COMPUTATIONAL PERFORMANCE STUDY OF SOME HEURISTICS FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

BY

ASANI EMMANUEL OLUWATOBI

(18PGCD000017)

MAY, 2021

COMPUTATIONAL PERFORMANCE STUDY OF SOME HEURISTICS FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

Ph.D THESIS

BY

ASANI EMMANUEL OLUWATOBI

(18PGCD000017)

SUPERVISOR

PROF. A.E. OKEYINKA

CO-SUPERVISOR

PROF. A.A. ADEBIYI

DEPARTMENT OF COMPUTER SCIENCE,

LANDMARK UNIVERSITY, OMU-ARAN

MAY, 2021

COMPUTATIONAL PERFORMANCE STUDY OF SOME HEURISTICS FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

ASANI EMMANUEL OLUWATOBI

(18PGCD000017)

A Thesis submitted to the Department of Computer Science, College of Pure and Applied Sciences, Landmark University, Omu-Aran. Nigeria.

In Partial Fulfilment of the Requirements for the Award of the Degree of Doctor of Philosophy (PhD) in Computer Science.

MAY, 2021

DECLARATION

I, **EMMANUEL OLUWATOBI ASANI,** a PhD student in the Department of Computer Science, Landmark University, Omu-Aran, hereby declare that this thesis entitled "**COMPUTATIONAL PERFORMANCE STUDY OF SOME HEURISTICS FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS**", submitted by me is based on my original work. Any material(s) obtained from other sources or work done by any other persons or institutions have been duly acknowledged.

EMMANUEL OLUWATOBI ASANI (18PGCD000017)

Signature & Date

CERTIFICATION

This is to certify that this thesis has been read and approved as meeting the requirements of the Department of Computer Science, Landmark University, Omu-Aran, Nigeria, for the Award of PhD Degree.

Prof A.E. Okeyinka Supervisor	Date
Prof. A.A. Adebiyi (Co- Supervisor)	Date
Dr. Mrs. M.O. Adebiyi (Head of Department)	Date
Prof. S.O. Olabiyisi (External Examiner)	Date

ABSTRACT

The Optimization Problem of solving complex, mostly impracticable problems with limited resources remains a research conundrum which has necessitated enormous amount of intervention over the years. In addressing Combinatorial Optimization Problems, many problems have been formulated, prominent among which is the Travelling Salesman Problem (TSP). While the exact approach to solving the TSP guarantees optimal solutions, more attention has been paid to approximate methods over the years because they address the limitations of exact techniques by generating solutions to complex problems within polynomial time p, especially with increasing solution space. Thus, a considerable amount of research efforts has gone into obtaining good lower bounds on the optimal tour quality of approximate methods of different classes such as the Tour Construction, Improvement, Compound heuristics and Metaheuristics.

The goal of this study is to investigate some Tour Construction heuristics with a view to understanding their implementation details and how they are applied to the solution process of the Travelling Salesman Problem, and to formulate a better solution in solving the Travelling Salesman Problem. Two classic Tour Construction heuristics were examined, namely the Nearest Neighbour Heuristic (NNH) and the Farthest Insertion Heuristic (FIH). The NNH solves the Travelling Salesman Problem using a greedy approach and suffers immensely from the "*curse of dimensionality*" phenomena. The FIH on the other hand is considered as the best performing Insertion heuristic and best among lower order complexity heuristics. However, its performance is impeded by the distance between its partial circuit and the new node to be inserted.

In order to circumvent the limitation of the NNH and FIH, a new insertion technique referred to as the Half Max Insertion Technique (HMIH) was evolved. The HMIH randomly pick one node from Q by $init(Q)$ and creates a partial circuit which is expanded with every iteration. The partial circuit is made up of the points u, v, w to form a minimum polygon. In the $(i + 1)th$ iteration, the insertion heuristics attempt to add one node into the current circuit by minimizing the increment of the total distance of the circuit. The method first determines the longest distance d_{max} of any node from either of $u \; or \; v$ and computes ${}^{1}/{}_{2}\, d_{max}$. The routine then finds a node w not in the subtour whose distance from either $u \; or \; v \;\approx {}^{1}/{}_{2}\, d_{max}$. An edge (u, v) of the subtour to which the insertion of w gives the smallest increase of length, that is for which $\Delta f = c_{ux} + c_{xv} + c_{wx} - c_{uvw} \; is \; smallest$ is determined and x is inserted between $u, v \; and \; w$. This process is iterated until a Hamiltonian cycle is formed.

The NNH, FIH and the newly devised HMIH were experimented on ten publicly available benchmark instances from the Travelling Salesman Problem Library (TSPLIB). The experimental results revealed that the Half Max Insertion Heuristic consistently outperformed both the FIH and NNH across a wide spectrum of benchmark instances with statistical significance of as much as 16% at some point. The average goodness value of the proposed HMIH was 86.9% as against 81.7% for the FIH and 74.5% for the NNH. Hence, the HMIH has a higher accuracy than both the FIH and NNH, and therefore yields a superior heuristic in tackling NP-Hard problems.

DEDICATION

"To him that stretched out the earth above the waters: for his mercy endureth for ever" *Psalm 136:6*

ACKNOWLEDGMENT

I like to acknowledge the immense contributions of the following people to the success of this work;

My supervisor, Prof. A.E. Okeyinka for being a father to me and a mentor, for the unshakable trust you showed in my abilities and for leading so that I could follow. My co-supervisor, Prof. A.A. Adebiyi for your support and understanding, thank you for your calm demeanour and for always pushing me to do better.

My priceless parents, Revd., and Revd. (Mrs.) Asani for their unconditional love and support, for guiding me to discover the greatest gift of all which is my salvation. The family of Pastor and Deaconess Olowe for their unconditional support and for nurturing the treasure that is my wife. My precious jewel, Oluwaseun Asani for believing in me, for bearing with me during the period of this work and for the many days and nights of prayers. My beloved son, Asher Ireoluwa and daughter, Anna Oluwatobi, thank you for rocking my world.

Uncle Remi Ajala who travelled all the way from Lagos to Osogbo to birth this dream by convincing me to study Computer Science. Prof. O.B. Longe, Prof. O.B. Akpor and my aunties, Abosede Olufunmi, Monisola Fasakin, Adebola Adelodun for being great mentors to me over the years. My brothers, Ikubanni Stephen, Ikubanni Peter, ẹ̀gbọ́n Sola Owolabi, Akinwale Oshodi, Opeyemi Matiluko, A.A. Adegun, Olabode Ajagbe and Ayo Aregbesola for providing timely supports and inspiration. Many thanks to Ayoola Joyce, Ayegba Peace, Adeojo Emmanuel and Musa Joshua for their contributions. To my senior colleagues as well as peers in the Department of Computer Science, Landmark University, Dr. B. Gbadamosi, Dr. M. Adebiyi, Dr. A. Kayode, Mrs. R.O. Ogundokun, Mr. N.O. Akande, Mr. J.K. Adeniyi, Mr. M. Arowolo Mr. A.E.

Adeniyi and Mr. P.O. Ehiedu, thank you for your support, this might not have been possible without you.

In conclusion, I like to pay tribute to Pastor J. Akintola and the memory of Late Deaconess Ruth Oladunni Akintola of Good Tidings Nursery and Primary School for believing in me at a very tender age and offering me scholarship in their institution of learning at a time when I would have dropped out due to financial difficulties. Thank you for offering that quality foundation upon which this achievement stands solid.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER ONE

1.0. INTRODUCTION

1.1. Background to the Study

The task of solving complex, often impracticable computational problems with limited resources remains a research conundrum that continues to generate interests in the field of Theoretical Computer Science. This scientific technique of finding the best solutions of cost functions is referred to as Combinatorial Optimization. In other words, Combinatorial Optimization is concerned with the task of obtaining the optimal or close to the optimal set of solutions of a finite set, subject to predefined conditions or constraints (Dowlatshashi *et al*, 2014). These sets of possible solutions can be depicted with formal mathematical notations or structures, such as graphs, trees, and matroids, among others.

Combinatorial Optimization spans the fields of Engineering, Bioinformatics, Artificial Intelligence, Mathematics, Operations Research, Computer Science to complete tasks such as memory register allocation, planning and scheduling, project management, internet data packet routing, protein structure prediction and so on. Models are built to formulate and solve real-life problems. Examples include the Travelling Salesman Problem (TSP), Satisfiability Problems (SAT), Graph Colouring Problems (GCP), Cutting Stock Problem, Minimum Spanning Tree (MST), Constraint Satisfaction Problem (CSP), Bin Parking Problem (BPP) and so on. (Neos, 2018; Becker and Buriol, 2019). Combinatorial Optimisation Problems (COP) are categorized as either P-problems or NP-hard problems. COPs whose solutions can be obtained in polynomial time are referred to as P-problems. They are mostly decision problems and their solution spaces can be built in polynomial time *p*. The COPs whose solutions are obtainable in non-deterministic polynomial time are referred to as NP-hard Problems (Woeginger, 2003). Some of these problems can be solved using either exact algorithms or approximate methods. However, because most of these problems are NP-hard problems and since the search space of the factorial number of solutions becomes so large that they are impractical to solve using exhaustive processing, the use of heuristics is often resorted to.

Combinatorial Optimization aims to provide solutions by deploying efficient algorithmic techniques whose runtime is bounded by a polynomial in the input size. Thus, in solving Combinatorial Optimisation Problems, the concerns are:

i. How quickly can one (or all) optimal solution(s) be obtained?

ii. And in cases where, due to complexities, the optimal solution is impracticable, what is the most appropriate solution that can be found using efficient algorithmic techniques?

In this study, the Travelling Salesman Problem is considered as a classic Combinatorial Optimization Problem. The Problem was first formulated in the nineteenth century and enhanced in the 1930s by M.M. Flood, and it has become the benchmark for several other techniques of optimization (Ajaz and Himani, 2016). The TSP is a shortest tour (or path) problem to find the optimal route while traversing a set of cities (or nodes), ensuring each city (or node) is visited exactly once before returning to the start node (or city), the tour thus made is referred to as Hamiltonian Cycle. It is assumed that the cost of the distance between any pair of cities is predefined. In this regard, the cost often refers to distance but may represent other notions such as time or money. A Hamiltonian cycle as depicted in Figure 1.1. refers to a graph cycle that traverses all the graph's vertices exactly once before returning to its starting vertex. The Travelling Salesman must traverse cities 1 $to\ n$ in a Hamiltonian cycle that is; Start from city 1, traverse the remaining $n-1$ cities in a specified order and then connect back to the starting city, having touched each of the cities only once at a minimal cost.

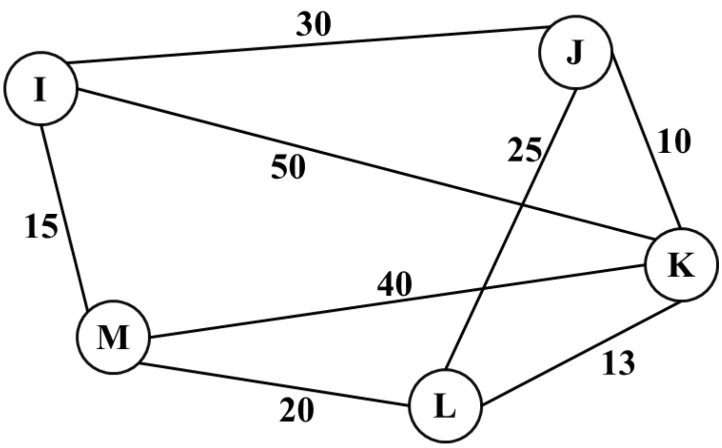

Figure 1.1. A Hamiltonian weighted graph around a network of five nodes
Where $n = number\ of\ nodes = 5$
$I, J, K, L, M = nodes/vertices$
$c_{I,J} = euclidean\ distance/tour\ cost\ between\ nodes\ I\ and\ J = 30$
$I - J - K - L - M - I = tour/hamiltonian\ cycle$

The distance $d(a,b)$ depicts the distance from the city $a\ to\ b$. Thus TSP is formally defined as presented in Equations (1.1) to (1.3);

$$F\ =\ min \sum_{a=1}^{n} \sum_{b=1}^{n} d_{ab} x_{ab} \tag{1.1}$$

$$where \sum_{b=1}^{n} x_{ab} = 1; a = 1,\ \ldots,\ n \tag{1.2}$$

$$and \sum_{a=1}^{n} d_{ab} = 1; b = 1,\ \ldots,\ n \tag{1.3}$$

The objective function is marked with F. With a limitation,

$$x_{a_1 a_2} + x_{a_2 a_3} + \ \dots + \ x_{a_r a_1} \leq r - 1 \qquad (1.4)$$

x_{ab} are the binary variables

$$x_{ab} = \begin{cases} 1 & if\ the\ salesman\ travels\ from\ city\ a\ to\ city\ b \\ 0 & if\ the\ salesman\ is\ not\ travelling\ from\ city\ a\ to\ city\ b \end{cases}$$

d_{ab} is the cost of moving from city *a* to city *b*.

The TSP has applications in several areas, most especially in varying areas of transportation. Being an NP-hard problem, the TSP has several solution algorithms broadly categorized into Exact Algorithms and Approximate Algorithms (heuristics). Solving TSP using Exact techniques involve the Explicit enumeration of the solution space; this is also known as brute force. Brute force obtains an optimal tour by exploring the entire search space and building all the possible solutions. There are instances where it is possible to solve the TSP efficiently, especially those with a small degree of search space, using exact algorithms. An example is the problem of obtaining the shortest route on a graph, based on some practically achievable assumptions. This can be tackled optimally in polynomial time by the "Dijkstra or Bellman-Ford algorithms" (Giovanni, 2017). More complex problems, with no "*efficient*" algorithms, may be approached by first modelling the problem as a Mixed Linear Programming (MILP) paradigm, then solving them using any suitable MILP solver such as Cplex, Gurobi, Xpress, AMPL, OPL and so on. This utilizes the general-purpose exact algorithms which guarantee optimal solutions at least hypothetically. The computational complexities of these techniques are exponential in nature, thus, the time required to provide their solutions grows exponentially with its solution space (Giovanni, 2017).

Although exact methods can potentially generate optimal tour, especially in theory, they are often impracticable and especially unsuitable for NP-hard problems with large solution space. For instance, the solution renown as the best performing exact technique is based on dynamic programming with a complexity of $O(2^n n^2)$, thus making it impracticable to solve TSP as the search space expands (Deudon *et al.*, 2018). This is a result of two practically related phenomena which are: 1. the complexity of COPs, which are generally NP-Hard in nature, and 2. the constraint of time. This explains the drive for the design, development, and deployment of heuristics. In contrast to exact techniques, heuristics provide solutions within polynomial p time.

Heuristics are approximate techniques that apply '*rules of thumb*' for solving Combinatorial Optimization Problems without necessarily guaranteeing optimal solutions. Heuristics provide approximate solutions within the constraint of polynomial time. Heuristic solutions are referred to as approximate because they make use of probabilities and some certain set of rules to finding solutions to problems. For an iterative procedure, heuristics can be used when an optimal solution is guaranteed to either obtain the solution with ease or make a decision within an exact procedure. In other words, the use of heuristics to solve the TSP and problems related to the TSP provides acceptable results that are not too far from the optimal and yet, are computationally affordable. Heuristics may be classified based on the atomicity of their solution procedures as Tour Construction, Improvement / Local Search Heuristics, and Compound Heuristics (Oliveira and Carravilla, 2009; Marti and Reinelt, 2011; Kyritsis *et al.*, 2018). The Tour Construction heuristics are stand-alone techniques that generate solutions by sequentially applying a set of predefined procedures to the problem space. These procedures describe the processes involved in stages of Initialization; Selection and; Insertion.

This study is focused on solving the TSP using Tour Construction heuristics. Tour construction heuristics are not only suitable for solving TSPs, they are equally central to the performance of the other classes of heuristics such as improvement techniques, compound heuristics, and metaheuristics. Construction heuristics serve as a seed for the development of some heuristics and can be used to build initial solutions for high performing techniques (Rao and Jin,2010; Huang and Yu, 2017; Lity *et al.*, 2017).

1.2. Statement of the Problem

Notwithstanding, the avalanche of computational techniques, many real-life problems of great importance remain largely unsolvable within the constraint of polynomial time, due to the intractability of Combinatorial Optimisation Problems and the limitations of exact algorithms in solving them in polynomial time. It has, therefore, become pertinent to study heuristics with a view to identifying the potentials for improving the possibilities of attaining the best trade-off between quality of the solution and computational time (Rego *et al.*, 2011; Abid and Mohammed, 2015). Heuristics algorithms play a prominent role in improving the search capability of exact algorithms.

A considerable amount of research efforts has gone into obtaining good lower bounds on the optimal tour quality for benchmark instances especially using Construction techniques (Bernardino and Paias, 2018; Kitjacharoenchaia *et al.*, 2019; Victor *et al.*, 2020; Babel 2020). The development of a high performing Tour Construction heuristics remains a research concern because they not only generate good approximate solutions for TSPs, they are equally central to the performance of the other classes of heuristics such as improvement techniques, compound heuristics, and metaheuristics. Construction heuristics serve as a seed for the development of some heuristics and can be used to build initial solutions for high performing techniques (Rao and Jin,2010;

Huang and Yu, 2017; Lity *et al.,* 2017). Construction heuristics generally generate better initial solutions in high performing improvement methods/metaheuristics than random initial solutions, thereby enhancing the quality of solutions (Ali, 2016; Neelima *et al.,* 2016; Wang *et al.,* 2016; Vaishnav *et al.,* 2017).

Existing tour construction methods typically fall short by between 10-30% in terms of solution quality with a worst-case complexity of $T(n) = O(n^2)$. The NNH for instance is fast, flexible, and simple to implement; it however solves the Travelling Salesman Problem using a greedy approach and suffers immensely from the "curse of dimensionality" phenomenon (Chen and Shar, 2018). The FIH on the other hand is renowned as the best performing lower-order complexity heuristic, yet suffers from a high upper bound of error with farther distance (Huang *et al.,* 2016). According to Huang *et al.,* (2016), if the distance can be reduced, the probability of attaining an optimal tour is higher. Rao and Jin (2010); Pichpibul and Kawtummachai, (2012) and Huang *et al.*, (2016) have identified the need for the development of a better performing tour construction technique. Thus, this study examines two classic construction heuristics, namely the Nearest Neighbour Heuristic and the Farthest Insertion Heuristic in order to evolve and experiment with a new and improved Tour Construction method which addresses the inherent limitations of the NNH and FIH. Additionally, an extensive performance evaluation of NNH, FIH and the newly developed heuristic is of great interest in this study.

1.2.1. Problem Formulation

Consider a *postal route problem*; suppose that a utility vehicle has to deliver agricultural products in m cities. The vehicle must complete a Hamiltonian cycle by touring cities $1\ to\ m$ exactly once and return to the starting city. The objective is to build the tour order which will guarantee minimal cost as the vehicle visits the cities/nodes from start till a Hamiltonian cycle is complete. The cost, in this case, refers to the distance or the tour length required to complete the cycle. Let $d_{a,b}$ be the distance from city $a\ to\ b$, given that the tour from $a\ to\ b$ traverses all the nodes with an edge, this is a complete graph. For each edge, therefore, a binary variable is associated.

$$x_{ab} = \begin{cases} 1,\ if\ (a,b) \in E\ is\ in\ the\ tour \\ 0,\ if\ otherwise \end{cases} \tag{1.5}$$

The total distance covered by the salesman can then be depicted as:

$$total\ distance = \sum_{(a,b)\in E} d_{ab}x_{ab} \tag{1.6}$$

The objective of the TSP is to minimize Equation (1.6), subject to two preconditions, which are:

i. For every node a, exactly two of the x_{ab} binary variables should be equal to 1.

ii. All the nodes must be connected to make the tour a complete graph.

Thus, the TSP can be mathematically described as:

$$minimize \sum_{(a,b)\in E} d_{ab}x_{ab} \tag{1.7}$$

$$subject\ to \sum_{b\in V} x_{ab} = 2\ \forall i \in V \tag{1.8}$$

$$\sum_{a,b\in S,\ a\neq b} x_{ab} \leq |S| - 1\ \forall S \subset V, S \neq \emptyset \tag{1.9}$$

$$x_{ab} \in \{0,1\}$$

1.3. Justification for the Study

While there are numerous instances of the Combinatorial Optimization Problems, the TSP is perhaps the most important of them all. Works on the TSP have catalyzed the emergence of several revolutionary concepts in the field of combinatorics and have led to notable advances in cutting edge researches in complexity theories and practices. Furthermore, the TSP has also become a standard testbed for the design and development of new, innovative techniques; numerous important methods devised to provide generic solutions to Combinatorial Optimization Problems were first tested on the TSP. These include cutting planes in integer programming, a precursor to high performing techniques such as the branch & cut methods, polyhedral approaches, branch & bound algorithms, as well as early local search algorithms. Other techniques such as Simulated Annealing, Ant Colony Optimization, and so on were first tested on the TSP. Thus, the outcome of this study is expected to further the frontiers of knowledge in the field of combinatorics and result in the development of an improved solution to the Travelling Salesman Problem and by extension, Combinatorial Optimization Problems.

