and 'd' is the distance between the two provided points. So given the example of trying to find the distance between the points (x_1 : 20833.3333, y_1 : 17100.0000) and (x_2 : 23616.6667, y_2 : 15866.6667) the equation will be formulated in the following way (figure 3.3):

$$d = \sqrt{(23616.6667 - 20833.3333)^2 + (15866.6667 - 17100.0000)^2}$$

$$d = 3044.348181$$

Figure 3.3: Euclidean distance example

Where the distance between the two given points is equal to 3044.348181.

The pseudocode for the proposed fitness function procedure is depicted in the figure 3.4. The fitness procedure has a main function labelled 'fit' which takes in 2 parameters for input which include a swarm in the form of a list of generated fly objects, and also a distance matrix table object.

A sub-function 'f' is defined within the main 'fit' function, and takes a single parameter of a fly object. A 'for loop' structure is then used to loop through each fly in the given swarm, and calls the 'f' function using the given fly in loop as the functions input.

```

FUNCTION fit(swarm, distanceTable):
    fitnessTable = []

    FUNCTION f(fly)
        fitness = 0
        FOR dimension IN range 0 to length of fly:
            IF dimension is NOT EQUAL to length of fly -1:
                THEN currentDimension EQUALS fly[dimension]
                AND nextDimension EQUALS fly[dimension + 1]
                AND cost EQUALS distanceTable[currentDimension][nextDimension]
                AND fitness EQUALS fitness PLUS cost
            ENDIF
            ELSE:
                currentDimension EQUALS fly[dimension]
                AND nextDimension EQUALS fly[0]
                AND cost EQUALS distanceTable[currentDimension][nextDimension]
                AND fitness EQUALS fitness PLUS cost
        OUTPUT fitness
    FOR fly IN swarm:
        fitnessTable.append(f(fly))
    OUTPUT fitnessTable

```

Figure 3.4: Fitness function pseudocode

The ‘f’ function makes use of a ‘for loop’ which loops through each dimension value in the inputted fly where the cost of travelling between the current dimension value and the next dimension value of the given fly is calculated using the inputted distance matrix table (where the distances between all nodes have been calculated using the Euclidean distance) and is added to the ‘fitness’ variable (which is initially set to 0) so that once the ‘for loop’ has completed executing the fitness value of the overall fly’s solution will be calculated. Each fly’s solutions fitness value is then appended and stored in the ‘fitness Table’ array and is outputted by the main ‘fitness’ function.

The use of the Euclidean distance for the fitness function is very suitable for the given problem, as the goal for the TSP is to find the most optimal path for the theoretical salesman to take which in other words is the path that requires the least amount distance to traverse all given nodes, hence finding the distance cost between different TSP nodes using the Euclidean distance function is crucial in order to produce an optimal path.

3.2.4 Updated update function

As previously mentioned above, the standard DFO algorithm’s update procedure is not compatible with the TSP due to the solution restrictions the TSP requires, hence an updated update procedure has been proposed which will allow the DFO algorithm to successfully work with the TSP.

```

FUNCTION update(swarm, bestNeighbourTable, fitnessTable, delta, citiesObj, visualise):

    bestFly = np.argmin(fitnessTable)
    cities = [i FOR i IN range(len(sswarm[0]))]

    FUNCTION mainUpdate(fly):
        bestNeighbour = bestNeighbourTable[fly]

        FUNCTION is_fly_valid():
            IF len(set(swarm(fly))) == len(swarm(fly)):
                return TRUE
            return FALSE

        FUNCTION update_dimensions():
            FOR dimension IN range(len(swarm[fly])):
                IF np.random.rand() LESS THAN delta:
                    swarm[fly][dimension] = random.choice(cities)
                CONTINUE

            u = np.random.rand()
            swarm[fly][dimension] = int(swarm[bestNeighbour][dimension] PLUS u TIMES (swarm[bestFly][dimension] MINUS swarm[fly][dimension]))

            IF swarm[fly][dimension] NOT IN cities:
                swarm[fly][dimension] = random.choice([i FOR i IN cities NOT IN swarm[fly]])

        update_dimensions()

    IF NOT is_fly_valid():
        missingDimensions = [dim FOR dim IN cities IF dim NOT IN swarm[fly]]
        duplicateDimensions = [(dim, count) FOR dim, count IN Counter(swarm[fly]).items() IF count MORE THAN 1]
        FOR dimension in duplicateDimensions:
            IF dimension[1] == 1:
                swarm[fly][swarm[fly].index(dimension[0])] = missingDimensions.pop()
            ELSE:
                FOR i IN range(dimension[1] - 1):
                    swarm[fly][swarm[fly].index(dimension[0])] = missingDimensions.pop()

    FOR fly IN range(len(swarm)):
        IF fly == bestFly: CONTINUE
        mainUpdate(fly)

```

Figure 3.5: Updated update function pseudocode

The proposed update procedure is depicted in the figure 3.5. The main update function takes in multiple parameters including a best neighbour table, fitness table, delta value (disturbance threshold), and a list of city objects.

The update function procedure when called initialises a variable named ‘bestFly’ and sets it to the best fly in the swarm using the inputted fitness table, in addition, a list containing integer values from 0 to the length of maximum TSP nodes available is initialised and labelled as ‘cities’, which is used during the updating procedure.

A ‘for loop’ then loops through each fly in the swarm and with each iteration an ‘if statement’ checks if the current fly in the loop is equal to the ‘bestFly’ variable, if found true the loop continues to the next fly in loop, this is known as the elitist strategy where the best fly in the swarm is not updated. However, if the ‘if statement’ returns as false then an embedded sub-function labelled ‘mainUpdate’ will be called using the given fly in loop as its input parameter.

The ‘mainUpdate’ function uses the best neighbour table to retrieve the inputted fly’s best neighbour and stores the value in the variable labelled ‘bestNeighbour’. The ‘mainUpdate’ function

then calls another embedded sub-function labelled ‘update_dimensions’ which is used to update the inputted fly’s dimensional values.

The ‘update_dimensions’ function still makes use of the same update formula from the original DFO algorithm which is depicted in figure 2.2.

The ‘update_dimensions’ function uses a ‘for loop’ to loop through each dimension value in the given fly and with each iteration an ‘if statement’ is used to check if a randomly generated value is less than the inputted ‘delta’ value (the disturbance threshold), if found true then the current dimension value in loop will be reset to a random value from the previously generated list of TSP nodes labelled ‘cities’, the function will then continue to the next dimension in loop and skip the updating process, this procedure is the disturbance mechanism of the algorithm. However, if the statement is returned as false, then the updating procedure will continue and update the dimension value based on the formula depicted in figure 2.2. Moreover, an out of bounds control check has also been implemented in the form of an ‘if statement’ which checks if the updated dimension value is within the bounds of the search space, if found false, then the dimension will be reset to a random TSP node value from the ‘cities’ list. It should also be noted that the updated dimension value is cast to an integer to ensure that the dimension value is not in the form of a decimal value which would violate the requirements of a valid TSP solution.

Once the ‘update_dimensions’ function has completed running and the given fly’s solution has been successfully updated, then the ‘is_fly_valid’ embedded sub-function will be called using an ‘if statement’. Unlike the original DFO algorithm the implemented ‘is_fly_valid’ function is an additional out of bounds control structure that has been added, which is used to check whether or not the entire fly’s solution is valid based on the principles of a TSP solution. The ‘is_fly_valid’ function does this by taking the current fly in loop from the ‘mainUpdate’ function and checks if the set value length of the fly’s solution is equivalent to the number of available TSP nodes.

If the ‘is_fly_valid’ function returns as true, the ‘mainUpdate’ function will complete executing and the inputted fly would now have a valid updated solution. However, if the ‘is_fly_valid’ function returns as false then the missing TSP nodes from the fly’s solution will be found by checking which values from the ‘cities’ list is not in the given fly’s solution, the missing values are then stored in a list labelled ‘missingDimensions’. Furthermore, duplicate TSP node values in the fly’s solution are also found by checking the count of each individual dimension value in the solution, and if a value is counted more than once then the duplicate value will then be appended to the ‘duplicateDimensions’ list object. A ‘for loop’ then loops through each value in ‘duplicateDimensions’ list object, and with each loop, the first index instance of the duplicate value is replaced by a popped value from the ‘missingDimensions’ list object, so that by the time the ‘for loop’ is completed running, the given fly’s updated solution will then be valid based on the principles of a TSP solution.

Chapter 4

Project Evaluation

4.1 How the proposed DFO algorithm can be tested and evaluated

In order to evaluate the effectiveness and suitability of the DFO algorithm's capability of solving the TSP, the DFO algorithm's performance based on its average accuracy and execution time will be evaluated, in order to achieve this, the DFO algorithm will be used on various TSP datasets, where the results will then be compared against the performance of other prominent algorithms which are often used to solve the TSP, the final results will then be analysed and evaluated.

Firstly, with the use of publicly available TSP datasets from the National Traveling Salesman Problems (NTSP) repository, which consists of 25 different TSPs provided with their optimal solutions will be applied to the DFO algorithm, and the average accuracy and execution times of solving the given datasets will be recorded where the average accuracy will be based on how close the final outputted solutions fitness value is to that of the provided optimal solutions fitness value.

Secondly, the DFO algorithm's performance will be compared to other prominent optimisation algorithms which have been used to solve the TSP time and time again, including the PSO, GA, and ACO on the same datasets from the NTSP repository where the average accuracy, execution time, and best outputted solutions will be analysed and evaluated similar to that of Madhumita Panda who had compared the performance of the GA, PSO, and SA algorithms applied to the TSP based using multiple datasets, where the performance was based on the obtained solutions and their respective run times (Panda 2018).

4.2 Experiments and results

4.2.1 Running the DFO algorithm on a TSP

The dataset

The dataset which will be used to conduct this experiment is the ‘wi29’ dataset in the form of TSPLIB file. The ‘wi29’ dataset is a TSP consisting of 29 real cities (nodes) from the Western Sahara region and was acquired from the NTSP repository.

The ‘wi29’ dataset itself consists of 29 different nodes numbered from 1 to 29 where each city is provided with their coordinate values. As mentioned previously the dataset is provided with its known best optimal solutions fitness value of 27603, in addition, the fitness function employed to represent the best optimal solution is the same as the one used in the proposed DFO algorithm, which is the Euclidean Distance, with the only small difference being that the fitness values for the dataset is calculated to 2 decimal points.

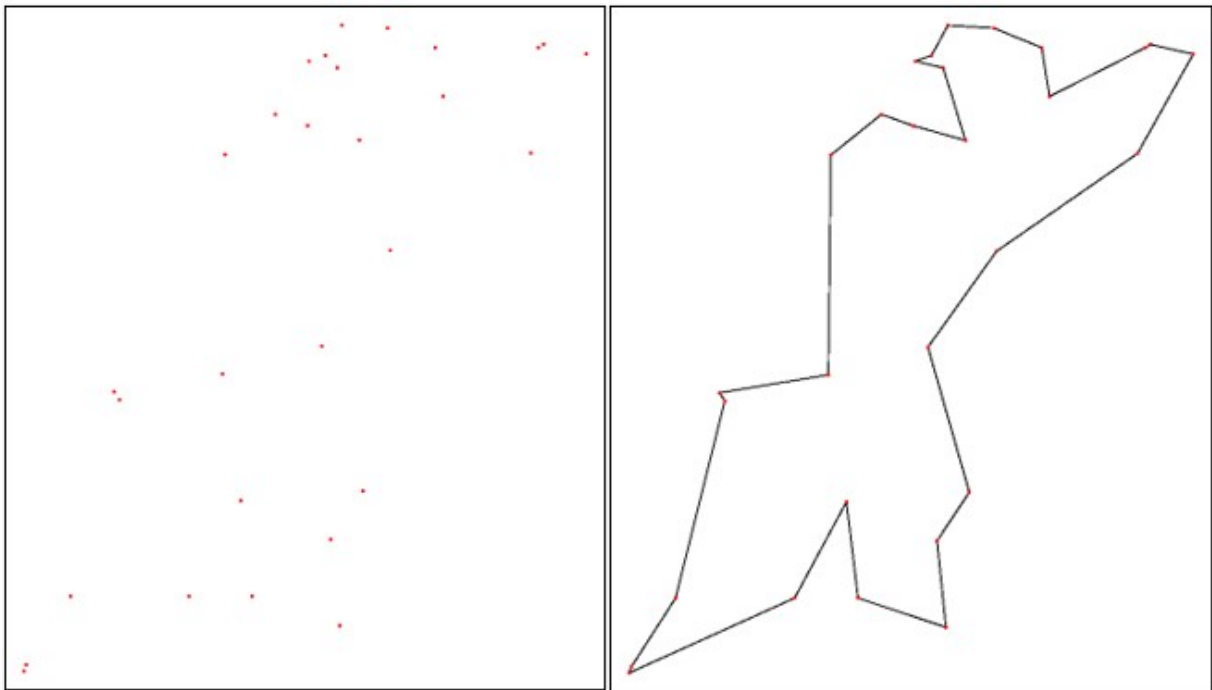


Figure 4.1: wi29 dataset point set and optimal solution

The figure displayed above (figure 4.1) depicts the point set and also a visual representation of how the known most optimal solution should look like.

Running the DFO algorithm on the dataset

For this experiment, the DFO algorithm will be run on the ‘wi29’ dataset 20 times using the default parameters of a swarm size of 100, a disturbance threshold value of 0.001, and a maximum iterations value of 1000. The results will be recorded and compared against the provided most optimal fitness value of 27603.

