

https://dspace.org/

Solving travelling salesman problem using iterative discrete flower pollination algorithms: a parallel processing perspective

Albertyn, Jandre Johan

2022
https://hdl.handle.net/10539/34948

Downloaded from a DSpace repository. https://dspace.org/

Solving Travelling Salesman Problem Using Iterative Discrete Flower Pollination Algorithms: A Parallel Processing Perspective

Jandre J. Albertyn

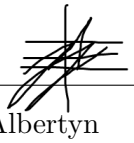
A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, June 2022

Declaration

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

Signed this 25th day of June 2022



Jandre J. Albertyn

Abstract

There has been a growing rate of new nature inspired meta-heuristic algorithms that have been developed for sub-optimally solving NP-Hard Problems. This has led to the development of the Flower Pollination Algorithm (FPA) by Yang in 2012, which has been able to outperform various common meta-heuristics. Due to its relative age it is still underdeveloped as compared to common meta-heuristics such as, the Genetic Algorithm (GA), Simulated Annealing (SA), and the Ant Colony Optimisation (ACO). This research investigates various parallel methods in order to design, implement and test a parallel iterative Discrete Flower Pollination Algorithm (PiDFPA) that can further optimise the FPA, and to investigate the performance of the PiDFPA on large maps in comparison to common parallelised meta-heuristic algorithms in the context of the Symmetrical Travelling Salesman Problem (TSP). Results within this research show that the PiDFPA is able to achieve the optimal tour length for small maps and is also able to overcome the weakness of the iDFPA when applied to large maps, where the PiDFPA is able to outperform its benchmark counterparts, the Parallel Ant Colony Optimisation (PACO) and the Parallel Genetic Algorithm (PGA), in terms of convergence rate and accuracy. An analysis is also done on the complexity as well as efficiency of the iDFPA and the suggested PiDFPA showing that the PiDFPA is able to exponentially increase the speedup of the iDFPA for TSPs as the map size increases. This is largely due to the adjustment of control variables that allow the PiDFPA to dynamically adjust its exploitation and exploration probability between each iteration along with minimizing communication overhead between agents by making use of the Nvidia CUDA grid architecture.

Acknowledgements

I would like to thank Prof. Ling Cheng from the University of the Witwatersrand as well as Dr Adnan Abu-Mahfouz from CSIR for the assistance and guidance they provided through the preparation of this dissertation.

Contents

Declaration **i**

Abstract **ii**

Acknowledgements **iii**

Contents **iv**

List of Figures **viii**

List of Tables **x**

List of Symbols **xii**

Nomenclature **xv**

1 Introduction **1**

1.1 Research Question 2

1.2 Research Significance 3

1.3 Scope and Research Objectives 3

1.4 Research Achievements 3

1.5 Dissertation Organisation 4

1.6 Conclusion 5

2 Literature review **6**

2.1 Combinatorial optimisation problems 6

2.2 Exact Optimisation algorithms 6

2.3 NP-hard problems 7

2.4 Meta-Heuristics 8

2.5 Nature Inspired Meta-Heuristics 8

2.5.1 Genetic Algorithm 9

2.5.2 Particle Swarm Optimisation 9

2.5.3 Flower Pollination Algorithm 9

3 Preliminaries **12**

3.1 Travelling Salesman Problem 12

3.2 Genetic Algorithm 14

3.3 Ant Colony Optimisation 16

3.4 Iterative Discrete Flower Pollination Algorithm 18

3.5 Parallelisation Methods 20

3.5.1 Master Slave Model 20

3.5.2 Parallel Independent Model 21

3.5.3 Multi-Colony Model 22

3.6 Conclusion 23

4 Parallel Iterative Discrete Flower Pollination Algorithm for Solving Symmetric Travelling Salesman Problem **24**

4.1 CUDA 24
4.1.1 Threads 24
4.1.2 Device Memory 25
4.1.3 Synchronization 25
4.2 CUDA implementation 25
4.3 Experimental Setup 26
4.4 Conclusion 28

5 Results **29**

5.1 Control Variable Adjustments 29
5.2 Accuracy Comparison 30
5.3 Convergence Rate Comparison 31
5.4 Serial and Parallel 32
5.5 Conclusion 33

6 Analysis **42**

6.1 Accuracy 42
6.1.1 Evaporation 42
6.1.2 Rejection Update 43
6.1.3 Best Tour Update 44
6.2 Rate of Convergence 46
6.2.1 Reduction in Complexity for Parallel vs Serial 46
6.3 Speedup and Efficiency 47
6.4 Performance against other Algorithms 48

6.4.1 Additional Complexity 48

6.4.2 Limited Shared Resources 49

6.4.3 Implementation Error 50

7 Conclusion **51**

7.1 Contributions 52

7.2 Future Work 52

7.3 Conclusion 52

List of Figures

3.1 TSPLib Bay29 example. Adapted from [58] 13

3.2 TSPLib Bay29 optimal tour. Adapted from [58] 14

3.3 Master Slave model 21

3.4 Parallel Independent model 22

3.5 Multi-colony model 23

4.1 CUDA GPU interaction 26

4.2 CPU vs GPU split 28

5.1 Tour length comparison for PiDFPA, PGA and PACO against optimal tour for the Berlin52 TSP map over 300 iterations 34

5.2 Tour length comparison for PiDFPA, PGA and PACO against optimal tour for the Att532 TSP map over 300 iterations 35

5.3 ITour length comparison for PiDFPA, PGA and PACO against optimal tour for the U1060 TSP map over 300 iterations 36

5.4 Box chart of the PiDFPA for 300 Iteration execution of the Berlin52 TSP map 37

5.5 Box chart of the PACO for 300 Iteration execution of the Berlin52 TSP map 37

5.6 Box chart of the PiDFPA for 300 Iteration execution of the Att532 TSP map 38

5.7 Box chart of the PACO for 300 Iteration execution of the Att532 TSP map 38

5.8 Box chart of the PiDFPA for 1000 Iteration execution of the U1060 TSP map 39

5.9 Box chart of the PACO for 1000 Iteration execution of the U1060 TSP map 39

5.10 Logarithmic time evaluation of the Berlin52 TSP map by the iDFPA and PiDFPA over 300 iterations 40

5.11 Logarithmic time evaluation of the Att532 TSP map by the iDFPA and PiDFPA over 300 iterations 40

5.12 Logarithmic time evaluation of the U1060 TSP map by the iDFPA and PiDFPA over 1000 iterations 41

6.1 PiDFPA α value adjustment for Att532 43

6.2 PiDFPA β value adjustment for Att532 44

6.3 PiDFPA γ value adjustment for Att532 45

6.4 PiDFPA α value adjustment for U1060 45

6.5 PiDFPA β value adjustment for U1060 46

6.6 PiDFPA γ value adjustment for U1060 46

6.7 iDFPA complexity analysis 47

6.8 PiDFPA complexity analysis 48

6.9 Iteration time cost comparison for PACO vs PiDFPA for Berlin52 over 300 iterations 49

6.10 Iteration time cost comparison for PACO vs PiDFPA for Att532 over 300 iterations 49

6.11 Iteration time cost comparison for PACO vs PiDFPA for U1060 over 1000 iterations 50

List of Tables

3.1 Crossover Example 16

5.1 Testing parameters used to find best and average solutions for TSPLIB problems 30

5.2 Accuracy comparison of PiDFPA against optimal solution 31

5.3 Accuracy comparison of PiDFPA against best PGA solution 31

5.4 Accuracy comparison of PiDFPA against best PACO solution 31

5.5 Execution time difference between PiDFPA and PGA 32

5.6 Execution time difference between PiDFPA and PACO 32

5.7 GPU speed and efficiency combination for different TSP maps . . . 33

List of Algorithms

1 Host code 27
2 Calculate best 27

List of Symbols

A	Normalisation constant
α, β, γ	Control constants
C	Cost matrix
c_{ij}	Cost from node i to node j
ΔC	Change in cost
D	Distance matrix
d_{ij}	Single distance between node v_i and v_j
d	Set of tour distances
d'	A subset of d containing the Rejection Update accepted tour distances
d_{prev}	The previous iteration's tour distance
d_{s_j}	Tour distance of tour s_i
d_{s^*}	Set of edges
$\Delta dist$	Change in distance
E	Graph edges
$\in$	Symbol to indicate in
η	Desirability level
F_L	Continuous Lévy Distribution

Γ	Gamma function
G	Graph
$g*$	Current best solution
$L\bar{V}$	Discrete Lévy Distribution
l^*	Minimum length
λ	Lambda - Lévy constant
m	Number of elements
N	Number of iterations
N_k^i	Unvisited nodes
N_{curr}	Current iteration
n	Number of cities
p	Selection probability
P	Probability
Φ	Step constant for Lévy Distribution
ρ	Switch probability
q	RU constant
r	Random number
r_{dist}	Radial threshold distance
S	Set of tours
s_n	Tour element at n
S_n^j	Tour element at n for j-th tour
s^*	Best tour

τ	Pheromone level
$\Delta\tau$	Change in pheromone level
U	Uniform distribution
V	Set of nodes or single tour
v_i	Single node in V
v_{prev}	Previous node
w	RU constant

Nomenclature

ACO ACO

BA Bat Algorithm

BTU Best Tour Update

CUDA Compute Unified Device Architecture

GA Genetic Algorithm

iDFPA iterative Flower Pollination Algorithm

MKP Multiple Knapsack Problem

NP-hard problem Non-deterministic polynomial time hard problem

PACO Parallel Ant Colony Optimisation

PGA Parallel Genetic Algorithm

PiDFPA Parallel iterative Flower Pollination Algorithm

PSO Particle Swarm Optimisation

RU Rejection Update

SA Simulated Annealing

TSP Travelling Salesman Problem

Chapter 1

Introduction

The Travelling Salesman Problem (TSP) was introduced by Karl Menger is 1930 [1]. The fundamental basis of the TSP is finding the shortest path that can be traversed from any point on a map so that each point on the map is visited only once. The TSP can therefore be defined as a combinatorial optimisation problem [2] as it involves the process of searching through a large set of discrete elements to find an optimal objective with a discrete set of solutions. The TSP is however also classified as being an NP-Hard Problem [3]. The TSP is considered to be NP-Hard since computing every solution, or finding the optimal path through brute force, quickly exceeds polynomial time as the set of elements to traverse increases [4].

Solving NP-Hard problems and finding the optimal solution for every problem is therefore infeasible which has led to the application of heuristic algorithms [5]. Heuristic algorithms only calculate a subset of the solutions based on probabilistic costs for each point within the TSP map [6]. During each iteration of the heuristic, the probabilistic costs are adjusted until the final iteration has been completed leading to either the optimal solution or a sub-optimal solution.

Heuristic algorithms can be split into two fields, namely specific heuristic algorithms, which are optimised for only single problems [7] and meta-heuristic algorithms, where specific algorithms are generalised to be applied to a wide set of problems [8].

The ability for meta-heuristics to find practical solutions for NP-Hard problems has sparked a wide interest in finding highly optimised meta-heuristics that can be applied over a range of problems such as the TSP, the Multiple Knapsack Problem (MKP) [9] and the Hamiltonian Path Problem [10], which form the basis of many modern day solutions for applications that involves cutting and packing planning [11], vehicle routing problems [12], logistic configurations [13], network planning and

scheduling [14].

One of the most promising new fields of meta-heuristics are nature based algorithms, which have been able to outperform classic heuristic optimisation methods such as the k-Means Heuristic [15] and iterated local search [16] in convergence rate and accuracy, due to the ability to quantify a multitude of complex techniques that have been part of the social or natural world for hundreds of years. This has led to the discovery and implementation of numerous algorithms such as the Bat Algorithm (BA) [17], the Ant Colony Optimisation (ACO) [18] and the Differential Evolution (DE) [19].

The Flower Pollination Algorithm (FPA), first introduced in 2012 [20], is a nature inspired algorithm that has been able to consistently outperform the common natural algorithms such as the Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) within continuous optimisation problems such as the non-parametric Friedman test [21]. The FPA has also been adapted to a discrete version in the iterative Discrete Flower Pollination Algorithm (iDFPA), where it could be benchmarked against one of the most popular NP-Hard combinatorial problems, namely the TSP, alongside the GA and ACO algorithms. The iDFPA was able to outperform both the GA and ACO in terms of accuracy of the sub-optimal solution within small map TSP problems [22].

Increased demands for real life applications have also led to different optimisation techniques being implemented on common nature inspired algorithms to enable those algorithms to be applied to large maps. One such technique is parallelisation of the ACO to form the Parallel Ant Colony Optimisation (PACO) method [23], which has been able to preform to the same level as the ACO in terms of accuracy but with much faster convergence times due to being able to perform various calculations simultaneously.

1.1 Research Question

How would the Parallel Combination of Ants and Evaluation of Solution Elements method perform in terms of convergence rate and accuracy of the optimal solution for parallelising the iDFPA and additionally, how would the PiDFPA compare to existing PACO solutions using the same parallelisation method for a large-scale TSP map?

1.2 Research Significance

The significance of this research is that it further develops methodologies used to solve problems that cannot be solved in real time. It aims to speed up the process used to calculate current Travelling Salesman Problems like planning the most efficient possible route for automated machinery to traverse a warehouse or for a bus driver to pick up a set of people. But because of the nature of NP-Hard problems, the solutions discovered in this research can be adapted to the MKP problem to assist in scheduling tasks for controlling smart grids to reduce the electrical consumption of an entire neighbourhood.

1.3 Scope and Research Objectives

The aim of this research is to introduce a way of increasing the rate of convergence and accuracy of meta-heuristics when applied to the TSP. The proposed solution for achieving this aim is to parallelise the existing iDFPA meta-heuristic using the Parallel Ants method and then comparing the result to the PACO method to investigate the performance of both methods.

1.4 Research Achievements

In this research a new algorithm was investigated, designed and developed. The newly formed PiDFPA was implemented on the Nvidia CUDA system. This version was able to show much more promising results than its original CPU design as it is able to solve TSP maps that contain more that 1000 location elements. The PiDFPA was also compared against parallel versions of common meta-heuristic algorithms namely the PGA and the PACO. By matching the base architecture of all of the parallel algorithms, the PiDFPA was able to outperform both the PGA and PACO in terms of sub-optimal solution accuracy and rate of convergence within the same amount of elapsed time.

1.5 Dissertation Organisation

Literature Review (Chapter 2)

This chapter provides a literature review for combinatorial optimisation problems and the formulation of NP-Hard Problems that led to the proposal of the TSP. This literature review also introduces some of the common meta-heuristics used to find sub-optimal solutions for NP-Hard Problems.

Preliminaries (Chapter 3)

This chapter provides a more detailed look at the exact problem that this research is trying to address along with a background into common current examples of meta-heuristics that have been developed to solve the problem. Thereafter the two elements that are used to create the PiDFPA are discussed.

Methodology (Chapter 4)

This chapter focuses on the architecture and methods used by the PiDFPA along with an experimental setup to demonstrate the functions being used by the PiDFPA and other benchmarking meta-heuristics.

Results (Chapter 5)

This chapter provides the results that are obtained from the PiDFPA comparison to the optimal results as well as the PACO and PGA optimisation meta-heuristics in a fair assessment of its performance metrics in terms of accuracy and rate of conversion.

Analysis (Chapter 6)

The chapter provides a deeper analysis regarding the results that are reported in the previous chapter to explain the performance metrics of the PiDFPA in terms of control variable adjustments and design architecture. By doing so, an understanding

can be achieved as to why certain trends appear within the results. This allows for additional future adjustments and testing to be considered.

Conclusion (Chapter 7)

The final chapter summarises the research, presents its achievements and presents possible future improvements.

1.6 Conclusion

This chapter serves as a broad introduction to the TSP and the Flower Pollination Algorithm and explains why researching these topics are important in terms of both practical implementations as well and the development of future optimisation methods as well as the current achievements of outperforming the PACO and PGA methods. This chapter then give a brief summary of each of the chapters that will be presented to guide the user through the research.

Chapter 2

Literature review

2.1 Combinatorial optimisation problems

Combinatorial optimisation is a prominent topic in the areas of computer science and mathematics as it forms the basis of many algorithms used in artificial intelligence [24], informatics [25] and algorithm theory [26]. Combinatorial optimisation involves the process of finding the minimum or maximum object from a set of feasible discrete solutions. The most common applications for combinatorial optimisation are finding the shortest path to visit a set number of locations, scheduling events as a function of time, allocating a finite number of resources and the design process for hardware construction [27].

Combinatorial optimisation problems can be solved by making use of either exact algorithms or heuristics algorithm [28]. Exact optimisation algorithms explore all the solutions within the problem domain to find the most optimal solution [29] whereas heuristic algorithms focus on actively exploring subsets of the solution set to find a near optimal solution which might not include the optimal solution [30].

2.2 Exact Optimisation algorithms

The two most common variants of optimisation algorithms are the branch-and-bound [31] method and the dynamic programming method [32]. Each problem that is introduced to the branch-and-bound method is split into smaller sub-problems where solving each of the smaller problems can be used together to solve the overall problem [33]. These smaller problems are subjected to calculations called bounding tests,

which are performed to see if an optimal result can be obtained from the smaller subset of problems. If the smaller problem can be solved to either determine that it contains part of the optimal solution or no parts of the optimal solution can be obtained, it is terminated. If the smaller problem still cannot be solved using the bounding tests, it is split (decomposed) into even smaller problems. This process continues until all of the sub-problems have been decomposed to their smaller parts and terminated [34].

The dynamic programming method also breaks the problem down into optimal substructures but instead of simplifying parts down until they can be solved, the dynamic programming method makes use of finding overlapping sub-problems to increase the speed at which a solution can be found by reducing the size of the solution set that needs to be evaluated [35].

Though both of these processes optimise the search space and resultant times, they both struggle with large problem spaces as can be shown in the the Held-Karp programming algorithm to solve the TSP [36]. The resultant algorithm was proved to exponentially outperform the brute force method, but still entered the non-deterministic polynomial-time limit for more that 20 elements [37].