1.4. Aim and Objectives

Given the intractability of some computational problems as well as the need to solve such problems using the available resources, the study of heuristics, both the existing and newly derived ones, has become prominent in theoretical Computer Science. In this study, the complexity of some heuristics is examined and evaluated and invoked to formulate a better solution in solving the Travelling Salesman Problem. The study therefore aims at improving on the performance of the NNH and FIH for solving Combinatorial Optimization Problems

The specific objectives of the study are to:

1. Implement some classical tour construction heuristics on the Travelling Salesman Problem;

2. Propose and implement a new heuristic model for solving the Travelling Salesman Problem;
3. Evaluate the performance of the three heuristics in (1) and (2) vis-à-vis solution quality and computational time;
4. Undertake a comparative study on the classical heuristics and the proposed heuristics.

1.5. Research Questions

The experiment is expected to answer the following questions;

1. What is the performance of the Nearest Neighbour Heuristic and Farthest Insertion Heuristic in terms of solution quality and time complexity for given instances and parameter set?
2. Can the quality of the result of a tour construction heuristic be improved upon to outperform the Farthest Insertion Heuristic which is the best performing lower-order complexity heuristic, while still retaining the same complexity of $0(n^2)$?
3. How does the improvement affect the computational time?

1.6. Overview of Research

The goal of this study is to investigate some approximate methods with a view to understanding their implementation details and how they are applied to the solution process of the Travelling Salesman Problems, to identify their limitations and ultimately device a new technique to circumvent these limitations and produce better solutions.

Given a tour distance d_{ab} and associated binary variable:

$$x_{ab} = \begin{cases} 1, \ if \ (a,b) \in E \ is \ in \ the \ tour \\ 0, \ if \ otherwise \end{cases} \tag{1.10}$$

An optimal solution is a solution in which:

$$tourcost = \sum_{(a,b)\in} d_{ab} x_{ab} \ is \ minimal \tag{1.11}$$

The objective is to minimize the tour length, that is, obtain a solution that is as close to the optimal solution as possible.

Thus, in achieving this goal, two tour construction heuristics were studied, namely, Nearest Neighbour Heuristic and Farthest Insertion Heuristic. The NNH readily comes to mind when solving the TSP and the FIH gives the best solution quality of all lower-order complexity heuristics. Tour construction heuristics were considered in this study, because of their importance both as viable solution techniques and as seed for the performance of other classes of heuristics. Relevant literature on these techniques were reviewed, then the methods were experimented on some benchmark instances and used to solve a hypothetical Travelling Salesman Problem. A new insertion technique, referred to in this study as the Half Max Insertion Heuristic (HMIH) was then derived with the potentials of outperforming existing state-of-the-art techniques.

All algorithms were implemented using the Java programming language.

The performances of the new and existing methods were evaluated using two measures:

i. Solution quality: the solution quality of a heuristic technique is determined by its tour cost relative to the optimal tour cost. The closer the tour cost is to the optimal cost, the better the quality of the technique.

ii. Computational speed approach: The computational speed is determined by computing the time taken to process the solution.

Table 1.1 maps the objective of this study to the materials and methods for achieving them.

Table 1.1. Mapping Objectives to Activities/Methods

OBJECTIVES	METHODOLOGY
Objective 1: To implement some classical heuristics on the Travelling Salesman Problem.	- Model the *postal route problem* as a Travelling Salesman Problem. - Obtain dataset (TSPLIB) - Generate a distance matrix as input to the program. - Implement the NNH and FIH in the Java Programming Environment.
Objective 2: To propose and implement a new heuristic in solving the Travelling Salesman Problem.	- Model the proposed insertion technique (Pseudocode, Flowchart) - Implement the technique on ten testbeds in the Java Programming Environment
Objective 3: To evaluate the performance of the existing heuristics considered and the proposed one.	- Computational speed approach - Generate cost and determine Solution quality (percentage deviation from optimal solution) - *Percentage Error (δ)*

	- *Quality improvement (Σ):* - *Goodness Value (ℊ):*
Objective 4: To carry out a comparative study on the classical heuristics and the proposed heuristics.	- Tables, Charts

1.7. Scope of the Study

This study aims to investigate the performance of some heuristics, including, Nearest Neighbour Heuristic and Farthest Insertion Heuristic on a Combinatorial Optimization Problem vis-a-vis their solution quality and complexity. The Combinatorial Optimization Problem considered is the Travelling Salesman Problem (TSP) due to its wide acceptability as the model testbed for new algorithmic ideas in solving COPs. The study is restricted to Tour Construction Heuristics. Other classes of heuristics are not covered in this study. A novel Tour Construction heuristic was designed and implemented and the result compared with that of existing methods.

Ten benchmark cases were considered from publicly available TSPLIB dataset because of the availability of optimal results for comparison. The data are categorized into three as follow:

- $no\_of\_nodes < 100$
- $100 < no\_of\_node < 1000$,
- $no\_of\_nodes \geq 1000$

Java Programming Language was used for implementation on a Windows Operating System platform.

1.8. Significance of the Study

This work is expected to extend the frontiers of knowledge in solving Combinatorial Optimization Problems, especially the Travelling Salesman Problem. By designing and implementing a new and improved construction tour technique, the solution quality of construction method would be enhanced, and this might impact positively on the performances of other classes of heuristics that depend on tour construction methods.

Results obtained in this study are good indices that can aid some crucial decisions of experts in to relevant domains such as route-finding, transportation, circuitry, VLSI design, logistics, pick-up and delivery of agricultural products, protein structure prediction, Printed-circuit-boards manufacturing, data transmission in computer networks, and so on.

1.9. Arrangement of the Thesis

This thesis is organised into five chapters. Chapter one includes an introduction to the study carried out, statement of the problem, justification for the study, aim and objectives, research questions, overview, significance, and scope of the study. The second chapter covers a review of fundamental concepts and existing related studies on Combinatorial Optimization Problems. Also contained in chapter two are detailed discussion on TSPs, variations of TSPs and methods that have been used to solve them. The concluding part of chapter two contains a detailed review of related literature that tackle the Travelling Salesman Problem using Tour Construction methods. Chapter three covers the description of the conceptual design, materials and method, as well as dataset, performance model and metrics. Chapter four focuses on testing, discussion of the results obtained, and evaluation of the techniques. The thesis is finally concluded in chapter five with summarized discussion of results, contributions to knowledge, recommendations, and suggestions for further work.

CHAPTER TWO

2.0. REVIEW OF LITERATURE

In this section, well-known Combinatorial Optimization Problems in literature such as the Travelling Salesman Problem (with the objective of minimizing the cost of completing a Hamiltonian cycle), the Knapsack problem (with the objective of maximizing gain using limited resources are reviewed). Other COPs in literature include, the Satisfiability Problem (SAT), the Graph Colouring Problem (GCP), the Cutting Stock Problem, the Minimum Spanning Tree (MST), Constraint Satisfaction Problems (CSP), Bin Parking Problems (BPP) and so on, (Neos, 2018; Becker and Buroil, 2019). Relevant literature on exact and approximate methods of solving COPs were also reviewed.

2.1. Combinatorial Optimization Problems

Combinatorial Optimization Problems often require the application of computational techniques to find optimal solutions within a finite set of possible solutions using limited resources, mostly defined in terms of space and time. Due to these constraints and their extremely large search space, exhaustive search methods are often not a realistic option in solving Combinatorial Optimisation Problems.

In formulating COPs, a finite set of variables with discrete domains is first defined; the aim is for the solution to satisfy a predefined set of constraints while optimizing an objective function. The optimality, based on some objective function that aims to either minimize or maximize is also stated. For instance, the objective may be to minimize distance, cost, time, weight, or maximize yield, efficiency, production, and so on.

A Combinatorial Optimization Problem is a four tuple (X, Σ, C, f) defined as follow (BUI, 2015):

- $X = \{x_1, \ldots, x_n\}$ is the finite set of variables;
- $D = \{ D(x_1), \ldots, D(x_n)\}$ is the set of domains of variables; consequently, $D(X_n)$ defines the domain of variable X_n;
- $C = \{C_1, \ldots, C_k\}$ is the set of constraints over variables;
- f is the objective function to be optimized.

Thus, given the objective function $f: D \rightarrow \mathbb{R}$ and $S \subseteq D$ as a set of feasible solutions x, defined according to some constraints $C = \{C_1, \ldots, C_k\}$, the generic optimization problem may be formulated as follow:

$$(OPT) \qquad minima|maxima\ f(x) \qquad (2.1)$$

$$subject\ to\ x \in S$$

Combinatorial Optimization Problems may be solved using either exact methods or heuristics. The objective of the non-heuristic (Exact) methods are to obtain optimal solutions through exhaustive searches while minimizing to a large extent, the computation time of the algorithm. For instance, Yu and Lin, (2004) designed and developed a service selection technique to optimize the user-centric QoS constraints of composite web services. They modelled the problem as a classic "*Multiple-Choice Knapsack Problem (MCKP)*" and applied their optimal solution to minimize service's end-to-end delay constraint. In the same vein, Grabrel *et al*, (2014) obtained an optimal solution to the Composite Web Services (CWS) problem using the 0-1 Linear Programming approach. The objective was to obtain an optimal solution in shorter computation time. They modelled the problem on a dependency graph and implemented their novel method on the CPLEX solver. The result obtained was optimal over wide-ranging benchmark instances and reduced response time for the transactional CWS.

Exact methods however tend to become grossly inadequate and incapable of dealing with NP-Hard COPs, especially as the solution space grows exponentially. NP-Hard problems, especially those with large solution space are impracticable for exact techniques and often result in combinatorial explosion (Deudon *et al.*, 2018). Heuristics are deployed to circumvent these short-falls. Heuristics do not guarantee optimal solutions but are able to obtain good enough results within the constraint of polynomial time.

Some Combinatorial Optimization Problems are reviewed in the following sub-sections.

2.1.1. The Knapsack Problem

The Knapsack Problem is a famous Combinatorial Optimization Problem applicable to real-life scenarios such as in capital budgeting, bin packing problems, and so on. The knapsack problem is illustrated as follows:

Suppose for instance, that a thief breaks into a shop with a container or a backpack, the problem he needs to solve is to fill his container with an optimal subset of goods or objects or items in the shop. This problem can be modelled mathematically (Martello and Toth, 1990; Kellerer *et al.*, 2004; Peasah *et al*, 2011; Christian and Cremaschi, 2018), if the items in the shop are numbered 1 $to\ n$ with a vector $X_i(i = 1, \dots, n)$ such that;

$$x_i = \begin{cases} 1 & if\ item\ i\ gets\ picked \\ 0 & otherwise \end{cases} \qquad (2.2)$$

Thus, given that p_i is the price on item i, and w_i is the weight of i, and k is the size of the knapsack, the problem is to select the vector x that satisfies the constraint;

$$\sum_{i=1}^{n} w_i x_i \leq k \qquad (2.3)$$

that optimizes the objective function

$$\sum_{i=1}^{n} p_i x_i \qquad (2.4)$$

The Knapsack Problem is a decision problem and can be modelled to suit wide-ranging application area of optimization such as Logistic, Investment, cutting problem and so on. Consequently, there are more than one variant of the Knapsack Problem. The Knapsack problem has equally been adapted as basis for or as subproblems to other Combinatorial Optimization Problems (Kellerer *et al*. 2010; Christian and Cremaschi, 2018). The Knapsack Problem (KP) may either be bounded or unbounded.

KP is said to be bounded, if there exists an upper limit lim_i, (represented as an integer variable) on each possible instance of item i that can be selected in the knapsack (Myasnikov *et al.*, 2015; Frenkel *et al.*, 2016). Thus:

$$\max \sum_{i=1}^{n} p_i x_i \qquad (2.5)$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \leq k \qquad (2.6)$$

$$lim_i \geq x_i \geq 0, x_i\ integral\ for\ all\ i$$

In contrast, there are no bounds on the selection of variable instance in unbounded Knapsack Problems. Thus:

$$\max \sum_{i=1}^{n} p_i x_i \qquad (2.7)$$

$$subject\ to \sum_{i=1}^{n} w_i x_i \le k \qquad (2.8)$$

$$x_i \ge 0, x_i\ integral\ for\ all\ i$$

The Multiple Knapsack Problem (MKP) is another variant of the Knapsack Problem. A Knapsack Problem is referred to as MKP if there exists a set of items n and a set of knapsacks m where each knapsack has an associated capacity k_i (Fukunaga, 2011; Balbal *et al.*, 2015; Martello and Monaci, 2020). Thus:

$$\max \sum_{j=1}^{m} \sum_{i=1}^{n} p_i x_{ij} \qquad (2.9)$$

$$\left.\begin{aligned} &subject\ to \sum_{i=1}^{n} w_i x_{ij} \le k_i, for\ all\ 1 \le j \le m \\ &\sum_{j=1}^{m} x_{ij} \le 1, for\ all\ 1 \le i \le n \\ &x_{ij} \epsilon \{0,1\}\ for\ all\ 1 \le j \le m\ and 1 \le i \le n \end{aligned}\right\} \qquad (2.10)$$

Notable variants of the MKP are the Multiple Knapsack Problem with Identical capacities - MKP-I, Multiple Subset Sum Problem with random capacities given by constraint, the Multiple Subset Sum Problem (MSSP-I) with Identical capacities and constraints (Kellerer *et al.*, 2004).

Other formulated variants of the Knapsack Problem include the Multiple-Choice Knapsack Problem – MCKP (Zhong and Young 2010; Bednarczuk *et al.*, 2018), the Quadratic Knapsack Problem (Fomeni *et al.*, 2020; Schulze *et al.*, 2020), the Subset

Sum Problem – SSP (Jain *et al.*, 2014; Xu *et al.*, 2020), the Multidimensional Knapsack Problem d-KP (Puchinger *et al.*, 2010; Laabadi *et al.*, 2019), the Set-Union Knapsack Problem – SUKP (Kellerer *et al.*, 2004).

From a computational point of view, the Knapsack Problem is intractable, thus the solution approaches include exact techniques (Gupta *et al.*, 2014; Hifi, 2014; Jain *et al.*, 2014; Leão *et al.*, 2014; Fomeni *et al.*, 2020) and approximation methods (Bansal and Deep, 2012; Bednarczuk *et al.*, 2018; Gurski *et al.*, 2019; Laabadi *et al.*, 2019; Martello and Monaci, 2020; Schulze *et al.*, 2020).

2.1.2. The Assignment Problem

The Assignment Problem, also referred to in Graph Theory as the Bipartite Perfect Matching Problem is described as follows:

Given n number of agents assigned to m number of tasks with associated cost. Also given that at most, one agent can be assigned to a task and vice-versa, the problem is to obtain the optimal way of assigning tasks to agents such that they perform as many tasks as possible with minimal associated cost (Singh, 2012; Faudzi *et al.*, 2018). In graph theory, the Assignment Problem is modelled as a weighted bipartite graph with the objective of obtaining the maximum matching, for which the sum of weights of the edges is minimal.

Formally, the Assignment Problem is defined by Silvano (2011) as follows:

Given a $(n \times n)$ cost matric of the ${}^{task}/_{agent}$ assignment $(\mathcal{C}_{ab})$ as in table 2.1.:

Table 2.1: A cost matrix of the ${}^{task}/_{agent}$ assignment

${}^{task}/_{agent}$	1	2	$\dots j$	$\dots n$
1	C_{11}	C_{12}	$\dots j$	$\dots j$
2	C_{21}	C_{22}	$\dots j$	$\dots j$
. . . i	. . . C_{i1}	. . . C_{i2}	... ... $\dots C_{ij}$	... ... $\dots C_{in}$
. . . n	. . . C_{n1}	. . . C_{n2}	... ... $\dots C_{nj}$	... ... $\dots C_{nn}$

$$C_{ij} = \begin{cases} 1 & \text{if row i is alloted to column j} \\ 0 & otherwise \end{cases} \quad (i, j = 1, \dots, n) \qquad (2.11)$$

The problem is to determine which ${}^{task}/_{agent}$ assignment will guarantee the minimum cost of completion of the task. This can be expressed mathematically as:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (2.12)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \; for \; i = 1, \dots, n \qquad (2.13)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \; for \; j = 1, \dots, n \qquad (2.14)$$

$$x_{ij} \in \{0,1\} \; for \; i, j = 1, \dots, n$$

The assignment is referred to as balance if the number of agents is the same as the number of tasks to be assigned, but referred as unbalanced if otherwise. On the other hand, the Assignment Problem is said to be a Linear Assignment if the cost of the assignment for all tasks is the same as the total costs for each agent (Ramshaw and Tarjan, 2012).

The assignment component of the Assignment Problem underlies its combinatorial structure. Thus, the solution approaches include exact techniques such as Integer Programming, Column generation, Hungarian method and so on (Ayorkor *et al.*, 2007; Qu *et al.*, 2009; Salehi, 2014; Shah *et al.*, 2015; Date and Nagi, 2016; Woumans *et al.*, 2016; Lesca *et al.*, 2019).

The Hungarian Algorithm for the $n \times n$ cost matrix to determine the optimal assignment is as follows (Ayorkor, *et al*, 2007; Shah *et al.*, 2015; Date and Nagi, 2016):

i. A bipartite graph $\{V, U, E\}$ $(where\ |V| = |U| = n)$ and an $n * n$ matrix of edge costs C

ii. initialization:

  (a) Start with an empty matching, $M_0 = \phi$.

  (b) Assign feasible values to the variables α_i and β_j as follows:

    1. $\forall v_i \in V,$ $\quad \alpha_i = 0$ (1) (2.15)

    2. $\forall u_i \in U,$ $\quad \beta_j = min_i(c_{ij})$ (2.16)

iii. Do this for n stages of the algorithm,

iv. After the n^{th} stage, output the matching: $M = M_n$.

Algorithmic Presentation of the Hungarian Process:

The flowchart for the Hungarian Method algorithmic stages is depicted in Figure 2.1.

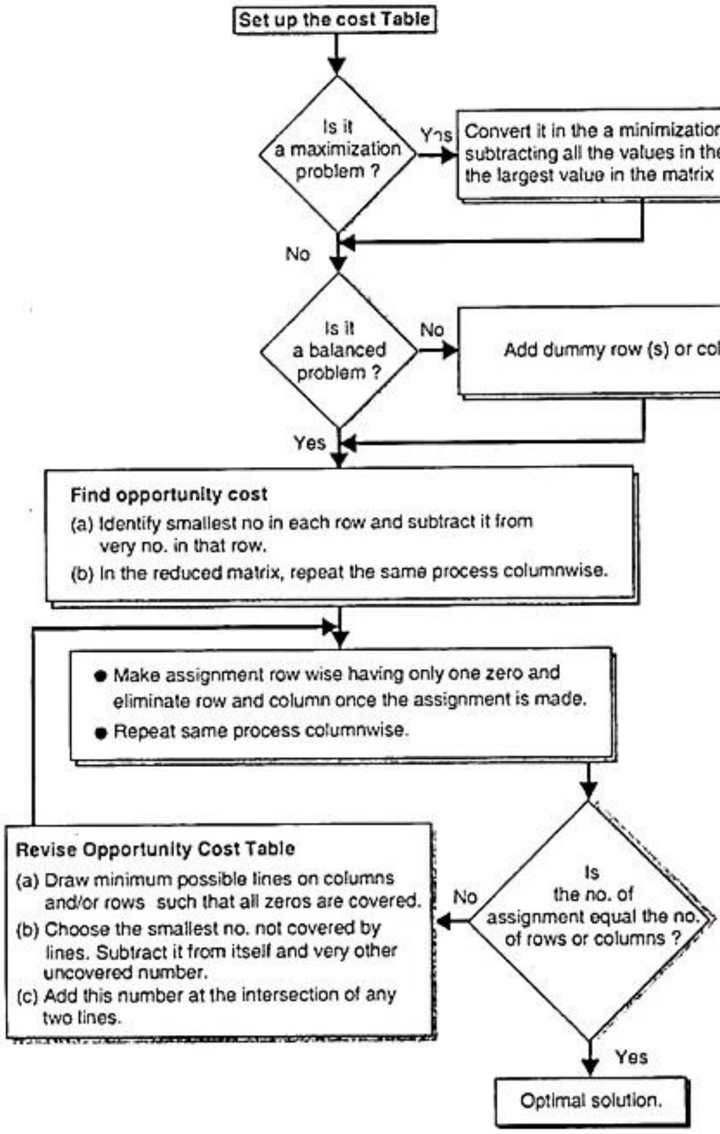

Figure 2.1. Flowchart of the Hungarian Method for solving Assignment Problems (Sengupta, 2017).