Execution number:	Final best solution fitness value:	Difference from best known solutions fitness value:
1	32354.09753099748	15.8483%
2	31318.186405933073	12.6107%
3	32750.37655488291	17.0575%
4	32042.816748520596	14.8873%
5	34207.367253772274	21.3698%
6	34008.60805967144	20.7935%
7	36381.28767875445	27.4389%
8	31668.682002603025	13.7188%
9	34728.8636689962	22.8643%
10	30591.16232612589	10.2696%
11	31529.468341355983	13.2802%
12	33726.84135182276	19.9702%
13	32963.617129827595	17.7016%
14	33929.13290344359	20.562%
15	35275.82051657107	24.4051%
16	31916.662378555004	14.4949%
17	30311.394300086176	9.3531%
18	34873.86463143929	23.2754%
19	30536.390122382498	10.0909%
20	32763.35843504962	17.0968%

Figure 4.2: Running DFO on wi29 dataset - table 1

	Fitness Value	Difference from known fitness value (%)
Average	32893.899917039555	17.4915%
Maximum	36381.28767875445	27.4389%
Minimum	30311.394300086176	9.3531%

Figure 4.3: Running DFO on wi29 dataset - table 2

The results from this experiment can be seen from the table in figure 4.2.

Furthermore, as seen from the second table displayed above (figure 4.3), the average, maximum, and minimum fitness value and percentage differences are calculated.

As seen from the table (figure 4.3), the average fitness value outputted from the 20 DFO executions on the given ‘wi29’ TSP dataset is 32893.899917039555 and the average difference from the optimal fitness value is 17.4915%. These values, particularly the average percentage difference between the outputted fitness values against the known optimal fitness value (17.4915%) is too high indicating that the average accuracy of the proposed DFO algorithm is not great, as a much lower percentage value would be desired and is ideal in order to show that the proposed algorithm can successfully solve any given TSP to a high degree of accuracy.

Moreover, although the minimum recorded fitness value was 30311.394300086176 with a percentage difference of 9.3531% from the known optimal solutions fitness value is a desirable result as it is closer to the know optimal solution, it can however be considered as an anomaly as the standard deviation for the average outputted solution fitness values is 1661.9981556246 which is very large and further proves that the DFO algorithm is not as efficient as it could be, as the algorithm is failing to output solutions to a high degree of accuracy which is of great importance for the TSP and therefore optimisation of the algorithms parameters is required to improve the accuracy of the solutions being outputted.

4.2.2 Optimising the parameters

In order to optimise the DFO algorithm to ensure a high degree of accuracy for the outputted solutions, a range of experiments have been devised based on the swarm size, disturbance threshold value, and the number of maximum iterations.

Test 1: Swarm (population) size parameter testing

For this first test, the DFO algorithms swarm size parameter was tested thoroughly were DFO algorithm was run a total of 400 different iterations in order find the most optimal swarm size parameter value. More specifically 20 different test iterations were executed where for each of the test iterations a further 20 DFO executions were executed with the purpose of finding the average, best, and worst fitness and execution time values. Each test iteration used the default maximum iterations value of 1000, and the default disturbance threshold value of 0.001. However, for the swarm size parameter, the value ranged from the default starting value of 100 up to 2000 in increments of 100 per each iteration.

Based on the results it was found that as the swarm size had increased with each iteration the average fitness value had also decreased (a lower average fitness value is desired). This can be seen from the graph in figure 4.5 were the average fitness had gradually improved as the swarm size had increased, which can also be seen from the table were the swarm size was set to 100 had resulted in the worst average fitness value of 33274.01254 and when set to 2000, it had resulted in the average fitness value of 30574.4948 which is an improvement of 8.45601%. Moreover,

Swarm Size	Average Fitness	Average Time	Best Fitness	Best Time	Worst Fitness	Worst Time
100	33274.01254	4.790929329	30344.72415	4.616040468	40774.39031	5.321198463
200	32028.10454	9.548601234	29351.60497	9.209075212	34561.70502	9.773201704
300	32302.04984	14.64599969	29068.91746	14.21820307	37884.39708	15.48148847
400	31154.58925	19.64687624	28318.0585	19.01328325	33944.90497	20.43260336
500	31522.40131	24.72436998	28229.56312	23.65432906	34777.92064	26.35593772
600	31479.03647	29.49424477	28746.00517	28.650455	34571.31115	30.88695908
700	31572.4528	34.43845832	28777.48541	33.13846564	36360.56835	38.72772479
800	31685.28044	39.57736669	29258.33896	38.20760751	34523.16082	40.77718663
900	30905.03446	44.3776477	28777.48541	42.75763273	33523.68025	46.55548859
1000	31359.98629	49.33591465	28588.06391	47.80376935	35607.63614	52.5248332
1100	31279.7045	55.15206431	28870.75143	52.69087052	37023.92578	62.94918108
1200	31462.37188	59.73059663	28229.56312	57.38092709	38288.80402	62.17200661
1300	30584.31161	64.39441884	28019.96875	61.76591396	34278.47396	66.01387143
1400	31644.19911	69.36122606	29068.91746	66.0088706	36799.71028	73.41953993
1500	31595.52498	73.90722525	28229.56312	70.79894996	37245.67616	76.64126611
1600	31375.51929	79.4837062	28082.02732	76.24217629	36194.78117	86.5675025
1700	30819.97664	83.92561818	28082.02732	80.81020522	33833.97468	88.36890817
1800	30893.86261	88.78350148	28189.75219	85.02715516	33833.97468	99.72046566
1900	30514.15716	93.28199009	27601.17377	90.00678062	33619.0142	95.93361259
2000	30574.4948	98.17580904	28777.48541	94.40826869	33697.71802	102.2380328

Figure 4.4: test 1 - results table

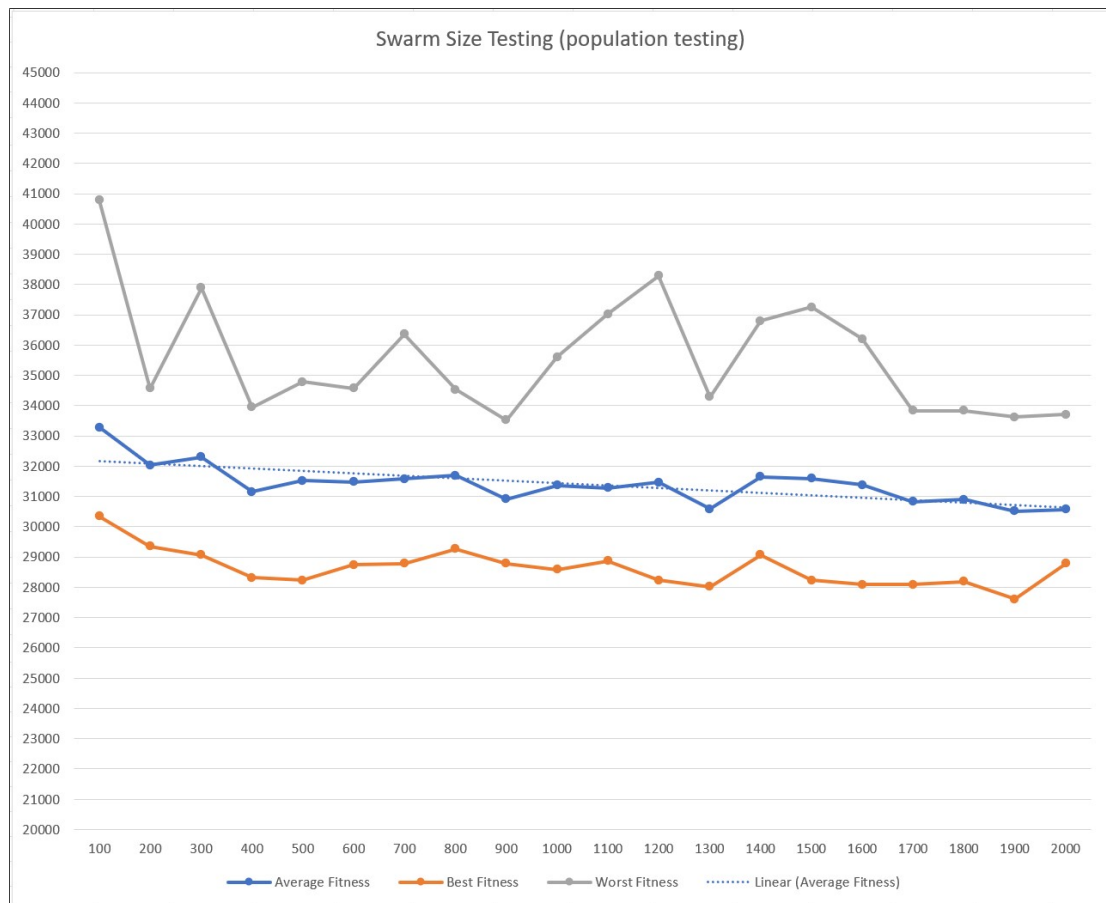


Figure 4.5: test 1 - graph 1

this finding is also further backed by the linear trend line for the average fitness value which is descending as can be seen from the graph (figure 4.5) further indicating that the greater the swarm size is, the lower the fitness value be.

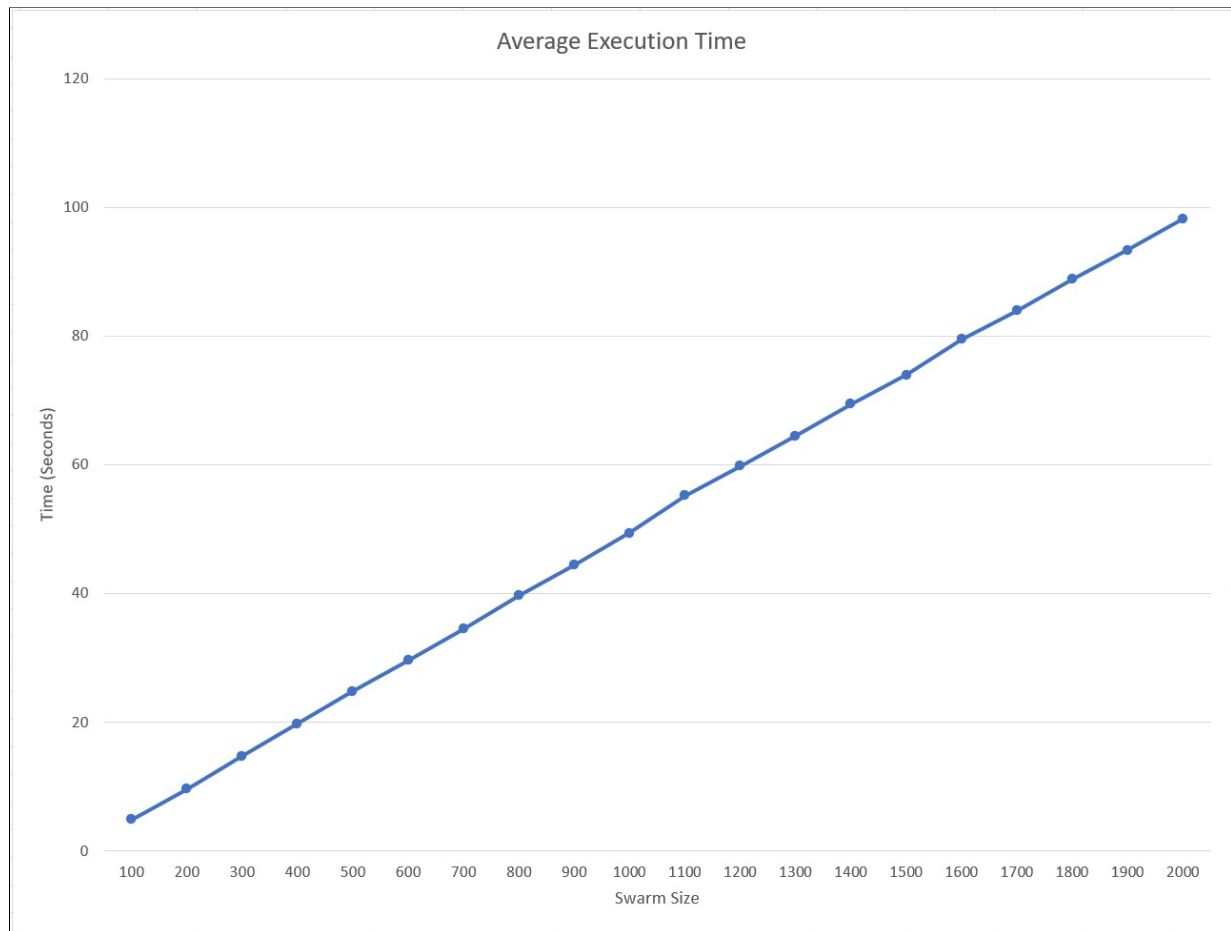


Figure 4.6: test 1 - graph 2

Furthermore, despite the finding that the greater the swarm size is the lower the fitness value will be, it was also found that the execution time is directly proportional to the swarm size, meaning that as the swarm size increases so does the required execution time which can be seen from the graph in figure 4.6, were the swarm size of 100 had resulted in an average execution time of 4.790929329 seconds, while the swarm size of 2000 had resulted in an average execution time of 98.17580904 seconds, which is an overall deterioration of 181.388%.

Overall, it was found that the greater the swarm size, the lower the fitness value will be, additionally it was also found that the swarm size value and execution times were directly proportional which can be seen from the graph in figure 4.6, were as the swarm size is increased the execution time is also equally increased. Furthermore, it was also found that the swarm size parameter

value set to 1900 resulted in the most optimal average fitness value being outputted, as it had resulted in the best average fitness value of 30514.15716 which is an improvement of 7.50612% compared to the initial average fitness value (32893.89991704) outputted using the default parameters. In addition, the use of the swarm size parameter set to 1900 had also resulted in the best overall fitness value outputted during the whole test which was the actual known solutions fitness value of 27601.17377 meaning that the proposed algorithm using the found best swarm size parameter was able to successfully solve the given ‘wi29’ TSP dataset.