2.3 NP-hard problems

Many practical real world optimisation problems like the TSP, the Multi-Knapsack Problem and the Hamiltonian Path Problem are all subsets of Non-Deterministic Polynomial-time hard (NP-hard) combinatorial optimisation problems. NP-Hard means that all the solutions for these problems cannot be found and evaluated in real time as they have an exponential growth in solution elements for the amount of elements presented in the problem [38].

NP-hard problems can be surmised as containing a series of choices at each node of the problem. Once a node is selected and the choice has been made all other nodes that could have been selected are excluded from the set of solutions that can be found. At each node a new set of choices and dependencies are generated making the scope of choices at each node is mutually exclusive. This means that the problem cannot be broken down into smaller parts where each part is evaluated and the combined into one solution. Which means that an optimal solution can only be found if each node is evaluated for every permutation of choice that can be made [39].

2.4 Meta-Heuristics

The solution to practically solving NP-hard problems can however be found in meta-heuristic-based optimisation algorithms [40]. Meta-heuristic algorithms form can be identified as approximate search algorithms, which is subdivided into single-based and population-based search algorithms. These meta-heuristics are able to sharply decrease the amount of time required to find solutions for NP-Hard problems due to only searching for a near-optimal (sub-optimal) solution within a subset of solutions. This is ideal for situations where computational capacity is limited or if only certain subsets of solutions can be obtained. Meta-heuristics are normally, but not always developed from heuristic algorithms. Heuristic algorithms are simple focused algorithms and have variables that are specifically tuned for exploitation or exploration. Exploitation focuses on finding the shortest path by favouring the most common solution elements whereas exploration tends to focus on solution elements that are not commonly selected favouring more diversity. Meta-heuristic algorithms are developed as more complex versions of heuristic algorithms that are able to make assumptions during the decision process to balance exploitation and exploration [41].

2.5 Nature Inspired Meta-Heuristics

Meta-heuristic algorithms can be constructed in various ways with most occurring through some form of balancing of exploration and exploitation by making use of existing specialised heuristics to solve general problems as previously mentioned. A popular modern field of meta-heuristics called nature inspired meta-heuristics was introduced by Glover and Greenberg [42] in 1989 to use natural and social systems to solve NP-Hard combinatorial optimisation problems. These nature inspired heuristics can be classified to have the following properties [43]:

- Can be explained in terms of a social or natural behaviour or phenomenon.
- Are able to use results from previous iterations to affect choices for subsequent iterations.
- Are used to solve non-deterministic polynomial time problems.

These characteristics can be presented in the following common nature-inspired meta-heuristic algorithms, namely, the Genetic Algorithm(GA), Particle Swarm Optimisation and the Flower Pollination Algorithm (FPA).

2.5.1 Genetic Algorithm

The GA was first presented by John Holland in 1975 [44]. Its basis in the natural world is to mimic the process of evolution. The GA groups the elements of a fitness function into subsets called genes. These genes can then be combined to form chromosomes where each chromosome has its fitness function determined by the arrangements of genes for each chromosome. The chromosomes then evaluate the fitness of other chromosomes to find a partner to create a new set of chromosomes through a process called selection. After selection has been performed, the pair of chromosomes exchange genes and create a new generation of chromosomes, before randomly mutating (swapping around) their own genes. The GA uses randomness in its mutation and crossover to favour exploration and looks for the best fitness in selection to favour exploitation [45].

2.5.2 Particle Swarm Optimisation

Kennedy and Eberhart did their first simulation on the Particle Swarm Optimisation in 1995 [46] based on the process of a flock of birds searching for corn that was presented by Heppner and Grenander in 1990 [47]. The PSO functions by spreading the particle (elements) around in a set search space. Each element then determines its own fitness function and plans its next move by evaluating its current fitness and historical fitness for each iteration. After a few iterations the swarm of elements start converging on a location, mimicking a flock of birds to minimise their fitness function.

2.5.3 Flower Pollination Algorithm

The Flower Pollination Algorithm (FPA) developed by Zhou [20] is a nature inspired algorithm that is modelled after the pollination process of plant reproduction. The FPA focuses on optimising the rate at which flowers are pollinated to enable optimal reproduction [48]. The FPA can be summarised by the following four rules:

- Rule 1 - Biotic and cross-pollination act as global pollination. The agents moving across global space follow the lévy-flight distribution.
- Rule 2 - Biotic and self-pollination are forms of local pollination.

- Rule 3 - Pollinators can develop a reproduction probability that is proportional to the similarity of two flowers involved.
- Rule 4 - Switching between local and global pollination is controlled by a switching probability. This is normally biased slightly towards local pollination.

The FPA rules defines an algorithm that has two modes of operation. The FPA either acts as a local search algorithm or as a global search algorithm for each point depending on its switching probability, which can be set to favour either exploration or exploitation. Lastly rule 3 can be explained in terms of the flowers being able to adjust their probability of reproducing with other flowers during each iteration. The novelty that the FPA introduces lies within the process it uses for its global search mode. The Lévy-flight distribution enables large walks which leads to a unique way of exploring the domain and allows for more diversity [49].

The simplicity of controlling the exploitation and exploration within the FPA as well as its performance in performing global searches has made the FPA a strong candidate for testing and further development. Some examples of how the FPA has been implemented, modified or hybridised can be seen in the following examples:

- The FPA was improved by the introduction of parallel and compact schemes, where it was able to outperform both the PSO and GA in terms of optimising problems within Wireless Sensor Networks [50].
- The FPA was used to optimise the Economic Load Dispatch and Combined Economic Emission Dispatch problems, where it outperformed bee colony and genetic algorithm implementations [51].
- The FPA has been hybridised with the Clonal Selection Algorithm and benchmarked against 23 different NP-Hard functions including Lévy's function, during which it was able to outperform the GA, the Bat Algorithm, the Firefly Algorithm, and Simulated Annealing [52].

The FPA has also attracted enough attention such that reviews [53] - [54] have been written to indicate the level of interest in the new algorithm as well as the vast amount of applications within which the FPA can be used to further optimise current solutions.

The literature review gives a background to the Flower Pollination algorithm and the TSP by introducing NP-Hard Combinatorial optimisation problems as well as

explaining their importance alongside the relevance of the meta-heuristics that can be used to find near-optimal solutions in real time for practical problems.

Chapter 3

Preliminaries

3.1 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is one of the most common combinatorial problems found within the fields of mathematics and computer science. The main reason for its popularity derives from how easy it is to describe the problem as well as its wide range of applications. Applications that use TSP include, drilling printed circuit boards, X-ray crystallography and vehicle routing problems [55]. The TSP can be classified into three different versions, namely the Symmetric Travelling Salesman Problem(sTSP), the Asymmetric Traveling Salesman Problem(aTSP) and the multiple Travelling Salesman Problem. The only version that will be considered for this research is the sTSP.

The TSP can be explained in terms of a salesman who needs to visit a number of cities. The salesman can visit these cities in any order but needs to visit each city exactly once and needs to determine the shortest path to drive. This can be mathematically represented as a complete weighted graph $G = (N, E)$ where G is the graph or domain space of the problem, N is the number of cities that need to be visited and E is the edges or paths between each city [56]. Each edge can be described to have a cost function of

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.1}$$

where i and j are two cities on the graph. A valid tour with the TSP can be modelled as $S = \{v_1, v_2, ..., v_i, ..., v_j, ..., v_n\}$ where $i \neq j$ given $(i, j) \in E$, S is the tour and

$v \in N$. The optimal solution is then represented as

$$min|d(v_n, v_1) + \sum_{i=1}^{n} d(v_i, v_{i+1})| \tag{3.2}$$

This simplicity can be deceiving as the TSP can be classified as an NP-Hard problem due to the exponential increase in the amount of calculations required to be performed every time the size of the TSP increases [57]. The cost of evaluating every value in the TSP is $(n-1)!$ which makes it unsolvable in polynomial time. An example of the TSP problem can be seen in Fig. 3.1, where the coordinates provided by TSPLiB [58] for the Bay29 problem have been illustrated as map of points. Each of these points can be linked to any other point in the path while generating a solution. The optimal solution for the Bay29 path is shown in Fig. 3.2.

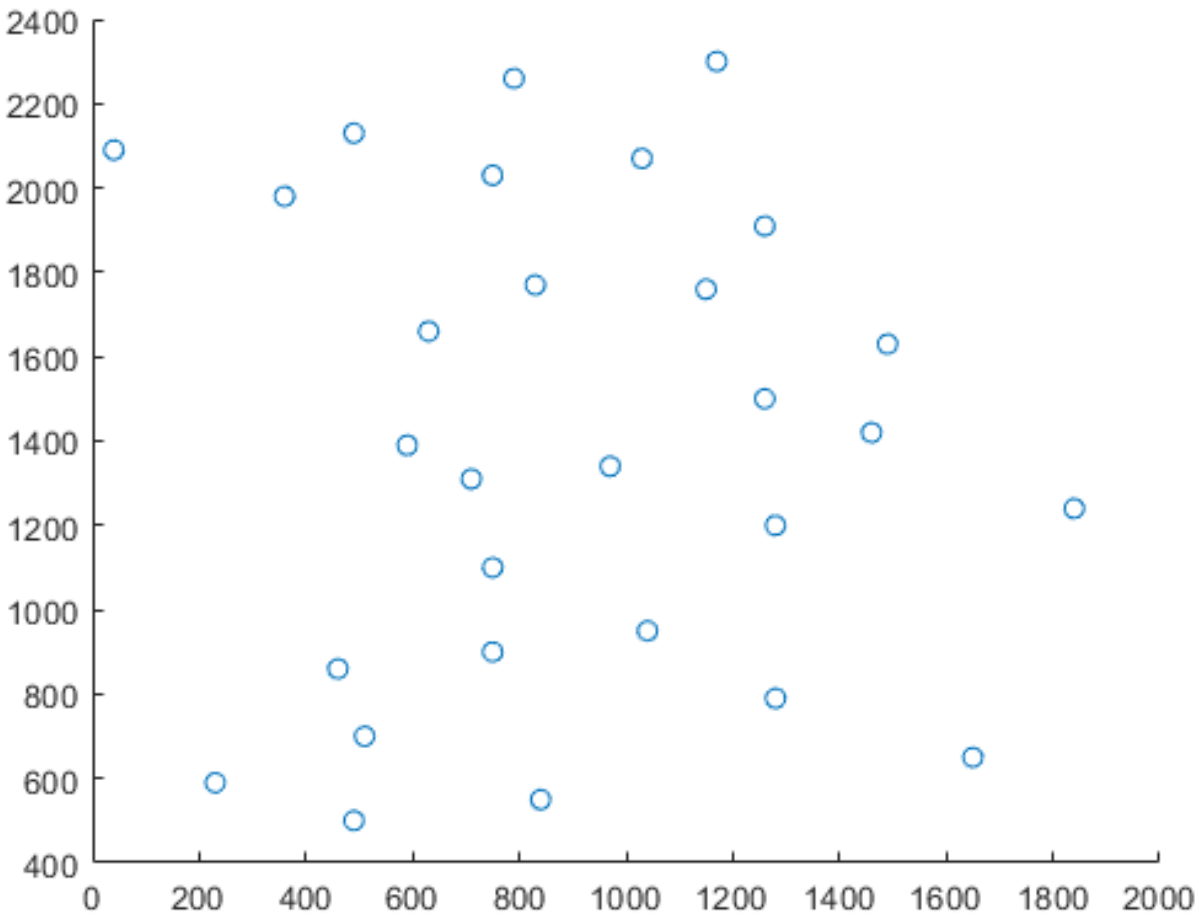


Fig. 3.1: TSPLib Bay29 example. Adapted from [58]

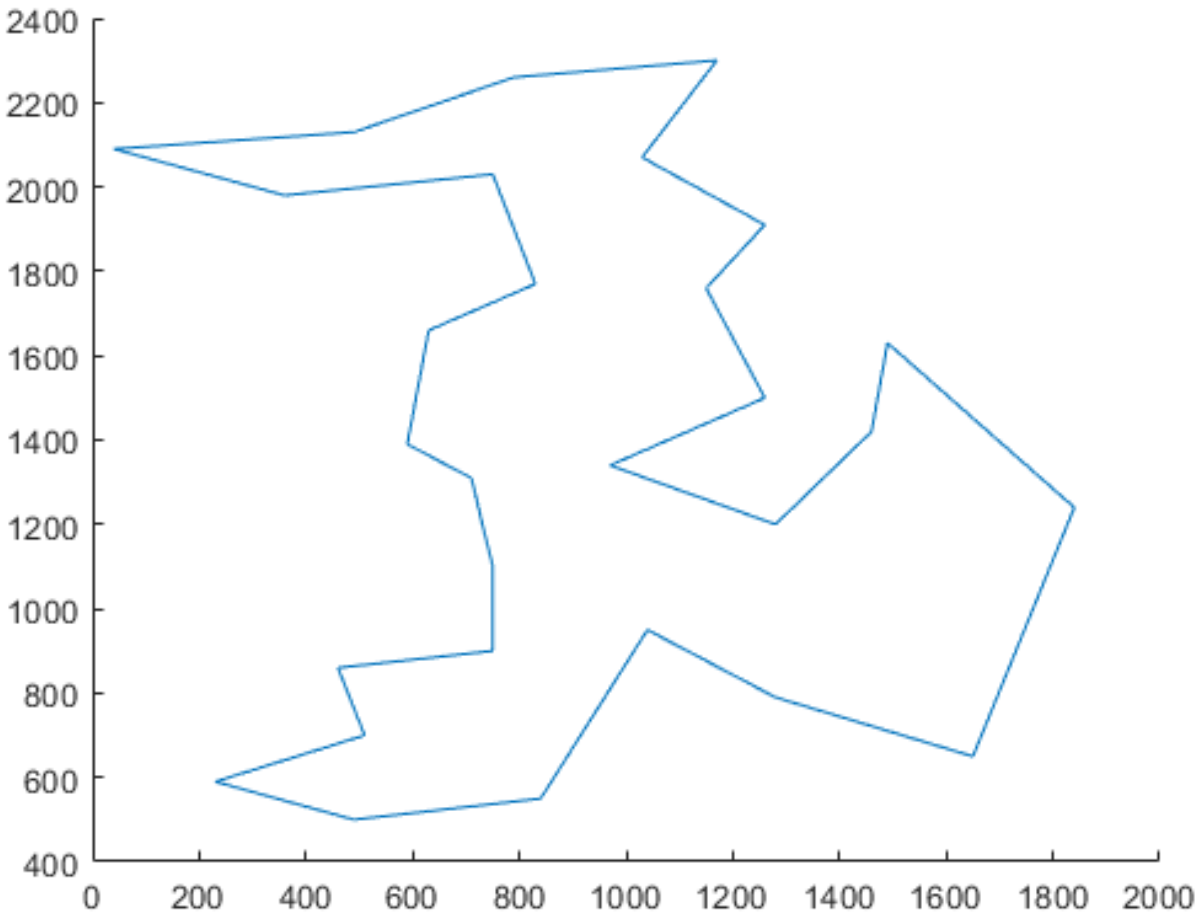


Fig. 3.2: TSPLib Bay29 optimal tour. Adapted from [58]

3.2 Genetic Algorithm

The GA can be used to summarise the Symmetric TSP, by equating each city to a gene and equating the tour distance between all the cities in a tour of a TSP instance as the objective function or, the fitness of the chromosome [44]. Each chromosome is constructed out of a set of genes such that each gene within the instance forms part of every chromosome, but cannot be allowed to appear in the same chromosome more than once as follows:

- $X_i \in C, 1 \leq i \leq n$
- $X_i \neq X_j$ when $i \neq j$ for $1 \leq i, j \leq n$

- $X_i = X_j$ when $i = j$ for $1 \leq i, j \leq n$

given that $C = \{1, 2, ..., n\}$ is the set of cities to be visited, X is a unique chromosome and n is the number of cities per instance. After each chromosome has been randomly populated by genes, the fitness function for a chromosome for instance chromosome X can be determined by:

$$f(X) = \sum_{i=1}^{n-1} D(X_i, X_{i+1}) + D(X_n, X_1) \tag{3.3}$$

where $D(X_i, X_j)$ is the distance between city X_i and X_j and $D(X_i, X_j) = D(X_j, X_i)$. After the random initialization has been completed, the selection process starts. The selection process is a greedy process as it mimics the "Survival of the fittest" [59] Darwinian approach by selecting chromosomes based on their fitness. Assume that p is the population scale or number of chromosomes within a generation G, such that $G = R_1, R_2, \ldots, R_p$, where R_1, R_2 and R_p can be defined as unique chromosomes X, Y, Z, such that $R_1 = X_1, X_2, \ldots, X_n$, $R_2 = Y_1, Y_2, \ldots, Y_n$ and $R_p = Z_1, Z_2, \ldots, Z_n$. The max fitness within each chromosome can then be determined:

$$M(R_i) = \max(f(R_1, f(R_2), \ldots, f(R_p)) + 0.1 - f(R_i), \quad 1 \leq i \leq p \tag{3.4}$$

A random variable v can then be uniformly generated as an upper limit for selection from:

$$0 \leq v \leq \sum_{j=1}^{k} M(R_j) \tag{3.5}$$

The local search is then done by solving for R_q as follows:

$$q = \min(k | \forall k \in \{1, 2, \ldots, p\}, v \leq \sum_{j=1}^{k} M(R_j)) \tag{3.6}$$

After the selection process has been completed, the crossover process can be started. This is done by selecting the two chromosomes which can be referred to as A and B. Two intersection points, i and j are then chosen so that $i < j < n$. To start the crossover process, a new temporary chromosome C, is introduced by replicating the values from B, such that $C = B$. All the genes that are shared between $(A_i, \ldots, A_j)$ and C are then removed for C after which a new chromosome E can be produced as follows:

$$E_m = \begin{cases} C_m, & 1 \leq m < i \\ A_m, & i \leq m \leq j \\ C_{m-j+i-1}, & j < m \leq n \end{cases} \tag{3.7}$$

A demonstration of this process can be shown in Table The same process can then be repeated where D is introduced as a copy of A where all the genes between $(B_i, \ldots, B_j)$ are removed from D after which F can be created where:

$$F_m = \begin{cases} D_m, & 1 \leq m < i \\ B_m, & i \leq m \leq j \\ D_{m-j+i-1}, & j < m \leq n \end{cases} \tag{3.8}$$

An example of the Crossover process can be seen in Table. 3.1, where chromosome A and B are used to form chromosome E. Chromosome C is created as $C = B = \{2, 4, 6, 8, 1, 3, 5, 7\}$. Since "i = 4" and "j = 6", the values from $\{A4, A5, A6\}$ which are $\{4, 5, 6\}$ are then removed from C to form $C = \{2, 8, 1, 3, 7\}$ after which Eq. (3.7) can be used to calculate E.