1. Every unmatched node in V is designated as the root node of a Hungarian tree.
2. In the equality sub-graph, Hungarian trees are grown at the exposed nodes. The indices i of nodes v_i found in the tree by the set I^*, and the indices j of nodes

u_j found by the set J^* are designated. If an augmenting path is constructed, go to (4), else the Hungarian trees cannot be grown any further, hence go to step (3).

3. New edges are included in the equality sub-graph by modifying α and β variables as;

$$\left.\begin{aligned} \theta &= \frac{1}{2}\min_{i\in I^*\, j\notin J^*}(c_{ij} - \alpha_i - \beta_j) \\ \alpha_i &\leftarrow \begin{cases} \alpha_i + \theta & i \in \mathrm{I}^* \\ \alpha_i - \theta & i \notin I^* \end{cases} \\ \beta_j &\leftarrow \begin{cases} \beta_j - \theta & j \in \mathrm{J}^* \\ \beta_j + \theta & j \notin J^* \end{cases} \end{aligned}\right\} \tag{2.17}$$

Go to step (2) to find an augmenting path.

4. The new matching, M_k (at stage k), is augmented by flipping unmatched and matched edges along the augmenting path selected. That is, $(M_{k-1} - P) \cup (P - M_{k-1})$, where M_{k-1} is the matching from the previous stage and P is the set of edges on the augmenting path selected.

Heuristics techniques such as TABU search, Graph Colouring heuristics, hyper-heuristics and so on are equally used in solving the Assignment Problem (Kaha and Kendall, 2010; Burke *et al.,* 2012; Sabar *et al.,* 2012; Abdul-Rahman *et al.,* 2017; Muklason *et al.,* 2017).

2.1.3. The Constraint Satisfaction Problem

Constraint Satisfaction Problems (CSP) have their root in artificial intelligence dating back to the 1970s, motivated by pioneering works in computer vision (Waltz, 1972; Mackworth 1977). The research scope has since been greatly widened to cover relevant application area in the domain of Artificial Intelligence and Operations Research such as temporal reasoning, scheduling and so on. The objective of the Constraint

Satisfaction Problem is to be able to map a value within a specified finite domain to a variable in such way that it satisfies all the constraint relating to the variable within its domain (Zamani, 2013; Kadri and Boctor, 2018; Sahu *et al.*, 2019). In essence, the solution is valid only when the derived value satisfies all constraints which is within the solution space. Roldán *et al.*, (2011) defined the Constraint Satisfaction Problem as a triple $P = (X, D, C)$, where

- $X = \{x_1, \dots, x_n\}$ is the set of variables within the domain D;
- $D = \{D_1, \dots, D_n\}$ is the set of finite domains containing the solution space for the possible values being searched.
- $C = \{C_1, \dots, C_c\}$ is the set of constraints. A constraint $c_1$is the condition defining the values which the set of variables $\{x_1, \dots, x_n\}$ can take simultaneously. In essence $c_1 \subseteq \{D_{i1}, \dots, D_{ik}\}$. Thus, $\{x_{i1}, \dots, x_{ik}\}$ determines the scope of c_1.

Several combinatorial problems in operations research can be modelled as a Constraint Satisfaction Problem. These include, but not limited to the Graph Colouring Problems, Time-tabling Problem and other resource allocation problems, eight-queens puzzle, the Boolean Satisfiability Problem, Scheduling Problems, Bounded-error Estimation Problems and so on.

Solutions to Constraint Satisfaction Problems on finite domains are characteristically obtained using a search procedure. The most common procedures are some form of backtracking, constraint propagation, and local search.

Most Constraint Satisfaction Problems are combinatorial and are thus NP-Hard. Finding a search solution that satisfies all the constraints will involve enumerating all the search space in exponential time at the worst case (Barto, 2015). Thus, solving them using exact techniques is intractable. Some exact techniques used to solve Constraint Satisfaction Problems include Integer Programming methods, Branch-and-Bound, Branch-and-Cut techniques and so on (Lorterapong and Ussavadilokrit, 2013; Peng *et al.,* 2014; Barto, 2015; Mostafa *et al,* 2015; Sitek and Wikarek, 2016). Heuristic and Metaheuristic methods have been equally deployed in solving the Constraint Satisfaction Problems (Roldán *et al.,* 2011; Zamani, 2013; Kadri and Boctor, 2018; Rutishauser *et al.,* 2018; Sahu *et al.,* 2019).

2.1.4. The Travelling Salesman Problem

The Travelling Salesman Problem is a vastly researched Combinatorial Optimization Problem. Its origin can be traced to the pioneering work in the 1800s of mathematicians W.R. Hamilton and Thomas Kirkman. Hamilton formulated a puzzle problem with the objective of completing a Hamiltonian cycle (Tutte, 2012). Works on the TSP was further enhanced in the 1930s by Karl Menger and M.M. Flood. Karl Menger defined the TSP and did some pioneering works on Brute-Force techniques as well as the Nearest Neighbour Heuristic. M.M. Flood formulated the TSP mathematically to solve the School Bus path finding problem.

The TSP is a shortest tour (or path) optimization problem with the objective to find the shortest route while visiting a set of cities (or nodes), ensuring each city (or node) is visited exactly once and regarding the Hamiltonian circuit, return to the start node or city. It is assumed that the cost of the distance between any pair of cities is predefined. In this regard, the cost often refers to distance but may represent other notions such as time or money. Given a complete weighted undirected graph $G\ (V, E)$, A Hamiltonian cycle refers to a graph cycle that traverses all the graph's vertices exactly once before returning to its starting vertex. The Travelling Salesman must traverse cities $1\ to\ n$ in a Hamiltonian cycle that is; Start from city 1, traverse the remaining $n-1$ cities in a specified order and then connect back to the starting city, having touched each of the cities only once at a minimal cost.

The distance $d(a,b)$ depicts the distance from the city $a\ to\ b$. Thus TSP is formally defined as below;

$$F = min \sum_{a=1}^{n}\sum_{b=1}^{n} d_{ab}x_{ab} \qquad (2.18)$$

$$\sum_{b=1}^{n} x_{ab} = 1; a = 1, \ \ldots, n \qquad (2.19)$$

$$\sum_{a=1}^{n} x_{ab} = 1; b = 1, \ \ldots, n \qquad (2.20)$$

The objective function is marked with F. With a limitation,

$x_{a_1a_2} + x_{a_2a_3} + \ \ldots + \ x_{a_ra_1} \leq r-1.$

$x_{ab}\, x_{ab}$ are the binary variables

$$x_{ab} = \begin{cases} 1 & if\ the\ salesman\ travels\ from\ city\ a\ to\ city\ b \\ 0 & if\ the\ salesman\ is\ not\ travelling\ from\ city\ a\ to\ city\ b \end{cases} \qquad (2.21)$$

d_{ij} is the cost of moving from city *a* to city *b*.

The TSP has applications in several areas, most especially in varying areas of transportation. Being an NP-hard problem, which is easily understood but computationally difficult to solve, the TSP has several solution algorithms broadly categorized into Exact Algorithms and Approximate Algorithms (heuristics).

The Travelling Salesman Problem is classified as either symmetric (STSP) or asymmetric (ATSP) problems, depending on whether the distance or cost to travel between any two cities is symmetric or asymmetric, respectively. In many cases, the STSP is seen as a subproblem of the ATSP but there are cases where the STSP and the ATSP are defined on separate graphs, that is, complete directed and undirected graphs. However, the ATSP can be converted to STSP by doubling the number of nodes in the given graph. There is an extension of the ATSP called multiple asymmetric travelling salesmen problem (mATSP), which requires the collective performance of multiple salesmen in touring each city exactly once at a minimal total cost. These categories of the TSP have been discussed below.

2.1.4.1.Symmetric Travelling Salesman Problems

The Travelling Salesman Problem is said to be Symmetric if the travel cost is the same between two nodes in both directions, that is, $d_{ab} = d_{ba}$, thereby reducing the number of possible solutions to half the initial (Hussain *et al.*, 2017; Arthanari and Qian, 2018). The STSP has half the solution space of the ATSP, thus it is considered as the more basic form of the TSP and often solved as benchmark cases for the TSP.

Exacts techniques such as Branch-and-Bound, Branch-and-Cut, Mixed Integer Linear Program, Dynamic Programming, Held Karp Algorithms and so on have been used to obtain optimal solutions for the STSP (Chauhan, 2012; Demez, 2013; Fischer *et al.,* 2014; Sundar and Rathinam, 2017; Dijck, 2018). Approximate methods have also been deployed in solving the STSP. They include heuristics such as NNH, Lin-Kernighan, Savings, k-opt techniques and so on. Metaheuristics methods include Simulated Annealing, local search, genetic techniques and so on (Demez, 2013; Fosin *et al.,* 2013; Kızılateş, 2015; Lim *et al.,* 2016; Hussain *et al.,* 2017; Kovács*et al.,* 2018).

Formulations for the STSP include the Dantzig, Fulkerson and Johnson formulation (Dantzig *et al.*, 1954), the Bellman formulation (Bellman, 1962), the Held-Karp formulation (Held and Karp, 1970), the Multistage Insertion formulation (Arthanari, 1983; 2000) and so on.

2.1.4.2.Asymmetric Travelling Salesman Problems

The Asymmetric Travelling Salesman Problem (ATSP) belongs to the class of NP-Hard Problems concerned with finding the distance from one point to another in a given space which differs from the inverse distance. There are various instances where the ATSP can be applied; for instance, in a vehicle routing problem, a delivery man uses a vehicle to travel through one-way streets in a city or minimizing the cost of petrol while driving through mountain roads. According to (Roberti and Toth, 2012), the ATSP can be defined formally as follows:

Given a directed graph $G = (V , A)$, where $V = \{1, \dots, n\}$ is the set of vertices, and $A = \{(i, j) : i, j \in V\}$ is the set of arcs, The ATSP is a non-symmetric cost matrix (Cij) which is defined on A.

Many ATSP formulations consist of an assignment problem with integrality and subtour elimination constraints. Such formulations include, the Dantzig, Fulkerson and Johnson (DFJ) formulation (Dantzig et al., 1954), the Fox, Gavish and Graves (FGG) formulations (Gavish and Graves, 1958), the Desrochers and Laporte (DL) formulation (Desrochers and Laporte, 1991), the Gouveia and Pires (GP) formulations (Gouveia and Pires, 1999) and the Sherali and Driscoll (SD) formulation (Sherali and Driscoll, 2002).

The ATSP has been solved using both the exact solution approaches (Ahmed, 2011; Roberti and Toth, 2012; Aguayo *et al.,* 2016; Campuzano *et al.*, 2020) and the approximate solution approaches (Arash *et al.,* 2010; Hyung-Chan *et al.,* 2010; Nima and Shayan, 2015; Barketau and Pesch, 2016; Basu *et al.*, 2017; Svensson *et al.*, 2018).

2.1.4.3.Multiple Travelling Salesman Problems

The Multiple Travelling Salesman Problem (mTSP) more adequately models real-life scenarios, as it can handle one or more salesmen. The mTSP contains a set of nodes, *m* salesmen at a base node, and the remaining nodes to be visited which are intermediate nodes. The mTSP finds tours for all *m* salesmen such that every intermediate node is visited exactly once and the total cost of visiting all nodes is minimized. A more detailed definition of the mTSP is: Given a graph with vertices V in which city, i, denotes the base city, an asymmetric distance matrix $[cij\ ], i;\ j\ \in V,$ and *m* salesmen located at the base city, determine *m* tours that start and end at the base city after collectively having visited city i exactly once, $\forall i\ \in V,$ while minimizing the total distance travelled (Cuevas *et al.*, 2020).

The Multiple Travelling Salesman Problem can be modelled as a relaxation of the Vehicle Routing Problems (VRPs) when side constraints are incorporated. Because of its amenability to real-life scenarios, a number of variations of the mTSP have been formulated in literature (Baranwal *et ai.,* 2017; Neos, 2018). They include the Non-Returning Multi-Travelling Salesmen Problem (Tang *et al.,* 2000), Returning Multi-Travelling Salesmen Problem (Gorenstein, 1970), Single-Depot Returning Multi-Travelling Salesmen Problem (Baranwal *et al.,* 2016), Multiple-Depot Returning Multi-Travelling Salesmen Problem (Oberlin *et al.,* 2009), Close Enough Travelling Salesmen Problem (CETSP) (Mennell, 2009; Assaf and Ndiaye, 2017).

The mTSP has been solved using both the exact solution approaches (Baranwal *et al.,* 2016; Assaf and Ndiaye, 2017; Baranwal *et al.,* 2017; Thenepalle and Singamsetty, 2019) and the approximate solution approaches (Shim *et al.,* 2012; Labadie *et al.,* 2014; Liu and Zhang, 2014; Necula *et al.,* 2015; Qing *et al.,* 2015; Shuai *et al.,* 2019).

2.2. Variants of the Travelling Salesman Problem

Several variants of the Travelling Salesman Problems have been studied by researchers, some of which have been considered as follows.

2.2.1. The Maximum Travelling Salesman Problem (MAX TSP)

The Maximum Travelling Salesman Problem (MAX TSP) sometimes referred to as the "taxicab ripoff problem" (Dudycz *et al.,* 2017), finds a Hamiltonian circuit with maximum total edge weight and uses its additive inverse to replace each cost edge (Jawaid and Smith, 2013). The MAX TSP is NP-hard, hence there exist some constants $\beta < 1$ such that obtaining a solution that guarantees better performance than β is NP-hard (Hassin and Rubinstein, 2000). If non-negative edge costs are required in the tour, it is possible to assign a constant to each of the edge costs with no effect to the optimal solutions of the problem edge (Punnen, 2007). Barvinok *et al.,* (2007) defined the MAX TSP as follows:

Given a weight matrix $w = w_{ij}$

The objective of the MAX TSP is to find a Hamiltonian cycle $i_0 \rightarrow i_2 \rightarrow, \dots, \rightarrow i_n \rightarrow i_1,$ for which the maximum value of $w_{i_1 i_2}+w_{i_3 i_4}+, \dots, ++ w_{i_{n-1} i_n} + w_{i_n i_1}$ is obtained, where ($i_1, \dots, i_n$) is the set of all possible combination of $\{1, \dots, n\}$.

The MAX TSP is unique because it contains some weights that the sign reversal does not preserve which are interesting and natural special cases. Also, some combinatorial and geometric problems can use MAX TSP methods.

The MAX TSP has been solved as a variety of related problems such as the Maximum Travelling Salesman Path Problem - Max TSPP (Monnot, 2005; Jawaid and Smith, 2015), the Maximum Scatter TSP (Hoffmann *et al.,* 2017; Kozma and Mömke, 2017;

Venkatesh *et al.*, 2019), the Maximum Metric Symmetric TSP (Kowalik and Mucha, 2007; 2008) the Maximum Latency TSP (Hassin *et al.*, 2009; Alamdari *et al.*, 2013) and so on.

Researches have deployed both the exact and approximate solution approach to the MAX TSP. Barvinok *et al.* (2003) derived polynomial-time algorithms in which the cities represent nodes of $\mathbb{R}^d$ for given distances d, computed based on either the polyhedral norm or quasi-norm; the computational time for the k-facet polyhedral was $O(n^{k-2} \log n)$. The solution was equally extended to solve the quasi model with a computational time of $O(n^{2k-2} \log n)$. The solution was then extended to solve the Tunnelling TSP as a derivative of the MAX TSP. Given a set $T = \{t1, t2, \ldots, tk\}\ of\ k \geq 2$ auxiliary objects the distances are computed using a special "*tunnel system"* distance function where all tunnels are bidirectional.

The approximate methods were able to obtain close approximations of the optimal solutions in polynomial time (Sergeev, 2014; Hoffman, 2016; Kozma and Momke, 2016; Dong *et al.*, 2017; Venkatesh *et al.*, 2019).

2.2.2. The Bottleneck TSP (BTSP)

The Bottleneck TSP is a special case of the Travelling Salesman Problem that obtains a tour that traverses each city exactly once with the objective of minimizing the farthest distance between any two adjacent cities on the tour. Given a weighted graph G, the objective of the BTSP is to keep the weight w of the weightiest edge w_{ab} as minimal as possible (Kao and Sanghi, 2009). Thus, the integer programming formulation of the BTSP is defined (Kabadi and Punnen, 2007; LaRusic, 2010) as follows:

$$Minimize \quad max\{w_{ab}x_{ab}, 1 \leq a, b \leq n, a \neq b\}$$

$$Subject\ to:$$

$$\sum_{a=1}^{n} x_{ab} = 1, b \in N \qquad (2.22)$$

$$\sum_{b=1}^{n} x_{ab} = 1, a \in N \qquad (2.23)$$

$$x_{ab} = 0\ or\ 1$$

$$\sum_{a \in S} \sum_{b \in \bar{S}} x_{ab} \geq 1 \forall S \subset N, \qquad (2.24)$$

$$where\ \bar{S} = N \backslash S.$$

The BTSP can be classified as either Symmetric or Asymmetric BTSP, depending on the nature of the cost matrix. The BTSP is Euclidean (EBTSP) if the tour costs from node to node is Euclidean. Other variants of the BTSP include the Constrained BTSP and the Maximum Scatter Travelling Salesman Problem (MSTSP). The Constrained BTSP places an additional restriction on the total weight of the tour (Malawski *et al.*, 2013; Van den Bossche *at al.*, 2013; Gahir, 2014; Wang *et al.*, 2016). The MSTSP obtains a tour T that traverses each nodes of the weighted graph G with the objective of maximizing the shortest edge in G (Hoffmann *et al.*, 2017; Kozma and Mömke, 2017; Venkatesh *et al.*, 2019).

Some application areas of the BTSP include the Assembly line sequencing, sequencing a One-State Variable Machine, Reconstructing Sequential Orderings from Inaccurate Adjacency Information and Sequencing Rivet Operations (LaRusic, 2010).

Researches have deployed both the exact and approximate approach in solving the BTSP. The approximate methods aim to obtain close approximations of the optimal solutions in polynomial time. For instance, LaRusic (2010) developed an approximate solution for the Symmetric Bottleneck Travelling Salesman Problem on a given graph G with cost matrix C. This was based on the assumption that a lower bound L had been computed on the optimal BTSP objective value using the "*Bottleneck Biconnected Spanning Subgraph Problem*" (BBSSP) lower bound. Extensive computational results were presented for problems of up to 31,623 vertices and the heuristic algorithm was able to obtain optimal solutions for almost all problems considered within a very reasonable computational time; this was achieved using randomization in a controlled way to guide the heuristic search. Helsgaun (2014) solved the BTSP with a Lin-Kernighan-Helsgaun (LKH) Algorithm. The author used the "1-tree approximation" technique to determine a possible edge set, then he deployed an extended search technique, and finally, outliers were pruned. The performance of the LKH was evaluated on a large BTSP test set, it found optimal results on instances with as much as 115,475 nodes in a reasonable time. With some modifications made, the LKH was able to solve BTSP instances of as much as one million nodes. Others such as (Kao and Sanghi, 2009; Ahmed, 2013; Pelaez *et al.*, 2016; Abdi *et al.*, 2017; Zhang and Sun 2017) reported encouraging performance of approximate techniques in solving the BTSP.

2.2.3. The Travelling Salesman Problem with Multiple Visits (TSPM)

As the name implies, the Travelling Salesman Problem with Multiple Visits (TSPM) finds a Hamiltonian tour that visits each nodes of the graph G more than once and complete the cycle at minimal cost. This in contrast to the classic Travelling Salesman Problem which must visit each node exactly once. Punnen (2007) showed that the

TSPM can be transformed to a classic TSP for a weighted graph G if the edge costs are substituted with the shortest path distances in G. Thus, given that the cycle is non-negative, it is possible to determine the shortest path distances between all pairs of vertices in G through high performing algorithmic techniques. In the event where G produces a non-negative cycle, the TSPM is said to be unbounded. Also, Oberlin *et al.*, (2009) and (Assaf and Ndiaye, 2017) converted a Multiple Depot TSPM (MDMTSP) into a Single, Asymmetric TSP. Oberlin *et al.*, (2009)'s work was premised on the condition that the cost of the edges satisfies the triangle inequality which was an improvement on the 2-Depot TSPM conversion earlier designed. A modified LKH heuristic was applied to test some computational results to determine the effectiveness of the conversion made for instances involving Dubins vehicles. The LKH heuristic was used because it is one of the best available solvers for the single Asymmetric TSP on the transformed graph. The computational results on instances test showed that the transformation was highly effective and produced quality, feasible solutions for large instances involving 50 Unmanned Aerial Vehicles and 500 targets in less than 20 seconds. Also, the cost of generating the feasible solution was on an average of about 3% away from its optimum.