It should be noted that the provided know best solution fitness value for the given dataset is 27603, however, the reason that the proposed DFO algorithm had outputted a lower fitness value is due to the provided optimal solution using the Euclidean distance rounded to the nearest whole number, which the proposed algorithm does not do, this difference is not of any great significance and will cause no overall issues.

Test 2: Disturbance threshold parameter testing

Disturbance Threshold	Average Fitness	Average Time	Best Fitness	Best Time	Worst Fitness	Worst Time
0.001	31373.58496	97.41974353	29251.28365	92.24078059	37460.32438	120.0401027
0.0009	30822.88837	94.85802009	28113.12222	91.45360303	33230.99609	100.9037321
0.0008	31093.31965	95.35571647	28563.95006	91.35458088	35749.77892	99.233356
0.0007	31294.36484	94.62076646	28101.63167	90.55340099	34033.56739	102.3290529
0.0006	30252.8914	94.50061469	27601.17377	91.58663273	33333.95678	103.5053179
0.0005	30977.02285	92.09969847	28588.06391	89.35312986	33478.26776	97.1268816
0.0004	31007.2559	92.51523522	27748.70958	89.72656918	34033.56739	95.95861793
0.0003	30830.29592	91.41454414	29068.91746	87.87279654	33975.25065	98.01208067
0.0002	31418.36689	91.25870912	28750.41584	86.82756066	37660.47514	95.18644357
0.0001	31932.78411	88.87637237	28019.96875	85.02715516	35471.35411	96.25868559
0.00009	31501.40769	89.3964649	29341.02759	84.99214745	33855.63028	95.59853673
0.00008	31372.6348	88.73984152	28500.8223	84.87012005	35191.97389	93.95616674
0.00007	32273.87501	88.37798536	29857.64157	83.81288171	37392.15553	95.05291843
0.00006	31780.11442	87.18489116	29862.05022	83.28576303	34924.92763	95.2824657
0.00005	30517.92716	86.8456902	27601.17377	82.76664615	34063.73335	93.29601789
0.00004	30719.7831	85.86199328	28588.06391	82.08649278	33480.33274	90.61341405
0.00003	31900.83112	86.86721987	28290.79048	81.82743454	34926.96286	93.57508087
0.00002	32252.64572	85.73673545	29068.91746	82.09149384	37380.26582	90.69643235
0.00001	31854.93577	83.34595176	29068.91746	80.38110852	35273.08659	91.0795188
0.000009	32068.19446	83.85154053	28082.02732	79.67494965	36611.59294	90.77345014

Figure 4.7: test 2 - results table

For this test, the proposed DFO algorithm’s disturbance threshold parameter will be thoroughly tested. Similar to that of the previous test the proposed DFO algorithm will be run a total of 400 different iterations in order to find the most optimal disturbance threshold value.

For each of the 20 test iterations, the maximum iterations value will be set to 1000 (default value), and the optimal swarm size parameter value will be set to 2000 based on the findings from the previous test. However, for the disturbance threshold parameter, the value will range from the default starting value of 0.001 down to 0.000009 in decrements of a single unit value per each of the 20 iterations.

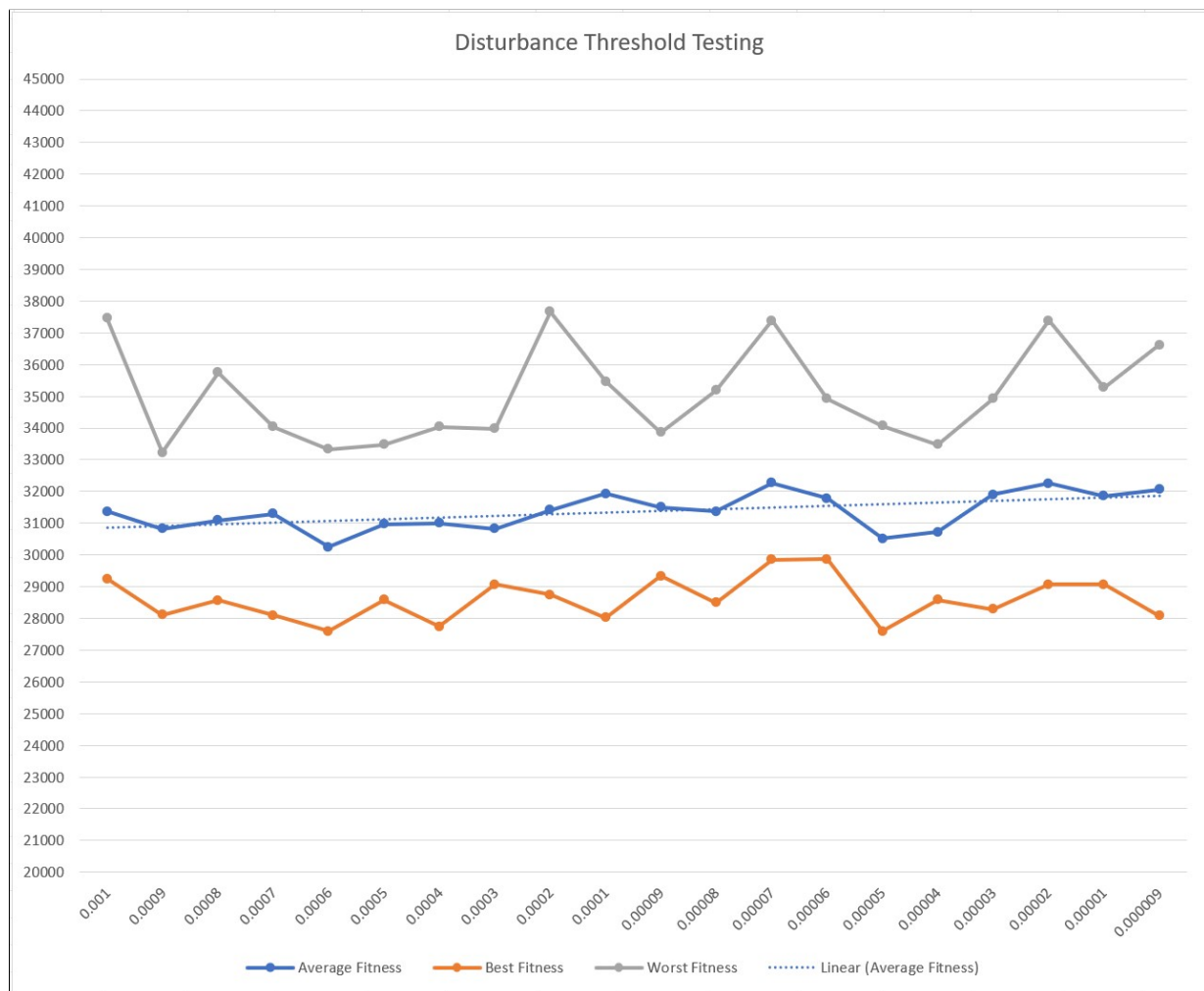


Figure 4.8: test 2 - graph 1

Moreover, it was found that as the disturbance threshold value was decreased the greater the average fitness value was which be seen from the graph in figure 4.8 were the average fitness value had gradually increased as the disturbance threshold value was lowered, this finding can also be seen from the table in figure 4.7, were the disturbance threshold value of 0.001 resulted in the average fitness value of 31373.58496, but when the disturbance threshold was set to 0.000009 had resulted in the average outputted fitness value of 32068.19446 which is a deterioration of 2.18975%. In addition, this finding is also further supported by the linear trend line fitted to the average fitness value which can be seen to be ascending which further indicates that a lower disturbance threshold will result in a greater average fitness value which is undesired.

Furthermore, it was also found that as the disturbance threshold value is decreased, the average required execution time is also decreased as can be seen from the graph depicted in figure 4.9

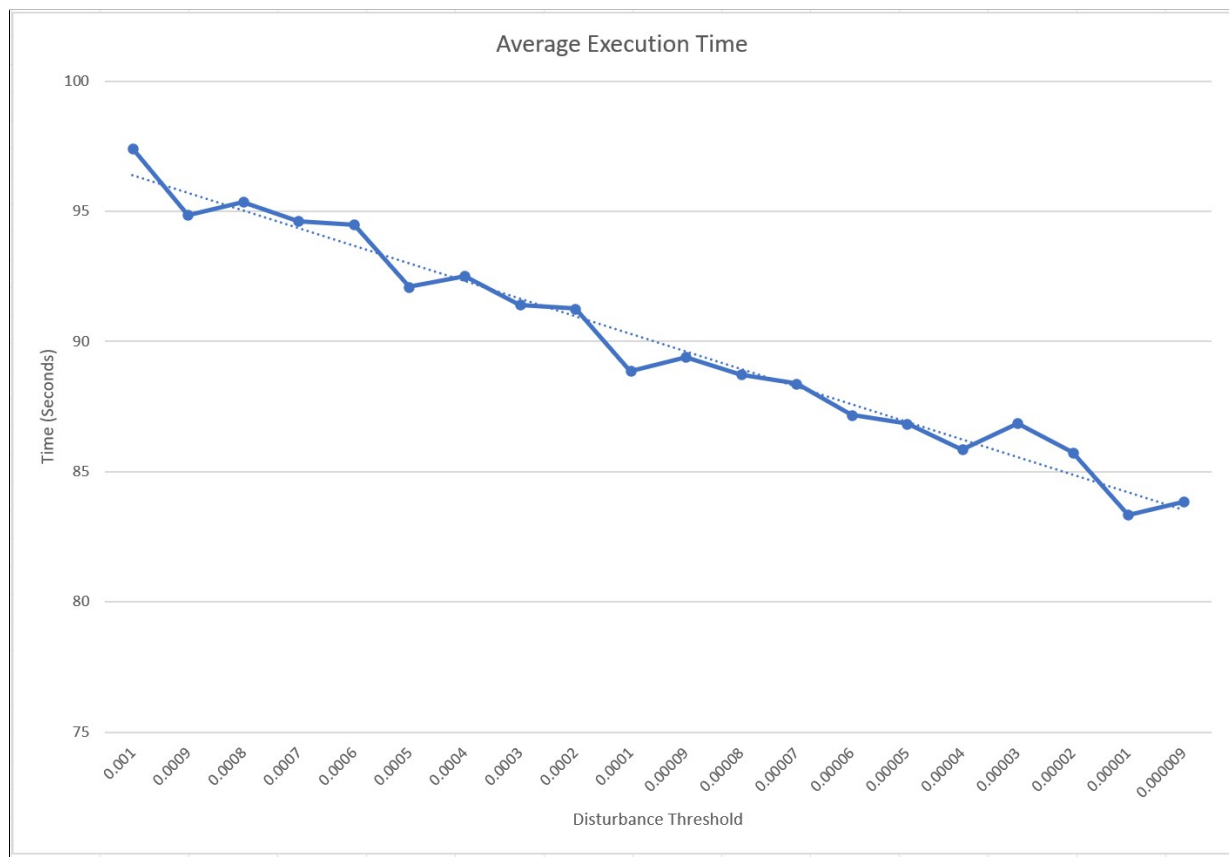


Figure 4.9: test 2 - graph 2

which shows the average execution time descending as the disturbance threshold is decreased. This finding can also be seen from the table (figure 4.7), as when the disturbance threshold value was set to the 0.001 the average execution time was 97.41974353 while when the disturbance threshold value was set to 0.000009 resulted in the average execution time of 83.85154053, which is an improvement of 14.9701%.

Overall, it was found that a lower disturbance threshold value will result in a higher fitness value, therefore meaning that using a higher disturbance threshold value will result in lower fitness values and is more ideal. Additionally, it also found that the disturbance threshold value is proportional to the execution time, as can be seen from the graph in figure 4.9, where a higher disturbance threshold value results in a larger execution time, and where a lower disturbance threshold value results in a lower execution time. Furthermore, it was also found that the disturbance threshold value set to 0.006 had resulted in the most optimal outputted average fitness value of 30252.8914 which compared to the initial average outputted fitness value (32893.899917039555) using the default parameter is an overall improvement of 8.36466%.

Test 3: Maximum iterations parameter testing

Max Iterations	Average Fitness	Average Time	Best Fitness	Best Time	Worst Fitness	Worst Time
1000	31086.15512	93.68292713	28981.44273	90.95649099	36502.20795	100.3326037
2000	31206.22025	185.274961	28777.48541	179.3734105	37308.928	197.0193849
3000	30451.34472	280.6383514	28588.06391	268.7360399	32163.16257	300.8372805
4000	30294.49491	378.3894986	27601.17377	357.0049391	32276.75481	403.4948997
5000	29986.65127	534.6214507	27601.17377	452.8570204	31916.66238	758.5849235
6000	30264.79734	667.0723463	27601.17377	555.6323242	32163.16257	809.9157968
7000	29948.42429	662.8948202	28082.02732	630.0619426	34033.56739	736.7736962
8000	30106.09981	759.9269507	27601.17377	723.4229751	32163.16257	810.7326446
9000	30331.36796	858.7724831	27601.17377	814.3834662	32845.38771	904.7043197
10000	29461.38045	1072.568167	27601.17377	918.7920256	31916.66238	1238.202251
11000	29976.80264	1152.149072	27601.17377	1044.86139	31802.03593	1305.848691
12000	30369.95012	1170.882744	28082.02732	1091.197828	32603.70464	1278.178952
13000	30343.22914	1290.403555	28042.21639	1155.26326	36723.89608	1474.001358
14000	30214.2151	1498.795588	27601.17377	1281.72275	34033.56739	1789.036789
15000	29856.1594	1399.179439	27601.17377	1342.560456	32371.03972	1444.307882
16000	30151.78951	1493.42377	27601.17377	1430.279725	32163.16257	1572.317393
17000	29909.51447	1742.472892	27601.17377	1525.290807	34033.56739	2339.801107
18000	29791.39479	1657.036781	27601.17377	1604.354434	31916.66238	1757.159363
19000	30083.82154	1939.285097	28588.06391	1706.547457	31916.66238	2231.656797
20000	30145.5856	1976.706365	27601.17377	1716.238638	32560.16284	2314.044121

Figure 4.10: test 3 - results table

For this test, the DFO algorithms maximum iterations parameter will be tested thoroughly. Similar to that of the previous tests the proposed DFO algorithm will be run a total of 400 different iterations in order to find the most optimal maximum iterations parameter value.