$i = 4$	$j = 6$
Chromosome	Genes
A	1,2,3,4,5,6,7,8
B	2,4,6,8,1,3,5,7
C	2,8,1,3,7
E	2,8,1,4,5,6,3,7

Table 3.1: Crossover Example

The final step for in the GA for every iteration is the mutation operation. Mutation simply involves randomly swapping the positions of two genes in a single chromosome, for example mutating $i = 2$ and $j = 3$ for $A = \{A_1, A_2, A_3, A_4, \ldots, A_n\}$ creates $B = \{A_1, A_3, A_2, A_4, \ldots, A_n\}$.

3.3 Ant Colony Optimisation

The ACO method models the travelling path of each ant within a colony. During each iteration, each ant is tasked to do a few different tasks. The ant needs to find its way to each city, or location of a piece of food, within the map. The k-th ant determines the probabilities of the points around it by:

$$p_{ij}^k(t) = \frac{|\tau_{ij}(t)|^\alpha . |\eta_{ij}|^\beta}{\Sigma_{l \epsilon N_i^k |\tau_{il}(t)|^\alpha . |\eta_{il}|^\beta}} \text{ if } j \epsilon N_i^k \tag{3.9}$$

given that $p_{ij}^k(t)$ is the probability of moving from its current position at i to position j, $\tau_{ij}(t)$ is the amount of pheromones on the path between city i and city j, with α

as the control factor to moderate the importance of the pheromones in the decision process, η_{ij} is the visibility or inverse distance between the current city and city j with β acting as a control variable to determine the importance of visibility within the system. Lastly l is all the points remaining in the neighbourhood (N_j^k) which have not yet been visited by the k-th ant. Mathematically we can determine that if $\alpha = 0$, only the η_{ij} will be used to determine the probability of the next city to visit, favouring nearby points. In the same way, we can see that setting $\beta = 0$ will result in only the $\tau_{ij}(t)$ being considered as a way for the ant to explore its new location. This however is based on the current pheromone levels and as such leads to stagnation since the ant is likely to just continue on the path it has been for previous iterations [60].

After each ant has completed its tour, the colony updates the amount of pheromones located between each city by:

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{3.10}$$

where $\Delta\tau_{ij}^k(t)$ is related only to the trail that ant k has walked during the current iteration, represented by

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{q}{L^k}, & \text{if arc}(i,j) \text{ is part of the tour for ant } k \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

where L^k is the length of the tour traversed by the k-th ant k, q is a control value that moderates the amount of pheromones that can be added to the colony by each ant and ρ is a control element such that $0 < \rho < 1$ to determine the amount of memory in the system after each iteration. Evaporating pheromones at a rate where $\rho = 1$ the algorithm will favour only paths walked during the previous iteration, whereas a ρ value close to zero leads to a system that will still favour paths that might no longer be relevant as ρ can be used to filter bad data out of the system to favour newer paths. After the colony has finished updating its pheromone levels, the ants restart the process of constructing their next tour, repeating the process until the final itertation has been reached [61].

The number of ants used to construct tours for each iteration are normally equal to the amount of cities in the TSP map, so that each ant starts on a different city, however the number of ants can be adjusted as a control element and then randomly assigned to different points on the map. After each iteration the tour length of each ant is compared to the current minimum tour length. At the end of the final iteration, the shortest overall tour length is considered to be the sub-optimal value for the ACO.

3.4 Iterative Discrete Flower Pollination Algorithm

The iDFPA developed by Strange, Yang and Cheng, [22], describes the formulation of a meta-heuristic algorithm used to solve the discrete space NP-Hard Traveling Salesman Problem. Since the FPA is used to solve for problems in continuous space, it was adapted and combined with the ACO algorithm to form the DFPA. DFPA hybridises the ACO algorithm and the FPA by making use of the local and global search elements from the FPA and the multi-agent elements from the ACO. The algorithm requires a distance matrix D to be constructed as an $n \times n$ matrix where n is the number of nodes for the given problem. The D matrix is then populated with a diagonal of zeroes and distance values d_{ij} to represent the distance between the i-th and j-th nodes. For a symmetric TSP the distance from node i to node j will always be the same as the distance from node j to node i. An $n \times n$ cost matrix C then produces an inverse of the distance matrix to determine the probability for each path. The algorithm uses m agents simultaneously to find the best solution. It does this by forming a Markov chain by using probabilities determined from the current node. The algorithm also makes use of a switching probability $\rho \in (0, 1)$ to switch between local and global search. Each m agent adds its tour to the set of tours $S = \{s_1, s_2, \cdots, s_m\}$ where $s_i = \{s_i^1, s_i^2, \cdots, s_i^n\}$ is an ordered sequence tour which is a permutation of $V = \{v_1, v_2, \cdots, v_n\}$. This enables the use of $V' = V$ where V' is a temporary set to represent the nodes still left within the tour. Next, randomly select s_i^1 from the list of nodes in V' and set $v_{prev} = s_i^1$ and remove it from V so that $V' = V' - \{s_i^1\}$. To determine the rest of the tour, first generate the normalization constant $A = \Sigma_{\forall v_k \in V'}\ c_i k$. The probability of each node in V' is then given by

$$\rho_{ij} = \frac{c_{ij}}{A} \tag{3.12}$$

A random number $r \sim U(0, 1)$ is generated. If $r > \rho$ then the global search algorithm is used to determine s_i^j else the local search algorithm is used. Finally, the j-th element is removed from the temporary tour variable so that $V' = V' - \{s_i^j\}$ and $v_{prev} = s_i^j$ after which the j variable is incremented to get the next element in the tour. Once all the tours in S have been determined, the minimum distance is chosen as the solution. For local search within DFPA, a search subset $V'' = \{v | d(v, v_{prev}) \leq r_{dist}, v \in V'\}$ can be used, where V'' is the ordered probability set of V'. The global search is used if there are no more nodes in the radial cluster. The global search makes use of a discrete Lévy distribution function $L(\bar{V})$ which is defined as

$$f(v_i; \bar{V}) = Pr(v_i) \tag{3.13}$$

$$f(v_i; \bar{V}) = F_L(\Phi\Sigma_{k=1}^{i} p_k) - F_L(\Phi\Sigma_{k=1}^{i-1} p_k) \tag{3.14}$$

where F_L is the Lévy CDF, p_k is the probability cost to move towards node k and Φ is a constant. After the discrete Lévy distribution is applied to the node $L(\bar{V})$, the next node $s_i^j \sim L(\bar{V})$ is generated using

$$l^* = \min\{l \in \{1, 2, \cdots, |\bar{V}|\} : (\Sigma_{i=1}^{l} p_i) - r \geq 0\} \tag{3.15}$$

Now that the DFPA can escape local optima with the FPA and can incorporate multiple agents, it is able to focus both on exploration and exploitation. The algorithm can still be improved however, by introducing memory into the system. The iterative Discrete Flower Pollination Algorithm (iDFPA) was created by adding both the best tour update and the rejection update to the DFPA process. This adds some extra complexity to the algorithm, but further increases the convergence rate [62]. The first element that is adjusted in the system is the cost matrix. The new cost matrix C^t represents the cost of the cost c_{ij}^t to travel between each node v_i and v_j during the t-th iteration of the process. Evaporation follows the formula

$$c_{ij}^{t+1} = (1 - \alpha)c_{ij}^t \tag{3.16}$$

where α controls the rate of evaporation. This is to allow for bad memory to be filtered out of the system. This then allows for use of the filtering processes known as the Best Tour Update (BTU) [63]. The best tour update allows for the cost matrix to be updated, but only if the path was travelled on the minimum distance tour s^*. The cost matrix is then updated as follows if arc(i, j) is part of s^*

$$c_{ij}^{t+1} = c_{ij}^t + \gamma \frac{d_{ij}}{d_{s^*}} \tag{3.17}$$

where the constant γ controls the magnitude of the best tour update. The last part of the iDFPA is the Rejection Update (RU). The RU also utilises the tour distance of the iDFPA. If a new tour distance is introduced, it is automatically added to the list of accepted tours $S' = s_1^{'}, s_2^{'}, \cdots$. If the tour is not shorter than the minimum tour, exponential annealing is introduced by

$$T_{curr} = e^{\frac{-\omega N_{curr}}{qN}} \tag{3.18}$$

where ω is a constant that controls the utilised region of the exponential function and q controls its shape. N_{curr} is the current iteration number and T_{curr} is the annealing

values corresponding to the current iteration. Tour distances that are longer than the shortest tour distance then have a probability of being accepted of

$$\rho = e^{\frac{-\Delta dist}{T_k \times d_{prev}}} \tag{3.19}$$

where $\Delta dist = d_{sj}$–d_{prev}. A random number $r \sim U(0,1)$ is then compared to ρ. If $\rho > r$ then the solution is accepted otherwise it is rejected. The rejection cost update is then shown as follows:

$$c_{ij}^{t+1} = c_{ij}^{t} + \beta \Sigma_{\forall s'_k} \in s' \frac{1}{d'_k} \tag{3.20}$$

where $d'_k \in d'$ and β is a constant that controls the magnitude of the Rejection Update on the cost matrix.

3.5 Parallelisation Methods

All the processes investigated above made use of single processor, sequential methods to solve for NP-hard problems. In [64] Randall investigated different parallel ACO architectures and compared the efficiency and speedup of a parallelised ACO compared to a serial implementation on the same computer. Pedemonte [65] furthered Randall's work by setting up a clearer classification of each model of common parallel methods along with classifying various case studies and research papers into five distinct parallel configurations. The most common of these configurations are the Master Slave model, the Parallel Independent Runs model, and the Multi-colony model.

3.5.1 Master Slave Model

The Master Slave model is one of the most common parallelisation methods found for introducing parallel strategies due to the simplicity in which the process can be setup. The model is presented as a system where the master processor stores and manages the pheromone matrix, best tour results and control variables. When required, the master processor sends all this data to its slave processors to perform iterations simultaneously after which the result is communicated back up to the master processor. One of the earliest implementations of the master slave system was performed by Talabi [66] where it was able to increase the rate of convergence of

the Tabu search over the same amount of iterations, whilst having a similar run time to the serial process. This did however not account for any efficiency metrics. Ibri [67], and Doerner [68] both indicated a sub-linear speedup and efficiency lower than 1 for the master slave system. This is due to the communication overhead of needing to communicate large amounts of data between master processing units and slave processing units. This process is presented in Fig. 3.3.

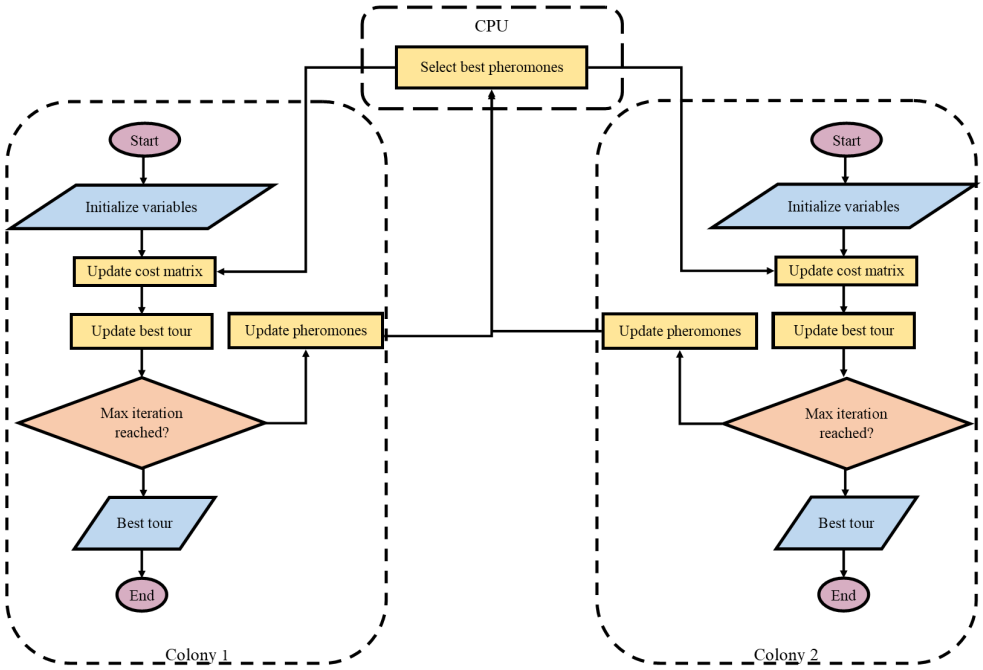


Fig. 3.3: Master Slave model

3.5.2 Parallel Independent Model

This method tries to minimise the overhead costs of communication which it achieves by allowing each slave processor to manage its own pheromone matrix, while the master processor communicates the best tour to each processor after a tour has been completed. Stützle [69] and Alba [70] both demonstrated much higher levels of efficiency by using this method in comparison to the mater-slave configuration whilst still being able to outperform the sequential operation in terms of accuracy, though the performance in terms of accuracy and convergence is not as high as other parallelisaton methods. This method is illustrated in Fig. 3.4 and shows how the colonies run on their own and only need to communicate with the CPU once they

have found the best tour for their iterations, cutting down on massive overheads in communication.

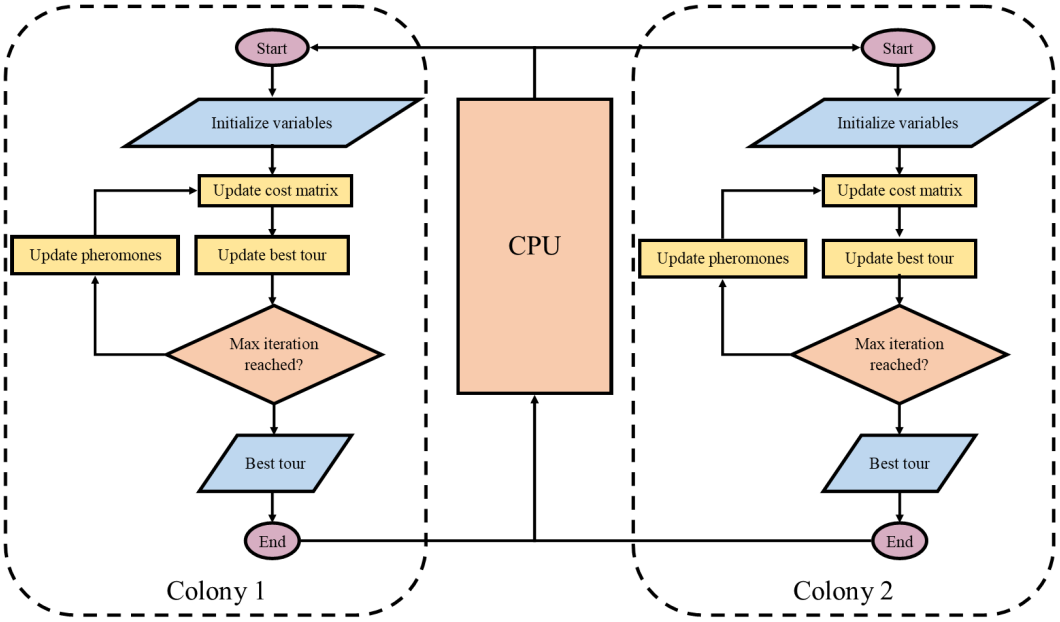


Fig. 3.4: Parallel Independent model

3.5.3 Multi-Colony Model

Multi-colony model operates by distributing the memory of an ant colony over a range of ants. The ants no longer need to communicate with a centralised processor, but are rather able to exchange data in a distributed system. This process was implemented by Chen for a the TSP [71] solution and Lucka for the Vehicle Routing Problem (VRP) [72], with both papers reporting a speedup and accuracy that outperformed the sequential results with a non-linear speedup and high efficiency. This process can be seen in Fig. 3.5, where the blue background represents both cores working on the same memory withing the GPU. The GPU does not need to fetch data once the process has been initialised, meaning that communication overheads are at a minimum.