An Open-Close Multiple Travelling Salesmen Problem with Single Depot (OCMTSP) was also proposed by (Thenepalle and Singamsetty, 2019) whereby all the salesmen are positioned at the base city to generate an optimal route such that all salesmen start from the base city and then visit a given set of cities exactly once but only the internal salesmen have to return to the depot city whereas the external ones need not return. An exact pattern recognition-based Lexi-Search Algorithm (LSA) was deployed to find optimal solutions for the simulated problem. Computational experiments were carried out, using arbitrarily generated test sets for OCMTSP. The performance of the LSA was evaluated and results indicated that the proposed technique was an efficient method in generating optimal and feasible solutions within reasonable times.

2.2.4. The Clustered TSP (CTSP)

In the Clustered TSP, the nodes (vertices) in a graph G are distributed into clusters $(Vi, V2, \ldots, Vn)$, the objective is to find a Hamiltonian tour in each cluster with optimum cost, ensuring all nodes within the same cluster are traversed contiguously. According to Punnen, (2007), the CTSP can be transformed to a classic Travelling Salesman Problem by adding a maximum cost M to the cost of each inter-cluster edge. Like the study carried out by (Ahmed, 2011), Potvin and Guertin (1996) proposed a genetic technique to solve the CTSP. The genetic algorithm used a sequence of integers, each integer representing a node, and new orderings from old ones were produced using specialized crossover and mutation operators. Problems with 500 vertices were used in the computational experiment performed on the genetic algorithm and it was able to solve them with an optimality of 5.5%. A computational comparison was also carried out on the proposed algorithm and the GENIUS heuristic. The results obtained showed that the proposed algorithm outperformed the GENIUS heuristic. Bazylevych *et al.*, (2007) suggested decomposition algorithms for solving CTSP which allow a considerable amount of decrease in the computation time. The CTSP model studied in this work was categorized into macro-modelling, micro-modelling, finding initial route and route optimization. Optimization of the route S_0^* was realized by means of the iterative improvement with minimization of its total distance: $D_0^* \rightarrow D_1^* \rightarrow D_2^* \rightarrow, \ldots, \rightarrow D_c^*$. Local and global optimization were the optimizations considered. The local optimization was obtained by using Scanning algorithm, which is an algorithm that scans (or finds) optimal or very good solution to some sub-problems of the whole problem. On the other hand, the global optimization was arrived at by the iterative revision of the whole route. Ahmed (2014) proposed a heuristic technique to solve the ordered CTSP. The technique was a hybrid genetic algorithm, with integrated modules such as sequential constructive crossover, 2-opt search, and local search. The initial sample space was generated using sequential sampling technique. The technique was experimented on some benchmark instances from TSPLIB. The efficiency of the technique was evaluated and compared with the exact partitioning algorithms. It was observed that the developed algorithm outperformed the other techniques based on quality of solutions and computational speed. Furthermore, the developed algorithm obtained optimal solutions for the instances with as much as 51 nodes.

Other variants of the Travelling Salesman Problem that have been formulated and solved in literature include, the Time-dependent TSP, the Black and White TSP, the Period TSP, the Resource constrained TSP, the Selective TSP, and the Angle TSP (Abeledo *et al.,* 2010; Godinho *et al.,* 2014; Arigliano *et al.,* 2018; Keskin *et al.,* 2019).

2.3. TSP Solutions

The Travelling Salesman Problem is relevant to several domain of knowledge and practices. Apart from the popular transportation and vehicle routing problems, the TSP is applied in the drilling and mask plotting of Printed Circuit Boards (PCB), overhauling gas turbine engines, X-Ray crystallography, Computer wiring, order-picking problem in warehouses and so on (Matai *et al.*, 2010). Being an NP-hard problem, which is easily understood but computationally difficult to solve, the TSP has several solution algorithms broadly categorized into Exact Algorithms and Approximate Algorithms. Solving TSP using Exact techniques involve the explicit enumeration of the solution space. Exact techniques guarantee optimal solutions at least hypothetically. However, as the solution space increases, the computational complexities of these techniques become exponential in nature and are thus impracticable and unsuitable for NP-hard problems with large solution space. Approximate techniques on the other hand guarantees good enough solutions within the constraint of polynomial time p. Some exact and approximate techniques are reviewed in the following subsections.

2.3.1. Exact Methods

Exact techniques, when used in finding solutions to TSPs try out all possible permutations of the solution, thus they have a complexity of $O(n!)$. Exact techniques such as Dijkstra or Bellman-Ford algorithms may be deployed to efficiently solve TSPs with small degree of search space (Giovanni, 2017). More complex problems, however, may require that the problem be first modelled as a Mixed Linear Programming (MILP) paradigm, before solving them using any suitable MILP solver such as Cplex, Gurobi, Xpress, AMPL, OPL and so on. While, exact methods can potentially generate optimal tour, especially in theory, they are often impracticable and especially unsuitable for NP-hard problems with large solution space. For instance, for a TSP of as little as $10\ nodes$, the execution time is about 3628800 which is impractical (Abdulkarim and Alshammari, 2015). The solution renown as the best performing exact technique is based on dynamic programming with a complexity of $O(2^n n^2)$, thus making it impracticable to solve TSP as the search space expands (Deudon *et al.*, 2018). This is a result the complexity of TSPs, and the constraint of time.

Some exact and approximate techniques are reviewed in the following subsections.

2.3.1.1.The Brute Force Algorithm

The Brute Force technique involves the explicit enumeration of the solution space. Brute force obtains an optimal tour by exploring the entire search space and building all the possible solutions. Although the Brute Force technique is simple to implement and guarantees optimal solution, it is however a naive approach, because it chooses the optimum solution from a wide search space of all possible solutions, thus in the worst case, the complexity expands exponentially until it becomes impracticable in polynomial time P, (Baidoo and Oppong, 2016).

The following are the stages involved in obtaining optimal solution by the brute-force technique (Saiyed, 2012):

1. Explore all the solution space.
2. Enumerate and plot all the feasible tours.
3. Compute the tour cost of each of the solutions.
4. Select the shortest tour.

The following pseudocode depicts the brute-force function:

```
Algorithm 2.1: Brute Force function
Input: Q: a TSP query of a set of points
Output: T: the TSP for Q
Generate first tour solution, TS
OptTour ← TS
OptCost ← Cost(TS)
while there exists more permutations of TS do
      generate a new permutation of TS
      if Cost(TS) < OptCost then
            OptTour ← TS
            OptCost ← Cost(TS)
      end if
end while
print OptTour and OptCost
```

$OptTour$ is optimal tour, $OptCost$ is the cost of the optimal tour.

Kolog (2012) compared the optimality of the brute force algorithm with the Tabu search algorithm for TSP. Results obtained from computational experiments indicated that the brute force algorithm outperformed the Tabu search as it produced a significantly high optimal solution but it could only work effectively in solving TSPs with less than 10 nodes compared to the TABU search which could find a solution without stern complexities.

Baidoo and Oppong (2016) performed a comparative evaluation of the Brute Force algorithm, the Greedy algorithm, the Branch-and-Bound technique, the Dynamic programming technique and the Nearest Neighbor Heuristic for solving TSP, with a focus on the distance traveled, execution time and effectiveness of these algorithms. Four test instances were used in the evaluation process, the Brute force approach gave the best results in all four instances but had a relatively low computational speed while the Nearest Neighbor Heuristic had the fastest computational speed but produced approximate values in all test instances. Conversely, the Dynamic programming algorithm produced optimal solutions within a considerable execution time. Regarding the given criteria, the researchers considered Dynamic programming as the best among the five algorithms.

2.3.1.2.The Branch-and-Bound (BB) Algorithm

The Branch-and-Bound algorithm is a decision technique for solving Combinatorial Optimization Problems. Given a list of vertices and a distance matrix, the Branch-and-Bound solution process breaks the problem into smaller sub-groups represented in a Branch-and-Bound tree. Dead nodes of the tree which cannot be further expanded are jettisoned based on the criteria set for the upper and lower bound approximation constraint. The upper bound is determined by first generating an initial solution and designating the solution cost as the upper bound. This is maintained recursively until a lower solution cost is generated (Baidoo and Oppong, 2016). The Branch-and-Bound (BB) solution process may be viewed as a mathematical model with a modular approach of initial constraint relaxation and incremental enumeration of solution. The quality of the bound determines the quality of the Branch-and-Bound technique (Matai *et al.*, 2010).

The Branch-and-Bound algorithm is called an "exact" algorithm because it guarantees an optimum solution, although it takes a lot of time (Chatting 2018). The Branch-and-Bound technique generally go through three procedures vis-à-vis Splitting, Bounding and Pruning. The steps performed by the Branch-and-Bound algorithm were enumerated by Chatting (2018) as follows:

Algorithm 2.2: Branch-and-Bound Algorithm

Input: Q: a TSP query of a set of points
Output: T: the TSP for Q

1. **Assign** a **bounding criterion** and **calculate** an overall **lower bound;**
2. **Set** an initial city, e.g., ***city* 1;**
3. **Evaluate** valid neighbours adjacent to the current city;
4. **Prune** any branches which now exceed the **bounding criterion;**
5. **Repeat** steps 3 and 4 until all branches reach a **'leaf';**
6. **Identify the optimum solution from those remaining.**

Hazra and Hore (2016) performed a comparative performance study on the algorithms for solving the TSP, namely Branch-and-Bound, Backtracking, and Dynamic Programming. The major factor for the comparison was the average running time of all three algorithms for solving TSPs of varying sizes. From the analysis, the Branch-and-Bound had a lesser running time than the Backtracking algorithm as it ignores sub-problems that are unproductive while the backtracking takes into consideration every possible path in solving the TSP problem. Although both the Backtracking and the Dynamic Programming algorithms are recursive, the Dynamic Programming had the least running time cost and gave the most optimal paths.

Droste (2017) studied the Branch-and-Bound and Ant Colony Optimisation algorithmic solutions for the Travelling Salesman Problem. The Branch-and-Bound algorithm was implemented using four different approaches. It was established that the addition of the constraints in order of increasing length instead of in lexicographic order is better for branching. While for bounding, the lower bound that adds up the two smallest allowed edges of each city performed better than a lower bound based on a $1 - tree$. The biggest instance for which an exact solution was found consisted of 23 cities. Also, the Ant Colony Optimisation algorithm was implemented, and results showed that for smaller instances (lesser than 100 cities), the algorithm performed well.

2.3.1.3.The Branch-and-Cut (BC) Algorithm

In the Branch-and-Bound algorithm, both cutting-plane and enumerative phases are separated, hence update about the existing partial linear definition of inequalities cannot be manipulated at the enumeration phase. Additionally, if the BB method terminates with a sub-tour solution, the whole enumerative procedures must be restarted from the beginning. Therefore, the branch-and-cut algorithm was developed to help overcome these loopholes of the BB algorithm due to its inflexibility (Padberg and Rinaldi, 1991). The BC algorithm is a combination of the BB algorithm and the cutting plane method. The BC algorithm is said to simultaneously compute for a series of increasing lower and decreasing upper bounds. In a situation where both coincide, the optimality of the feasible solution is proven and even if this does not occur, the bounds help to proffer quality guarantee on the best solution (Ascheuer *et al.,* 1999).

The first procedure of the Branch-and-Cut technique is the initialization stage where the linear programming relaxation for the problem is defined. In this phase, a cutting plane method is iteratively deployed until the termination criteria is reached, that is no more inequalities. The best solution of this phase is stored as the initial solution. The next phase is the Branching phase. Here, a binary branching (0 or 1) of the fractional variable is carried out to generate two new nodes. In the third phase, a new linear programming relaxation is introduced and deployed iteratively until the enumeration process is complete. During the iteration, the optimal solution module is updated with solutions with better cost, but left unchanged if otherwise (Dijck, 2018).

Dumitrescu *et al.* (2010) solved a simulated integer linear programming TSP with Pickup and Delivery (TSPPD) using a separation procedure involving a Branch-and-Cut technique. The computational results obtained indicated that the BC algorithm could find optimal solutions for instances with up to 35 pickup and delivery requests.

Battarra *et al.*, (2014) carried out a performance study on three variations of the Branch-and-Cut technique. They are the Branch-and-Cut algorithm with a compact formulation that considers two sets of two-index binary variables and a polynomial number of constraints, the Branch-and-Cut algorithm with a formulation that considers three index variables, and the Branch-Cut-and-Price with a path interpretation of the preceding formulation. When enhanced with sub-tour elimination and trivial constraints, the first formulation is not empirically dominated, the second formulation was proven to have theoretically and empirically dominated the previous, while the third dynamically introducing ng-paths to the formulations to generate columns. It was observed that the third algorithm could find optimality for all the benchmark instances used.

2.3.1.4. The Branch-and-Price (BaP) Algorithm

The Branch-and-Price is a high performing exact method based on integer programming for solving the Travelling Salesman Problem (Christiansen *et al.,* 2013; Gendreau *et al.,* 2014). The technique employs a similar approach of integer relaxation as the Branch-and-Cut technique. However, in the BaP technique, the rows are excluded, and column generation is emphasized. A large portion of feasible solutions, represented by the columns, most of which have insignificant associated variables for obtaining optimal solutions are excluded. This limits the number of columns that require efficient handling to manageable sizes. Thus, column generation can be applied throughout the Branch-and-Bound tree (Barnhart, 1998). The BaP typically consist of two subproblems, namely the master and the pricing. The pricing subproblem is solved to evaluate profitable columns whose cost is minimal, after which the integer programming is then reoptimized. This procedure is done iteratively until the condition for branching is reached, such that no more profitable columns are obtained (Savelsbergh, 2001; Feillet *et al.,* 2010).

Jepsen (2011) solved the Vehicle Routing Problem using a hybrid technique called the Branch-and-Cut-and-Price. The problem was formulated as a Mixed-integer Programming model. The edges of the VRPTW were assigned a fixed cost for the pilot test and was experimented on an instance of 50 nodes. The technique outperformed CPLEX and obtained solution within a reasonable time.

Kozanidis, (2018) modelled an aircraft routing problem as a TSP and solved it using the Branch-and-Price algorithm. Only optimal air-routes were considered and fed into the master process iteratively. The experimental results showed a promising performance by the model.

2.3.1.5. The Cutting Plane Algorithm

Just like the branching techniques, the Cutting Plane technique belong to a class of integer programming solution protocols in which a LP relaxation of the problem is tightened and improved through the introduction of Cutting Planes (Stratopoulos, 2017). Any problem that can be reduced to integer programming can be solved by the cutting-plane method. The technique solves some Integer Programming relaxation by minimizing the cost of the solution space through a process of iterative refinement:

$$Min\ c^T x, subject\ to\ x \in S; \qquad (2.25)$$

The Cutting Plane technique can obtain approximate solutions for complex problems where optimal solutions cannot be obtained (Applegate *et al.*, 2001; Mitchell, 2008).

2.3.1.6. The Dynamic Programming (DP) technique

The Dynamic Programming (DP) method is an Optimization technique that finds optimal or feasible solutions to Optimization Problems including the Travelling Salesman Problem. The DP solution process involves recursively breaking problem into simpler manageable modules or "sub-problems" and recursively solving them optimally. This DP technique is able to efficiently deals with iterative computations or processes by the process called "*memoization"*. This involves storing sub-solutions into a table. Dynamic programming requires a very smart formulation of the problem and simple thinking (Baidoo and Oppong, 2016). An essential feature of the dynamic programming technique, as described by Fachini and Armentano (2018), is to model the Optimization Problem in *phases* adaptable to an *"optimal sub-structure"* and recursively generate optimal sub-solutions which is then mapped and updated per iteration using a "*resource extension function"*.

Allaoua (2017) integrated Genetic Algorithm (GA) with Dynamic Programming (DP) to solve the TSP. From the experimental results performed on some test instances, it was observed that the combined GA-DP algorithm significantly minimized the computational effort, produced an improved solution quality of the GA, and avoids early premature convergence of GA.

A DP algorithm for TSP is given below (Bouman *et al.*, 2018):

Algorithm 2.3: A Dynamic Programming Algorithm for TSP

Input: Set of cities $\boldsymbol{V}$, an arbitrary city $\boldsymbol{v \epsilon V}$, and cost function $\boldsymbol{C}$.
Output: T: the TSP for V

Initialize $\boldsymbol{D_{TSP}}$ **with values** ∞;
Initialize a table $\boldsymbol{P}$ **to retain predecessor cites;**
Initialize $\boldsymbol{v}$ **as an arbitrary city in** $\boldsymbol{V}$;
Foreach $\boldsymbol{w \epsilon V}$ **do**
    $\boldsymbol{D_{TSP}}(\{\boldsymbol{w}\}, \boldsymbol{w}) \leftarrow \boldsymbol{v}$;
    $\boldsymbol{P}(\{\boldsymbol{w}\}, \boldsymbol{w}) \leftarrow \boldsymbol{v}$;
    For $\boldsymbol{i} = 2, \dots, |\boldsymbol{V}|$ **do**
        For $\boldsymbol{S} \subseteq \boldsymbol{V}\ \boldsymbol{where}\ |\boldsymbol{S}| = \boldsymbol{i}$ **do**
            For $\boldsymbol{w \epsilon S}$ **do**
                $\boldsymbol{z} \leftarrow \boldsymbol{D_{TSP}}(\boldsymbol{S} \backslash \{\boldsymbol{w}\}, \boldsymbol{u}) + \boldsymbol{c}(\boldsymbol{v}, \boldsymbol{w})$;
                if $\boldsymbol{z} < \boldsymbol{D_{TSP}}(\boldsymbol{S}, \boldsymbol{w})$ **then**
                    $\boldsymbol{D_{TSP}}(\boldsymbol{S}, \boldsymbol{w}) \leftarrow \boldsymbol{z}$**;**
                    $\boldsymbol{P}(\boldsymbol{S}, \boldsymbol{w}) \leftarrow \boldsymbol{u}$**;**
                end if;
            end loop;
        end loop;
end loop.
return path obtained by backtracking over cities in $\boldsymbol{P}$ starting at $\boldsymbol{P}(\boldsymbol{V}, \boldsymbol{v})$;

2.3.1.7.The Dijkstra's Algorithm

The Dijkstra's algorithm helps to solve Optimization Problems by considering node weight when computing the shortest path. It has an algorithmic complexity of $O(n^2)$. The Dijkstra's algorithm has several advantages which include obtaining the shortest path every pair of vertices, between two vertices through several nodes specific, and from a given vertex to all other vertices (Ratnasari, 2013).

Nath (2016) developed Dijkstra's and bitonic algorithms to help solve the TSP. For test instances of small and medium sizes, optimal solutions were obtained. However, for test instances of larger sizes, the proposed bitonic approach generated the best feasible solutions. Therefore, the proposed bitonic approach outperformed the Dijkstra's algorithms and was concluded to be an efficient methodology for the TSP. however, it was also observed that the bitonic algorithm had lesser computational speed in comparison to the Dijkstra's algorithm as the test instance size increased.

Syahputra (2016) simulated a logistic system using the Travelling Salesman Problem and solved the problem using the Dijkstra algorithm. The technique was experimented on an instance of 60 nodes. From the computational results, the Dijkstra's algorithm gave 100% accuracy in solving the TSP.

Ginting *et al.,* (2019) modelled the efficient delivery of items by logistic companies as a classic Travelling Salesman Problem. They obtained an optimal solution using a modified Dijkstra technique. The Dijkstra algorithm was modified to recognize the priority of some clusters of routes based on their distance and weight. The experimental outcome of the method on some instances yielded a comparative efficiency of 47.8% and with the computation time of 48.1%. This showed that the modified method with priority outperformed the state-of-the-art Dijkstra technique.

2.3.1.8.The Bellman-Ford Algorithm

The Bellman-Ford Algorithm also known as the Ford-Fulkerson Algorithm is a dynamic programming technique that extends the Dijkstra technique by including negative node in its computations. Just like the Dijkstra's algorithm, it finds the shortest path in a bottom-up approach (Patel and Baggar, 2014). Figure 2.2 shows a model of the Bellman-Ford technique depicted by a "*single source*" route finding solution for clustered nodes.