For this test, each of the 20 test iterations the DFO algorithm will use the swarm size parameter value of 2000 based on the findings from test 1. The disturbance threshold value will be set to 0.001 based on the findings from test 2. However, for the maximum iterations parameter, the value ranged from the default starting value of 1000 up to 20000 in increments of 1000 per each of the 20 iterations.

Based on the results, it was found that as the maximum iterations value was increased the lower the average fitness value will be. This finding can be seen from the graph in figure 4.11 which shows that as the maximum iterations value is gradually increased the average fitness value decreases. This finding can also be seen from the table in figure ??, as when the maximum iterations was set to 1000 the average outputted fitness value was equal to 31086.1551, but when the maximum iterations value was set to 20000 the average outputted fitness value was equal to 30145.5856, which is an overall improvement of 3.07216%. In addition, the linear trend-line fitted to the average fitness value shows that it is in a downwards trend which further indicates that a larger maximum iterations value will result in a lower fitness value.

Furthermore, it was also found that as the maximum iterations value is increased, the average required execution time is also increased which can be seen from the graph displayed in figure

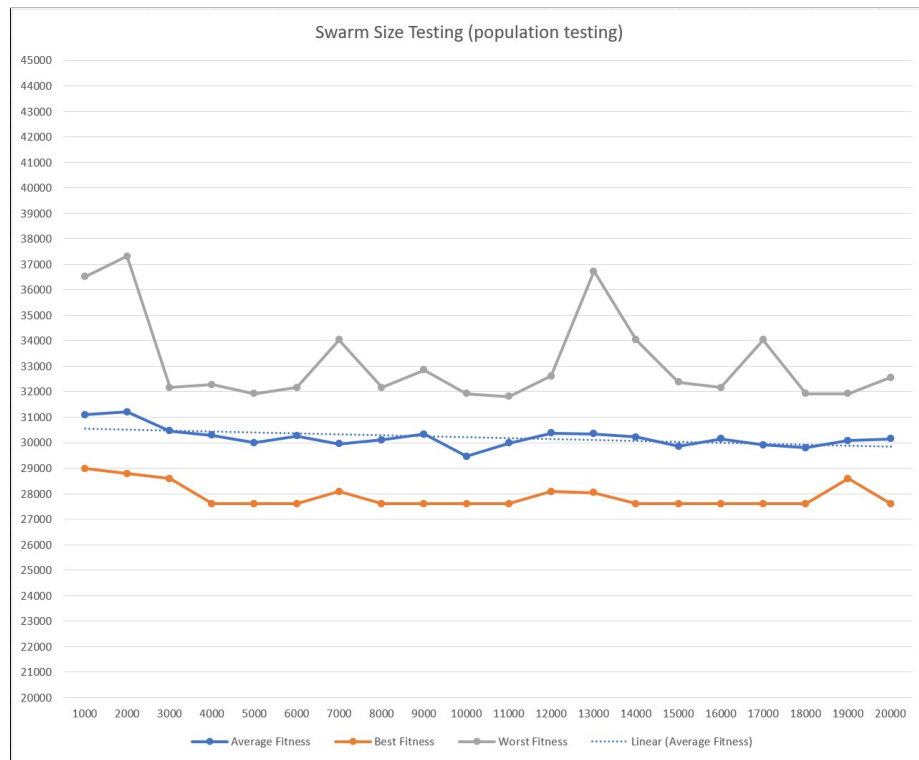


Figure 4.11: test 3 - graph 1



Figure 4.12: test 3 - graph 2

4.12 which shows that the average execution time is ascending as the maximum iterations value is increased. This finding can also be seen from the table displayed in figure 4.10, as when the maximum iterations value is set to 1000 the average required execution time was 93.68292713,

but when the maximum iterations value was set to 20000, the average required execution time was equal to 1976.706365, which is an overall increase of 2009.9963734662%.

Overall, it was found that a larger maximum iterations value will result in a lower fitness value. Moreover, it was also found that the maximum iterations value is proportional to the execution time, as can be seen from the graph in figure 4.12, where a low maximum iterations value had resulted in a low average execution time, and where a higher maximum iterations value had resulted in a larger average execution time. Furthermore, it was also found that the maximum iterations value set to 10000 had resulted in the best optimal solution with a fitness value of 29461.38045 which compared to the initial average outputted fitness value (32893.899917039555) using the default parameters is an overall improvement of 11.0096%.

Test Results

Based on the results it was found that as both parameters are increased in value, the average outputted fitness value decreases in value which can be seen from test 1, where it was found that when the swarm size was set to 100, the average fitness value was equal to 33274.01254 while when set to 2000, the average fitness value was equal to 30574.4948 which is an improvement of 8.45601%. Moreover, this finding can also be seen from test 3, where for the maximum iterations testing, it was found that when the maximum iterations was set to 1000, the average fitness value was equal to 31086.1551, while when was set to 20000, the average fitness value was equal to 30145.5856, which is an overall improvement of 3.07216%.

However, for the disturbance threshold parameter, based on test 2, it was found that decreasing the disturbance threshold value would result in a higher fitness value, thus making the default disturbance threshold value of 0.001 the most optimal choice to use. Overall, it was found that the most optimal parameter values to use include setting the swarm size to 2000, setting the disturbance threshold to 0.001, and the maximum iterations value to 20000, which had resulted in the average outputted solution fitness value of 30145.5856, which compared to the initial average outputted solution fitness value of 32893.899917039555 is an overall improvement of 8.71934%. However, despite this improvement of 8.71934%, the average outputted fitness value is still far from the known best fitness value of 27603 by 8.80571%, also the average execution time of 1976.706365 seconds using the found optimal swarm size and maximum iterations values is too large, which makes using the proposed DFO algorithm not feasible for any real use case scenarios due to its extremely long required execution times. Therefore, it is clear that further optimisation is required to further improve the average accuracy and execution time of the proposed algorithm.

4.2.3 Further Optimisation

In order to further optimise the average outputted fitness value and execution time of the DFO algorithm, a series of tests were devised based on the findings from the previous section (section 4.2.2) where it was found that although using a large swarm size and maximum iterations value leads to better outputted solution fitness values, they also contribute to the extremely high average

execution time, as both parameters are directly proportional to execution time, meaning that as both parameters are increased, the execution is also respectively increased. Based on this, 2 tests will be conducted, were for test 1 the swarm size will be increased, and the maximum iterations will be decreased, and for test 2 the swarm size will be decreased while the maximum iterations will be increased.

Test 1: Increasing swarm size decreasing maximum iterations

Test Number	Swarm Size	Maximum Iterations	Average Fitness Value	Average Execution Time
1	1000	1000	32163.38186	55.35136955
2	1000	900	31632.2813	47.52190578
3	1000	800	30383.03116	43.43708544
4	1000	700	31197.87239	37.95575099
5	1000	600	31145.50688	33.62177463
6	1000	500	31763.50333	27.12491076
7	1000	400	31612.36618	22.3253294
8	1000	300	32191.23264	16.22065415
9	1000	200	32879.31055	11.17176752
10	1000	100	34666.98464	5.826612544
11	2000	1000	31698.89565	110.3085511
12	2000	900	31481.88089	101.3078228
13	2000	800	30852.31339	84.07133973
14	2000	700	30697.99059	72.52885261
15	2000	600	30602.60476	62.7308018
16	2000	500	30436.72751	53.13477039
17	2000	400	31827.31645	47.0413739
18	2000	300	32014.75454	43.43269928
19	2000	200	33461.75163	28.93981924
20	2000	100	33329.28249	14.74675136
21	3000	1000	30957.10257	191.7707185
22	3000	900	30745.16399	143.6207235
23	3000	800	31088.56695	126.5593057
24	3000	700	30765.91219	111.3787919
25	3000	600	31189.32222	95.93971355
26	3000	500	31236.93672	79.63744111
27	3000	400	31399.54571	61.95100694
28	3000	300	32585.78222	46.90796766
29	3000	200	31872.2617	31.18212497
30	3000	100	33058.89771	16.24005883
31	4000	1000	31062.37413	204.0618907
32	4000	900	30841.5708	186.6781958
33	4000	800	30917.38591	164.8380808
34	4000	700	30547.64191	146.5266905
35	4000	600	31118.26109	125.0632964
36	4000	500	31378.23151	104.5946606
37	4000	400	31837.39969	86.59544866
38	4000	300	31231.32531	66.39445703
39	4000	200	31214.68593	45.13158374
40	4000	100	33170.0102	23.72537982
41	5000	1000	30475.85491	274.8282812
42	5000	900	30123.15474	289.0387485
43	5000	800	30610.96941	260.3992143
44	5000	700	31199.1783	220.7091164
45	5000	600	30748.32228	187.1337951
46	5000	500	31056.17097	154.0307209
47	5000	400	31975.94148	111.485317
48	5000	300	32205.3945	98.20813334
49	5000	200	31874.77912	62.21640797
50	5000	100	33439.00618	28.34818664

Test Number	Swarm Size	Maximum Iterations	Average Fitness Value	Average Execution Time
51	6000	1000	29512.75278	323.1124918
52	6000	900	30847.49216	279.5818356
53	6000	800	30655.18315	246.0986418
54	6000	700	31351.98368	216.3963015
55	6000	600	31086.43716	185.6651776
56	6000	500	30840.71339	156.3871314
57	6000	400	30449.24684	125.2914259
58	6000	300	32345.5975	94.32680049
59	6000	200	32307.22849	63.74256046
60	6000	100	33164.50319	32.11213398
61	7000	1000	30601.87818	356.1215361
62	7000	900	30811.68376	320.7169449
63	7000	800	30803.20393	286.5915481
64	7000	700	30150.73497	248.0196246
65	7000	600	30663.19409	214.4760599
66	7000	500	31242.74712	179.5967528
67	7000	400	30550.97436	142.775193
68	7000	300	31208.05112	107.0789332
69	7000	200	31139.20326	72.1932601
70	7000	100	33688.36924	36.52552862
71	8000	1000	30656.79813	401.1080007
72	8000	900	30358.47746	364.4333588
73	8000	800	31136.15208	331.9446016
74	8000	700	30510.66367	313.9837701
75	8000	600	29732.10974	261.2867139
76	8000	500	30554.82637	214.1631639
77	8000	400	30887.03372	173.8062151
78	8000	300	31258.82599	132.6376555
79	8000	200	32667.79976	89.57500131
80	8000	100	33985.208	45.50726902
81	9000	1000	9550.997149	482.9103493
82	9000	900	30411.22081	403.448851
83	9000	800	30062.0268	360.9861038
84	9000	700	30053.88417	318.3928789
85	9000	600	31324.91357	271.5951857
86	9000	500	31959.45128	222.6506096
87	9000	400	31696.01747	182.5460656
88	9000	300	31162.11834	137.1183408
89	9000	200	31133.54105	92.4736326
90	9000	100	33260.16975	47.03939695
91	10000	1000	30946.77349	493.3381412
92	10000	900	29725.24271	444.0353668
93	10000	800	30366.28621	395.1451208
94	10000	700	31413.37167	346.777973
95	10000	600	30564.28506	295.8589265
96	10000	500	31227.41016	245.2641536
97	10000	400	30905.75243	198.2628594
98	10000	300	30908.4273	150.2171727
99	10000	200	31498.00161	104.2389055
100	10000	100	33169.15272	54.69607718

Figure 4.13: test 1 - results table

As seen from the results depicted in the table (figure 4.13), for this first test a total of 100 different sub-test iterations were run, where with each of the test iterations 10 different DFO executions were executed, resulting in a total of 1000 different DFO executions being run. Furthermore, every 10 test iterations the swarm size parameter value was increased in increments of 1000, ranging from 1000 up to 10000, resulting in 10 different swarm sizes being thoroughly tested. Additionally, with each of the 10 different swarm size values being tested, the maximum iterations value was also decreased in decrements of 100, ranging from 1000 down to 100 and the

outputted average solution fitness values and average execution times were recorded and stored in the table (figure 4.13).

Based on the results, it was ultimately found that using the swarm size parameter value of 8000 in conjunction with the maximum iterations parameter value of 1000 had resulted in the overall best average solution fitness value of 29512.75278 with an average required execution time of 323.1124918 seconds, as can be seen from test number 51 in the displayed table (figure 4.13).

Furthermore, the outputted average fitness value of 29512.75278 using the given parameters was found to be 3.33652% better than that of the previously found best average fitness value of 30514.1571 using the optimal parameters from the previous section (section 4.2.2), additionally, it was also found to be within a 6.9186421041191% accuracy range of the known best solutions fitness value for the 'wi29' dataset which is 27603.

Moreover, the average execution time of 323.1124918 seconds using the given found optimal parameters from this test was also found to be an 83.653996490268% improvement compared to the average execution time of 1976.706365 seconds using the previously found optimal parameters from section 4.2.2.

Test 2: Decreasing swarm size increasing maximum iterations

Test 2 was carried out in a similar manner to that of test 1 where 100 different test iterations were run with a further 10 DFO executions per each test iteration, however for this test, instead of the swarm size being increased and the maximum iteration value being decreased with each test iterations like in test 1, instead the swarm size was decreased and the maximum iterations value was decreased.