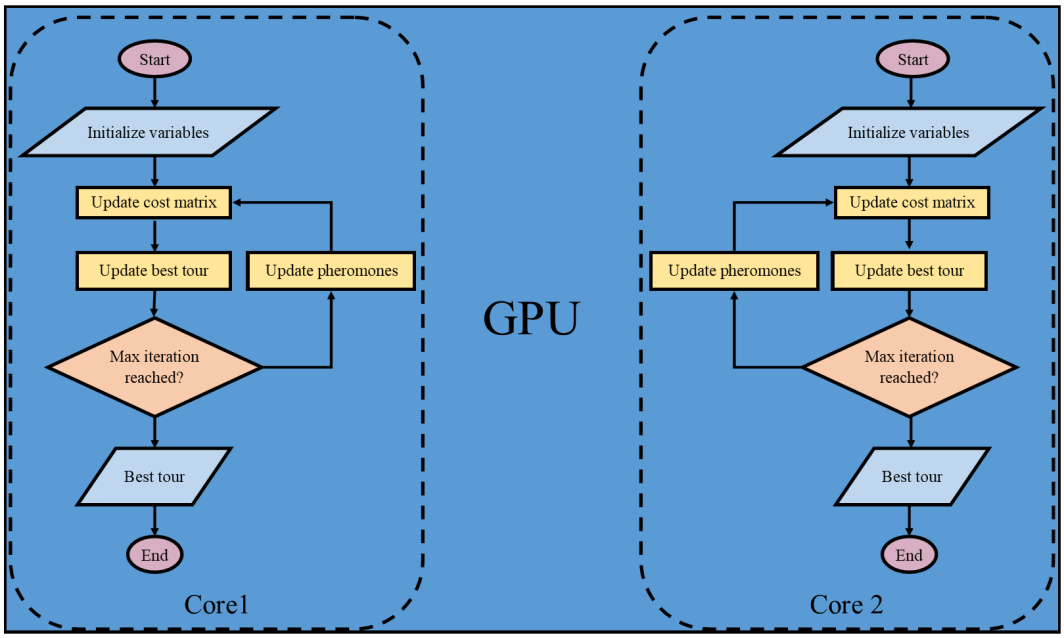


Fig. 3.5: Multi-colony model

3.6 Conclusion

Preliminaries served as a way to introduce current solutions within the framework of the Flower Pollination Algorithm to provide the building blocks that are used in constructing the PiDFPA. This was done by investigating current implementations of the iDPFA and common parallelisation techniques used by algorithms to solve the same set of problems.

Chapter 4

Parallel Iterative Discrete Flower Pollination Algorithm for Solving Symmetric Travelling Salesman Problem

4.1 CUDA

A brief introducing to the CUDA architecture is provided before the implementation specifics are provided to introduce the main structure of the CUDA system. The CUDA system can be summarised in terms of threads, memory and synchronization.

4.1.1 Threads

Each thread in the CUDA system can be seen as a processor. Multiple threads are combined to create a block which are grouped together into grids. The number of threads in a block can be set when initialising the CUDA kernel, but the amount is limited by the GPU's architecture. Each block contains its own set of memory that cannot be accessed by threads within another block. Threads within the same block are also able to interact with each other but cannot communicate with threads in different blocks.

4.1.2 Device Memory

There are three main types of memory accessible by threads within the CUDA system. The first type is global memory. This is memory that is stored within the RAM or CPU cache of the computer that is using the CUDA GPU. This memory is extremely slow to access in terms of CUDA execution time but normally has much more storage capacity than any other form of memory accessible by the CUDA threads. The next level of memory is shared memory which is the memory that is allocated to each block. As stated above, shared memory is localised to each block and threads can only access and edit shared memory within their own block. Lastly register memory is memory that is only accessible by the thread itself. This memory is used to store information that the thread requires to do calculations and is extremely limited.

4.1.3 Synchronization

Each block within a CUDA grid is executed at the same time, after which the next grid is executed. That means that grids need to wait for every thread in the previous grid to be terminated before it can start. Blocks in the same grid however are all executed at the same time so that each block can execute its instructions asynchronously from other blocks within the same grid.

4.2 CUDA implementation

To implement the Multi-colony model, the CUDA programming library, illustrated in Fig. 4.1, was used. The initial code starts from the Host(CPU) which sends instructions down to the Device(GPU) by making use of a kernel. This kernel carries instructions to divide the Device into different grid sizes and blocks. Each grid on the device manages a number of blocks and threads. The more blocks used per grid, the less memory is shared per grid. This results in a trade off between GPU memory and the amount of processors used for larger maps.

Within each block, each processor is allocated to a thread number to give it a unique identity. The kernel allows the host to send the cost matrix to the device to store the data in its shared memory to allows the GPU to execute all tours without needing to fetch data from the CPU memory or memory stored within RAM. Once the next

block within the sequence starts, the previous block's memory is lost. Only data stored within shared memory is able to be accessed by the next block.

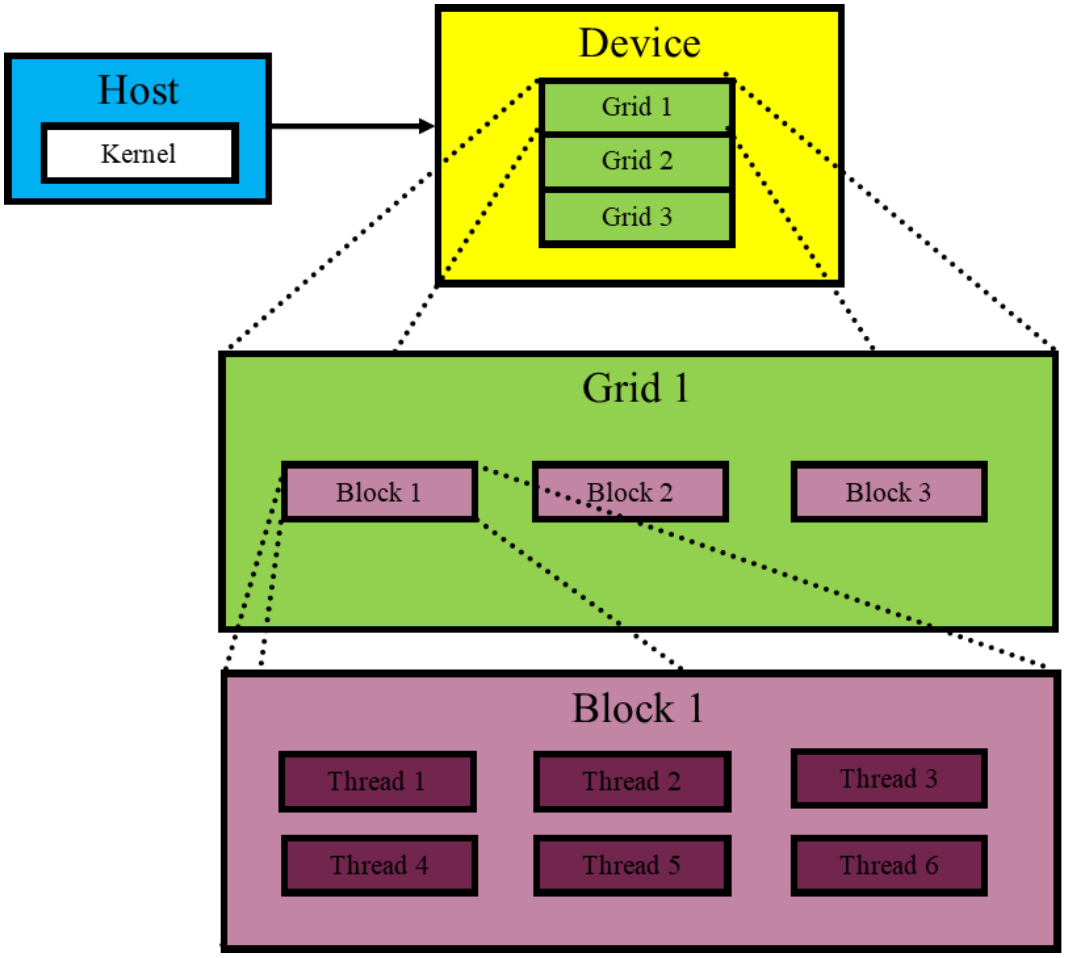


Fig. 4.1: CUDA GPU interaction

4.3 Experimental Setup

Algorithm 1 provides a high level overview of the the PiDFPA. The Host(CPU) imports the TSP map and initialises the control variables to generate the cost matrix. The cost matrix is then stored in shared memory within the device where it can be accessed by the threads. The Host then runs a set of device calls during each iteration to allow the Device to update the cost matrix within the shared memory, to calculate the tours for all the agents assigned to the problem, and then to adjust the pheromone values within the device memory based on the acquired tours. None of the memory is explicitly sent back to the CPU as everything is initialised at the start

of the program to enable the threads to easily access all the information it needs to complete its current function.

Algorithm 2 provides a high level overview of the global and local search that each agent uses to determine how it should traverse the TSP map.

Algorithm 1 Host code

```
1: Import TSP Map coordinates
2: Calculate initial cost matrix
3: Set Shared Memory
4: while iterations ≤ maxIterations do
5:     Call device to update cost matrix in shared memory from pheromones
6:     Call device to calculate tours for current iteration
7:     Call device to determine best tour for current iteration
8:     Call device to perform best tour and rejection update for current iteration
9:     Call device to adjust pheromone values
10: end while
```

Algorithm 2 Calculate best

```
1: Select random starting city
2: while currentPathSize ≤ numberOfCities do
3:     calculate current probability matrix from pheromone and cost matrix for
   current city
4:     sort probability matrix
5:     if ρ ≤ generatedprobability then
6:         Global search
7:     else
8:         Local search
9:     end if
10:    store selected city
11: end while
```

The process of these algorithms are combined in Fig. 4.2 to demonstrate how the host and device interact with each other.

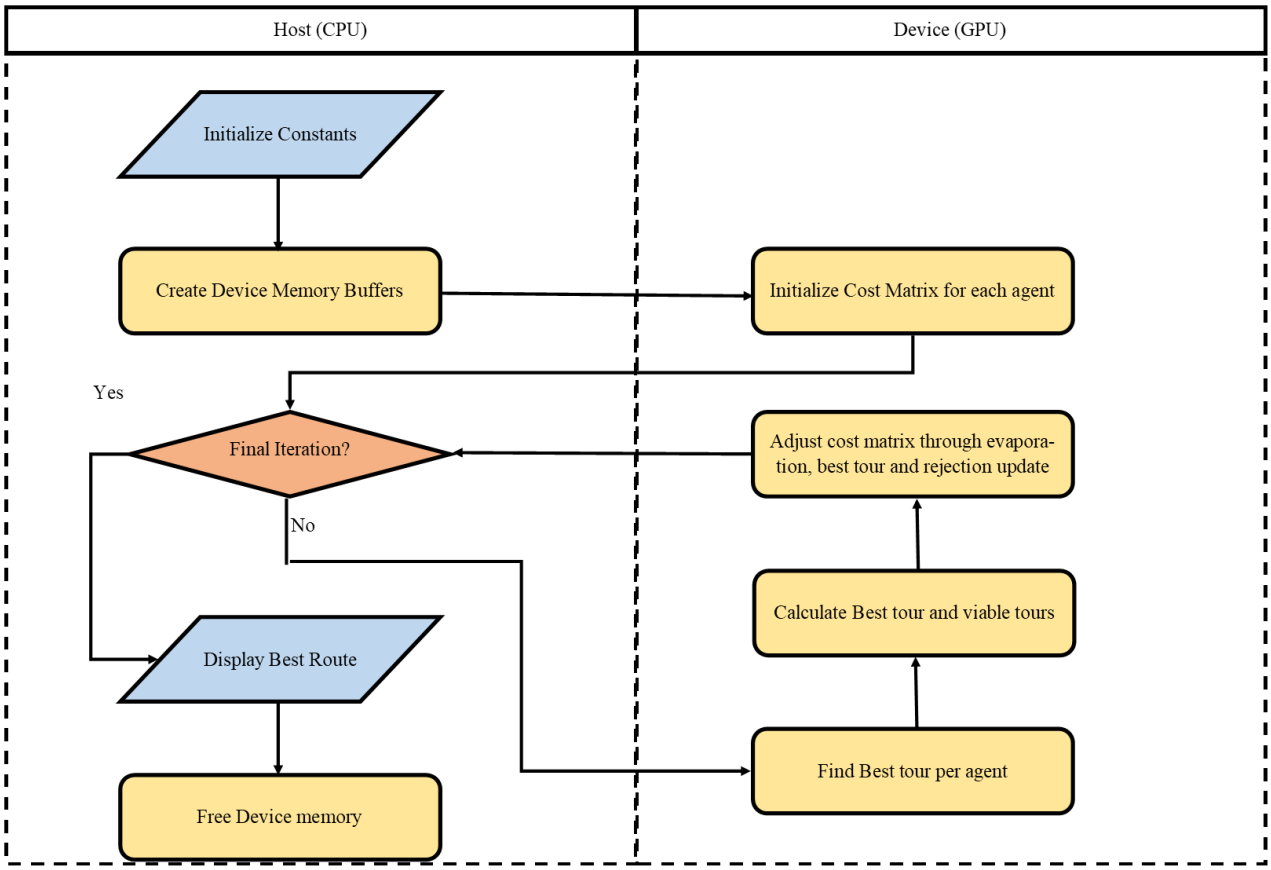


Fig. 4.2: CPU vs GPU split

4.4 Conclusion

This chapter introduced a high level overview of the PiDFPA by presenting the framework used to implement the code to adjust the iDFPA according to the Multi-colony model. This is done by allowing the iDFPA process to be distributed among a large number of processors where it can make use of its shared memory to enable multiple agents to solve the same problem simultaneously.

Chapter 5

Results

The TSPLIB open source library will be used in all comparisons as it provides a large set of maps of varying sizes that have all been optimised based on the specifications of the Symmetric TSP and as such is an ideal basis for benchmarking the accuracy of new meta-heuristics. The first performance metric to be assessed is therefore the accuracy of the sub-optimal solutions achieved by the PiDFPA against three differing map sizes. To fairly assess the PiDFPA, it is tested alongside other common parallel nature inspired meta-heuristics namely the PACO and PGA implementations. By matching the parallel architecture and optimising the control variables for each of the benchmarking algorithms, it would be possible to determine the trade-offs between these parallel meta-heuristics over a range of TSPLIB problems. The second performance metric that can be assessed is the rate of convergence of the PiDFPA, this deals with the efficiency of the meta-heuristics and will be compared to the ACO method by evaluating the time each meta-heuristic algorithm requires to perform the number of iterations determined by the window of the sub-optimal solution. The final performance metric to be tested is the comparison of the speedup that can be achieved by the PiDFPA against the serial iDFPA. This is achieved by comparing the time each algorithm requires to complete the amount of iterations for a given problem.

5.1 Control Variable Adjustments

Due to the nature of the size of the problems being investigated, the ρ value is set to be 0.1 in order to more heavily favour global search to increase the initial rate of convergence. The remaining control variables are then determined by running the PiDFPA on each map with a range of values starting from low to high for each control

TSP Map	PiDPFA	PACO	PGA
Berlin52	$m = 52$ $\rho = 0.1$ $\alpha = 0.2$ $\beta = 0$ $\gamma = 0.1$	$m = 52$ $\alpha = 0.65$ $\beta = 5$ $\rho = 0.5$	$m = 52$
Att532	$m = 532$ $\rho = 0.1$ $\alpha = 0.05$ $\beta = 0.5$ $\gamma = 0$	$m = 532$ $\alpha = 0.65$ $\beta = 9$ $\rho = 0.5$	$m = 532$
u1060	$m = 1060$ $\rho = 0.1$ $\alpha = 0.01$ $\beta = 0.5$ $\gamma = 0$	$m = 1060$ $\alpha = 0.9$ $\beta = 9$ $\rho = 0.5$	$m = 1060$

Table 5.1: Testing parameters used to find best and average solutions for TSPLIB problems

variable. The control variable combinations that show the most promising result can then be used to optimise the values to be used. Once a promising combination is found, two of its control variables are set to be constant while the last variable is changed to determine the effects of the variable pair. The variables chosen for the final results are shown in Table 5.1.

5.2 Accuracy Comparison

As can be seen in Table 5.2, the PiDFPA performs excellently in terms of accuracy for small problems as it is able to consistently achieve a near-optimal results, averaging a solution that is within 97% of the optimal solution, and with the potential to achieve a completely optimised solution within the 300 iterations. This accuracy is however not afforded to larger maps as the increase of map size shows a tendency for a decrease in accuracy of the PiDFPA. The PiDFPA is however not the only meta-heuristic to face decreased performance as the map size grows larger. The GA has a mixed performance due to its inherent weakness in terms of performing local searches. This is shown in Fig. 5.1, 5.2 and 5.3, where the PGA algorithm quickly converges to a point but then has to slowly adjust to find a better solution.

Lastly the PACO algorithm can be seen to closely match the PiDFPA in terms of accuracy due to both systems involving complex ways to both focus on local and global searches where the PiDFPA has a small advantage in converging to a point.

To ensure that the results shown in the Fig. 5.1, 5.2 and 5.3, were reliable, the algorithms were repeated thirty times and the results were compiled into box charts shown in Fig. 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9. The blue boxes shows values within interquartile range of the average value where the black lines reveal outliers for the values at each point. From this it can be seen that on average, the PiDFPA and PACO still follow the same trend as the best result as the PiDFPA gradually decreases its sub-optimal values where the PACO converges to a point and needs to wait multiple iterations decrease its best tour length.

Problem	Optimal	Best Tour	Error(%)	Avg Tour Error (%)
Berlin52(300 It)	7542	7542	0	-2.74
Att532(300 It)	27686	30769	-11.14	-12.28
U1060(1000 It)	224094	257991	-14.13	-15.74

Table 5.2: Accuracy comparison of PiDFPA against optimal solution

Problem	PGA Best	PiDFPA Best	Diff(%)	Avg Diff(%)
Berlin52(300 It)	8658	7542	12.89	10.49
Att532(300 It)	125761	30769	75.53	75.28
U1060(1000 It)	2776570	257991	90.71	90.65

Table 5.3: Accuracy comparison of PiDFPA against best PGA solution

Problem	PACO Best	PiDFPA Best	Diff(%)	Avg Diff(%)
Berlin52(300 It)	7597	7542	0.72	1.55
Att532(300 It)	31037	30769	0.86	0.53
U1060(1000 It)	263165	257991	1.97	2.42

Table 5.4: Accuracy comparison of PiDFPA against best PACO solution

5.3 Convergence Rate Comparison

The PiDFPA is able to still keep up with both the benchmarking algorithms as seen in Table 5.5 and 5.6. The PiDFPA, PGA and PACO show remarkably similar results

with the PGA being favoured on smaller maps and the PiDFPA showing slightly better results for larger maps against the PACO whilst still staying close to the PGA.