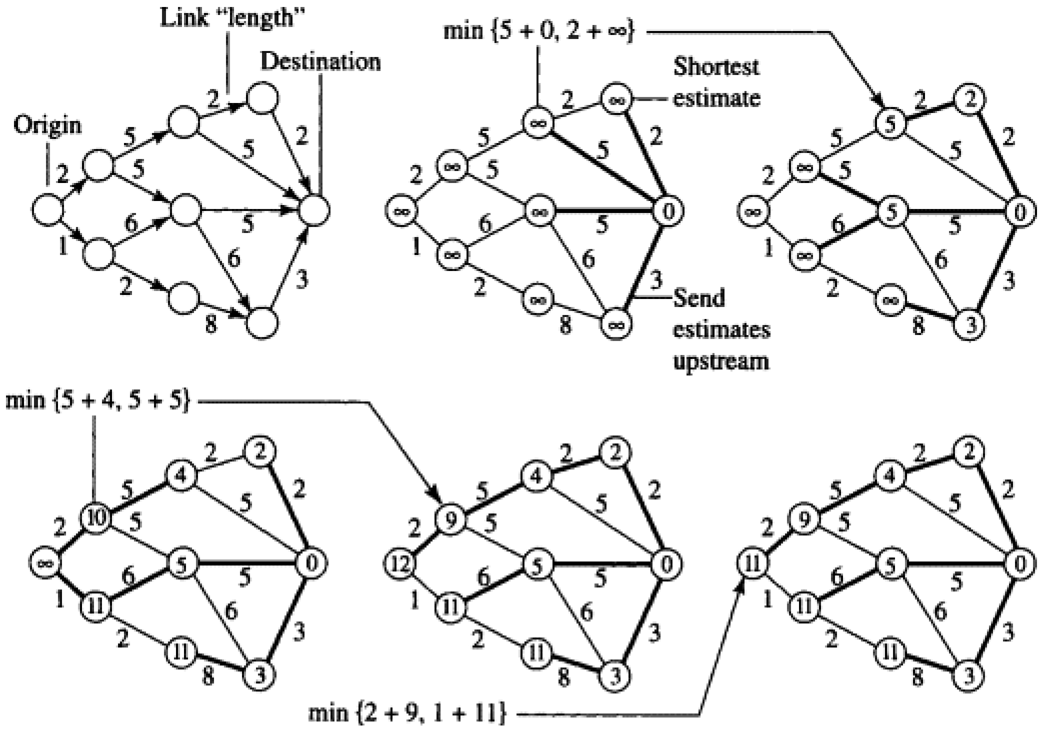

Figure 2.2: A distributed route-finding technique for clustered nodes modelling the Bellman-Ford Method (Walrand and Varaiya, 2000)

Yuan (1999) solved the Constrained Quality of Service Path Routing Problem using a modified Bellman-Ford technique. The modification was in the introduction of constraints to the path and granularity of the computations. Additionally, the technique was mapped uniformly. This technique was experimented on two test cases and yielded promising result in improving the worst-case of the Constrained QoS routing problem.

Patel and Baggar (2014) carried out a performance comparative study on both Dijkstra Algorithm and the Bellman-Ford Algorithm, in solving the shortest path problem in GIS application for the government sector, emergency system, Business sector, etc. From the study performed on both algorithms, the Dijkstra algorithm outperformed the bellman ford algorithm in a very lager network and has wider use in the real-time application of GIS technology.

2.3.2. Approximate Techniques

Approximate techniques generally refer to heuristics and metaheuristics.

Heuristics are approximate techniques that apply '*rules of thumb*' for solving Combinatorial Optimization Problems without necessarily guaranteeing optimal solutions. Heuristics provide approximate solutions within the constraint of polynomial time. Heuristic solutions are referred to as approximate because they make use of probabilities and some set of rules to finding solutions to problems. For an iterative procedure, heuristics can be used when an optimal solution is guaranteed to either obtain the solution with ease or make a decision within an exact procedure. In other words, the use of heuristics to solve the TSP and problems related to the TSP provides acceptable results that are not too far from the optimal and yet, are computationally affordable. A good heuristic must be effective, that is, must always lead to a solution, must be able to obtain 'good enough' approximate solutions, easy to implement, and flexible. Aside from the need to solve hard problems in polynomial time p, other motivations for using heuristic methods in literature (Oliviera and Carravilla, 2009; Marti and Reinelt, 2011; Giovanni, 2017; Kyritsis *et al*, 2018) include:

i. Unavailability of optimal methods for solving the problems
ii. The heuristic is part of a broader optimal solution procedure
iii. Incompatibility of existing exact solutions to available hardware
iv. The heuristic is more amenable to complexities than the available exact technique and can integrate complex constraints that are difficult to model.

Heuristics may be classified based on the atomicity of their solution procedures. In this regard, heuristics are classified as Tour Construction, Improvement / Local Search

Heuristics, and Compound Heuristics (Oliveira and Carravilla, 2009; Marti and Reinelt, 2011; Kyritsis *et al.*, 2018).

Heuristics may also be classified based on their solution paradigm, into space-partitioning-based heuristics, edge-based heuristics, or node-based heuristics, (Huang *et al.*, 2016; Huang and Yu, 2017). The space-partitioning-based heuristics, build solutions by first splitting the nodes into subsets $(s_1, s_2, \ldots, s_n)$ based on their paired distances, the nodes within the same subset are then connected into the tour path, and then the Hamilton tour for S is obtained by coupling the Hamiltonian paths of subsets $(s_1, s_2, \ldots, s_n)$. A common example under this category is the Strip and Hilbert technique. Edge-based heuristics build solutions by first determining the edge with the smallest distance and then placing it into the circuit. Most heuristics under the edge-based category are built on the Minimum Spanning Tree (MST), they include multiple fragment heuristic, double-MST (DMST), the Christofides algorithm (Chris), and so on. In the third category, the node-based heuristics build the tour by expanding the nodes one at a time till all the nodes have been inserted. Node-based heuristics must first decide which node to be used as the initial node, then determine the succeeding node to explore in each iteration, and where it will be inserted. Some known node-based heuristics include the Addition techniques, the Nearest Neighbour techniques, the insertion heuristics, the convex hull-based insertion heuristics, and so on. Apparently, node-based heuristics are chiefly tour construction techniques as well (Huang *et al.*, 2016; Huang and Yu, 2017).

Approximate techniques may also refer to Metaheuristics. Unlike heuristic techniques which are designed to solve specific optimization problems, metaheuristics are general purpose approximate computational techniques for solving optimization problems and may require few modifications to solve a given problem (Abdel-Basset *et al.*, 2018).

The most popular and widely researched metaheuristics are the nature inspired metaheuristic.

Some approximate techniques are reviewed in the following subsections.

2.3.2.1.Tour Construction Heuristics

Tour Construction heuristics are stand-alone techniques that generate solutions by sequentially applying a set of predefined procedures to the problem space. These procedures describe the processes involved in stages of Initialization; Selection and; insertion. The construction heuristic techniques have been used extensively in solving classic combinatorial optimization problems. Common techniques include the Nearest Neighbour Heuristic, the Nearest Insertion, Cheapest Insertion, Random Insertion, Addition heuristics, Savings Heuristics, and so on. Some well-known constructive heuristic methods are described briefly in Table 2.2.

Table 2.2. Description of some well-known tour construction heuristics

HEURISTICS	**DESCRIPTION**
Nearest Neighbour Heuristic	The NNH starts its tour with a single subtour of node/city i, chosen randomly or purposively and then iteratively add the next node $k+i$ not yet chosen but closest to subtour until all the nodes have been added to the tour. This technique is naïve and result in the occurrence of outliers as the search space and nodes increase. The NNH has a complexity of $O(n^2)$ and yields tours whose qualities are within 25%-30% of the Held-Karp lower bound. (Rosenkrantz,

	et al., 1977; Rao and Jin,2010; Huang and Yu, 2017; Lity *et al.*, 2017).
Nearest Insertion Heuristic	The NIH belong to the class of Insertion Heuristics. The Insertion heuristics starts from an arbitrary point to form a sub tour or partial circuit. Nodes not already in the sub tour are then inserted based on predefined criteria such that the increment to the total distance of the sub tour is minimized. Given the sub tour T_i, and given that x is the next node to be inserted, then the insertion technique inserts x between x_i^* and x_j^* in T_i according to: $$(x_i^*, x_j^*) = \underset{(x_i x_j) \in T_i}{\operatorname{argmin}}\, c(x_i, x_j, x)$$ The NIH obtains a tour solution by first building its subtour; initial node i and a node j nearest to i to form a partial circuit $T = i - j - i$. The next node $x^* = argmin_{v \notin T_i}\{d(x, x_i), \forall x_i \in T_i$ is then added iteratively till a Hamiltonian tour is formed (Huang *et al.*, 2016; Huang and Yu, 2017).
Farthest Insertion Heuristic	The FIH obtains a tour solution by first building its subtour; initial node i and a node j nearest to i to form a partial circuit $T = i - j - i$. The next node $x^* = argmax_{v \notin T_i}\{d(x, x_i), \forall x_i \in T_i$ is then added iteratively till a Hamiltonian tour is formed. The FIH solution when evaluated: $S_{FIH}/S_{OPT} \leq [\log n] + 1$ The FIH is executed in $O(n^2)$ computational effort and since the algorithm runs n times starting it has a complexity of $O(n^2)$. (Rosenkrantz, *et al.*, 1977; Huang *et al.*, 2016; Huang and Yu, 2017; Lity *et al.*, 2017)

Cheapest Insertion Heuristic	This is similar to the Nearest Insertion heuristic. Start at node i (arbitrary or fixed), find cities $k, i \, and \, j$ (i and j being the extremes of an edge belonging to the partial tour and k not belonging to that tour) for which $C_{ik} + C_{kj} - C_{ij}$ is minimized. If all nodes have been selected STOP, else repeat the process. Analysis by Rosenkrantz *et al.*, (1977) shows that the complexity of cheapest Insertion is $T(n) = O(n^2 \log n)$. An experimental evaluation of the solution is ${S_{CIH}}/{S_{OPT}} \leq 2$. (Fan, 2011; Cruz *et al.*, 2012).
Random Insertion Heuristic	The Random Insertion Heuristic starts by choosing two arbitrary nodes $i, j \in T$, and form a sub tour $i - j - i$. Then, iteratively and arbitrarily chooses a node k of T that is yet to be added to the cycle such that the increase in the total cost of the tour is minimal. The loop terminates when all nodes have been included in the tour (Goetschalckx, 2011; Anbuudayasankar *et al.*, 2014).

Others tour construction methods are described below:

The Greedy Heuristic:

The Greedy heuristic is a technique with a '*simplest improvement*' approach. The Greedy method's solution paradigm is to obtain a global optimum by first obtaining local optimal solution at each stage of the problem. This technique is naïve and usually fall short of obtaining global optimum, although locally optimal solutions are often reached. It is thus a good approximate technique. In solving the TSP, the Greedy method iteratively adding a sorted node set starting with the minimum weight until the tour is completed (Matai *et al.*, 2010). Techniques such as Prim's MST, Kruskal's MST, Dijkstra Shortest Route Algorithm, Huffman Coding and so on employ the greedy solution paradigm. The complexity of the greedy heuristic is $O(n^2 \log_2(n))$ (Ejim, 2016; Jain and Prasad, 2017). Figure 2.3. provides an illustration of the greedy technique on six nodes instance.

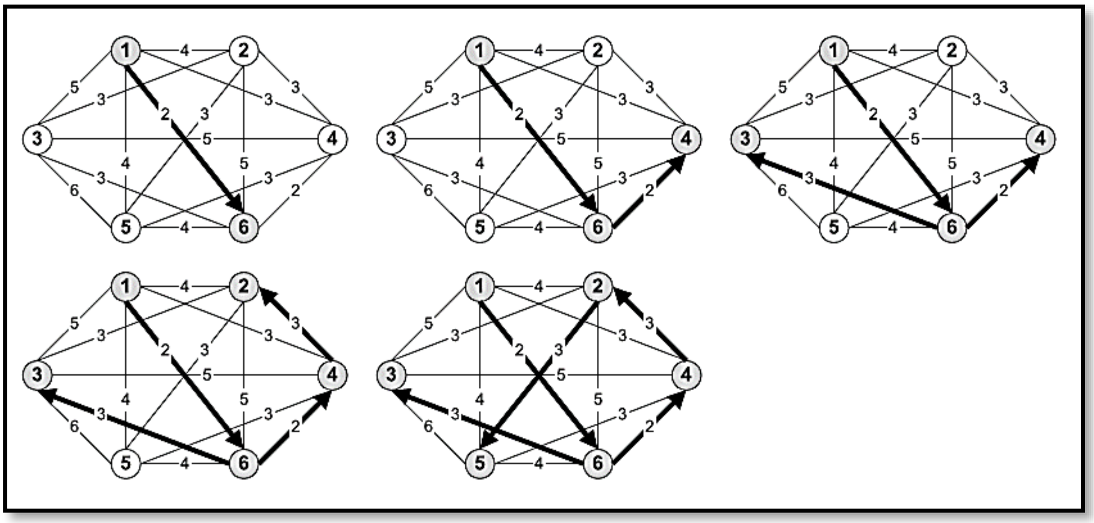

Figure 2.3. An illustration of the greedy technique on six nodes instance (Oliveira and Carravilla, 2009)

Abdulkarim and Alshammari (2015) used the Genetic Algorithm and the Greedy Heuristic to solve a TSP. The computational experiment consists of three test instances with 20, 100, and 1000 cities within the US border. The results obtained showed that the Greedy Heuristic's complexity was higher than that of the Genetic technique, due to the higher number of iterations it took before the solution was reached. However, the Greedy technique outperformed the Genetic Algorithm in terms of solution quality.

The Christofides Heuristic:

The Christofides algorithm was named after Nicos Christofides. This technique specializes in solving symmetric TSPs, which of course, satisfy the triangle inequality criteria. The Christofide techniques guarantees solutions which are $3/2$ of the Held Karp lower bound (Štencek, 2013). The complexity of the Christofide techniques is $O(n^3)$ (Chauhan *et al.*, 2012).

The steps involved in the Christofides algorithm are given below (Matai *et al.*, 2010):

Algorithm 2.4: The Christofides Algorithm	
Input: Set of nodes $\boldsymbol{S}$, an arbitrary city s$\boldsymbol{\epsilon S}$, and cost function $\boldsymbol{C}$. **Output:** T: the TSP for S	
Step 1:	For a set of nodes $\mathbf{S_{1,2,\ldots,n}}$, generate a Minimal Spanning Tree (MST).
Step 2:	For a set $O \in S$ of nodes having odd degree, generate a Minimum-Weight Matching (MWM) and merge the MST with the MWM to create a "multigraph" M.
Step 3:	Generate an "Euler cycle" from M, while avoiding visited nodes.

Research efforts extending the base Christofide technique have further improved the performance of the method (Xu *et al.*, 2011; An *et al.*, 2012; Genova and Williamson, 2017; Xu and Rodrigues, 2017).

The Clarke-Wright Savings Heuristic:

The Clarke-Wright Savings Heuristic is reputed for solving the Vehicle Routing variant of the Travelling Salesman Problem (Chauhan *et al.*, 2012). The Clarke-Wright Savings Heuristic excels in handling the Vehicle Routing Problem because of its flexibility and abilities to handle divers constraints. Its experimental performance of approximately 2.5% over the optimal solution is equally high compared with some methods such as the Nearest Neighbour Heuristic; its time and space complexities are $O(n^2 \log(n))$ and $O(n^2)$ respectively (Chauhan *et al.*, 2012; Jeřábek *et al.*, 2016). The Clarke-Wright

Savings Heuristic's approach to the Vehicle Routing Problem can be thought of as an iterative refinement process. The process starts by finding an initial solution which is then refined through a series of stepwise activities, thus giving room for the gradual introduction, monitoring and control of constraints.

Chauhan *et al.,* (2012) and Kampf *et al.,* (2015) detailed the procedure used by the Clarke-Wright Savings heuristic as in Algorithm 2.5:

Given a network S of nodes n and connecting edges e, where is the starting node n_0 and $n_{i=1,2,\ldots,x}$ are the delivery nodes. Each of the nodes have attaches constraints, and the vehicle has limited capacity, the objective is to generate sets of paths subject to some constraints, that traverses each node and return to the starting node with a single ride, without exceeding the capacity of the carrier at minimal cost.

Algorithm 2.5: Clarke-Wright Savings Algorithm

Input: Set of cities $\boldsymbol{n}$
Output: T: the TSP

1. Form preliminary solution: select two feasible routes, $(\boldsymbol{n_0}-\boldsymbol{n_i}-\boldsymbol{n_0})$ ***and*** $(\boldsymbol{n_0}-\boldsymbol{n_j}-\boldsymbol{n_0})$ not yet in the hub and connect them to the hub
2. Determine the savings coefficient for each pair of non-hub nodes by computing the cost difference if the vehicle bypassed the hub, rather than going through it.
3. The non-hub pairs of nodes are then passed through iteratively in decreasing order of savings, performing the bypass so long as it does not create a cycle of non-hub nodes or cause a non-hub node to become adjacent to more than two other non-hub nodes.
4. Terminate if when only two non-hub cities remain connected to the hub, in which case we have a true tour.

Pichpibul and Kawtummachai (2012), proposed a Clarke-Wright (CW) Savings technique in solving the Capacitated Vehicle Routing Problem that traverses all nodes, but does not necessarily complete the Hamiltonian Cycle. The proposed technique followed four procedures: firstly, the Clarke-Wright model was modified, then an open-path was constructed, thirdly, the selection process was implemented in two phases, and finally, route post-refinement. Experimental results showed that the proposed technique did better than the classical CW method.

Addition Heuristics:

Addition Heuristics and Insertion Heuristics both solve the TSP by adding nodes to partial tours based on some expansion rules. However, unlike the Insertion technique which considers all the insertion points of the subtour, the Addition heuristic consider only edges connecting the node u nearest to the node v that is to be inserted. Expectedly, addition heuristics bettered the insertion techniques in terms of complexity but fall short of insertion techniques in terms of solution quality (Bentley 1992; Huang and Yu, 2016). Like insertion techniques, there are four types of addition heuristics, namely Nearest Addition Heuristic, Cheapest Addition Heuristic, Farthest Addition Heuristic and Random Addition Heuristic. The complexity of these techniques is $O(n^3)$.

2.3.2.2.Improvement/Local Search Methods

Improvement techniques build an initial solution, which is then iteratively refined until the termination criterion is achieved at which stage, there is no way to further improve it. This is derived from the concept that by iteratively refining solutions, the quality of the solution can be enhanced to be as close to the optimal solution as possible. Some common improvement heuristics include the Lin Kernighan, the 2-Opt, 3-Opt, and k-Opt algorithms, and so on. Some Improvement techniques are discussed as follow:

a. Simulated Annealing (SA):

Simulated Annealing (SA) is primarily an arbitrary local search algorithm, which is similar to the TABU Search approach, but differs in that it does not allow path exchange that deteriorates the solution (Matai *et al.,* 2010). The Simulated Annealing technique has a complexity $O(n^2)$ with a large constant of proportionality because it uses the 2-Opt neighbourhood search. The primary difference of the SA from the 2-Opt is that the local optimization algorithm is often restricted to their search for the optimal solution in a downhill direction which means that the initial solution is changed only if it results in a decrease in the objective function value. However, the 2-opt algorithm works well when the problem size is less than 50 cities. The Simulated Annealing (SA) algorithm obtains good tour quality because of its modular approach of going from one solution to the next (Abid and Muhammad, 2015).

b. The 2-Opt and 3-Opt algorithms:

The 2-Opt procedure was first formulated by Croes in 1958 based an earlier work by Flood in1956 (Saiyed, 2012). The 2-Opt procedure improves an initial tour through a process of comparison of all admissible pair of valid edges and substitution based on some criteria. This swapping procedure iteratively refines the tour until the route converges to a locally optimal solution, in which case it is no longer possible to reduce the tour length. At this point of local optimum, this procedure would have transformed all crossing edges into non-crossing ones (see Figure 2.4.) (Matai *et al.,* 2010).

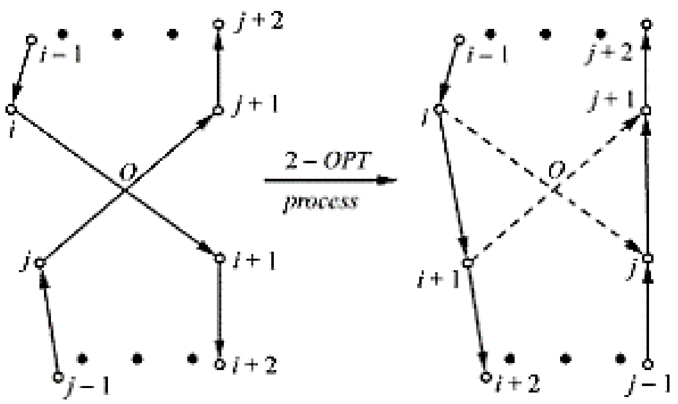

Figure 2.4. A Schematic Illustration of the 2-OPT Procedure (Yang *et al.,* 2008)

A naive implementation of 2-Opt runs in $O(n^2)$. This involves selecting an edge (c_1, c_2) and searching for another edge (c_3, c_4), completing a move only if $dist(c_1, c_2) + dist(c_3, c_4) > dist(c_2, c_3) + dist(c_1, c_4)$, (Saiyed, 2012).

The 3-Opt algorithm's structure is similar to that of the 2-Opt, except that it removes three edges. The search is exhausted when no more 3-opt moves can improve tour quality. A 3-Optimal tour is also a 2-Optimal tour.