After this test was successfully completed and analysed it was found that using the swarm size parameter value of 800 in conjunction with the maximum iterations value of 9000 had resulted in the best overall results from conducting this test, as the average outputted solution fitness value using these given parameters was equal to 29383.35891 with an average execution time of 381.8523666 seconds, as can be seen from test number 83 displayed in the table (figure 4.14).

Furthermore, the outputted average fitness value of 29383.35891 using the given parameters found in this test was found to be 3.8484306490064% better than that of the previously found best average fitness value of 30514.1571 using the optimal parameters from the previous section (section 4.2.2), additionally, it is also found to be within a 6.449874687534% accuracy range of the best known solutions fitness value for the 'wi29' dataset which is 27603.

Moreover, the average execution time of 381.8523666 seconds using the given optimal parameters found from conducting this test was also found to be an 80.682393027049% improvement compared to the average execution time found using the previously found optimal parameters from section 4.2.2, which was equal to 1976.706365 seconds.

Test Number	Swarm Size	Maximum Iterations	Average Fitness Value	Average Execution Time
1	1000	1000	30918.88383	54.4414644
2	900	1000	31369.76944	46.96257989
3	800	1000	30801.27017	42.91886873
4	700	1000	32371.87503	37.85752878
5	600	1000	31996.75599	32.51322472
6	500	1000	31435.53935	27.4970948
7	400	1000	32652.05122	21.28889611
8	300	1000	32639.10593	16.45770745
9	200	1000	32481.0327	10.40099373
10	100	1000	33174.17236	5.225777054
11	1000	2000	30213.84286	106.5184967
12	900	2000	31112.31993	96.24948344
13	800	2000	32313.35556	84.12825255
14	700	2000	30690.01892	71.14754143
15	600	2000	30522.23021	60.8365752
16	500	2000	31180.14828	50.3511431
17	400	2000	30642.18404	41.48354549
18	300	2000	31614.43724	34.35052545
19	200	2000	31489.89366	27.16836812
20	100	2000	31527.85297	13.24184496
21	1000	3000	30340.23453	199.4460746
22	900	3000	30080.52095	143.1831062
23	800	3000	30551.45493	124.0171687
24	700	3000	31122.64248	108.897027
25	600	3000	31309.41989	93.05736425
26	500	3000	31094.02641	77.00224724
27	400	3000	30388.04985	61.62908382
28	300	3000	31160.17775	45.73845453
29	200	3000	30537.79791	30.05287037
30	100	3000	31133.66332	14.76132541
31	1000	4000	30614.52934	200.3354977
32	900	4000	30823.60604	182.7900244
33	800	4000	29871.53816	162.2855006
34	700	4000	31382.72488	141.124288
35	600	4000	32079.18837	121.1602353
36	500	4000	30445.72676	100.5400979
37	400	4000	31251.8198	81.95751979
38	300	4000	30860.3488	61.45486743
39	200	4000	30744.30112	41.27663307
40	100	4000	31675.57213	19.85049968
41	1000	5000	30494.25229	264.2578883
42	900	5000	31677.89083	259.1981829
43	800	5000	31029.02635	251.231092
44	700	5000	30318.5438	225.3552482
45	600	5000	31381.13012	181.1024755
46	500	5000	30597.50966	151.6020304
47	400	5000	31052.18071	120.5057214
48	300	5000	29892.40583	90.6649303
49	200	5000	31493.11829	52.20325952
50	100	5000	31751.41277	24.62494757

Test Number	Swarm Size	Maximum Iterations	Average Fitness Value	Average Execution Time
51	1000	6000	30306.36544	339.3217512
52	900	6000	30538.85192	273.9099069
53	800	6000	30704.70571	242.4141624
54	700	6000	30242.59957	209.0639986
55	600	6000	31324.12655	179.6722273
56	500	6000	30422.4002	149.5091824
57	400	6000	30790.68896	119.6494062
58	300	6000	30333.3294	90.12530386
59	200	6000	31247.32888	59.86888771
60	100	6000	30579.82334	29.35881402
61	1000	7000	32197.91168	352.2329023
62	900	7000	31189.42649	316.4003872
63	800	7000	30327.84268	275.7049044
64	700	7000	30421.12721	251.3409228
65	600	7000	30587.69593	207.5168342
66	500	7000	29937.03437	172.5933329
67	400	7000	30889.9602	139.3007681
68	300	7000	30928.5421	102.7535981
69	200	7000	29894.57418	68.96951764
70	100	7000	30411.48852	33.61680593
71	1000	8000	31336.18159	391.8727569
72	900	8000	31048.04349	350.359569
73	800	8000	29887.59749	312.8336468
74	700	8000	30772.04681	271.3378104
75	600	8000	30189.7531	247.0195434
76	500	8000	30850.10257	214.8431702
77	400	8000	30546.11234	171.8646167
78	300	8000	31171.13639	126.6649971
79	200	8000	30687.00762	81.75304255
80	100	8000	31094.40289	40.02387674
81	1000	9000	31242.67799	466.8637107
82	900	9000	30352.27202	439.9579832
83	800	9000	29383.35791	381.8523666
84	700	9000	30658.04786	349.2912344
85	600	9000	29490.19891	268.1165567
86	500	9000	30274.72235	220.0552554
87	400	9000	30965.80248	173.740491
88	300	9000	30406.50047	132.0459837
89	200	9000	31021.56682	87.11319675
90	100	9000	30994.54461	42.98658407
91	1000	10000	30190.90418	491.3969046
92	900	10000	29895.98844	442.0694411
93	800	10000	30147.49304	390.0547234
94	700	10000	29440.92745	338.5390591
95	600	10000	29993.24124	296.837623
96	500	10000	29674.53723	244.3233925
97	400	10000	30878.77487	192.1041779
98	300	10000	30138.35708	146.2053881
99	200	10000	31624.12202	96.17506664
100	100	10000	30752.34044	47.8060698

Figure 4.14: test 2 - results table

Test Results

Based on the test results it was found that the most optimal parameters in test 1 were setting the swarm size parameter to 6000 and the maximum iterations parameter to 10000, as using these parameters had resulted in the average solution fitness value of 29512.75278 with an average execution time of 323.1124918 seconds. Moreover, it was also found that the most optimal parameters for test 2 were using the swarm size parameter value of 800 and the maximum iterations value of 9000, which had resulted in the average solution fitness value of 29383.357791 with an average execution time 381.85236 seconds.

Moreover, it was found that test 2 had resulted in the best overall average solution fitness value of 29383.35891 with an average execution time of 381.8523666 seconds, which is an 0.44036446070147% improvement compared to the best average fitness value of 29512.75278 found in test 1. Furthermore, it should be noted that this outcome was to be expected, because as the number of maximum iterations grows it would mean that the update procedure of the algo-

rithm will be called and executed more often, where with each update procedure call, the given swarm will converge towards an optimal solution, unlike in test 1 where the maximum iterations value is lower meaning that the update procedure will be called fewer times, which will ultimately mean that the swarm will not converge towards an optimal solution as much as to that in test 2.

However, it should be noted that despite the fact that test 2 had resulted in the most optimal average solution fitness value of 29383.35891 with an average execution time of 381.8523666 seconds, it was found that the optimal results from test 1 had resulted in a far better execution time of 323.1124918 seconds, which is an overall reduction of 15.382874208241% compared to the average execution time of test 2.

Overall, it was found that test 2 had ultimately resulted in the best overall average fitness value of 29383.35891, thus making the swarm size value of 800 paired with the maximum iterations value of 9000 the most optimal parameters to use for the proposed DFO algorithm, as using these parameters had resulted in successfully managing to decrease the average fitness value down from 30145.5856 to 29383.35891, and had also successfully resulted in managing to decrease the average execution time down from 1976.706365 seconds to 381.8523666 seconds.

4.2.4 Resolving the local optimum trap

Although using the optimal parameters, the proposed algorithm's average outputted fitness value and execution time have significantly improved, during observations taken during multiple executions of the proposed algorithm on the 'wi29' TSP dataset, it was observed that the DFO algorithm suffers from the local optimum trap, where the swarm is trapped to a single found optimal solution and is unable to converge towards a better solution, as can be seen from the example in figure 4.15.

As seen from figure 4.15, the algorithms swarm of flies had fixated on a single solution with the fitness value of 28777.485412655158 (local optimum trap). The fixated solution fitness value of 28777.485412655158 was originally found by the swarm during iteration 700, meaning that the algorithm was unable to converge towards a better solution fitness value for the remainder of the 24300 iterations, this is however not the known best fitness value of 27603 and therefore ultimately proves that the DFO algorithm suffers from the local optimum trap. In order to try to resolve the issue of the local optimum trap, 2 different approaches were devised, the first approach being the implementation of an oscillating disturbance threshold, and the second approach being the implementation of multiple swarms.

Test 1: Oscillating disturbance threshold approach

For this first test, an oscillating disturbance threshold approach had been implemented which automatically oscillates the disturbance threshold value every 100 iterations from 0.001 to 0.0006, with these values being based on section 4.2.2. The approach being implemented is intended to

```

Iteration 20600, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 20700, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 20800, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 20900, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21000, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21100, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21200, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21300, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21400, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21500, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21600, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21700, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21800, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 21900, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22000, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22100, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22200, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22300, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22400, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22500, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22600, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22700, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22800, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 22900, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23000, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23100, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23200, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23300, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23400, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23500, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23600, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23700, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23800, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 23900, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24000, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24100, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24200, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24300, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24400, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24500, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24600, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24700, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24800, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009
Iteration 24900, Best fly index: 0, Fitness Value: 28777.485412655158, Delta Value: 0.0009

Final best fitness: 28777.485412655158
Final best fly: [23, 26, 24, 19, 25, 27, 28, 20, 22, 21, 18, 14, 17, 16, 13, 12, 11, 10, 9, 5, 1, 0, 4, 7, 3, 2, 6, 8, 15]
--- 731.6373963356018 seconds ---

```

Figure 4.15: Local optimum trap - Example

cause more disturbance every 100 iterations within the algorithms swarm which would allow for the swarm to explore for a more better possible optimal solutions, and then for the following 100 iterations, the swarm then would converge towards a possibly better solution in the given search space. The DFO algorithm was then run on the ‘wi29’ dataset 20 times using the oscillating disturbance threshold approach and the results are presented in figure 4.16.

Based on the results, it was that found that the average fitness value using this approach is equal to 30616.54559 which is within 10.3524% accuracy range of the best-known fitness value of 27603, and the best overall fitness value outputted was 28042.21639 which is within 1.57863% accuracy range of the best-known fitness value. Moreover, the average outputted fitness value of 30616.54559 from using this approach was found to be an increase 4.11063% compared to the previous best average outputted fitness value of 29383.35891 using the optimal parameters found in section 4.2.3.

Execution	Fitness Value	Execution Time
1	31458.40189	346.6642454
2	30445.97623	361.5675976
3	31782.50387	369.0296137
4	29965.18173	343.7605917
5	31380.56546	381.2952576
6	30927.24114	360.0770178
7	29965.18173	360.6601486
8	28777.48541	359.1128011
9	28042.21639	358.8597426
10	31683.75093	346.8017032
11	31227.47624	359.0507247
12	31463.77431	354.8705091
13	29931.69751	369.0682385
14	30203.04122	350.8095164
15	31802.03593	351.5450983
16	30632.08118	364.1099246
17	29216.45326	378.069066
18	33832.90874	390.274812
19	31458.40189	356.9913225
20	31782.50387	351.8967083

Figure 4.16: Approach 1 - Oscillating Disturbance Threshold Approach results

Test 2: Multiple swarms approach

For this second test, a multiple swarms approach had been implemented which simply uses multiple swarms of flies to solve the same given TSP where the swarm with the best overall solution based on its fitness value will be returned. More specifically, 20 different instances of the DFO algorithm were run with 5 swarms running per instance on the ‘wi29’ dataset. The results from the 20 DFO executions using the multiple swarms approach is presented are the following figure (figure 4.17).

Based on the test results, it was found that the average outputted solution fitness value using this approach is equal to 29065.7257 which is a 5.16237% difference from the known best solutions fitness value 27603, and the best overall fitness value outputted was 28082.02732 which has a difference of 1.72049% from the best-known solutions fitness value. Moreover, the average outputted fitness value of 29065.7257 from using this multiple swarms approach was found to

Execution	Best Fitness Value	Average Execution Time
1	28588.06391	341.9233906
2	30446.03527	330.769423
3	28588.06391	331.581111
4	28670.85388	340.8111327
5	29357.13782	332.8929002
6	29216.45326	336.4717054
7	28082.02732	334.0381582
8	28588.06391	334.6713004
9	29951.30187	336.1466336
10	28470.10134	345.33653
11	28229.56312	342.7532573
12	28588.06391	336.8985929
13	28588.06391	336.1796408
14	28082.02732	344.2554567
15	31202.89739	338.7518408
16	28588.06391	391.5527804
17	29258.33896	359.7719803
18	30534.02354	331.5816047
19	29216.45326	335.6425202
20	29068.91746	338.7572205

Figure 4.17: Approach 2 - Multiple Swarms Approach results

be 1.08687% better than the previous best average outputted fitness value of 29383.35891 using the optimal parameters found in section 4.2.3.

Test Results

Overall, it was ultimately found that the second approach of using multiple swarms was the most superior approach to resolving the local optimum trap, as the oscillating disturbance threshold was not successful in resolving the local optimum trap and had resulted in a higher average outputted solution fitness value compared to that found in section 4.2.3. Furthermore, it was also found that, not only did the multiple swarms approach successfully resolve the local optimum trap but also further improved the average outputted fitness value (of the 'wi29' dataset), where the outputted solution resulted in having a 1.08687% improvement compared to the previous outputted fitness value of 29383.35891 using the optimal parameters found in section 4.2.3.

4.2.5 Testing the DFO on TSP datasets

For this experiment, the proposed DFO algorithm using the optimal found parameter values and also the multiple swarms approach will be tested in a series of different TSP datasets ranging in size, from 29 up to 194 nodes.