Problem	PiDFPA runtime(ms)	PGA runtime(ms)	Diff(%)
Berlin52(300 It)	11189	10037	-10.29
Att532(300 It)	2669261	2568456	-3.92
U1060(1000 It)	63415107	60143625	-5.43

Table 5.5: Execution time difference between PiDFPA and PGA

Problem	PiDFPA runtime(ms)	PACO runtime(ms)	Diff(%)
Berlin52(300 It)	11189	9915	-12.84
Att532(300 It)	2669261	3071079	13.08
U1060(1000 It)	63415107	65994455	3.9

Table 5.6: Execution time difference between PiDFPA and PACO

5.4 Serial and Parallel

Two of the most common metrics to measure the performance of parallel meta-heuristics, as to compare them to their serial parent meta-heuristics, are speedup and efficiency. Speedup refers to the rate of increase in speed to process the same amount of data in a serial configuration versus a parallel configuration. It is suggested that the same computer is used to run the comparisons between Serial and Parallel meta-heuristics in most cases as the system clock as well as wall clock can be used to ensure that both meta-heuristics can be quantified against the strength of the CPU performing the simulations [73].

To calculate speedup, we can use the following formula:

$$speedup = \frac{\text{Time taken for serial code}}{\text{Time taken for parallel code}} \tag{5.1}$$

One issue discovered and reported by [13], was that inaccurate results were obtained when using average speeds for parallel processing time. As such CPU time should be used to determine the time elapsed for the serial code, whereas a separate timer on a different system should be implemented to record the time for the parallel code, and this should only be calculated once the entirety of the parallel code has been completed. After speedup has been obtained, the efficiency can be determined by

TSP Map	Serial(ms)	Parallel(ms)	Speedup	Efficiency(%)
Berlin52	78	33.8	2.3	1.8
Att532	628942	8946	70.3	54.9
u1060	9706287	62371	155.6	122

Table 5.7: GPU speed and efficiency combination for different TSP maps

using the following formula:

$$efficiency = \frac{Speedup}{\text{Number of Processors}} \tag{5.2}$$

The setup for comparing the serial and parallel time testing was done on a computer with a Nvidia Geforce 1080 TI card, containing 3584 cores with a clock speed of 1.6GHz along with a 6 core CPU running at 3.5GHz. This would mean that it is highly unlikely to expect any form of linear speedup. It should also be taken into consideration that due to shared memory constraints, only 128 of the processors could be used for tests being computed on the GPU.

This is supported by the results in Table 5.7, where it can be seen that the process for computing the PiDFPA can speed up the iDFPA by a factor of 2 up to a factor of 155. This process is however only efficient for large maps as smaller maps tend to bottleneck the resources required to achieve a much faster solution. These factors will be discussed in the next chapter.

5.5 Conclusion

The results indicate that the PiDFPA has the ability to outperform both the PACO and PGA meta-heuristics with regards to accuracy over every map size that it was benchmarked against. It was also shown that the PiDFPA was able to achieve the result without the trade-off of slower run times as the time taken for all three meta-heuristics are within the same range. The results for the PiDFPA were also positive with regards to speedup and efficiency especially in larger maps.

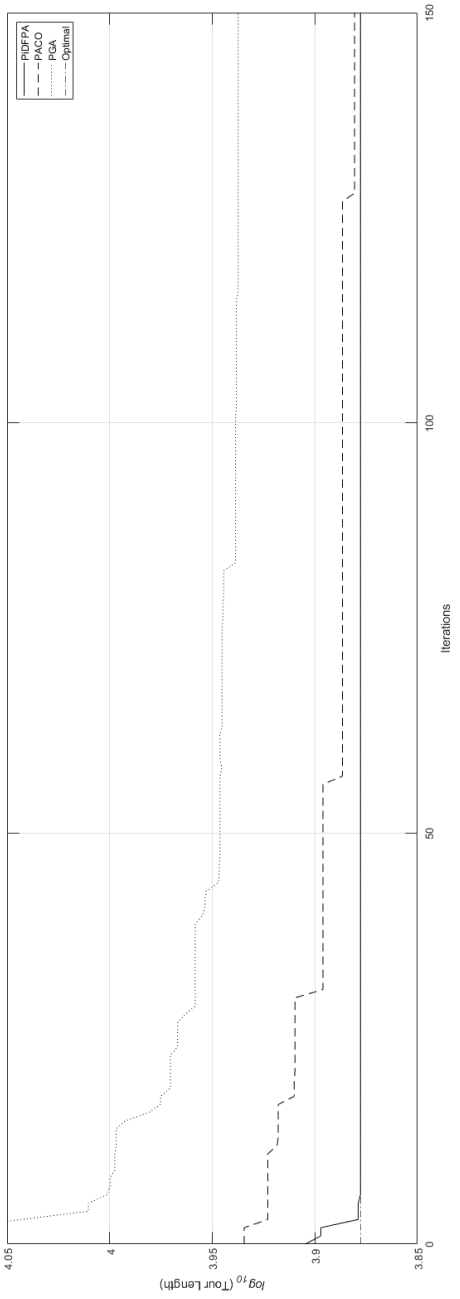


Fig. 5.1: Tour length comparison for PiDFPA, PGA and PACO against optimal tour for the Berlin52 TSP map over 300 iterations

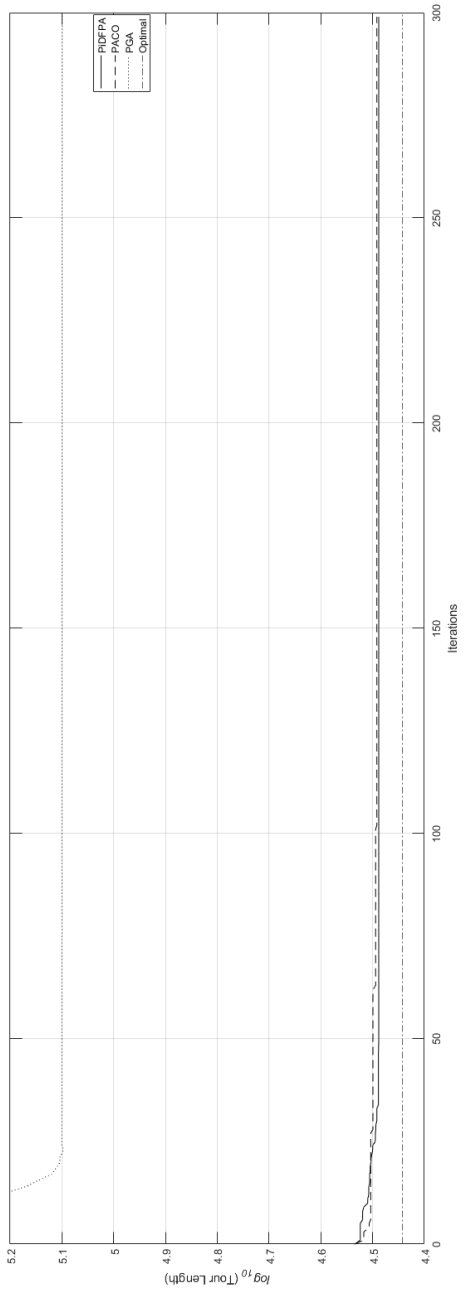


Fig. 5.2: Tour length comparison for PiDFPA, PGA and PACO against optimal tour for the Att532 TSP map over 300 iterations

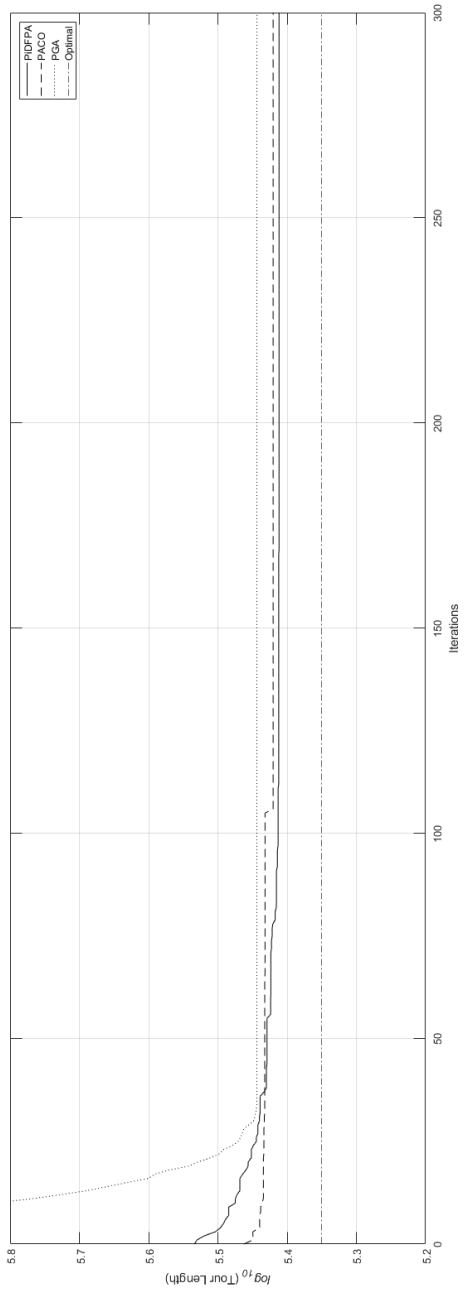


Fig. 5.3: ITour length comparison for PiDFPA, PGA and PACO against optimal tour for the U1060 TSP map over 300 iterations

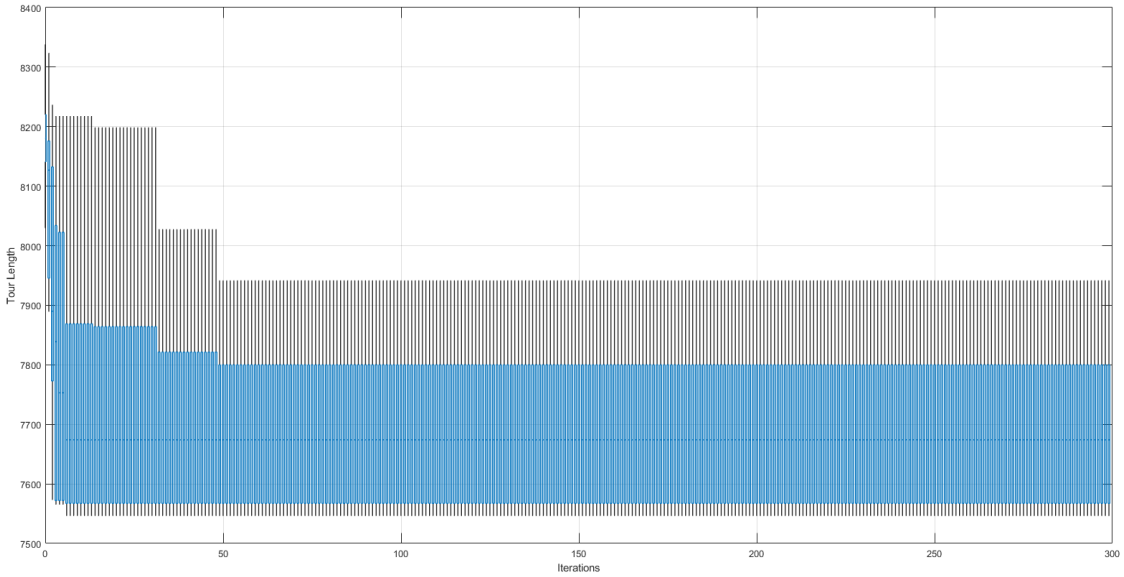


Fig. 5.4: Box chart of the PiDFPA for 300 Iteration execution of the Berlin52 TSP map

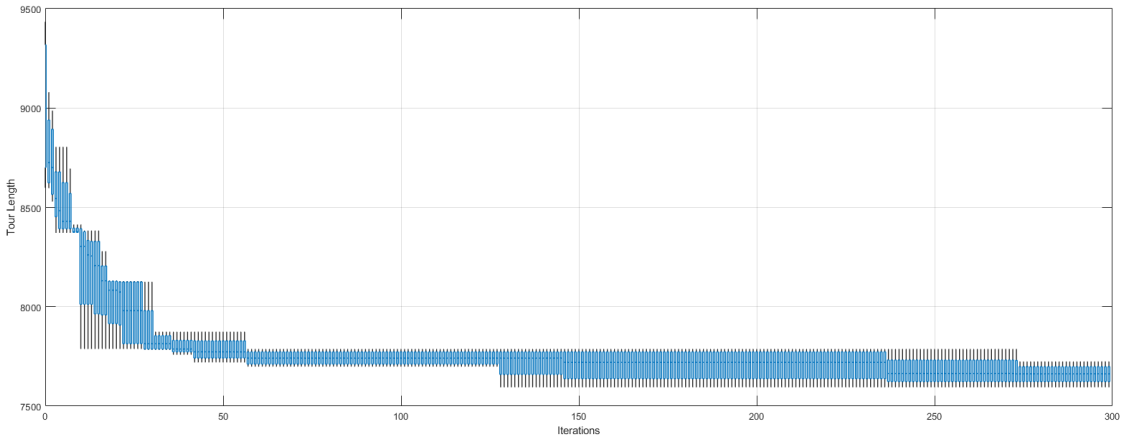


Fig. 5.5: Box chart of the PACO for 300 Iteration execution of the Berlin52 TSP map

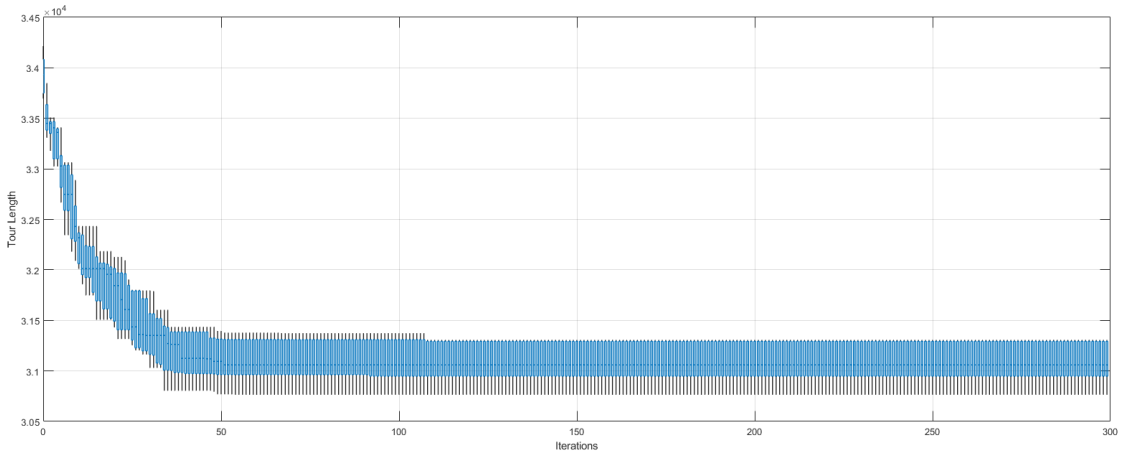


Fig. 5.6: Box chart of the PiDFPA for 300 Iteration execution of the Att532 TSP map

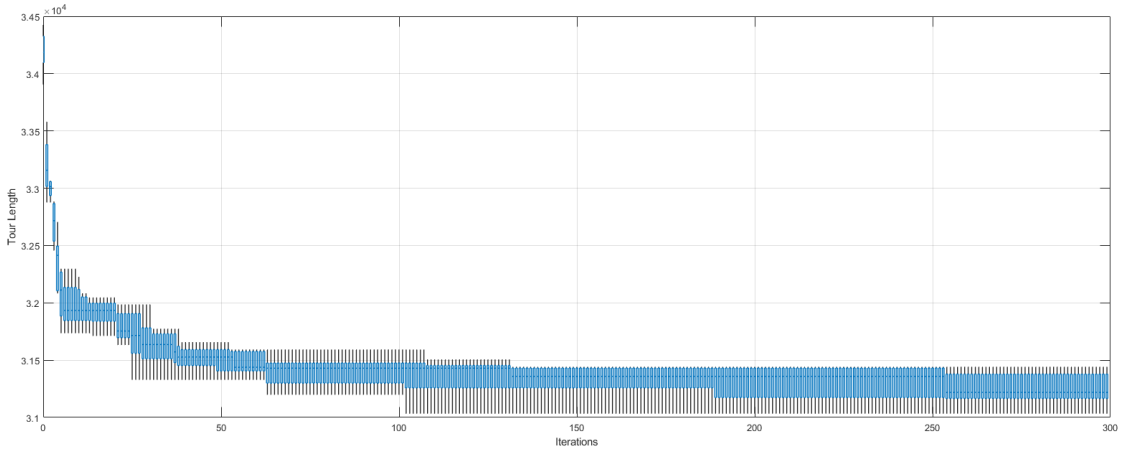


Fig. 5.7: Box chart of the PACO for 300 Iteration execution of the Att532 TSP map

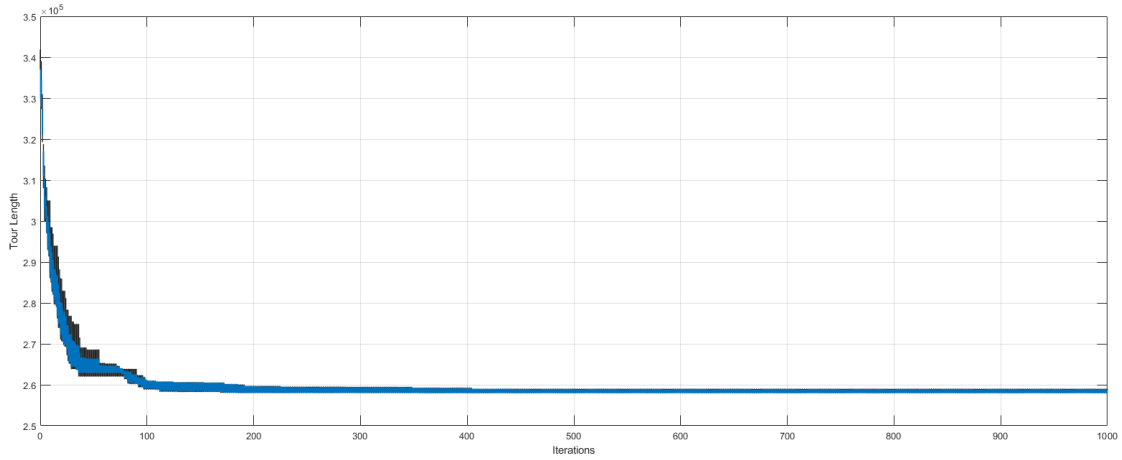


Fig. 5.8: Box chart of the PiDFPA for 1000 Iteration execution of the U1060 TSP map