Neissi and Mazloom (2009) made use of both local search heuristics and genetic local search algorithms; a 2-Opt algorithm and a 3-Opt algorithm, to solve the TSP. From the evaluation and comparison of the run time behaviour and fitness of their approach, 2-Opt had better fitness for solving the TSP while it was observed that with the 3-Opt algorithm the solution converges to the global optimum in more time. Hence, they recommended that the 2-Opt algorithm be used in getting the optimum arrival time and the 3-Opt algorithm be used to get global optimum where it is important.

c. k-Opt Algorithms:

The k-Opt move is applied to improve the generated tour from obtained tour construction heuristic. The exchange heuristic for $k > 3$ will take more computational time as compared to that of 2-Opt and 3-Opt exchange heuristic. For instance, a 4-Opt move, which is referred to as "the crossing bridges", cannot be sequentially constructed using 2-Opt moves and for this to be possible two of these moves would have to be illegal (Matai *et al.*, 2010).

Helsgaun (2009) implemented a general k-Opt sub-move for a variant of the Lin–Kernighan heuristic, LKH-2. The computational experiments performed showed that the implementation was both effective and scalable for Euclidean test instances from 10,000 to 10,000,000 cities. It was noted that the use of general k-Opt sub-moves depends on the candidate graph, except the candidate graph is sparse hence the instance should not be heavily clustered else this will lead to time consumption. Thus, the runtime of the method increases almost linearly with the problem size.

d. Lin-Kernighan:

The Lin-Kernighan (LK) Algorithm is renown as a high performing technique for obtaining optimal or approximate solutions for the TSP, although its implementation is complex. The creation of the LK was based on the static K in the K-Opt method. The LK is a variable k-way exchange heuristic that introduces a powerful variable-Opt algorithm to its implementation and dynamically changes the value of K during its execution (Chauhan *et al.*, 2012). The time complexity of LK is approximately $O(n^{2.2})$, making it slower than a simple 2-Opt (Papadimitriou, 1992; Matai *et al.,* 2010).

Lau (2002) developed a Search and Learning Algorithm (SLA*-TSP) for solving TSPs which applies the heuristic estimation approach and made a comparison of the proposed algorithm's computation time and solutions with the Nearest Neighbour Heuristic and Lin-Kernighan Heuristics. SLA*-TSP proffered more suitable results than Nearest Neighbour heuristics and almost the same solutions as Lin-Kernighan Heuristics. It however performed woefully in computational time as compared to the other algorithms while Nearest Neighbour heuristics had the best computation time record. The poor computational time performance of SLA*-TSP was attributed to its dynamic tour construction and the inefficient data retrieval in its program.

2.3.2.3.Compound Heuristics

The constructive and local search methods form the foundations of the Compound heuristic procedures. In this approach, two or more constructive and improvement heuristics are applied separately and the best solution is chosen (Frederickson *et al,* 1978; Yao, 1980; Landston, 1987). Examples include CCAO (Convex Hull, Cheap Insertion, Largest Angle and OR-Opt) (Golden and Stewart, 1985), GENIUS (Gendreau *et al, 1992*) among others.

2.3.2.4.Metaheuristics

Metaheuristic algorithms are special form of heuristics used for solving specific but complex optimization problems. They are classified either as metaphor based or non-metaphor based (Damghanijazi and Mazidi, 2017). They differ mainly in the techniques used in simulating the selected phenomenon behaviour in the search area. Examples of metaphor-based metaheuristics include: Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Water Waves Optimization (WWO), Clonal Selection Algorithm (CLONALG), Chemical Reaction Optimization (CRO), Harmony Search (HS), Sine Cosine Algorithm (SCA), Simulated Annealing (SA), Teaching–Learning-Based Optimization (TLBO), League Championship Algorithm (LCA), and so on (Matai *et al.,* 2010; Chauhan *et al.,* 2012; Abdulkarim and Alshammari, 2015. Some non-metaphor-based metaheuristics include TABU Search (TS), Variable Neighbourhood Search (VNS) (Matai *et al.,* 2010; Basu, 2012; Damghanijazi and Mazidi, 2017).

a. Ant Colony:

Ant Colony Optimization is a meta-heuristic technique whose principle was inspired by the behaviour of real ants that find food resources by laying a trail of a chemical substance called ‘pheromone’ along the path from the nest to the food source (Chauhan *et al.,* 2012). The amount of available pheromone determined if new ants are encouraged to trail on the same path. Shorter routes to food sources have higher amounts of pheromone. As time goes by, most of the ants are directed to use the shortest path. The medium of indirect communication is referred to as ‘stigmergy’ (Dorigo *et* *al.*, 1999), in which the concept of positive feedback is exploited to find the best possible path, based on the experience of previous ants (Chauhan *et al.*, 2012).

Gupta (2013) carried a comparative performance analysis of some meta-heuristics in solving the classic and Random Travelling Salesman Problem. Two classical meta-heuristics (TABU search and Simulated Annealing), two evolutionary techniques (Genetic and Memetic), and four nature-inspired algorithms (Ant Colony Optimization, Bee Colony Optimization, Firefly, and Cuckoo-Search) were considered. The performances of these meta-heuristic algorithms were compared based on quality of the tour solution. It was observed that the Nature-inspired algorithms outperformed both Traditional and Evolutionary algorithms and obtained optimal solutions for some instances. Particularly, the Cuckoo Search algorithm produced the best solutions in terms of solution quality.

Droste (2017) studied the Branch-and-Bound algorithm and the Ant Colony Optimisation algorithm for solving the TSP. The computational results indicated that the Branch-and-Bound algorithm could not solve for test instances with more than 23 cities. While the Ant Colony Optimisation algorithm provided solutions for instances with nodes of almost 100 cities. Result showed that the accuracy of the Ant Colony technique decreases with increasing number of nodes.

b. Genetic Algorithm:

Genetic Algorithm (GA) is a heuristic algorithm that simulates the evolution principles in finding solutions to complex problems that cannot be solved with any other exact algorithms. These evolution principles include inheritance, mutation, natural selection, hybridization – for “selective breeding” of a solution of a basic problem. A basic GA starts with a randomly generated population of candidate solutions for different problems. The candidates are saved and are then mated to produce offspring, while some go through a mutating process, and as the population develops, the solutions are improved (Matai *et al.,* 2010). The algorithm calculates the fitness function for each member of the population expressing the quality of solution for all members. By selecting the fittest candidates for mating and mutation the overall fitness of the population improves (Abdulkarim and Alshammari, 2015). The algorithm terminates after a considerable improvement to the quality has been achieved or after a time-out. Applying GA to the TSP involves implementing a crossover routine, a measure of fitness, and a mutation routine. A good measure of fitness is the actual length of the solution (Štencek, 2013).

Using GA for TSP has disadvantages of premature convergence and poor local search capability. These problems can be circumvented by integrating other high performing techniques such artificial immune systems into it (Abid and Muhammad, 2015).

Gupta and Kakkar (2012) solved the Travelling Salesman Problem using a modified Genetic Algorithm. The Parallel search-and-learn technique, Hybrid Method, Neural Network Techniques, TABU search were used as a curtail the complexity of the Genetic Algorithm and generate and optimized solution.

AlSalib *et al.,* (2013) investigated the performance of the Genetic algorithm and Nearest Neighbour Heuristic in terms of cost and running time, using four datasets of varying cities. It was observed that the Nearest Neighbour Heuristic proffered very suitable results for datasets with less than 50 cities, its results were either close to or better than the optimal solutions. It produced solutions farther away from the optimal for large datasets but recorded an overall better execution time which was lesser than a second for all four instances used. Genetic Algorithm, on the other hand, was more stable as near-optimal tour costs producing solutions that were much closer to each other and proved to have an overall lesser amount of errors, as computed by the MED formula, hence indicating that it performed better than the Nearest Neighbour Heuristic.

Damghanijazi and Mazidi (2017) carried out a comparative performance analysis of five meta-heuristics including Hill Climbing, Simulated Annealing, PSO, Ant Colony, and Genetic Algorithm in solving the classic Travelling Salesman Problem. The execution time and space complexities were also compared. Computational results showed that the Simulated Annealing and Hill Climbing solutions stopped at the local minimum and thus had poorer tour quality than the other methods. The other algorithms gave better solutions while GA achieves the optimal solution in the shortest time. The hill-climbing method has the lowest memory consumption.

c. TABU Search:

The TABU Search is an iterative refinement technique based on local search, also known to be a neighbourhood-search algorithm that begins with an initial solution to the problem and searches for the best solution in the neighbourhood of the existing solution using a 2-opt exchange mechanism. It then designates the best solution in the neighbourhood as the current solution and iteratively refines the process until the termination criteria is met which may either be due to execution time, maximum iteration count conditions, or solution quality objectives, or all (Basu, 2012). The challenge with using a simple neighbourhood search approach (either 2-opt or 3-opt exchange heuristic), is that the procedure can easily get stuck in a local optimum. To avoid this, the TABU search keeps a TABU list containing bad solution with a bad exchange (Matai *et al.*, 2010).

Misevičius *et al.*, (2005) used a variant of the TABU search scheme, the fast iterated TABU search (ITS) meta-heuristic, to solve the TSP. ITS obtains near-optimal solutions by combining intensification (standard TABU search) and diversification properly. The fast-iterated TABU search technique obtained promising results for the TSP instances considered from the TSPLIB. It was observed that the FITS outperformed the random multi-start (RMS) algorithm based on 2-opt moves, the Simulated Annealing algorithm, the straightforward TABU search algorithm, and the iterated TABU search (ITS) algorithm, especially, on the smaller TSP instances.

Erdogan *et al.*, (2012) developed three metaheuristics to solve the Travelling Salesman Problems with Pickups, Deliveries, and Handling Costs. The metaheuristics solutions used were based on the TABU search, Iterated Local Search, and the Iterated TABU search. The three heuristics experimented on some test instances and their performances were documented and compared. The computational results indicated that the hybrid of TABU search with exact Dynamic Programming performed best, but using the approximate linear time algorithm considerably decreases the CPU time at the cost of slightly worse solutions.

2.3.3. The Held-Karp Lower Bound

The Held-Karp (HK) lower bound is used in testing the performance of any new TSP heuristic. It is the solution to the linear programming (LP) relaxation of the standard integer programming formulation of the Travelling salesman problem (Matai *et al.,* 2010). Surprisingly, there is no readily available LP code for evaluating HK lower bound for problems larger than a few hundred cities. Also, Linear Programming implementations (even efficient ones) do not scale well and rapidly become impractical for problems with many thousands of cities (Valenzuela and Jones, 2001); HK lower bound is averagely 0.8% below the optimal tour length.

The HK lower bound can be evaluated as a 1-tree relaxation, where a 1-tree on an $n\ city$ problem is defined as follows (Valenzuela and Jones, 1997):

A $1 - tree$ is a connected graph with vertices $1, 2, \dots, n$ consisting of a tree on the vertices $2, 3, \dots, n$ together with two edges incident with $city\ 1$.

Evaluation of a Held-Karp lower bound requires the computation of a sequence of Minimum $1 - trees$, where:

A Minimum $1 - tree$ is a Minimum Spanning Tree (MST) on the vertices $2, 3, \dots, n$ together with the two lowest-cost edges incident with $city\ 1$.

A tour is simply a $1 - tree$ in which each vertex has degree 2. If a minimum $1 - tree$ is a tour, then it is a tour of minimum cost.

2.4. Related State-of-the-Art Tour Construction Solutions

Research works done on state-of-the-art tour construction methods such as the Nearest Neighbour Heuristic, Nearest Insertion Heuristic, Cheapest Insertion Heuristic, Random Insertion Heuristic and Farthest Insertion Heuristic were reviewed in this section.

Generally, the Nearest Neighbour Heuristic can solve the TSP in good time, with less-than-optimal solution quality. Experimentally,

$$T_{NNH}/T_{OPT} \approx 1.26 \qquad (2.26)$$

Where T_{NNH} = tour cost of the Nearest Neighbour Heuristic and T_{OPT} = cost of the optimal tour.

Thus, recent literature focus on using the Nearest Neighbour Heuristic either as part of a hybrid method as in (Rao and Jin,2010; Huang and Yu, 2017; Lity *et al.*, 2017) or as a seed technique in a metaheuristic for building initial solutions (Lingling and Ruhan, 2012; Bernardino and Paias, 2018; Kitjacharoenchaia *et al.,* 2019). The works reviewed in this section fall in the latter category. The literature considered span the period 2011-2020.

Rego *et al.,* (2011) used the Nearest Neighbour Heuristic to build an initial tour in their experimental survey of some leading techniques. They identified important implementation success factors and experimented a total of nine high performing heuristics on different instances of both symmetric and asymmetric TSPs. These methods included four derivatives of the Lin-Kernighan heuristic and two variants of the stem and cycle (S&C) technique for the implementation of the Symmetric TSP; while three generalized LK and S&C methods were used for implementation on the Asymmetric TSPs. The LK variants used on the Symmetric TSPs include the Johnson and McGeoch Lin–Kernighan (LK–JM), the Neto's Lin–Kernighan (LK–N), the Applegate, Bixby, Chvatal, and Cook Lin–Kernighan (LK–ABCC) and the Applegate, Cook and Rohe Lin–Kernighan (LK–ACR). The stem and cycle considered include the Rego, Glover, and Gamboa stem-and-cycle (S&C-RGG) and S&C-RGG+. The three generalized techniques used on the Asymmetric TSPs are; Kanellakis–Papadimitriou heuristic (KP-JM), Rego, Glover, and Gamboa stem-and-cycle (S&C-RGG) and Rego, Glover, and Gamboa doubly-rooted S&C (DRS&C-RGG). All the generalized methods' implementation used the Nearest Neighbour Heuristic to build their initial tour. Their findings revealed that S&C approaches clearly outperformed the basic LK implementations in terms of solution quality, while the LK performed better in terms of time.

Lingling and Ruhan (2012) developed a hybrid metaheuristic algorithm for solving large-scale vehicle routing problem, the algorithm was a combination of Nearest Neighbour Heuristic and TABU algorithms. The Nearest Neighbour Heuristic was used to generate the initial routes while the TABU was used for the intra and cross-exchange routes. The testbed used in the experiments carried out was from a dataset of 6772 customers in the central and suburb of Suizhou city. The performance evaluation revealed that the proposed algorithm evidently benefited from the introduction of the Nearest Neighbour Heuristic in generating the initial tour and was able to efficiently provide minimum cost for delivery.

Fischer *et al.* (2014) introduced an extension of the Travelling Salesman Problem (TSP), referred to as Quadratic TSP (QTSP). Three Exact algorithms (an exact approach based on a polynomial transformation to a TSP, branch-and-bound algorithm and branch-and-cut) and seven approximate algorithms (Cheapest-Insertion Heuristic, Nearest-Neighbour Heuristic, Two-Directional Nearest-Neighbour Heuristic (2NN), Assignment-Patching Heuristic (AP), Nearest-Neighbour-Patching Heuristic (NNP), Two-Directional Nearest-Neighbour-Patching Heuristic (2NNP) and Greedy Heuristic (GR)) were used to solve the QTSP. From the computational evaluation, the branch-and-cut approach was seen to be capable of solving large real-world instances with up to 100 nodes and provided optimality in a reasonable time of about ten minutes. Although the running times of exact algorithms were reasonable, they were not as fast as heuristics which took less than or equal to ten seconds to solve the largest instances. The variants of the Nearest Neighbour Heuristic presented did well in terms of computational speed but fell short in comparison to the exact methods in terms of solution quality.

Lity *et al.,* (2017) modelled the product ordering process of the incremental Software Product Line (SPL) analysis as a Travelling Salesman Problem (TSP). The aim was to optimize product orders and thereby improve the overall SPL analysis. Products were modelled as nodes in a graph and the solution-space information defines edge weights between product nodes. Existing graph route-finding heuristics were used to obtain the path with minimal costs. The first heuristic deployed was the Nearest Neighbour Heuristic. The nodes were analyzed in order of their similarity, so the Nearest Neighbour Heuristic path was built by adding the product (*node*) that is most similar to the last node on the path. However, it was observed that the quality of the approximation was poor because it first greedily added all the similar nodes and later suffered the curse of dimensionality when not so similar nodes were to be added. To circumvent this, a lookup was introduced to examine the next node to be added with respect to the already computed path. Thereafter, two insertion heuristics namely Nearest Insertion Heuristic and Farthest Insertion Heuristic were deployed to insert the remaining product to the existing path created by the Nearest Neighbour Heuristic. The proposed method was simulated on a prototype and evaluated for applicability and performance; a significantly more optimized SPL process was reported.

Bernardino and Paias (2018) used a modified Nearest Neighbour Heuristic to generate an initial solution as part of the procedure of the Iterated Local Search implementation. They worked on the Family Travelling Salesman Problem (FTSP), which is a variant of the classic Travelling Salesman Problem (TSP). They set out by formulating the FTSP, the objective being to traverse a stated number of nodes in each cluster at a minimum cost. The FTSP sub tour was then modelled both as compact and non-compact models. Three compact models were created namely; Single-Commodity Flow model (SCF), the Family-Commodity Flow model (FCF), and the Node-Commodity

Flow model (NCF). The non-compact models proposed were the Connectivity Cuts (CC) model, the Rounded Visits (RV) model, and the Rounded Family visits (RFV) model. These models were then compared analytically and experimented using C++ programming language. Iterative Local Search (ILS) was implemented on C++ to provide upper bounds for instances that cannot be solved using exact techniques. The first stage of the Iterative Local Search implementation included the use of a modified Nearest Neighbour Heuristic to build an initial solution, after which local search was deployed to arrive at a local optimum. A perturbation was then used to escape the local optimum before the extra nodes accrued were extracted based on removal criteria. The performance of the Iterative Local Search validated the known research hypothesis that construction tour heuristics produce quality initial solutions. The models were implemented on publicly available benchmark instances and the experimental results were documented. Results showed that non-compact models did better than their counterpart compact ones.

In the study by Kitjacharoenchaia *et al.*, (2019), the Nearest Neighbour Heuristic and two other heuristics were used to build an initial solution for their proposed model. Motivated by the increasing adoption of drones to achieve fast and flexible delivery, the authors conducted a study to simulate a drone delivery system formulated as a Multiple Travelling Salesman Problem (mTSP) to minimize time. They implemented the Mixed Integer Programming (MIP) to solve the problem and thereafter proposed a new technique called the Adaptive Insertion Algorithm (ADI). The Adaptive Insertion Algorithm was implemented in two phases. An initial solution on only truck tours was built using three heuristics (namely the Nearest Neighbour Heuristic, Genetic Algorithm, and Random Cluster/tour). The mTSP solution was generated from the initial tour in the second phase. The method was then experimented on a single truck, multiple trucks, and a single truck and drone system and the solution compared with the existing MIP solution. The system reported a promising, competitive performance. It could be deduced that solutions generated from the initial solution by heuristics such as Nearest Neighbour Heuristic hold promising performances.

Nikolas *et al.,* (2019) presented *k*-Repetitive-Nearest-Neighbour (*k*-RNN) algorithm which is an extension of the well-known Nearest-Neighbour Heuristic. The procedure for the *k*-Repetitive-Nearest-Neighbour was to begin a search tour with permutations of *k* nodes and then continue the search using the NNH from that point on, after which the optimal tour is obtained. From the experiments conducted on numerous instances, it was observed that there was an increase in the quality of the solution obtained when the value of *k* increases, meanwhile the running time increased by a factor of *n.* Experimental results showed that for 2-RNN the solutions' quality remains relatively stable at approximately 10% to 40% above the optimum.

Víctor *et al*., (2020) solved the Euclidean TSPs of small and large data sizes with an efficient heuristic that is based on the Girding Polygon which doesn't take up much computer memory space and produces approximate results that are near-optimal. The computational performance of the proposed approximate heuristic was compared to that of Nearest Neighbour Heuristic which is also an approximate heuristic. The proposed heuristic outperformed the Nearest Neighbour Heuristic with an average percentage error of 16.89% while that of the Nearest Neighbour Heuristic an average percentage error of 26.55%. The technique also had a standard deviation of 0.05%, while the Nearest Neighbour Heuristic had a standard deviation of 0.04%. Even though the proposed algorithm didn't produce optimal solutions for the instances used, it gave an approximate solution which was significantly better than the Nearest Neighbour Heuristic.