Datasets

The datasets used for this are from the NTSP repository consisting of 25 different TSP instances based on real cities from various regions of the world, and also the MP-TESTDATA repository which is a library of various TSP datasets also consisting of real cities from various regions of the world.

The datasets being used for this experiment include the following:

- wi29 - consisting of 29 nodes, with an optimal solution fitness value of 27603.
- dj38 - consisting of 38 nodes, with an optimal solution fitness value of 6656.
- berlin52 - consisting of 52 nodes, with an optimal solution fitness value of 7544.365901904087.
- eil76 - consisting of 76 nodes, with an optimal solution fitness value of 538.
- eil101 - consisting of 101 nodes, with an optimal solution fitness value of 629.
- qa194 - consisting of 194 nodes, with an optimal solution fitness value of 9352.

Running the DFO algorithm on the datasets

DFO				
Dataset:	Fitness value:	Execution time (Seconds):	Known optimal fitness value:	Difference from know solution: (%)
wi29	28777.48541	374.0188942	27603	4.08%
dj38	6810.967445	502.6467488	6656	2.28%
berlin52	11717.47927	1214.864012	7544	35.62%
eil76	1076.253703	3319.498206	538	50.01%
eil101	1625.570997	4963.637083	629	61.31%
qa194	21514.73872	11055.66686	9352	56.53%

Figure 4.18: Running DFO algorithm on datasets

After successfully running the DFO algorithm using the multiple swarms approach on the 6 TSP datasets mentioned in the previous subsection (4.2.5), the outputted results based on the algorithm's final best outputted fitness value, execution time, and percentage difference from each of the datasets best known solutions fitness value are displayed in the table depicted in figure 4.18.

Based on the findings from this experiment, it was found that the proposed DFO algorithm works very well with small-scale TSPs ranging up to 40 nodes, as can be seen from the table (figure 4.18),

where for 'wi29' dataset the DFO algorithm had managed successfully output a solution with a fitness value of 28777.48541 which is a difference of 1174.48541 from the known best solutions fitness value of 27603. Additionally, the outputted fitness value of 28777.48541 is also within 4.08% accuracy range of the know optimal solutions fitness value.

Furthermore, this finding can also be seen from the table (figure 4.18), where for the 'dj38' dataset, the DFO algorithm had successfully output a solution with a fitness value of 6810.967445, which is a difference of 154.967445 from the known best solutions fitness value of 6656. Additionally, the outputted fitness value of 6810.96744 is also within 2.28% accuracy range of the know optimal solutions fitness value.

However, despite the fact the DFO algorithm can successfully solve and optimise any given small-scale TSP to a very high degree of accuracy, the algorithm is unfortunately not able to do the same for any given large-scale TSP as can be seen from the table (figure 4.18) were for the 'berlin52' dataset, the algorithm had managed to output a solution with a fitness value of 11717.47927 which is a difference of 4173.47927 from the datasets known best solutions fitness value of 7544, which is an overall difference of 35.62% from the optimal solution. Moreover, this finding can also be seen from where the DFO algorithm was used on the 'qa194' dataset, had resulted in an outputted solution with the fitness value of 21514.73872 which is an overall difference of 56.53% from the datasets known optimal solutions fitness value of 9352.

Moreover, it was also found that there is a correlation between the execution time of the DFO algorithm and the number of nodes in the given TSP were as the number nodes increases, the execution time of the DFO algorithm also increases, as can be seen from the graph in figure 4.19, were when the number of nodes was equal to 29, the execution time was equal to 374.0188942 seconds, while when the number of nodes was equal to 194, the execution time was equal to 11055.66686 seconds.

4.2.6 Comparing the DFO against other algorithms

For this experiment, the PSO, ACO, and GA algorithms were used to solve the same 6 datasets mentioned in section 4.2.5, the results from the aforementioned algorithms were then compared against the results from using the DFO algorithm on the same 6 datasets. The results from conducting this experiment are displayed in the tables in figure 4.20.

Based on the results, it was found that the DFO algorithm was able to significantly outperform the PSO and ACO algorithm on all 6 TSP datasets as can be seen from the graph in figure 4.21 were the DFO algorithm had resulted in outputting solutions with fitness values that are closest to the known optimal solutions fitness value for each of the 6 TSP datasets.

An example of this finding can be seen from when each of the 3 aforementioned algorithms

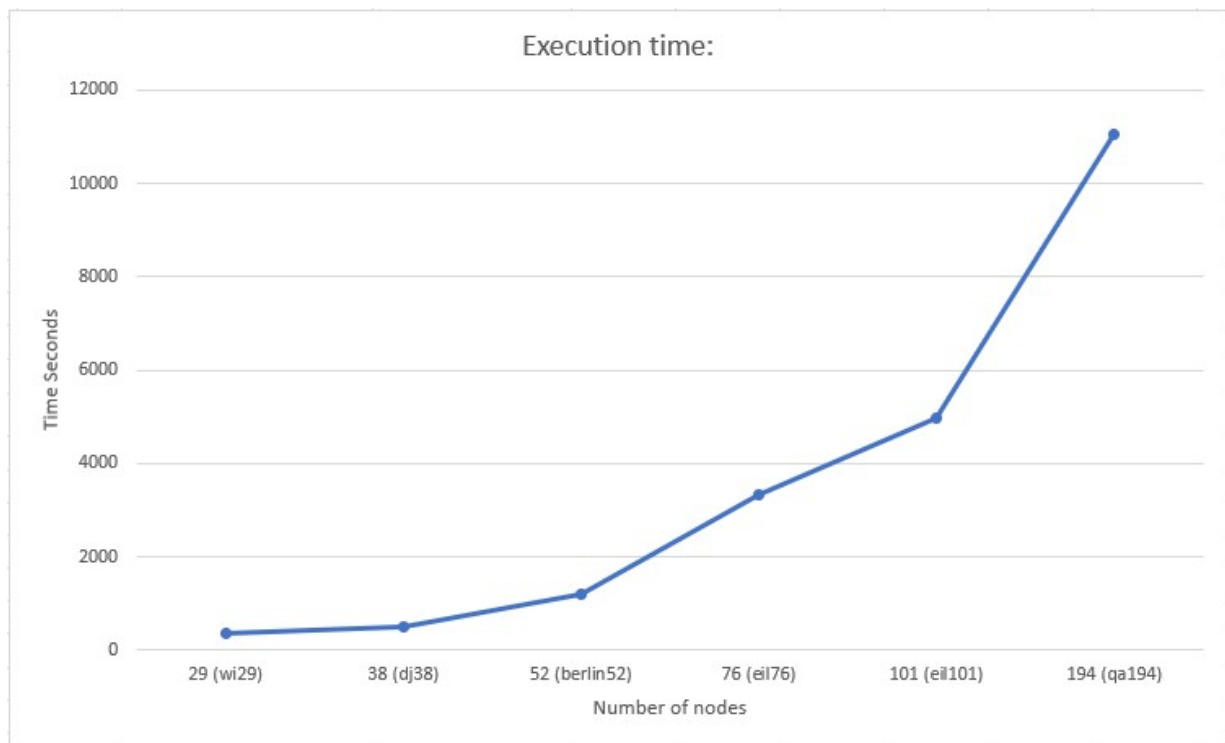


Figure 4.19: Execution time based on number of nodes

were used on the 'qa194' TSP dataset which is the largest dataset based on the number of nodes that was used during this experiment, were the DFO algorithm had successfully outputted a final best solution with a fitness value of 21514.73872, which is a far more accurate result compared to the solutions outputted by the PSO and ACO algorithms. The PSO algorithm had outputted a final best solution with a fitness value of 54092.78232 which is an overall difference of 151.42198110784% compared to the solution outputted by the DFO algorithm, while the ACO algorithm had outputted a final best solution with a fitness value of 36455.57302, which although is better than the PSO algorithms solution by 48.38000842923%, it is still outperformed by the DFO algorithm by 69.444646734711%.

Additionally, this finding can also be seen when the aforementioned algorithms were used on the 'berlin52' dataset, which based on the findings from the previous section (section 4.2.5), was the dataset used when the proposed DFO algorithm had reached its limitation based on the number of nodes the given dataset had consisted of, as it was found that the DFO algorithm was unable to solve the 'berlin52' TSP dataset to a high degree of accuracy, as the outputted solution from the DFO algorithm had an overall difference of 35.62% from the known optimal solution.

However, despite the DFO algorithm successfully outperforming the PSO and ACO algorithms in terms of outputted solution fitness values, it was found that the DFO had resulted in having the highest execution times compared to the PSO and ACO algorithms as can be seen from the

DFO				
Dataset:	Fitness value:	Execution time (Seconds):	Known optimal fitness value:	Difference from know solution: (%)
wi29	28777.48541	374.0188942	27603	4.08%
dj38	6810.967445	502.6467488	6656	2.28%
berlin52	11717.47927	1214.864012	7544	35.62%
eil76	1076.253703	3319.498206	538	50.01%
eil101	1625.570997	4963.637083	629	61.31%
qa194	21514.73872	11055.66686	9352	56.53%

PSO				
Dataset:	Fitness value:	Execution time (Seconds):	Known optimal fitness value:	Difference from know solution: (%)
wi29	37792.84551	281.7261939	27603	26.96%
dj38	9887.528074	411.3292649	6656	32.68%
berlin52	13534.54726	651.0240855	7544	44.26%
eil76	1416.272665	1182.044517	538	62.01%
eil101	1901.44659	1944.259791	629	66.92%
qa194	54092.78232	5776.103014	9352	82.71%

ACO				
Dataset:	Fitness value:	Execution time (Seconds):	Known optimal fitness value:	Difference from know solution: (%)
wi29	35866.38523	3.971890211	27603	23.04%
dj38	13300.539	6.534465313	6656	49.96%
berlin52	13929.68611	12.11471701	7544	45.84%
eil76	1390.599148	25.53972864	538	61.31%
eil101	2169.715475	47.19658566	629	71.01%
qa194	36455.57302	170.9403427	9352	74.35%

GA				
Dataset:	Fitness value:	Execution time (Seconds):	Known optimal fitness value:	Difference from know solution: (%)
wi29	29093.30678	166.9364448	27603	5.12%
dj38	6657.684628	227.4430163	6656	0.03%
berlin52	7544.365902	185.8916969	7544	0.00%
eil76	566.0104659	471.2056956	538	4.95%
eil101	703.1931811	680.5796595	629	10.55%
qa194	10469.01062	1902.33509	9352	10.67%

Figure 4.20: Comparing the PSO, ACO, and GA algorithms against the DFO algorithm

tables in 4.20. For example, when using the 'qa194' dataset, the DFO algorithm had an elapsed execution time of 11055.66686 seconds, while the PSO and ACO had elapsed execution times of 170.9403427 seconds and 1902.33509 seconds, which outperform the DFO algorithm, by 6367.5586145294% and 1012.8649094489% respectively.

Moreover, it was found that the DFO algorithm although reaching its limitation had resulted in successfully outputting a solution with a fitness value of 11717.47927 for the 'berlin52' dataset, which significantly outperforms the results outputted by PSO and ACO algorithms. Were, the PSO algorithm had outputted a solution with a fitness value of 13534.54726 for the given dataset which is outperformed by the DFO algorithm by 15.507328394872%, while the ACO algorithm had outputted a solution with a fitness value of 13929.68611 for the given dataset which is outperformed by the DFO algorithm by 18.879545583357%.

Furthermore, despite the findings that the DFO algorithm was able to noticeably outperform

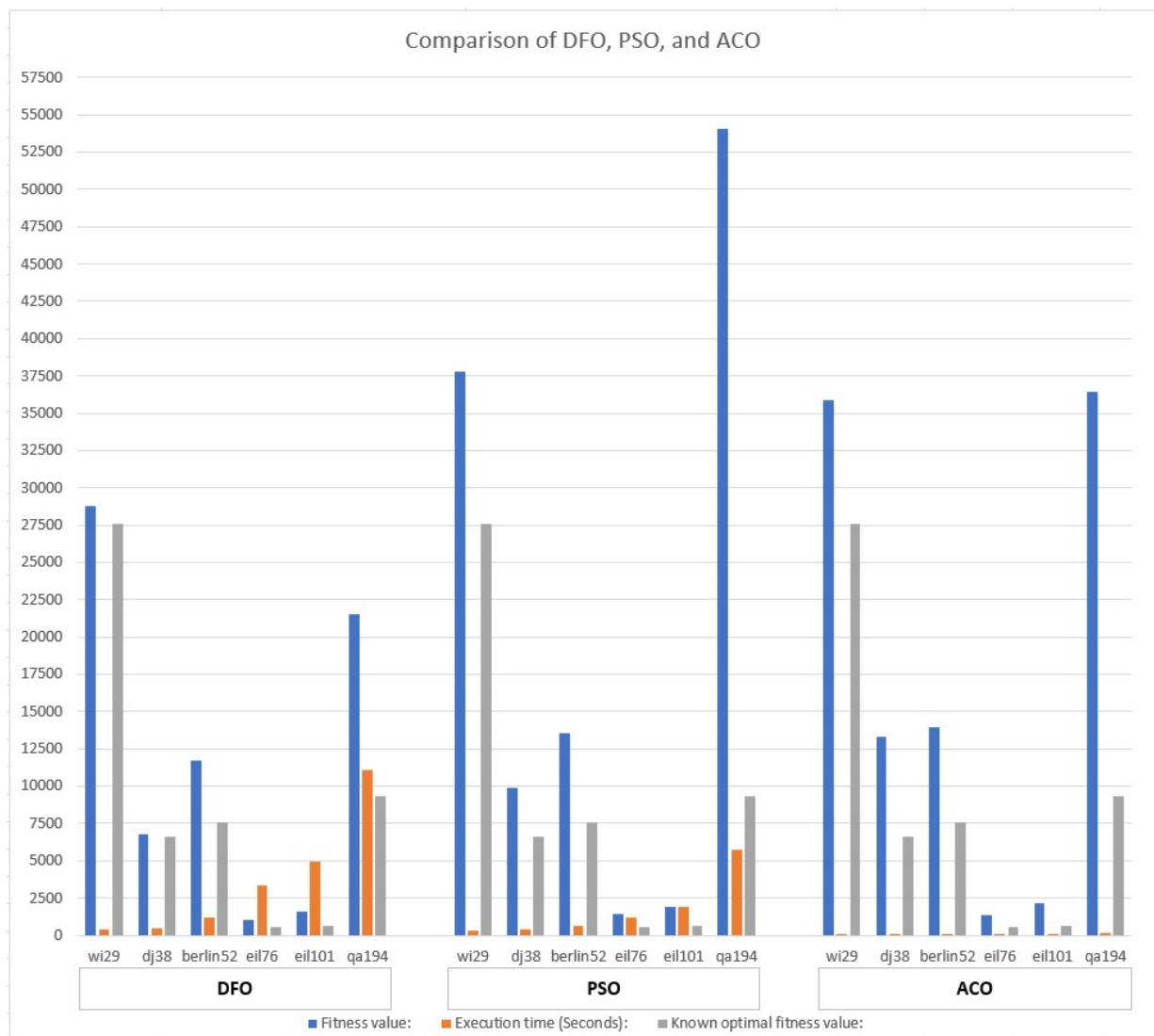


Figure 4.21: Comparing the DFO, PSO, and ACO algorithms

the PSO and ACO algorithms on all 6 TSP datasets used (section 4.2.5), the DFO algorithm and also the PSO and ACO algorithms were significantly outperformed by the GA on 5 out of the total 6 TSP datasets used, which can be seen from the graph in figure 4.22.