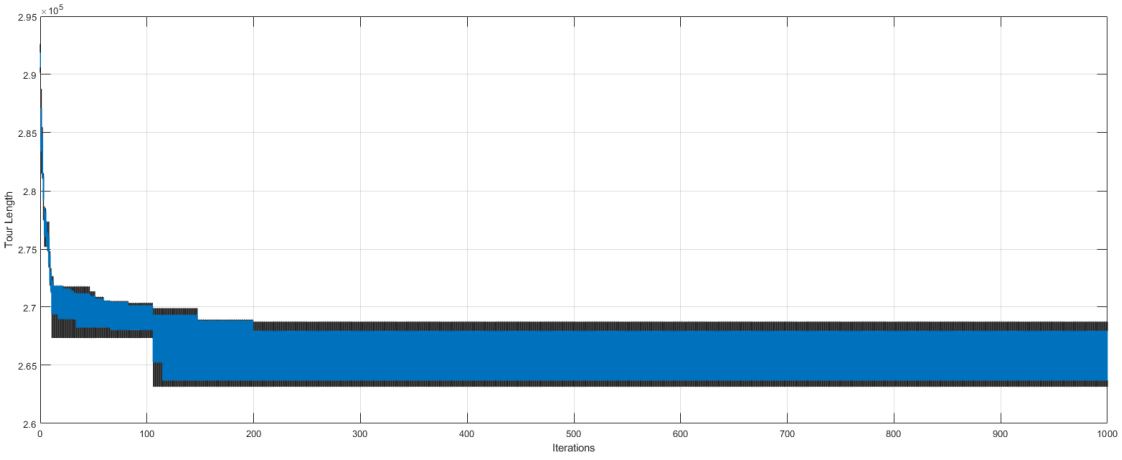


Fig. 5.9: Box chart of the PACO for 1000 Iteration execution of the U1060 TSP map

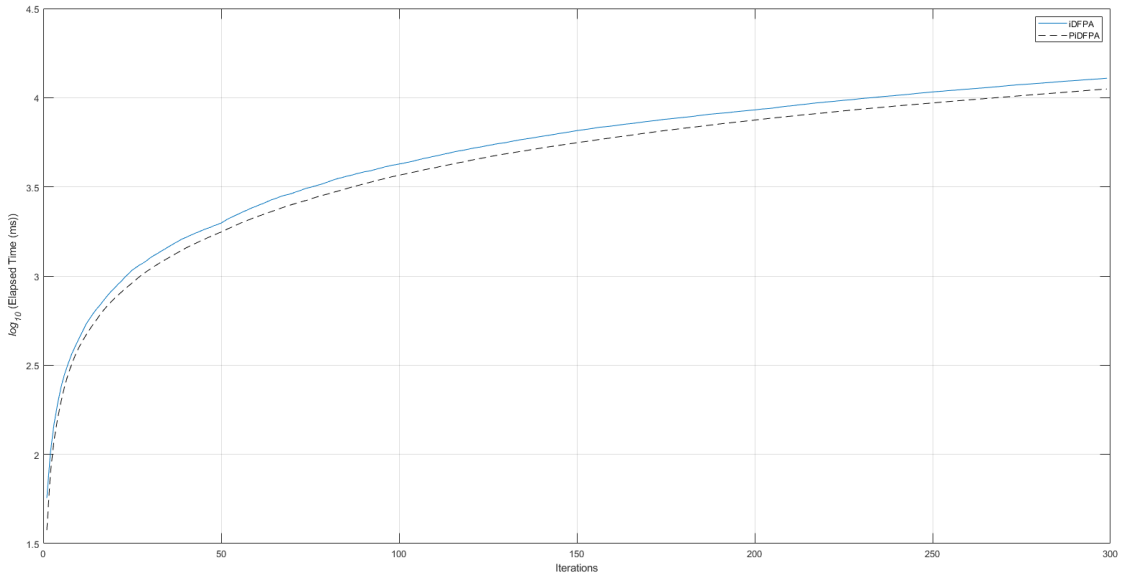


Fig. 5.10: Logarithmic time evaluation of the Berlin52 TSP map by the iDFPA and PiDFPA over 300 iterations

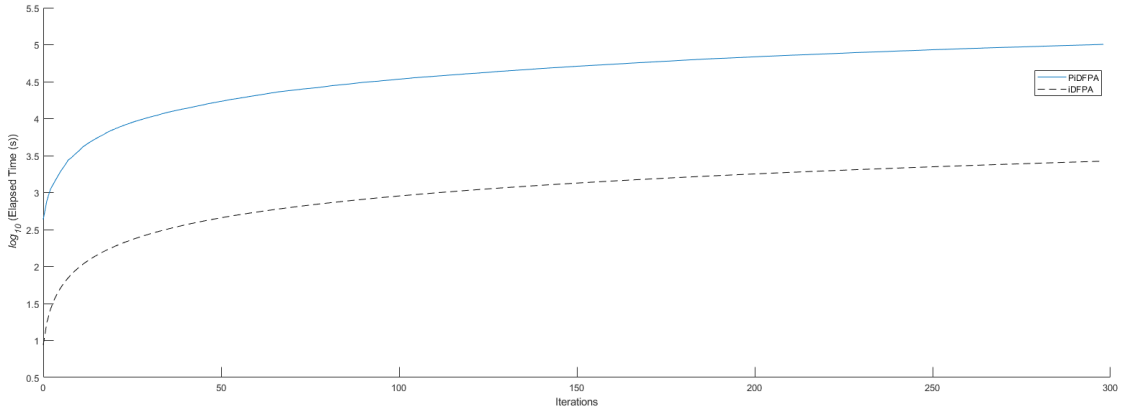


Fig. 5.11: Logarithmic time evaluation of the Att532 TSP map by the iDFPA and PiDFPA over 300 iterations

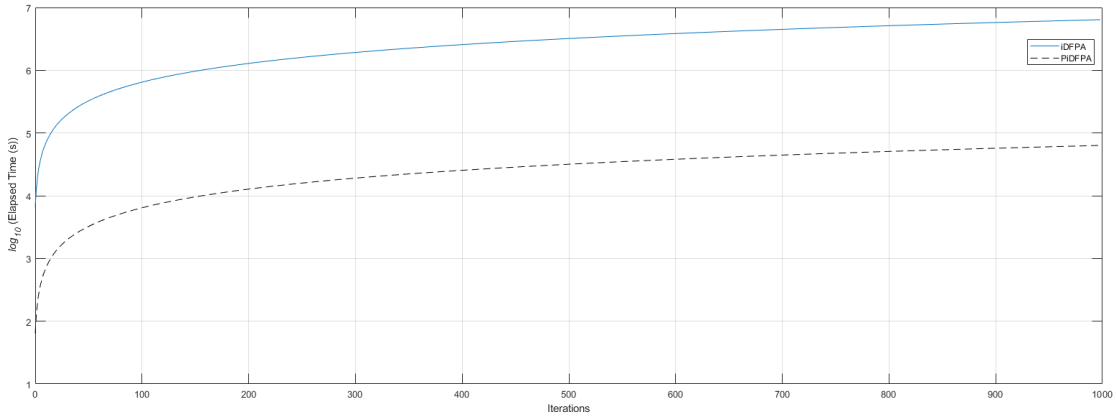


Fig. 5.12: Logarithmic time evaluation of the U1060 TSP map by the iDFPA and PiDFPA over 1000 iterations

Chapter 6

Analysis

As demonstrated in the previous chapter, the PiDFPA has outperformed both the PACO and PGA in most situations over the Berlin52, Att532 and U1060 TSP instances. There is however still a lot of room for improvement both for achieving a more accurate near-optimal solution as well as speed. In this chapter, the three main performance metrics, namely accuracy, convergence rate and efficiency will be discussed to support the results in the previous chapter as well as discussing probable causes of loss in performance in each category.

6.1 Accuracy

The biggest success of the PiDFPA is its ability to demonstrate that the iDFPA methodologies can be applied to a large map and still outperform the most common benchmarking algorithms in terms of accuracy as seen in Fig. 5.1, 5.2 and 5.3. This is partially due to the fact that the PiDFPA is quite versatile at adjusting itself based on the control variables provided. Each of the control variables assessed in Fig. 6.1, 6.2, 6.3, 6.4, 6.5 and 6.6, holds a unique property in terms of steering the FPA algorithm to improve its performance.

6.1.1 Evaporation

One of the sensitive values that can be adjusted for the PiDFPA is the evaporation constant. This can also be shown in the PACO system where the most favoured solution for α was a value that was close to 1. This is due to the PiDFPA and PACO operating in two different ways. As seen in Eq. (3.9), the PACO uses $|\tau_{ij}(t)|^{\alpha}$ in

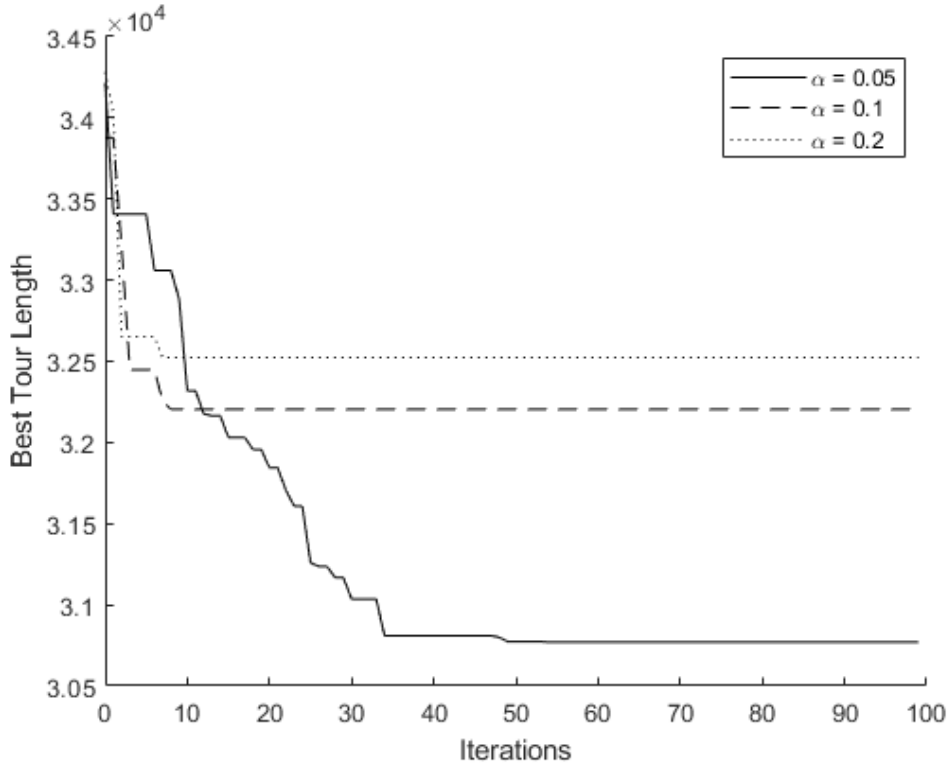


Fig. 6.1: PiDFPA α value adjustment for Att532

order to balance the visibility variable. In the PiDFPA, as shown in Eq. (3.16), $(1-\alpha)c_{ij}^{t}$ is used to directly control the cost function of every city within the instance. Evaporating these instances too early can lead to a loss in global search capability for both the PiDFPA and PACO. Some evaporation is still required in both instances as bad memory is still a problem that can stagnate a system. Although not implemented in this version of the PiDFPA, an adjustable evaporation constant can possibly assist in allowing the PiDFPA to increase the evaporation constant after a few iterations of stagnation before adjusting it down again to allow for new memory to pass into the system.

6.1.2 Rejection Update

The rejection update which is influenced by β is unique compared to the other control variables in the system as it follows a filtering period during which local search results are heavily favoured. This is due to Eq. (3.18) $T_{curr} = \epsilon^{\frac{-\omega N_{curr}}{qN}}$ which allows the Rejection Update to switch modes based on the current iteration of the PiDFPA. Keeping β between zero and one thereby allows the PiDFPA to remain dynamic

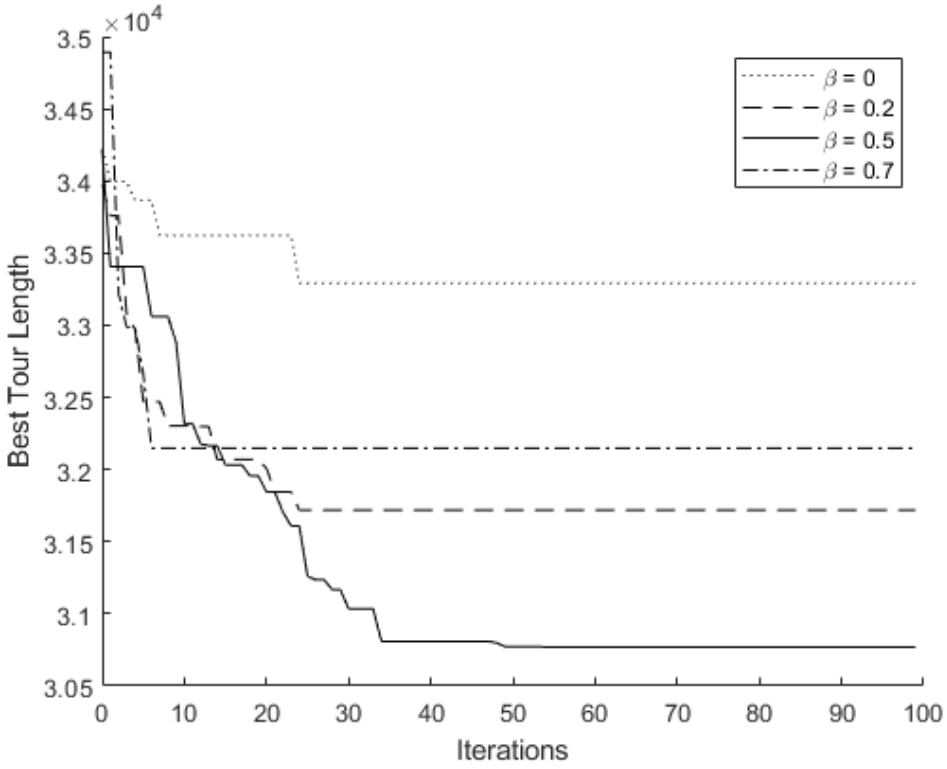


Fig. 6.2: PiDFPA β value adjustment for Att532

whilst not too heavily favouring local searches.

6.1.3 Best Tour Update

The best tour update is controlled by γ which is applied to each best result that has been found per iteration. This is quite dangerous especially early on in the earlier iterations as there is a very low α value assigned to the larger maps when considering PiDFPA. As such gamma is often low or set to zero to prevent a value in a tour from being increased constantly because of old data.