Fontaine *et al.*, (2020) conducted an experimental study to ascertain the effectiveness of the human strategies in solving the Vehicle Routing Problem (VRP) compared to that of heuristic techniques. Motivated by the need to understand the strengths and limitations of the human decision making especially in completing the Travelling Salesman related tasks such as clustering and route building, the discrete choice model was developed to evaluate the underlying motivation of participants in their choices of some attributes during the tour building process of clustering and route finding. Their work was based on three (3) hypotheses which are: one, the complexity of the problem has an impact on the solution quality of the participants, two, the participants follow certain strategies during problem-solving, and three, Feedback requested by the participants has a significant impact on the performance. A total of 112 respondents, aged between 18 and 32 years, participated in the experimental study, most of who are novices in routing. The costs of the attributes by each participant and instance were also evaluated using multinomial logistic regression to determine how much each attribute contributes to the individual choices when clusters and routes are built. The analysis also included the splitting of the clustering and routing performance to be able to independently compute the optimal TSP solution for each cluster and then compare the results with the actual routes of the participant. The humans' performance was then compared to the performances of the Nearest Neighbour Heuristic, the Sweep Heuristic, and the Savings Heuristic. Their findings showed that while humans, were more often than not unable to generate optimal solutions, they typically perform better than the worst cases of these heuristics and worse than their best cases irrespective of size and vehicle capacity. Additionally, they reported that poor clustering led to poor solutions in the Nearest Neighbour Heuristic and others. They concluded by recommending that interface design should avoid too much feedback options, but rather focus more on obtaining good clusters to foster better solutions.

In summary, the Nearest Neighbour Heuristic is widely used in literature because of its speed and simplicity. Efforts have been made to modify the Nearest Neighbour Heuristic for better performance. It has also been used as part of hybridized solutions or used to build the initial solution of metaheuristics. While the Nearest Neighbour Heuristic is preferred for its speed and simplicity, its greedy approach of adding the lowest cost nodes first, however, means that it suffers what is called the "*curse of dimensionality*" because as the search space and nodes increase, more and more outliers are seen. The term "*curse of dimensionality*" is often used to describe the phenomenon that as the dimensionality increases, leading to larger search space, the sparsity of data results in more outliers.

The Insertion heuristics starts from an arbitrary point to form a sub tour or partial circuit. Nodes not already in the sub tour are then inserted based on predefined criteria such that the increment to the total distance of the sub tour is minimized (Huang *et al.*, 2016; Huang and Yu, 2017). Suppose that node x is to be added to edge (x_i, x_j), and given the cost function $c(x_i, x_j, x)$, then,

$$c\left(x_i, x_j, x\right) = d(x, x_i) + d\left(x, x_j\right) - d\left(x_i, x_j\right) \qquad (2.27)$$

Each insertion technique method aims to add a node to an edge (that is between two nodes) at a minimal cost. Given the sub tour T_i, and given that x is the next node to be inserted, then the insertion technique inserts x between x_i^* and x_j^* in T_i according to:

$$(x_i^*, x_j^*) = \underset{(x_i x_j) \in T_i}{\operatorname{argmin}} c\left(x_i, x_j, x\right) \qquad (2.28)$$

Insertion techniques are desirable because of their speed, ease of implementation, quality of solutions, and the fact that they can be easily modified to handle complex constraints (Daamen and Phillipson, 2015). There are 4 generally known insertion techniques vis Nearest Insertion Heuristic, Cheapest Insertion Heuristic, Random Insertion Heuristic, and Farthest Insertion Heuristic. Others include Priciest Insertion, quick insertion, and greatest angle insertion (Goetschalckx, 2011; Anbuudayasankar *et al.*, 2014).

Insertion techniques can be used to get a good tour construction solution. According to Rosenkrantz *et al.*, (1977), Insertion techniques find $O(log\ n)$ approximate solutions. Insertion techniques are also used as an initial solution for improvement heuristics as well as metaheuristics; insertion techniques have been proven to significantly improve the performance of 2-Opt methods when used as initial solutions (Englert *et al.*, 2014). Other researchers have presented new insertion techniques, either as a modification of state-of-the-art methods or as novel efforts. Experimentally, the Farthest Insertion Heuristic has been known to outperform the Random Insertion, the Cheapest Insertion, and the Nearest Insertion Heuristic in that order (Rosenkrantz *et al.*, 1977; Lawler et al. 1985; Babel 2020).

Daamen and Phillipson (2015) presented a simulated TSP called an Edge Disjoint Circuits Problem (EDCP) with an intent to compare the approach of integrating both clustering and disjoint routing with separate approaches in obtaining better optimal solutions. The Insertion and the Cluster First-Route Second (CFRS) heuristics were applied to finding initial solutions for the EDCP as they were well-known techniques, from literature, for constructing feasible solutions for Vehicle Routing Problem (VRP). Insertion heuristics are known to be fast in generating good solutions, easy to implement, and extendable in dealing with complicated constraints. Hence, the initial solution was found using an insertion heuristic and enhanced using local search until the maximum time was exceeded or a local optimum was found. Three orders of insertion heuristic were tested, namely: random, non-disjoint insertion cost, and disjoint cost orders. For all testbeds considered, the random order had the highest average cost while the disjoint cost order had the lowest average cost with considerably larger computation time. Using various test instances, the developed insertion heuristic was compared with the CFRS Heuristic, the insertion heuristic gave an average cost between 27% lower and 3% higher than the average cost of the CFRS heuristic. The insertion heuristic performed quite well compared to the CFRS heuristic. There were only two instances where the CFRS heuristic performed better, in terms of average cost, with a difference of around 2% - 3%.

In a bid to obtain an approximate or optimal solution, Laha and Gupta, (2016) used an insertion technique to improve a proposed penalty-based construction algorithm. The algorithm was based on a Hungarian penalty method used for assigning a resource to an activity on a one-to-one basis that lowers a cost matrix to a penalty cost matrix. The proposed method used, was subdivided into three processes; the initial process used the Hungarian penalty method to derive a set of instances, the initial schedule was constructed in the second process and the third process improved on the proceeding processes using an insertion technique. To evaluate the efficiency of the proposed algorithm in terms of quality and computational speed, a comparative performance was carried out using seven well-known heuristics; Nearest Neighbour Heuristic, Farthest Insertion Heuristic, Cheapest Insertion Heuristic, Gangadharan and Rajendran (1993), Framinan and Nagano (1194), and Laha *et al.,* (2014). Average relative percentage deviation (ARPD), and percent of optimal solutions (for small problem sizes) or percent of best heuristic solutions (for large problem sizes) were the measurement metrics used.

The proposed method produced the best ARPD followed by the Cheapest Insertion Heuristic, it also had the best percentages for optimal and approximate solutions followed by the cheapest insertion heuristic.

Balseiro *et al.* (2011) used insertion heuristics to enhance the performance of an Ant Colony System algorithm which solves the Time-Dependent Vehicle Routing Problem with Time Windows (TDVRPTW), this led to a hybrid algorithm called a Multiple Ant Colony System algorithm hybridized with Insertion Heuristics (MACS-IH). The Ant Colony System algorithm produces results that were less than optimal at the final stages because there was a significant number of unrouted nodes, hence the reason for introducing the Insertion heuristics which helped to reduce the number of unvisited routes. The Insertion heuristics used in this study were the Sequential Nearest Neighbour Heuristic and the Parallel Nearest Neighbour Heuristic. The 56 Time-Dependent Solomon instances were used as testbeds for the proposed hybridized algorithm for four different solutions were constructed using a sequential NN heuristic, a sequential NN heuristic plus local search, a parallel NN heuristic and a parallel NN heuristic plus local search. The best metrics seen from results are the sequential NN heuristic plus local search and the parallel NN heuristic plus local search which improved the quality of the solutions produced by the constructive heuristics and the local search. The parallel NN heuristic measured the urgency of delivery but had the least impact. Three new insertion heuristic were formed: Local Search + Insertion (LSI), Local Search + MDL (LSMDL), and Local Search + MFT (LSMFT). The LSI explores all possible solutions and tried inserting the unrouted nodes into them, however, it fails to include tougher clients that require multiple successive changes in the solution before they can be served. Hence, the minimum delay metric (MDL) was introduced to measure the difficulty of inserting a new node. The maximal free time

(MFT) of a solution was used as a measure to find the maximum contiguous waiting time within a route which creates an allowance for possible insertions and in turn enhances the performance of the LSMDL.

Fan (2011) worked on The Vehicle Routing Problem with Simultaneous Pickup and Delivery with emphasis on Customer Satisfaction (VRPSPDCS); this is a VRPTWSPD. The work was motivated by the need to improve the decision-making abilities to optimize the efficiency of their supply chain. These included decisions that bothered on strategies for designing optimal routing network to potentially minimize cost as well as decisions capable of improving customer evaluation by taking into consideration customers' time windows. These requirements were modelled as a VRPDPDCS and an improvement heuristic was proposed. The first stage of the proposed method was to generate an initial solution through the Cheapest Insertion heuristic. The second stage which was the improvement solution was done via the TABU search procedure. The model was tested on six testbeds; the result showed promising performance. The improvement techniques discussed relied on the performance of the Cheapest Insertion Heuristic used to build the initial solution.

Wang and Chen (2012) proposed a Co-Evolution Genetic Algorithm (CEA) to help get a better solution method for a Simultaneous Delivery and Pickup Problem with Time Windows (SDPPTW). The proposed algorithm was developed using variants of the Cheapest Insertion method (CIM). It was noted that the typical generic algorithm was challenged with quick convergence that does not produce optimal results or low computational speed in obtaining convergence at optimal results, to overcome this issues, the CEA consecutively employed two separate evolutions that helped to keep the algorithm's ability to perform wide searches via Reproduction, Recombination and Selection, while increasing the computational speed in obtaining optimal results via

Reproduction, Local Improvement, Crossover, and Selection. The two variants of CIM used were the Multi-Parameter Cheapest Insertion Method (MPCIM) and the Random Seeds Cheapest Insertion Method (RSCIM). The MPCIM was used to speed up the global search process, it used a modified Insertion Criterion of Mester *et al.*, (2007), where the cost-saving threshold of values in range 0.2– 1.4 were tried in increments of 0.2 units for 100 customers. The RSCIM randomly generated the order of nodes for route expansion to widen the initial population search of the genetic algorithm for globally acceptable solutions. In evaluating performance, fifty-six 100-customer SDPPTW were used as testbeds and the experimental results showed that the CEA produced quality results in a better computational speed in comparison to the typical genetic algorithm.

Cruz *et al.*, (2012) proposed an improvement of the GENIL heuristic proposed by Souza *et al.*, (2011) in solving the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD). The algorithm was a hybrid of eight (8) heuristic techniques namely Cheapest Insertion, Cheapest Insertion with multiple routes, GENIUS, Variable Neighbourhood Search (VNS), Variable Neighbourhood Descent (VND), TABU Search (TS) and Path Relinking (PR). The Cheapest Insertion, Cheapest Insertion with multiple routes and the GENIUS, procedures were used to build the initial solution; this was in contrast with the design of GENIL which deployed the two other variations of cheapest insertion namely Route-by-Route Cheapest Insertion, Cheapest Insertion with Multiple Routes and a modified GENIUS heuristic to generate the initial solution. The Variable Neighbourhood Descent (VND) and the TABU Search (TS) were deployed as the local search procedure; the VND was iterated until there was no improvement in the search then the TS was called. The PR technique linked a high performing solution generated during the search to a local optimum after every iteration of the VND.

Thereafter, the Candidate List strategy was deployed as the removal procedure. The proposed method was experimented on available benchmark instances. The experimental result of the new technique outperformed the GENIL method. Its result was also compared with heuristic methods in literature by (Souza *et al.,* 2011; Subramanian *et al.*, 2011; Zachariadis *et al.,* 2010) and outperformed all except Subramanian *et al.,* (2011). The performance of the variated Cheapest Insertion heuristic in generating the initial solution was an important factor in the performance of the method.

Wang and Chen (2013) solved a flexible delivery and pickup problem with time windows (FDPPTW) using a co-evolution genetic algorithm (CEA) with a modified Cheapest Insertion Method (CIM) to improve the solution method. In a bid to solve the challenges of inflexible mix and reduced access time of vehicle routing problems with backhaul and time windows, to reduce the total distance covered and quantity of vehicles, the FDPPTW was modelled as a mixed binary integer programming problem. The model was implemented using CEA to generate approximate solutions in better time and fifty-six 100-customer FDPPTW testbeds gotten from the SDPPTWs in Wang and Chen (2012) were used in the experimental evaluation. A modified CIM called Random Seeds Cheapest Insertion Method (RSCIM) was employed in generating the random nodes used as the initial routes in contrast to providing separate routes individually. Also, the CEA results for the FDPPTWs was compared to that of the Wang and Chen (2012), it was observed that the CEA developed in FDPPTW scheme had a high computational speed and better results hence more flexible and economical. The FDPPTW achieved its goal of overcoming the shortcomings of the existing schemes for the delivery and pickup problems.

Morais *et al.*, (2014) proposed the use of a greedy tour construction heuristic based on the Nearest Insertion Heuristic to build an initial tour as part of the implementation of the Iterated local search (ILS) named X-ILS in solving the Vehicle Routing Problem with Cross-Docking (VRPCD). The technique added the node with the least increasing cost to the tour in what was referred to as the 2S-NI heuristic to builds the pickup and delivery path for the of the vehicle simultaneously. Six local search procedures were deployed to arrive at a local optimum. Thereafter, the process of perturbation was applied to the local optimum. Finally, a removal strategy was done to extract extra nodes. This process, which is the standard ILS was implemented with a slight modification of keeping a set of elite solutions, instead of a single current solution and tabu-search was not used. Results showed that the novel technique outperformed the existing ILS technique.

Weiler *et al.*, (2015) proposed a modification of otherwise deterministic approaches to solving the Probabilistic Travelling Salesman Problem (PTSP). The PTSP is a variant of the Travelling Salesman Problem (TSP), in which a probability function is assigned to a node, based on its possibility of visit. This was used to model an a-priori tour of cities most likely to be visited to minimize cost with respect to tour length. Thus, a real tour can be built based on this model where nodes that do not have to be traversed were skipped. In solving this problem, five deterministic construction tour techniques were considered and analyzed, namely Nearest Insertion Heuristic, Farthest Insertion Heuristic, Nearest Neighbour Heuristic, Radial Sorting Heuristic, and Space-Filling Curve. The Nearest Insertion and the Farthest Insertion Heuristics were then modified as Probabilistic Nearest Insertion (PNI) and Probabilistic Farthest Insertion (PFI) respectively. The PNI and PFI methods mirrored their deterministic counterparts by inserting nodes nearest or farthest to the last inserted node from the points modelled by the a-priori tour and with an evaluation of objective function on each possible position. This approach gave a better-quality result, but with a complexity of $O(n^4)$. To circumvent this problem, 'delta 1-shift' local search procedure embedded in a neighbourhood structure, proposed by Bianchi *et al.*, (2005) was introduced. This reduced the complexity to $O(n^3)$. The proposed method was experimented on benchmark instances from TSPLIB using the Xeon E5649, 2.53 GHz Quad-Core, running on Ubuntu 12.04.5 LTS (Precise Pangolin). The PFI outperformed the PNI; this was consistent with assertions in literature of the superiority of the Farthest Insertion technique over the Nearest Insertion Technique. Both the PFI and PNI outperformed their deterministic counterparts, albeit at a longer time.

Huang *et al.*, (2016) proposed a Sketch First approach to solving the Travelling Salesman Problem (TSP) in Location-Based Services (LBS). The idea was to find the optimal tour by mimicking the human cognitive approach of undertaking a global sketch for some chosen subset of the node T and then insert other nodes not in the initial tour based on a global-to-local refinement approach. The study started by exploring 7 existing tour construction heuristics namely the Farthest Insertion Heuristic, Nearest Insertion Heuristic, Cheapest Insertion Heuristic, Random Insertion Heuristic, the Nearest Neighbour Heuristic, the Nearest Addition Heuristic, and the Farthest Addition Heuristic. The insertion was done through local refinement. While the Farthest Insertion Heuristic was identified as the best performing techniques of the construction techniques considered, it was deemed unsuitable for this technique because the farther a node was from the current circuit, the higher the risk of error. The system was experimented on some benchmark instances and compared with existing methods. The performance of the techniques was encouraging.

In a proposed two-part search technique for solving TSPs by Jamal *et al.* (2017), a heuristic approach was used to find optimal results by first identifying an infeasible solution, then searching through a two-part space and narrowing the search space into a primal where a feasible solution can be obtained. Some insertion heuristics were used to find initial solutions for the proposed dual local search (DLS) framework, namely: Random, Farthest, and Nearest Insertions Heuristics. From Computational evaluation of the proposed DLS framework against the classical primal local search framework, the DLS performed significantly better than the insertion heuristics and also outperformed them with about 35% of optimal solutions and a range of 23% to 79% of approximate solutions.

Oliver *et al.* (2017) developed a hybridized heuristic of standard insertion and local search techniques with integer programming for solving the vehicle routing problem modelled by the Windy Rural Postman Problem with Zigzag Time Windows (WRPPZTW). A push forward technique with a constructed graph of ordered edges having priority costs was used in the proposed hybridization, a cheapest insertion method incorporating the priority costs was used to generate the order of insertion on the solution route while an integer program was used to complete the solution route for the Windy Rural Postman Problem. From the experiment performed on testbeds of over a hundred edges, the computational performance of the hybrid heuristic was compared to an exact method called BNC (Nossack *et al.*,2017). The hybrid produced 0.67% better solutions than BNC for a smaller number of edges, even though the hybrid is scalable to large instances, its performance requires an improvement

Lity *et al.,* (2017) made use of the Nearest Insertion (NEARIN) and the Farthest Insertion (FARIN) heuristics to generate near-optimal results for the optimal product order. These heuristics were picked on the basis that they perform well when generating approximate solutions for TSPs. From the computational evaluation performed with an optimal TSP Solver, both heuristics were seen to have produced good near-optimal solutions for products between 100 and 500 but as the number of products increased, their computational speed reduced. However, the TSP solver had way-less computational speed in generating results in contrast with the approximated algorithms.

Mário Mestria (2018) developed a hybrid method to solve the Clustered Travelling Salesman Problem (CTSP) based on Iterated Local Search (ILS) and Greedy Randomized Adaptive Search Procedure (GRASP) with integrated construction heuristic. This study was motivated by assertions in literature such as (Caserta & Voß, 2010) that a combination of two or more heuristics holds the promise of getting more robust and better results. Thus, the author proposed a new hybrid heuristic (VNRDGILS) that ran iterations of metaheuristics and included local search and specified perturbation strategies. The search procedure was greedy and randomized and could adapt to varying neighbourhood insertions. The neighbourhoods were added randomly to provide a basis for comparison with methods with deterministic neighbourhood additions. The constructive heuristic was based on modified Nearest Insertion Heuristic. The technique was experimented on different instances based on data with different levels of granularity. The result was compared with four other approximate methods and an exact method. Results obtained showed that the new heuristic outperformed a similar hybrid method with deterministic neighbourhood addition. It also outperformed four other heuristic methods considered in the study. Performances of these heuristic methods were also predicated on the performance of the modified Nearest Insertion method.

Babel, (2020) studied adaptations of some existing techniques such as Farthest Insertion Heuristic, Nearest Insertion Heuristic, Nearest Neighbour Heuristic, and so on, to solve the Dubins Travelling Salesman Problem (DTSP). The DTSP is a variant of the TSP concerned with determining the shortest "curvature-constrained closed path" through a set of destinations in a plane. The objective was to devise a suitable technique to optimize the headings of the targets of an open or closed sub tour, given a predefined sequence by discretely labelling the headings and building a make-shift network from which the shortest path could be created. Thus, a 3-tier algorithm with a differing number of heading was proposed. The first tier uses the sequence of targets generated from the initial solution of the Euclidean TSP. New targets were then iteratively inserted into the open sub tour in the second tier until all the targets in the tier had been added, then the circuit was closed. In the third tier, there were fewer targets to be added, this was done iteratively as well until all the targets had been added. The Farthest Insertion Method was deployed as one of the insertion procedures for adding targets in the K-insert algorithm. Other Insertion Algorithms included the Random Insertion, Nearest Insertion, and the Cheapest Insertion Heuristics. The methods were implemented in a simulated environment and compared with state-of-the-art methods. The performances of the methods were greatly influenced by the turning radius of the vehicle, as well as the abilities of the insertion technique. For a smaller radius, the Farthest-2-Insert had the best performance. This was closely followed by the Random-2-Insert technique, the Cheapest-2-Insert techniques, and finally the Nearest-2-Insert. This result was consistent with the experimental report on the Farthest Insertion Heuristic as the best performing insertion technique. The results of the method were competitive with respect to solution quality and running time.

CHAPTER THREE

3.0. METHODOLOGY

3.1. Research Approach – Introduction

Combinatorial Optimisation Problems (COP) are mostly NP-Hard, therefore, recurse is made to the use of heuristics for solving them. The goal of this study is to investigate some approximate methods, with a view to understanding their implementation details and how they are applied to the solution process of the Travelling Salesman Problem. And to consequently evolve a new and improved technique, with the potentials of outperforming existing state-of-the-art techniques. Tour construction heuristics were considered in this study, because of their importance both as solution techniques and as seed for the performance of other classes of heuristics. In this regard, two classic Tour Construction Techniques were considered, namely the Nearest Neighbour Heuristic and the Farthest Insertion Heuristic.