As seen from the graph (figure 4.22), the only TSP dataset in which the GA algorithm was unable to outperform the DFO algorithm was the 'wi29' dataset, although it should be noted that the DFO algorithm had only outperformed the GA algorithm by a very small and insignificant figure, as the DFO algorithm had outputted a solution with a fitness value of 28777.48541, while the GA had outputted a solution with a fitness value of 29093.30678, which is an overall difference of 1.0974599257038%.

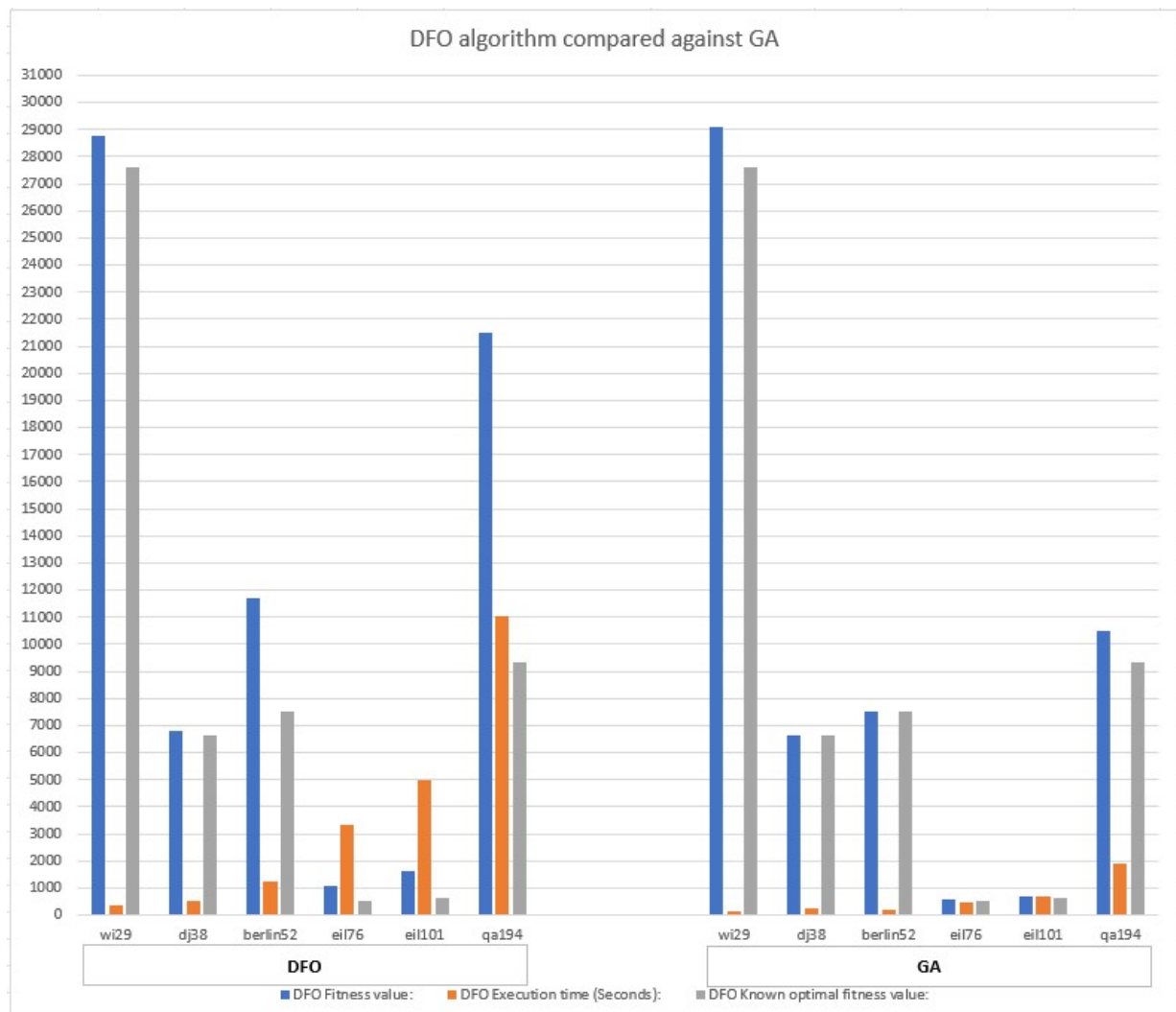


Figure 4.22: Comparing the DFO against the GA

However, for the remainder 5 datasets, the DFO algorithm was unable to compete against the GA, which had completely outperformed the DFO algorithm as can be seen from the graph (figure 4.22) were the 2 algorithms were used to solve the 'berlin52' TSP dataset, which is the dataset where the DFO algorithm had reached its limitation. Using this dataset, the DFO algorithm had outputted a solution with a fitness value of 11717.47927, while the GA had successfully outputted the known optimal solution with a fitness value of 7544.365902 which outperforms the DFO algorithm by 55.314302384163%.

Moreover, this finding can also be seen when the DFO and GA algorithms were used on the 'qa194' TSP dataset, where the DFO algorithm had outputted a solution with a fitness value of 21514.73872, while the GA had successfully outputted a solution with a fitness value of 10469.01062 which outperforms the DFO algorithm by 105.50880595057% and is also within

an accuracy range of 10.67% of the known optimal solution.

In addition, it was also found that not only had the GA outperformed the DFO and PSO algorithms in regards to outputted solution fitness values, but also elapsed execution times as can be seen from figures 4.20 and 4.21, were for the 'qa194' dataset was used, the DFO algorithm had an elapsed execution time 11055.66686, while the PSO and GA algorithm had elapsed execution times of 5776.103014 seconds and 1902.33509 seconds, further showing that the GA outperforms the DFO and ACO algorithms in regards to execution time by 481.16295694256% and 91.403561072292%.

Overall, it was found that the DFO algorithm is able to noticeably outperform the PSO and ACO algorithms when used on small-scale TSPs ranging up to a maximum of 40 nodes, as can be seen from figure 4.20 were when the DFO algorithm was applied to the 'dj38' dataset, the algorithm was able to successfully solve the TSP to an accuracy range of 2.28%, while when the proposed algorithm was applied to the 'berlin52' dataset the outputted solution was within a accuracy range of 35.62%, which is not to the high degree of accuracy that was achieved using the previous dataset, therefore indicating that the DFO algorithm is only effective when solving small-scale TSPs consisting of a maximum 40 nodes.

In addition, it was also found that the DFO algorithm had also resulted in the overall worst executions time for all tested TSP datasets compared to the PSO, ACO, and GA algorithms which can be seen in figures 4.20 and 4.21.

Furthermore, it was also found that the GA algorithm had significantly outperformed all of the other algorithms including the DFO in regards to outputted solution fitness values, with the GA algorithm outputting solutions to an extremely high degree of accuracy with some of the datasets being solved to 100% accuracy such as the 'wi29' and 'dj38' datasets as can be seen from figure 4.20, whilst the ACO had resulted in having the lowest elapsed execution throughout all the datasets used compared against the DFO, PSO, and GA algorithms.

4.3 How the project can be further improved based on this evaluation

Based on the finding from all of the experiments conducted in this paper, it was found that the DFO algorithm is best suited to be used on small-scale TSPs ranging up to a maximum of 40 nodes as this is where the DFO algorithm excels in solving the TSP to a very high degree of accuracy. However, as seen from section 4.2.6, the DFO algorithm had resulted in the worst elapsed execution times compared to any of the used algorithms which include the PSO, ACO, and GA algorithms. Moreover, it was also found that the although having a greater execution time throughout the use of all 6 datasets (section 4.2.5), the DFO had successfully managed to outperform the PSO and ACO algorithms, but despite this success, the GA had significantly outperformed the DFO algorithm to a very high standard of accuracy, with 2 out of 6 datasets the GA algorithm had managed to successfully solve to 100% and 99.97% accuracy ('dj38' and

'berlin52'). So therefore, in order to further improve the proposed DFO algorithm based on this evaluation of the performance and accuracy of the algorithm, it would be intended to further research and develop a better and more efficient update strategy for each fly in the swarm, in order to reduce the algorithms average elapsed execution time. It would also be intended to further improve the accuracy for large-scale TSPs, as currently, the algorithm is only effective and efficient at solving small scale TSPs consisting of up to a maximum of 40 nodes, which can also possibly be achieved by researching and developing a new and more efficient update strategy.

Chapter 5

Summary and Conclusions

5.1 Summary of the Project

To summarise this project, the DFO algorithm was proposed to solve the TSP, with the intention of optimising logistical routes within the logistical industry which can lead to shorter travel distances and times, which ultimately would result in saving time, fuel, and money, and also allow for more deliveries to be made in the same amount of time as before optimisation.

The motivation behind the use of the DFO algorithm to solve the TSP for logistical optimisation, is based on the findings that historically researchers have taken various conventional approaches to solve the TSP, including the PSO, ACO, and GA algorithms. However, despite these conventional approaches having been proven to be effective for solving the TSP, the motivation behind the use of the DFO algorithm is that it has never been previously applied to optimise and solve the TSP, thus making it a new and unique approach to solving the TSP. Additionally, Mohammad Majid Al-Rifaie found the DFO algorithm to be "more efficient in 84.62% and more reliable in 90% of the 28 standard optimization in benchmarks used" during his research and experiments (Al-Rifaie 2014). Moreover, it was also found from the research of Michael King and Mohammad Majid Al-Rifaie that the DFO algorithm was able to outperform the PSO, GA, and DE algorithms with the finding that the DFO algorithm "converges to better solutions in 71.05% of problem set" (King & Al-Rifaie 2017).

This paper had tried to answer a series of research questions which include, can the DFO algorithm be used to successfully optimise and solve any given TSP. Secondly, is the DFO algorithm able to successfully solve the TSP to a high degree of accuracy while maintaining a low required execution time, and the thirdly whether or not the DFO algorithm will be able to outperform other conventional algorithms used repetitively over the years to solve the TSP which include the PSO, ACO, and GA algorithms.

Next, it was found that the standard DFO algorithm is not suitable for solving the TSP due to the nature of how TSP solutions are formulated and the standard updating procedure of the DFO

algorithm is not compatible of updating. More specifically, based on the research conducted it was found that the TSP has various solution requirements which must be met in order to make a valid TSP solution, which includes ensuring that the solution only consists of unique values and that a solution can only consist of integer values from the search space, which is an issue because it was found that the standard update procedure can often result in outputting an updated dimension value in the form of a decimal value which would not be valid. Therefore, due to the standard DFO algorithm not being compatible with the TSP, a new compatible version of the DFO algorithm had been proposed which is compatible with solving the TSP.

The proposed TSP compatible DFO algorithm had then successfully been developed based on the original DFO algorithm and its standard update procedure, but with the correct changes and alterations made based on the requirements of a TSP solution to be compatible and suitable for solving the TSP. In order to achieve this, a city generation feature, relevant fitness function, and a compatible update function were successfully implemented. Firstly, a city generation feature was implemented which reads in inputted TSP files and creates TSP nodes based on the inputted file alongside with a distance matrix table which is generated to find the distance between any two given nodes, and is also used by the fitness function and ultimately the new update procedure. Secondly, the fitness function that was implemented was the Euclidean Distance which can be seen in figure 3.2, the fitness function is simply used to find the total length of a given solution using the Euclidean Distance formula. Thirdly, an updated update procedure was then implemented which is compatible with the requirements of a valid TSP solution, the updated update procedure was based on the original standard update procedure but with the added alterations of, firstly rounding the updated dimension in order to make it an integer value, as decimal values are not compatible, next the out of bounds control was also altered to select a random node from the list of available nodes if the updated dimension value is found to be out of bounds, and lastly the biggest change made was the inclusion of a out bounds control for the entire fly, which replaces all duplicate dimension node values in the given flies solution with a missing node value, thus making the solution a valid TSP solution.