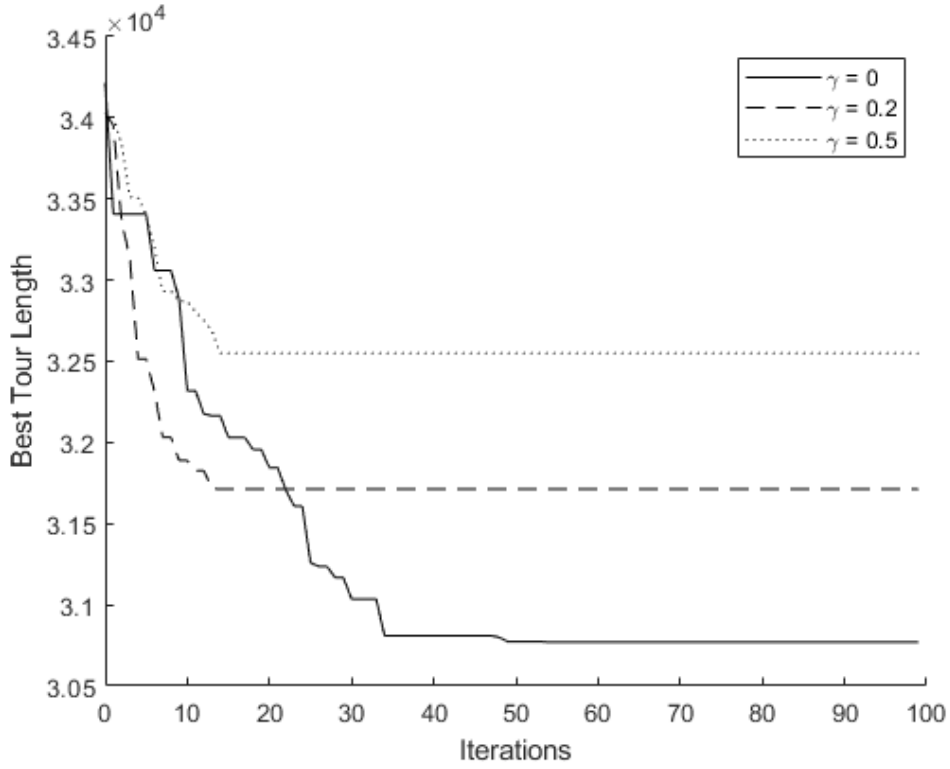


Fig. 6.3: PiDFPA γ value adjustment for Att532

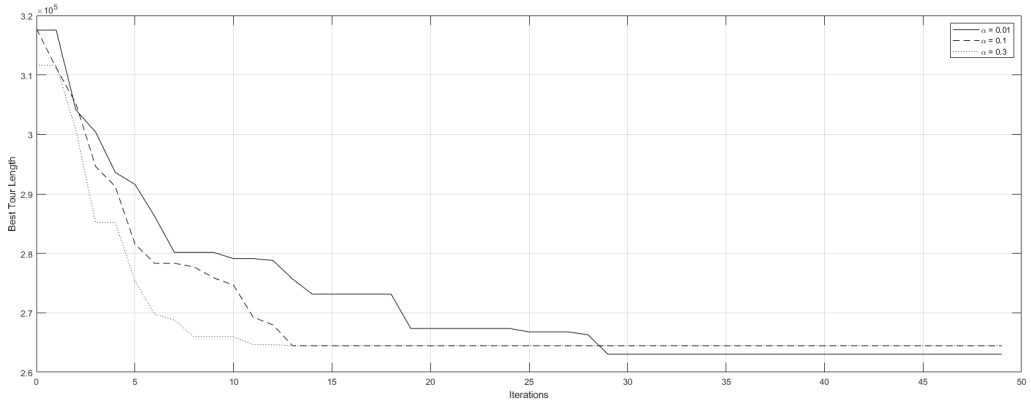


Fig. 6.4: PiDFPA α value adjustment for U1060

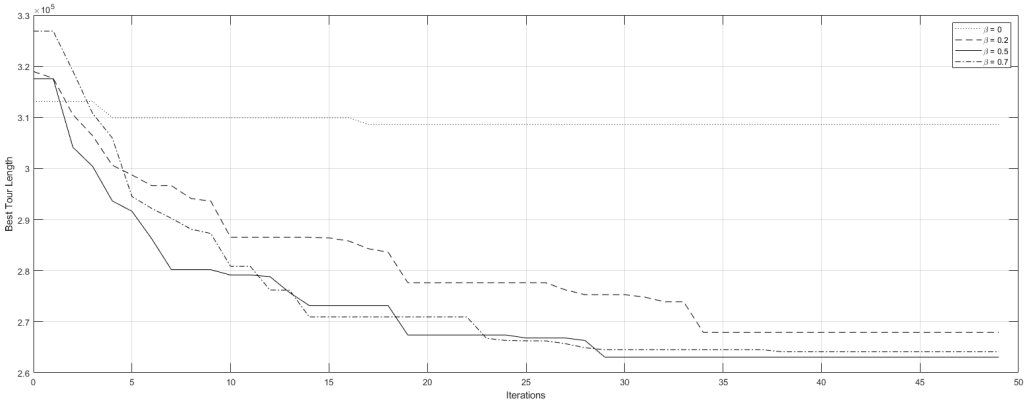


Fig. 6.5: PiDFPA β value adjustment for U1060

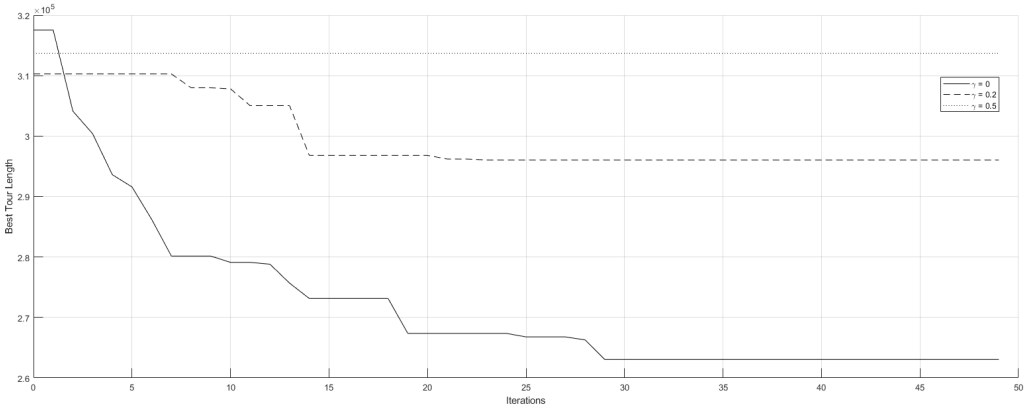


Fig. 6.6: PiDFPA γ value adjustment for U1060

6.2 Rate of Convergence

6.2.1 Reduction in Complexity for Parallel vs Serial

The biggest factor that allows the PiDFPA to be applied to a larger scale map is found in the parallel coding method. Instead of a single processor running through each iteration for each agent, the agents can be divided up into grids. These grids then run each of their agents on a single thread and combine the result when each agent has completed its tour while allocating a shared memory pool to all the threads. The same can be applied in the combination of the best path for each agent. Instead of calculating the tour path for each agent on the CPU, the process can be spread over multiple threads on the GPU allowing for faster calculations of the best tour

and a faster update for the cost matrix as each tour and cost matrix element are calculated separately and simultaneously.

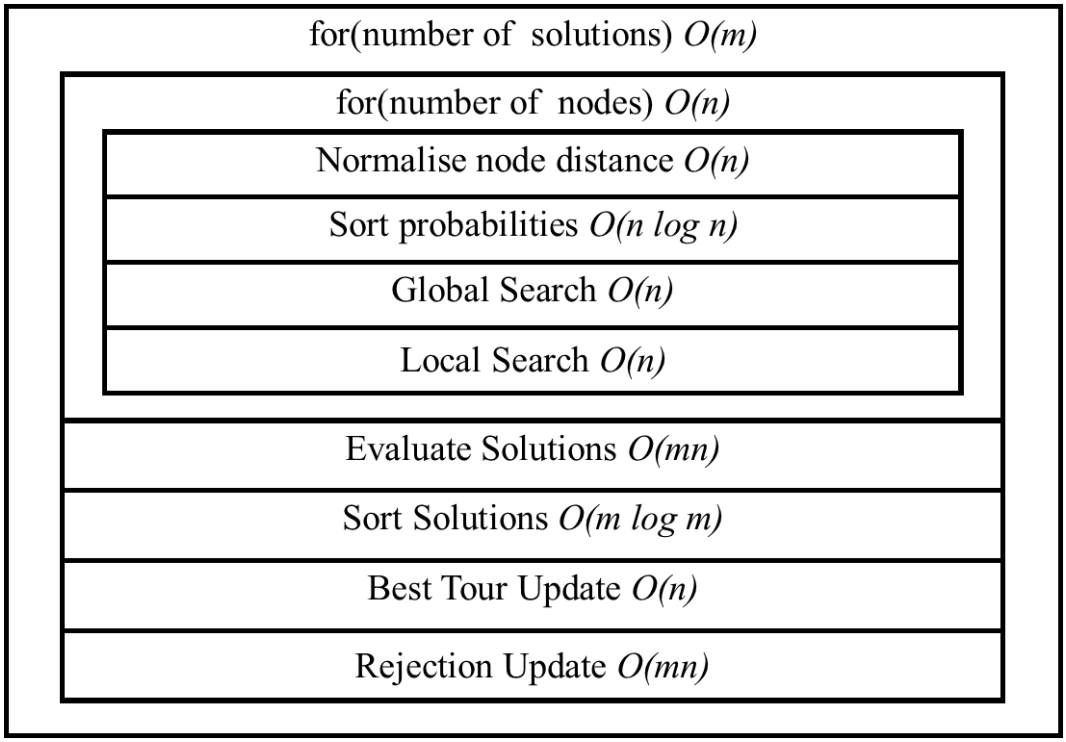


Fig. 6.7: iDFPA complexity analysis

Fig. 6.7 shows that the iDFPA has a complexity of $O(n^3 \log n)$. The PiDFPA can however be reduced to $O(n^2 \log n)$ due to the process allowing for actions to be run in parallel instead of waiting for each agent to finish its task before starting the next agents' task if the number of parallel processors $p \sim n$.

6.3 Speedup and Efficiency

The complexity comparison above is supported by the speedup that can be observed in Table 5.7. When looking at the table it can also be noted that there is an abysmal 1.8% efficiency value for the Berlin52 instance. This however changes drastically when the map size starts increasing, as can be seen for the u1060 map where the efficiency has increased to 122%.

To put the speedup into context, it means that each processor in the GPU that is working on the problem is outperforming the single core on the GPU by a factor of

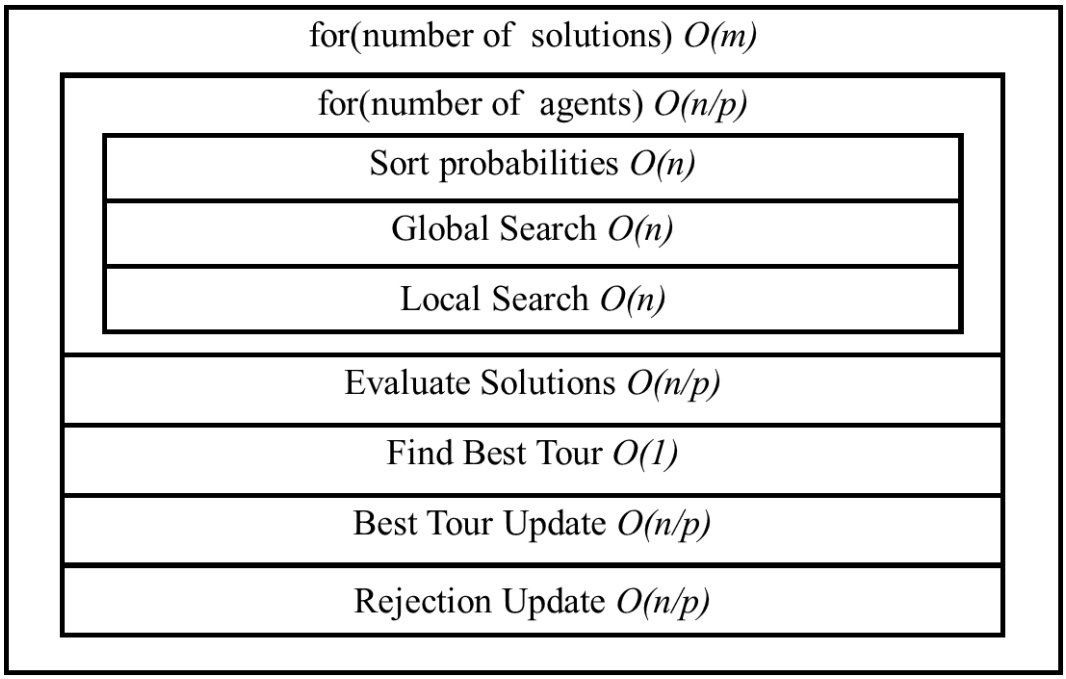


Fig. 6.8: PiDFPA complexity analysis

1.22. The reason for this increase is most likely due to the fact that the problem space of a larger TSP map is non-linear, which supports the complexity analysis done in Fig. 6.7 and 6.8. The speedup is therefore due to the CPU working on a problem that is exponentially more complex and time consuming.

6.4 Performance against other Algorithms

The PiDFPA is able to perform on a similar level to the PACO as shown in Fig. 6.9, 6.10 and 6.11 due to their complexity being of a similar magnitude. The PiDFPA does however underperform slightly with regards to speed compared to the PGA as shown in Table 5.5.

6.4.1 Additional Complexity

The main reason for the loss in speed can probably be attributed to the requirement for a sort function to be used in the code for the PiDFPA in order to arrange the cities for each ant to allow for the correct calculation of the Lévy-distribution.

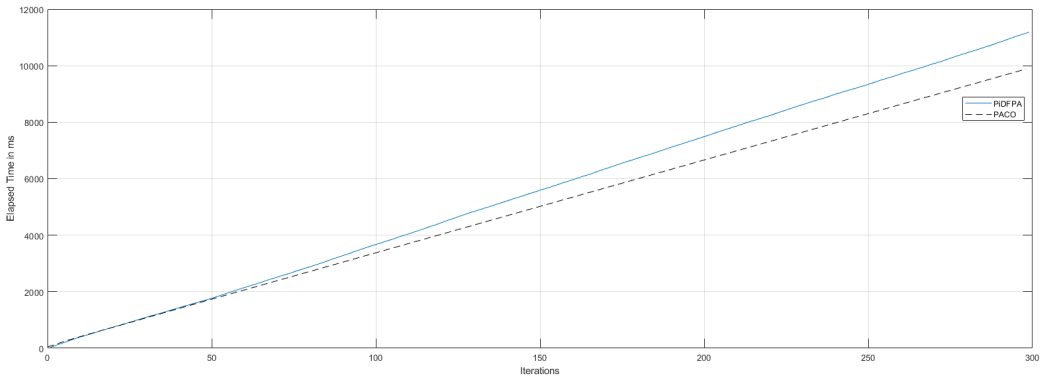


Fig. 6.9: Iteration time cost comparison for PACO vs PiDFPA for Berlin52 over 300 iterations

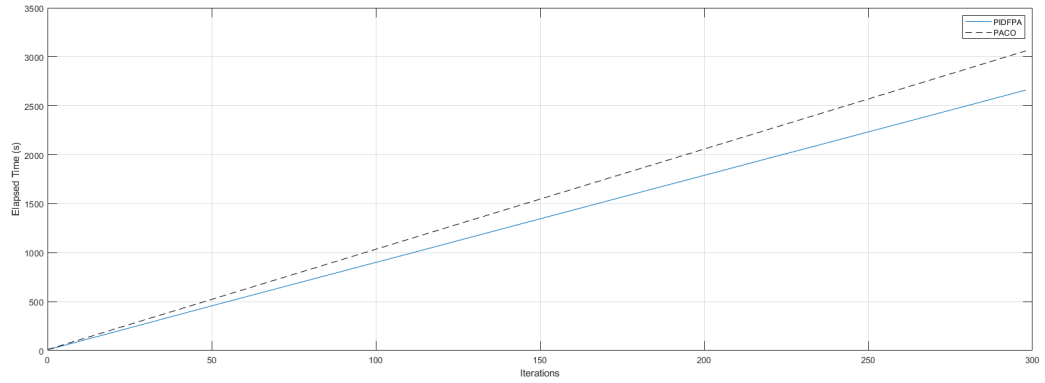


Fig. 6.10: Iteration time cost comparison for PACO vs PiDFPA for Att532 over 300 iterations

6.4.2 Limited Shared Resources

Another considerable bottleneck might be the memory that is required to be kept in the system for the PiDFPA and PACO as they both require to keep track of a pheromone map. This limits the shared memory that can be provided by the CUDA system resulting in a limitation on the amount of grids that can be assigned to solving a problem [74].

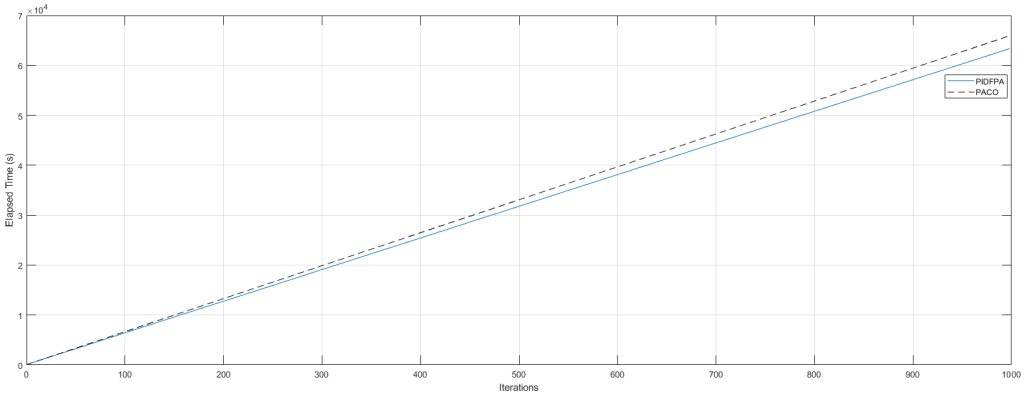


Fig. 6.11: Iteration time cost comparison for PACO vs PiDFPA for U1060 over 1000 iterations

6.4.3 Implementation Error

The analysis done within this chapter shows how the PiDFPA was able to outperform the PACO and PGA by looking at the variables that allow control over the PiDFPA to stop it from converging to a local optima too quickly. This was clearly demonstrated in the U1060 benchmark where the PiDFPA was set to have a slower convergence rate as compared to that of the PACO and PGA, which enabled it to use its global search step to allow for a more accurate sub-optimal solution. The analysis also presents reasons for why the PiDFPA did not perform even better due to possibly lacking further coding optimisation and/or the inability to adjust for certain control during the PiDFPA process.

Chapter 7

Conclusion

The research presented focuses on the question, “How would the Parallel Combination of Ants and Evaluation of Solution Elements method perform in terms of convergence rate and accuracy of the optimal solution for parallelising the iDFPA and additionally, how would the PiDFPA compare to existing PACO solutions using the same pluralization method for a large-scale TSP map?” The answers to that question were provided by first introducing the general history of NP-Hard problems and the TSP as well as functional ways that are used to solve the TSP problem practically by favouring sub-optimal solutions that have the possibility to contain the optimal solution. To further assist in the understanding of the research, a more detailed background was provided, in which the TSP problem was explained in relation to two common methods of solving the TSP, after which the formulation of a new parallel meta-heuristic was provided along with key steps which are required to solve the TSP using the PiDFPA. The results of this new meta-heuristic has been found to outperform both the PACO and PGA in terms of accuracy and convergence rate within 300 iterations of the Berlin52 and Att532 problems and 1000 iterations of the U1060 problem, the difference in running time for the PiDFPA and its parent meta-heuristic, the iDFPA, is also provided, which is analysed in terms of speedup and efficiency to show that the PiDFPA is able to exponentially outperform the iDFPA in the time domain as the TSP size increases. A comprehensive analysis was also provided to explain which factors significantly contributed to this performance of the PiDFPA along with possible future considerations.

7.1 Contributions

In this research a new algorithm was investigated, designed and developed using CUDA based GPU architecture. The resultant algorithm is called the PiDFPA, since it is a parallel version of the FPA.