The DFO algorithm was then evaluated on its performance which was achieved by conducting a series of experiments and tests in order to optimise the algorithm's parameters where it was found that the most optimal parameters is setting the swarm size to 800, setting the maximum iterations to 9000, and setting the disturbance threshold value to 0.001, as sing these parameters resulted in an average outputted solution fitness value of 29383.35791 on the 'wi29' dataset, which is a 4.08% difference from the known optimal solutions fitness value of 27603.

Moreover, it was also found that the DFO algorithm suffers from the local optimum trap, where the algorithms swarm is fixated on converging towards a single solution and is unable to converge towards a more optimal solution, in order to resolve this issue, 2 different approaches were tested which included an oscillating disturbance threshold approach, and also a multiple swarms approach. After testing both approaches, it was found that the multiple swarms approach had ultimately resulted in being the better approach, as it had successfully resulted in an average outputted solution fitness value of 29065.725763746 which was found to be within 5.2991550329529% accuracy the known solutions fitness value of 27603, and also was found to

be 13.170750265828% better than the initial average outputted solution fitness value of 32893.899917039555 which was found using the default parameters.

Furthermore, the DFO algorithm using the found optimal parameters in conjunction with the multiple swarms approach was then tested on 6 different TSP datasets against the PSO, ACO, and GA algorithms. After successfully conducting this experiment, it was found that the DFO algorithm was able to successfully solve 2 of the 6 datasets ('wi29' and 'dj38' datasets), to a very high degree of accuracy although requiring a very high elapsed execution time for each dataset.

It was also found that the DFO algorithm is only able to effectively and efficiently solve small-scale TSPs ranging up to a maximum of 40 nodes, and for anything above 40 nodes, the algorithm is unable to solve it to a high degree of accuracy as can be seen from figure 4.20, when the DFO algorithm was used on the 'berlin52' dataset the algorithm had outputted a solution with a fitness value with a difference of 35.62% from the known best solutions fitness value, further indicating that the DFO algorithm is only effective at solving small scale TSPs. Furthermore, despite this finding, the DFO algorithm was found to successfully outperform the PSO and ACO algorithm on all datasets, although it should be mentioned that both the PSO and ACO algorithms had significantly lower elapsed execution times. In addition, although the DFO outperformed the PSO and ACO in regards to outputted solutions, the GA algorithm had managed to outperform all 3 of the other algorithms including the DFO in all datasets, with it successfully solving all datasets to an extremely high degree of accuracy, where it had solved 2 of the 6 datasets to 100% and 99.7% accuracy.

Overall, this project aimed to effectively and efficiently solve the TSP using the DFO algorithm, in order to optimise logistical routes to find the shortest possible tour for a given set of logistical locations to a high degree of accuracy whilst maintaining a short execution time. Additionally, this project aimed to evaluate the effectiveness and efficiency of using the DFO approach towards solving the TSP against other conventional approaches that have been used frequently by researchers including the PSO, ACO, and GA algorithms.

The project had successfully achieved these goals, as a TSP compatible DFO algorithm using a city generation feature, Euclidean Distance based fitness function, and an updated update procedure was proposed and successfully developed. After finding the optimal parameters and also using the multiple swarms approach the DFO algorithm was able to successfully solve any given TSP. Although it should be mentioned that the DFO algorithm is only able to solve small scale TSPs to a high degree of accuracy. Moreover, despite successfully solving small-scale TSPs, the DFO algorithm was found to require high execution times.

Furthermore, the project also successfully achieve the goal of evaluating the effectiveness and efficiency of the proposed DFO algorithm against the PSO, ACO, and GA algorithms. This goal was met in section 4.2.6, where it was found that the proposed DFO algorithm had successfully outperformed the PSO and ACO algorithms by outputting better optimal solutions based on their fitness values. However, despite this success, it should also be mentioned that the DFO algorithm resulted in having the highest elapsed execution times of all the tested algorithms. Moreover, it

was ultimately found that proposed DFO algorithm was unable to outperform nor match the performance of the GA algorithm which had outputted the best overall solutions whilst successfully doing it in very short elapsed execution times.

5.2 Conclusions Drawn from the Project

To conclude, this project was overall a great success, as this project had successfully answered the intended research questions and had also achieved all its targeted goals.

Firstly, this project managed to successfully answer the key research question of can the DFO algorithm which has never previously been used on the TSP able to successfully solve the TSP. This research question was ultimately answered by proposing and developing a TSP compatible DFO algorithm using an updated update procedure in conjunction with a city generation feature and also a fitness function based on the Euclidean Distance formula, as the original DFO algorithm was not designed nor compatible to work with the TSP. The proposed algorithm was then tested on a range of different publicly available TSP datasets and was found to be able successfully optimise and solve them, although it should be mentioned, whilst using the default parameter values, the algorithm was outputting sub-optimal solutions as can be seen from section 4.2.1.

Secondly, this project has also successfully managed to answer the second research question, of whether or not the DFO algorithm is able to solve any given TSP to a high degree of accuracy while maintaining a relatively low execution time. It was initially found that the DFO algorithm was outputting solutions with sub-optimal fitness values when using the default parameters, as can be seen in section 4.2.1 where the average outputted fitness value for the 'wi29' dataset was 32893.899917039555 which is a difference of 17.4915% from the best known solutions fitness value.

However, in sections 4.2.2 and 4.2.3 the DFO algorithm was optimised based on finding its optimal parameters (section 4.2.3) and also with the use of the multiple swarms approach which was found to resolve the local optimum trap issue (section 4.2.3), whilst also further improving the average outputted solution accuracy of the DFO algorithm, was found to be able to successfully solve the 'wi29' TSP dataset within an accuracy range of 5.16237%, which is an overall improvement of 12.32913% compared to the initial findings.

Furthermore, this research question was also answered in section 4.2.5, where the DFO algorithm was tested on 6 different datasets ranging from 29 up to 194 nodes (section 4.2.5), where it was found that the DFO algorithm was able to solve the 'wi29' and 'dj38' datasets to a very high degree of accuracy of 4.08% and 2.28%, however, despite this, it was found that the DFO algorithm reaches its limitations around the 40 nodes mark, as can be seen in figure 4.18, where for the 'berlin52' dataset the outputted solution had a massive difference of 35.62% from the known optimal solution, moreover for the remainder of the datasets ranging in nodes sizes from 76 to 194, the algorithm had also struggled to achieve a solution to a high degree of accuracy.

Thus to answer the research question, it was found that the DFO algorithm is only able to solve small scale TSPs to a high degree of accuracy while also maintaining a reasonable execution time.

Thirdly, the final question this project has successfully managed to answer is, can the DFO algorithm outperform other conventional algorithms when solving the TSP, including the PSO, ACO, and GA algorithms. This research question was successfully answered in section 4.2.6, where the DFO algorithm was compared against the aforementioned against the 6 datasets from section 4.2.5. Based on the findings from section 4.2.6, it was found that the DFO algorithm was able to only successfully outperform the PSO and ACO algorithms on all dataset instances including those consisting of more than a maximum of 40 nodes, however, it was also found that DFO was unable to outperform the GA algorithm, as the GA algorithm had significantly outperformed the DFO algorithms on all datasets, where some datasets the GA algorithm had successfully found the known optimal solution.

Overall, this project was a success as it was found the DFO algorithm using the found optimal parameters and also the multiple swarms approach is able to successfully solve small scale TSP ranging up to a maximum of 40 nodes effectively and efficiently, to a high degree of accuracy while maintaining an acceptable execution time, and it was also found that the DFO algorithm was able to successfully outperform the PSO and ACO algorithms which have been used numerous times as conventional approaches to solving the TSP, meaning that the DFO algorithm can therefore be used to successfully optimise small scale logistical routes. However, despite successfully outperforming the PSO and ACO algorithms, the GA algorithm was found to be a superior approach to solving the TSP, as it had outperformed the DFO algorithm in both outputted solution fitness values and also required execution times.

Bibliography

- Al-Rifaie, M. M. (2014), Dispersive flies optimisation, in ‘2014 Federated Conference on Computer Science and Information Systems’, IEEE, pp. 529–538.
- Al-Rifaie, M. M. (2020), ‘Dispersive flies optimisation’, <https://github.com/mohmaj/DFO>.
- Deng, W., Xu, J. & Zhao, H. (2019), ‘An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem’, *IEEE access* **7**, 20281–20292.
- Du, P., Liu, N., Zhang, H. & Lu, J. (2021), ‘An improved ant colony optimization based on an adaptive heuristic factor for the traveling salesman problem’, *Journal of Advanced Transportation* **2021**.
- Gao, W. (2020), ‘New ant colony optimization algorithm for the traveling salesman problem’, *International Journal of Computational Intelligence Systems* **13**(1), 44–55.
- Ghiani, G., Adamo, T., Greco, P. & Guerriero, E. (2020), ‘Lifting the performance of a heuristic for the time-dependent travelling salesman problem through machine learning’, *Algorithms* **13**(12), 340.
- Graeme, W. (2021), ‘Uk’s lorry driver shortage ‘could push food prices up’, as supply chain crisis hits confidence – as it happened’, *The Guardian* .
URL: <https://www.theguardian.com/business/live/2021/aug/26/uk-shortages-supply-chains-jobs-cars-ftse-100-sterling-dow-business-live>
- Guo, J. & Sato, Y. (2020), ‘A standardized bare bones particle swarm optimization algorithm for traveling salesman problem’, *International Journal of Machine Learning and Computing* **10**(3).
- Hamilton, W. R. (n.d.), ‘Travelling salesman problem’.
- Juneja, S. S., Saraswat, P., Singh, K., Sharma, J., Majumdar, R. & Chowdhary, S. (2019), Travelling salesman problem optimization using genetic algorithm, in ‘2019 Amity International Conference on Artificial Intelligence (AICAI)’, IEEE, pp. 264–268.
- Kennedy, J. & Eberhart, R. (1995), Particle swarm optimization, in ‘Proceedings of ICNN’95-international conference on neural networks’, Vol. 4, IEEE, pp. 1942–1948.

- King, M. & Al-Rifaie, M. M. (2017), 'Building simple non-identical organic structures with dispersive flies optimisation and a* path-finding', *AISB 2017: Games and AI* pp. 336–340.
- Lisa, O. (2021), 'Uk facing summer of food shortages due to lack of lorry drivers', *The Guardian*.
URL: <https://www.theguardian.com/business/2021/jun/25/uk-facing-summer-of-food-shortages-due-to-lack-of-lorry-drivers>
- Mele, U. J., Gambardella, L. M. & Montemanni, R. (2021), 'A new constructive heuristic driven by machine learning for the traveling salesman problem', *Algorithms* **14**(9), 267.
- Mirjalili, S., Mirjalili, S. M. & Lewis, A. (2014), 'Grey wolf optimizer', *Advances in engineering software* **69**, 46–61.
- Ning, J., Zhang, Q., Zhang, C. & Zhang, B. (2018), 'A best-path-updating information-guided ant colony optimization algorithm', *Information Sciences* **433**, 142–162.
- Osaba, E., Yang, X.-S., Diaz, F., Lopez-Garcia, P. & Carballedo, R. (2016), 'An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems', *Engineering Applications of Artificial Intelligence* **48**, 59–71.
- Panda, M. (2018), 'Performance comparison of genetic algorithm, particle swarm optimization and simulated annealing applied to tsp', *International Journal of Applied Engineering Research* **13**(9), 6808–6816.
- Saeed, A. L. A. A. M. (2019), 'Using genetic algorithm and ant colony optimization for solving traveling salesman problem'.
- Saji, Y. & Barkatou, M. (2021), 'A discrete bat algorithm based on lévy flights for euclidean traveling salesman problem', *Expert Systems with Applications* **172**, 114639.
- Shin, W., Tan, T. R., Stoller, P., Yew, W. & Liew, D. (2020), 'Issues on the logistics challenges in the pandemic period', *J. Crit. Rev* **7**(8), 776–780.
- Sopto, D. S., Ayon, S. I., Akhand, M. & Siddique, N. (2018), Modified grey wolf optimization to solve traveling salesman problem, in '2018 International Conference on Innovation in Engineering and Technology (ICIET)', IEEE, pp. 1–4.
- Strak, Ł., Skinderowicz, R. & Boryczka, U. (2018), 'Adjustability of a discrete particle swarm optimization for the dynamic tsp', *Soft Computing* **22**(22), 7633–7648.
- Sureja, N. M. (2020), 'Solving random travelling salesman problem using firefly algorithm', *international Journal of Innovative Technology and Exploring Engineering (IJITEE)*.
- Wang, R., Xia, K. & Sandrine, M. (n.d.), Grey wolf optimizer with multi-strategy optimization and its application on tsp, in 'Proceedings of the 2020 International Conference on Internet Computing for Science and Engineering'.

- Wei, B., Xing, Y., Xia, X. & Gui, L. (2021), 'A novel particle swarm optimization with genetic operator and its application to tsp', *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)* **15**(4), 1–17.
- Xiong, H. (2021), 'Research on cold chain logistics distribution route based on ant colony optimization algorithm', *Discrete Dynamics in Nature and Society* **2021**.
- Xu, C., Gordan, B., Koopialipour, M., Armaghani, D. J., Tahir, M. & Zhang, X. (2019), 'Improving performance of retaining walls under dynamic conditions developing an optimized ann based on ant colony optimization technique', *IEEE Access* **7**, 94692–94700.
- Yang, X.-S. (2010), A new metaheuristic bat-inspired algorithm, in 'Nature inspired cooperative strategies for optimization (NICSO 2010)', Springer, pp. 65–74.
- Yang, X.-S. (2012), 'Nature-inspired metaheuristic algorithms: success and new challenges', *arXiv preprint arXiv:1211.6658*.
- Yuan, Y., Li, H. & Ji, L. (2021), 'Application of deep reinforcement learning algorithm in uncertain logistics transportation scheduling', *Computational intelligence and neuroscience* **2021**.