The PiDFPA is able to solve large TSP maps that contain more that 1000 location elements and can be tested with a one-to-one ratio of agents-to-TSP map size as well as being able to run the map over any size of iteration. This is a remarkable change to the iDFPA which struggled to optimise maps over the size of 300 elements within a feasible time.

The PiDPFA was tested against parallel versions of common meta-heuristic algorithms namely the PGA and the PACO over three maps ranging from a small map of 52 elements to a sizeable map of 1060 elements, where it is able to outperform both the PGA and PACO in terms of accuracy.

7.2 Future Work

Having more time will allow for testing the variables in more depth to ensure that the results can be verified for a large iteration size with at least 30 or more test samples for each combination of optimisation variables. This can also be made easier by finding a way to either sort the values more efficiently or to presort each map for each agent. The PiDFPA can also be adapted and applied to other NP-Hard problems like the Hamiltonian Path Problem or the MKP to find a generalised version of the PiDFPA.

7.3 Conclusion

The answer to the research question is able to provide an optimal solution that has an error rate of less than 15% when compared to the optimal solution. It has also been thoroughly tested against the PACO and PGA methods where it is able to outperform both methods in terms of accuracy and rate of converge to the sub-optimal result in the same range of time as the PACO and PGA. This result was verified by testing the three methods extensively with a large iteration size over three benchmarking problems varying in size. It can also noted that there is a large speedup and efficacy

that is achieved when compared to the iDPFA which is what enabled it to be applied to larger maps, which will allow the PiDFPA to be tested against even more maps that were presented within the context of this dissertation and can potentially provide even stronger performance results with increased variable and code optimisations.

References

[1] D. Davendra, M. Metlicka, and M. Bialic-Davendra, "CUDA Accelerated 2-OPT Local Search for the Traveling Salesman Problem," in *Novel Trends in the Traveling Salesman Problem*, London, United Kingdom, IntechOpen, 2020, pp. 7-28, [Online], Available: `https://www.intechopen.com/chapters/72774`, doi: 10.5772/intechopen.93125.

[2] T. Narwadi, and S. Subiyanto, "An Application of Traveling Salesman Problem Using the Improved Genetic Algorithm on Android Google Maps," in *American Institute of Physics Conference Proceedings*, vol. 1818, no. 1, 2017, p. 020035, [Online], Available: `https://doi.org/10.1063/1.4976899`.

[3] J. Gudmundsson, and C. Levcopoulos, "Hardness result for TSP with neighborhoods," Department of Computer Science, Lund University, Sweden, 2000, [Online], Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.5944&rep=rep1&type=pdf`.

[4] K. D. Boese, "Cost versus distance in the traveling salesman problem." Los Angeles: UCLA Computer Science Department, 1995.

[5] M. Lazarova, and P. Borovska, "Comparison of parallel metaheuristics for solving the TSP," in *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, 2008, pp. II-12, doi: 10.1145/1500879.1500899

[6] T. Veale, and M. T. Keane, "The competence of sub-optimal theories of structure mapping on hard analogies," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, vol. 1, 1997, pp. 232-237.

[7] A. Schrijver, "On the history of combinatorial optimization (till 1960)," in *Handbooks in operations research and management science*, vol. 12, 2005, pp. 1-68.

[8] V. Zharfi, and A. Mirzazadeh, "A novel metaheuristic for travelling salesman problem," in *Journal of Industrial Engineering*, vol. 2013, Jul. 2013, Art no. 347825.

[9] C. Chekuri, and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," in *SIAM Journal on Computing*, vol. 35, no. 3, 2005, pp.713-728.

[10] W, Bozejko, and M. Wodecki. "Theoretical properties of multimoves in metaheuristics in aspect of involutions," 2006.

[11] S. Martello, D. Pisinger, and D. Vigoet, "The Three-Dimensional Bin Packing Problem," in *Operations Research*, vol. 48, no. 2, Informs, 2000, pp. 256–267, [Online], Available: `http://www.jstor.org/stable/223143`.

[12] S. Kumar, and R. Panneerselvam, "A Survey on the Vehicle Routing Problem and Its Variants," in *Intelligent Information Management*, vol. 4 no. 3, 2012, pp. 66-74, doi: 10.4236/iim.2012.43010.

[13] X. Zhang, Z. Hu, and S. Mahadevan, "Bilevel Optimization Model for Resilient Configuration of Logistics Service Centers," in *IEEE Transactions on Reliability*, vol. 71, no. 1, 2022, pp. 469-483, doi: 10.1109/TR.2020.2996025.

[14] F, Yang , T. Jin, T.-Y. Liu, X. Sun, and J. Zhang, "Boosting dynamic programming with neural networks for solving np-hard problems," in *Proceedings of The 10th Asian Conference on Machine Learning*, vol. 95, PMLR, 2018, pp. 726-739.

[15] R. R. Mettu, "Approximation algorithms for np-hard clustering problems". Ph.D dissertation, Deparmtent of Computer Sciences., The University of Texas at Austin, 2013. Accessed on: March 31, 2022. [Online]. Available: `http://repositories.lib.utexas.edu/bitstream/handle/2152/782/metturr026.pdf?sequence=2&isAllowed=y`

[16] J. Žerovnik, "Heuristics for NP-hard optimization problems - simpler is better!?" in *Logistics, Supply Chain, Sustainability and Global Challenges*, vol. 6, no. 1, 2015, pp.1-10, https://doi.org/10.1515/jlst-2015-0006.

[17] X.-S. Yang, "A new meta-heuristic bat-inspired algorithm", in J.R. González, D.A.Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, (eds) *Nature Inspired Cooperative Strategies for Optimization(NICSO 2010)*, vol. 285, Springer, Berlin, Heidelberg, Germany, 2010, pp. 65–74, doi: 10.1007/978-3-642-12538-6_6.

[18] M. Dorigo, M. Birattari and T. Stutzle, "Ant colony optimization," in *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, Nov. 2006, pp. 28-39, doi: 10.1109/MCI.2006.329691.

[19] K.V. Price, R.M. Storn, and J.A. Lampinen, "Differential Evolution a Practical Approach to Global Optimization," Springer, Berlin, Heidelberg, Germany, 2005, pp. 37-134, doi:10.1007/3-540-31306-0.

[20] X.-S. Yang, "Flower pollination algorithm for global optimization," in J. Durand-Lose, and N. Jonoska, (eds) *International conference on unconventional computing and natural computation*, vol. 7445, Springer, Berlin, Heidelberg, Germany, 2012, pp. 240–249, doi: 10.1007/978-3-642-32894-7_27.

[21] M, Abdel-Basset, and L.A. Shawky, "Flower pollination algorithm: a comprehensive review," in *Artificial Intelligence Review*, vol. 52, no. 4, 2019, pp. 2533-2557, doi: 10.1007/s10462-018-9624-4.

[22] R. Strange, A. Y. Yang, and L. Cheng, "Discrete Flower Pollination Algorithm for Solving the Symmetric Travelling Salesman Problem," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 2130-2137, doi: 10.1109/SSCI44817.2019.9002797.

[23] M. Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," in *Applied Soft Computing*, vol. 11, no. 8, 2011, pp. 5181-5197, doi: 10.1016/j.asoc.2011.05.042.

[24] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, "Heuristics from nature for hard combinatorial optimization problems," in *International Transactions in Operational Research*, vol. 3, no. 1, 1996, pp. 1-21.

[25] D.K. Agrafiotis, V.S. Lobanov, and F.R. Salemme, "Combinatorial informatics in the post-genomics era,"in *Nature Reviews Drug Discovery*, vol 1, no. 5, 2002, pp. 337-346.

[26] C.H. Papadimitriou, and K. Steiglitz, "Combinatorial optimization: algorithms and complexity," Dover Publications, Mineola, New York, 1998, pp. 165-192.

[27] J. Žerovnik, "Heuristics for NP-hard optimization problems: simpler is better!?," in *Logistics & Sustainable Transport*, vol. 6, no. 1, 2015, pp. 1-10, doi:10.1515/jlst-2015-0006.

[28] J. Puchinger, and G.R. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification," in J. Mira, and J.R. Álvarez, (eds) *Artificial Intelligence and Knowledge Engineering Applications: A*

Bioinspired Approach. IWINAC 2005. Lecture Notes in Computer Science, vol 3562. Springer, Berlin, Heidelberg, 2005, pp. 41-53, doi: 10.1007/11499305_5.

[29] L.A. Wolsey, and G.L. Nemhauser, "Integer and combinatorial optimization," vol. 55, John Wiley & Sons, New York, 1995, pp. 333-334.

[30] C. Ogwumike, M. Short and M. Denai, "Near-optimal scheduling of residential smart home appliances using heuristic approach," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, 2015, pp. 3128-3133, doi: 10.1109/ICIT.2015.7125560.

[31] Z.A. Lomnicki, "A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem," in *Journal of the operational research society*, vol. 16, no. 1, 1965, pp. 89-100.

[32] S. Tanaka, and S. Fujikuma, "A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time," in *Journal of Scheduling*, vol 15, no. 3, 2012, pp. 347-361.

[33] G, Laporte, and Y Nobert, "A branch and bound algorithm for the capacitated vehicle routing problem," in *Operations-Research-Spektrum*, vol 5, no. 2, 1983, pp. 77-85.

[34] T. Ibaraki, "Branch-and-bound procedure and state—space representation of combinatorial optimization problems," in *Information and Control*, vol. 36, no. 1, 1978, pp. 1-27.

[35] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proceedings of the 12th International Conference on International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 261-268, doi: 10.1016/B978-1-55860-377-6.50040-2.

[36] D.S. Johnson, L.A. McGeoch, and E.E. Rothberg, "Asymptotic experimental analysis for the Held-Karp traveling salesman bound," in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, Vol. 341, San Francisco, ACM Press, 1996, pp. 341-350.

[37] F, Yang , T. Jin, T.-Y. Liu, X. Sun, and J. Zhang, "Boosting dynamic programming with neural networks for solving np-hard problems," in *Proceedings of The 10th Asian Conference on Machine Learning*, vol. 95, PMLR, 2018, pp. 726-739.

[38] S.M. Abdulrahman, "Using swarm intelligence for solving NP-hard problems," in *Academic Journal of Nawroz University*, vol 6, no. 3, 2017, pp. 46-50.

[39] M. Hidalgo-Herrero, P. Rabanal, I. Rodríguez, and F. Rubio, "Comparing problem solving strategies for NP-hard optimization problems," in *Fundamenta Informaticae*, vol 124, no. 1-2, 2013, pp. 1-25.

[40] S. Almufti, R. Marqas, P. Shivan, and A. Sallow, "Single-based and Population-based Metaheuristics for Solving NP-hard Problems," in *Iraqi Journal of Science* vol. 62, no. 5, 2021, pp. 1-11.

[41] X.-S. Yang, S. Deb, and S. Fong. "Metaheuristic algorithms: optimal balance of intensification and diversification," in *Applied Mathematics and Information Sciences* vol. 8, no. 3, 2014, p. 977.

[42] F. Glover, and H. J. Greenberg. "New approaches for heuristic search: A bilateral linkage with artificial intelligence," in *European Journal of Operational Research* vol. 39, no. 2, 1989, pp. 119-130.

[43] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, "Heuristics from nature for hard combinatorial optimization problems," in *International Transactions in Operational Research*, vol. 3, no. 1, 1996, pp. 1-21.

[44] J.H. Holland, "Genetic algorithms: Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand," in *Scientific American*, vol. 267, no. 1, 1992, pp. 66-73.

[45] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," in *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 2, 2005, pp. 126-142, doi: 10.1109/TEVC.2005.843751.

[46] J. Kennedy, and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942-1948, doi: 10.1109/ICNN.1995.488968.

[47] I.L, Bajec, and F,H. Heppner. "Organized flight in birds," in *Animal Behaviour*, vol. 78, no. 4, 2009, pp. 777-789.

[48] E. Nabil, "A modified flower pollination algorithm for global optimization," in *Expert Systems with Applications*, vol. 57, 2016, pp. 192-203.

[49] L. Fredriksson, "A brief survey of Lévy walks: with applications to probe diffusion," M. S. thesis, Karlstad University, Karlstad, Sweden, 2010.

[50] T. T. Nguyen, J. S. Pan, and T. K. Dao, "An Improved Flower Pollination Algorithm for Optimizing Layouts of Nodes in Wireless Sensor Network," in *IEEE Access*, vol. 7, 2019, pp. 75985-75998, doi: 10.1109/ACCESS.2019.2921721.

[51] A. Y. Abdelaziz, E. S. Ali, and S. M. Abd Elazim, "Combined economic and emission dispatch solution using flower pollination algorithm," in textitInternational Journal of Electrical Power & Energy Systems, vol. 80, 2016, pp. 264-274.

[52] E. Nabil, "A modified flower pollination algorithm for global optimization," in textitExpert Systems with Applications, vol. 57, 2016, pp. 192-203.

[53] Z. A. A. Alyasseri, A. T. Khader, M. A. Al-Betar, M. A. Awadallah, and X.-S. Yang, "Variants of the flower pollination algorithm: a review," in *Nature-Inspired Algorithms and Applied Optimization. Studies in Computational Intelligence*, vol. 744, Springer, Cham, 2018, pp. 91-118.

[54] H. Chiroma, N. L. M. Shuib, S. A. Muaz, A. I. Abubakar, L. B. Ila, and J. Z. Maitama, "A review of the applications of bio-inspired flower pollination algorithm," in *Procedia Computer Science*, vol. 62, 2015, pp. 435-441.

[55] M. Grötschel, and M. W. Padberg, "On the symmetric travelling salesman problem I: inequalities," in *Mathematical Programming*, vol. 16, no. 1, 1979, pp. 265-280.

[56] G, Singh, R. Mehta, and S. Katiyar, "Implementation of travelling salesman problem using ant colony optimization," *in Journal of engineering research and applications*, vol. 6, no. 3, 2014, pp. 385-389.

[57] R.M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of computer computations*, Plenum Press, New York, 1972, pp. 83-103.

[58] G. Skorobohatyj, "MP-TESTDATA - The TSPLIB Symmetric Traveling Salesman Problem Instances," elib.zib.de, [Online]. Available: `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html` (accessed Jun. 22, 2022).

[59] N. M. Razali, and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering*, vol. 2, no. 1, International Association of Engineers, Hong Kong, China, 2011, pp. 1-6.

[60] Ş. Gülcü, M. Mahi, Ö. K. Baykan, and H. Kodaz, "A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem," in *Soft Computing*, vol. 22, no. 5, 2018, pp. 1669-1685.

[61] J. Yang, X. Shi, M. Marchese, Y. Liang, "An ant colony optimization method for generalized TSP problem," in *Progress in Natural Science*, vol. 18, no. 11, 2008, pp. 1417-1422.

[62] J. Pedro Schmitt, F. Baldo, and R. Stubs Parpinelli, "A MAX-MIN Ant System with Short-Term Memory Applied to the Dynamic and Asymmetric Traveling Salesman Problem," in *7th Brazilian Conference on Intelligent Systems (BRACIS)*, 2018, pp. 1-6, doi: 10.1109/BRACIS.2018.00009.

[63] P. Lalbakhsh, B. Zaeri, and A. Lalbakhsh, "An improved model of ant colony optimization using a novel pheromone update strategy," in *IEICE Transactions on Information and Systems*, vol. 96, no. 11, 2013, pp. 2309-2318.

[64] M. Randall, and A. Lewis, "A parallel implementation of ant colony optimization," in *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, 2002, pp. 1421-1432.

[65] M, Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," in *Applied Soft Computing*, vol. 11, no. 8, 2011, pp. 5181-5197.

[66] E. G. Talbi, O. Roux, C. Fonlupt, and D. Robillard, "Parallel ant colonies for the quadratic assignment problem," in *Future Generation Computer Systems*, vol. 17, no. 4, 2001, pp. 441-449.

[67] S. Ibri, H. Drias, and M. Nourelfath, "A parallel hybrid ant-tabu algorithm for integrated emergency vehicle dispatching and covering problem," in *International Journal of Innovative Computing and Applications*, vol. 2, no. 4, 2010, pp. 226-236.

[68] K. F. Doerner, R. F. Hartl, G. Kiechle, M. Lucka, and M. Reimann, "Parallel ant systems for the capacitated vehicle routing problem," in J. Gottlieb, G. R. Raidl, (eds) *European Conference on Evolutionary Computation in Combinatorial Optimization EvoCOP 2004. Lecture Notes in Computer Science*, vol. 3004, Springer, Berlin, Heidelberg, 2004, pp. 72-83.

[69] T. Stützle, "Parallelization strategies for ant colony optimization," in A. E. Eiben, T. Bäck, M. Schoenauer, H. P. Schwefel, (eds) *Parallel Problem Solving from Nature — PPSN V. PPSN 1998. Lecture Notes in Computer Science*, vol. 1498, Springer, Berlin, Heidelberg, 1998, pp. 722-731.

[70] E. Alba, G. Leguizamon and G. Ordonez, "Analyzing the behavior of parallel ant colony systems for large instances of the task scheduling problem," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 14 pp.-, doi: 10.1109/IPDPS.2005.109.

[71] L, Chen, HY Sun, and W. Shu "Parallel implementation of ant colony optimization on MPP," in *2008 International Conference on Machine Learning and Cybernetics*, vol. 2, IEEE, 2008, pp. 981-986.

[72] M, Lucka, and P. Stanislav, "Parallel posix threads based ant colony optimization using asynchronous communication," in *Proceedings of the 8th International Conference on Applied Mathematics*, vol. 2, 2009, pp. 229-236.

[73] D. L. Eager, J. Zahorjan and E. D. Lazowska, "Speedup versus efficiency in parallel systems," in *IEEE Transactions on Computers*, vol. 38, no. 3, 1989, pp. 408-423, doi: 10.1109/12.21127.

[74] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 163-174, doi: 10.1109/ISPASS.2009.4919648.