In achieving the objectives of this study, a review of existing approaches in solving the Travelling Salesman Problem was conducted. A hypothetical *postal route problem* was then formulated as a classic TSP problem. The postal towns are representative of the vertices. The vertices are connected by the edges, while the distances between the postal vertices measured in kilometre are the path costs. These parameters (vertices and the path cost) were used to generate the distance matrix which is the input to the program. The aim of the algorithms is to generate a Hamiltonian tour of the postal towns with minimal cost.

The heuristics were implemented on ten benchmark test sets as follows:

- 2 test sets with *no_of_nodes<100*

- 5 test sets with $100<no\_of\_node<1000$
- 3 test sets with $no\_of\_nodes \geq 1000$

All the algorithms were implemented using Java programming language.

The performances of the new and existing methods were evaluated using two approaches:

i. Solution quality: this is determined by computing the algorithm's tour cost relative to the optimal tour cost. The closer the tour cost is to the optimal cost, the better the quality of the technique.
ii. Computational speed approach: The computational speed is determined by computing the time taken to process the solution.

3.2. Building the Dataset

The heuristics were implemented on ten publicly available benchmark instances from TSPLIB, made available by Heidelberg University on http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/. The TSPLIB repository was chosen because of the wide-range of test cases available (for example, the datasets contain instances of pathfinding problems, drilling problems, programmed logic array and so on) and because the optimal cost of each of the instances have been computed and made available, thereby creating a basis for comparison of solution qualities against the optimal cost. There were 3 groups of instances tested; Group 1: instances whose nodes are less than 100. Group 2: instances whose nodes are more than 100 but less than 1000. Group 3: instances whose nodes are more than or equal to 1000.

The instances in group 1 are as follow:

att48 is a dataset of 48 US capital cities, it was generated by Padberg and Rinaldi, the optimal tour length of 33523. ***eil51*** is a dataset for 51 arbitrary cities problem by Christofides and Eilon, the optimal tour length is 426.

The instances in group 2 are as follow:

eil101 is a tour of 101 arbitraty cities generated by Christofides and Eilon with an optimal tour of 629. ***ch130*** and ***ch150*** are tours of 130 and 150 arbitrary cities respectively, compiled by Churritz. The computed optimal tour lengths are 6110 and 6528 respectively. ***pr439*** is a tour of 439 arbitrary cities by Padberg/Rinaldi, the optimal tour is 107217. ***rat 783*** is a Rattled grid problem of 783 nodes generated by Pulleybank with optimal tour length of 8806.

The instances in group 3 are as follow:

dsj 1000 has 1000 nodes with an optimal tour length of 18659688. ***u2319*** is a Drilling problem generated by Reinelt with 2319 nodes and optimal tour length of 234256. ***pcb3038*** is a Drilling problem of 3038 nodes generated by Juenger and Reinelt with an optimal tour length of 137694.

All the datasets used, were generated in the EUC 2D format and thereafter converted to FULLMATRIX format.

3.3. System Design

3.3.1. Framework for Tour Construction Heuristics

This study focused on tour construction heuristics. Tour Construction heuristics follow predefined process in building tours. These processes are carried out in three steps namely initialization, selection, and insertion. These processes differ from one method to the other, and they play a part in the performances of these different techniques. The initialization phase may start with a single node as in the Nearest Neighbour Heuristic, or may involve a subtour as in insertion techniques. Figure 3.1 depicts the generic framework for tour construction techniques.

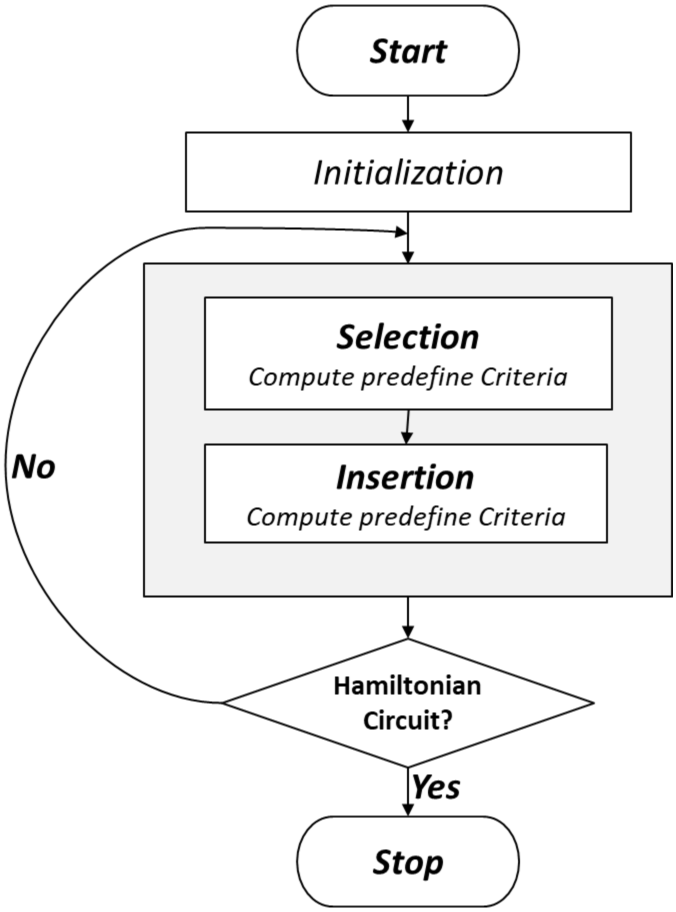

Figure 3.1. Generic framework for tour construction heuristics.

The step-wise activities continue iteratively until all the nodes have been added and the Hamiltonian cycle completed.

3.3.2. The Program Flow and Building Blocks

The program flow consists of three building blocks/phases. They include the input module, implementation module and the output module. Figure 3.2 shows the conceptual framework for this study.

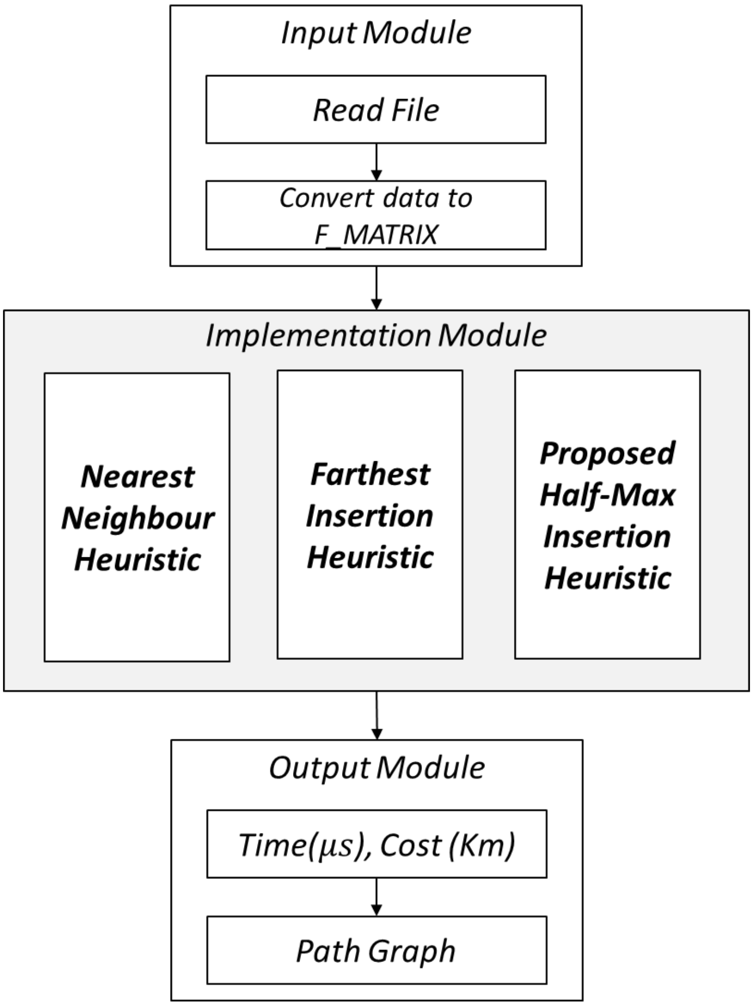

Figure 3.2. Research Conceptual Framework

The first phase was the input phase, where the input module reads the problem instance and prepares it for the implementation phase. At the input phase, the dataset was collected in the EUC_2D format and converted to distance matrix if the format is incompatible. Other formats for the dataset include the ATT, GEO, LOWER DIAG ROW, UPPER DIAG ROW, UPPER ROW, and FULL MATRIX. The EUC_2D format has $n\ rows$ of $\{i, x_i, y_i\}$ where n the number of nodes and $i \in \{1,2,3, \dots, n\}$ is computer using the formula:

$$d_{ij} = \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right\rfloor \qquad (3.1)$$

The dataset was thereafter converted to the FULL MATRIX input format which is compatible to the program using the following algorithm;

Algorithm 3.1: Algorithm for converting TSP dataset from EUC_2D to FULL MATRIX format

Input: E: EUC_2D
Output: FULL MATRIX data format

```
Start
Int f_matrix [x][y];
   for (int i = 0; i < x; i++)
     for (int j = 0; j < y; j++)
         read data from file
         dist_matrix = round(sqrt((x1 − x2)^2 + (y1 − y2)^2))
         Matrix [i][j] = dist_matrix
         Return dist_matrix
       Next Loop
    Next Loop
End
```

Figure 3.3 shows the flowchart of the input phase. This phase include reading of the input dataset, conversion of the dataset to the FULL_MATRIX format which is the format acceptable to the program and computation and generation of the distance matrix.

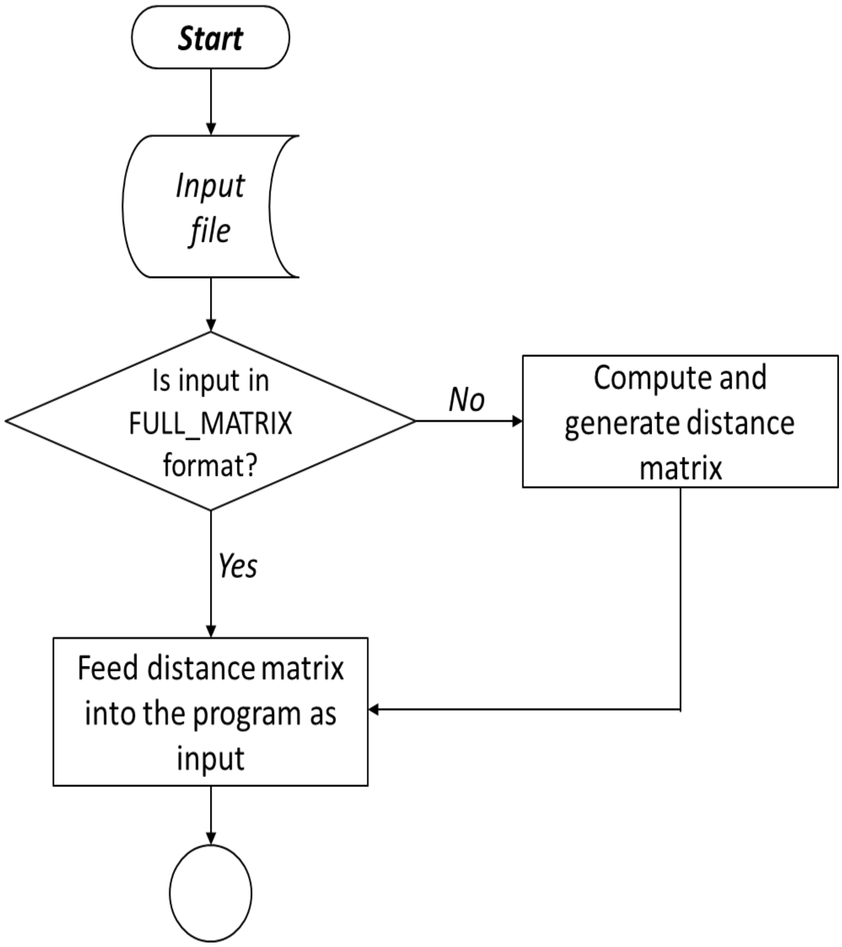

Figure 3.3. Flowchart of the input phase

The second phase was the implementation phase where the data is supplied to the implementing modules for implementation. The heuristics were implemented on the formatted instances using the Java Programming Language.

The output phase was the third phase. The program outputs the computational speed, the tour cost, and the tour path. The tour path was then further transformed graphically.

3.4. Research Materials and Methods.

3.4.1. Research Methods

Three methods were experimented on in this study. The first two were existing state-of-the-art construction techniques, namely, the Nearest Neighbour Heuristic and the Farthest Insertion Heuristic, while the third was the proposed insertion technique.

The NNH readily comes to mind when solving the TSP and FIH gives the best solution quality of all construction heuristics.

The objective of this study is to minimize the tour length, that is, to obtain solutions which are as close to their corresponding optimal solutions as possible.

Thus, given a tour distance d_{ab} and associated binary variable, earlier presented in Equations (1.10) and (1.11).

$$x_{ab} = \begin{cases} 1, \ if \ (a, b) \in E \ is \ in \ the \ tour \\ 0, \ if \ otherwise \end{cases} \tag{3.2}$$

An optimal solution is a solution in which:

$$tourcost = \sum_{(a,b)\in} d_{ab} x_{ab} \ is \ minimal \tag{3.3}$$

In obtaining the solution, the following assumptions were made:

i. Distances are nonnegative and symmetric

ii. Distances satisfy the triangle inequality, such that for every $a, b, c \in V$;

$$tourcost_{ac} \leq tourcost_{ab} + tourcost_{bc} \tag{3.4}$$

This means that, the cost of moving directly from vertex a to vertex c is not more than the cost of going via some intermediate vertex b. Without these assumptions, the bound given for the objective function value may not be valid.

Both the NNH and the FIH as well as the proposed technique are node-based which follow the following procedure (Huang and Yu, 2017):

Algorithm 3.2: Node-based Heuristic

Input: Q: Set of node

Output: T: the TSP for Q

```
Start
    T ← init (Q);
    While T does not contain all nodes in Q do
        V ← select (Q, T);
        Insert (v, T);
    Return T;
Stop
```

3.4.1.1.The Nearest Neighbour Heuristic

The Nearest Neighbour Heuristic is a classic tour construction heuristic for solving the Travelling Salesman Problem. It is equally one of the oldest and most widely used heuristics. It is simple, flexible, and fast. The NNH tries to solve the Travelling Salesman Problem using a greedy approach. The NNH starts with a city/node and builds the remaining tour by joining the node closest to the starting node to the tour. This process is iterated for all the nodes that are not yet part of the tour until the tour is fully build and a Hamiltonian circuit is formed. This process is greedy in nature; thus, the performance is relatively poor.

The pseudocode for the Nearest Neighbour Heuristic is as follow:

Algorithm 3.3: A Pseudocode for the Nearest Neighbour Heuristic

Input: set of nodes $V_{n=1,2,\ldots,n}$

Output: Path T

```
Start
Select an arbitrary node k ∈ V
Set Path ← k
   While {Path} ≠ {V} do
      Find node k + 1 ∉ Path such that dist(Path, k + 1) is minimal
      set Path ← k + 1
   End while
T ← path
return T
End
```

Figure 3.4 depicts the Nearest Neighbour Heuristic procedure in a flowchart as follow:

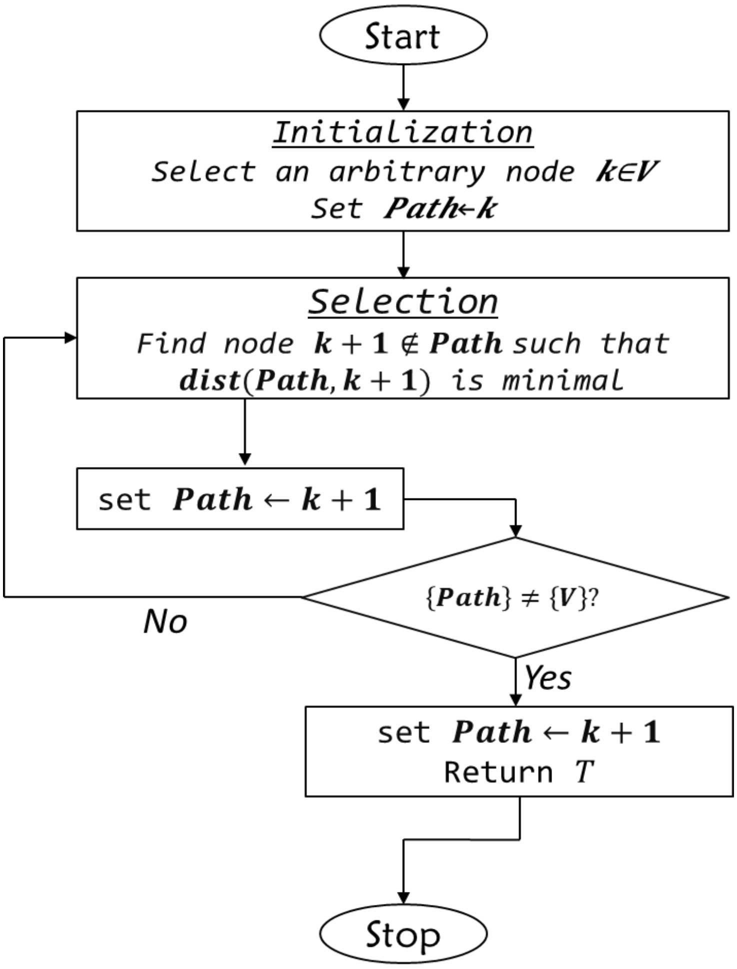

Figure 3.4. Flowchart of the Nearest Neighbour Heuristic.

Analytically, Rosenkrantz, *et al.*, (1977) had shown that for a TSP instance of nodes n, the approximation ratio/solution quality of the NNH is at most:

$$f_s/f_{OPT} = \frac{1}{2}[\log(n)] + \frac{1}{2} \tag{3.5}$$

of the optimal length, where f_s is the length of a tour by the solution and f_{OPT} is the optimal tour length.

The worst-case complexity of the NNH is $T(n) = O(n^2)$. However, in practice, the NNH can solve the TSP in good time, with much better solution quality. Experimentally, the NNH typically yield much better solutions than the worst case suggests. NNH often yield tour quality that is within $25\% - 30\%$ of the Held-Karp lower bound (Víctor *et al.*, 2020). The performance of the NNH greatly suffers from a phenomenon called the "*curse of dimensionally*" resulting from its greedy approach to solving the TSP. The NNH adds the lowest cost nodes as priority, and consequently, as the search space and nodes increase, more and more outliers are seen. Recent literature focus on using the NNH either as a part of a hybrid method as in (Huang and Yu, 2017; Lity *et al.*, 2017) or as a seed technique in a metaheuristic for building initial solutions (Rego *et al.*, 2011; Bernardino and Paias, 2018).

3.4.1.2.The Farthest Insertion Heuristic

The Farthest (also called Furthest) Insertion Heuristic belong to the family of insertion heuristics. Other known insertion heuristics include the Nearest Insertion Heuristic, Random Insertion Heuristic, Cheapest Insertion Heuristic, Priciest Insertion Heuristic, Quick insertion Heuristic, and Greatest angle Insertion Heuristic (Goetschalckx, 2011; Anbuudayasankar *et al.*, 2014).

Insertion heuristics starts from an arbitrary point to form a sub tour or partial circuit. Nodes not already in the sub tour are then inserted based on predefined criteria such that the increment to the total distance of the sub tour is minimized. (Huang *et al.*, 2016; Huang and Yu, 2017).

Suppose that node x is to be added to edge (x_i, x_j), and given the cost function $c(x_i, x_j, x)$, then,

$$c(x_i, x_j, x) = d(x, x_i) + d(x, x_j) - d(x_i, x_j) \qquad (3.6)$$

Each insertion technique method aims to add a node to an edge (that is between two nodes) at a minimal cost. Given the sub tour T_i, and given that x is the next node to be inserted, then the insertion technique inserts x between x_i^* and x_j^* in T_i according to:

$$(x_i^*, x_j^*) = \underset{(x_i x_j) \in T_i}{\operatorname{argmin}}\, c(x_i, x_j, x) \qquad (3.7)$$

Insertion techniques are desirable because of their speed, ease of implementation, quality of solutions, and the fact that they can be easily modified to handle complex constraints. Insertion techniques can be used to get a good tour construction solution. According to Rosenkrantz *et al.,* (1977), Insertion techniques find an $O(log\ n)$ approximate solutions.

The Farthest Insertion Heuristic (FIH) chooses the next node

$$x^* = argmax_{v \notin T_i}\{d(x, x_i), \forall x_i \in T_i \qquad (3.8)$$

The following pseudocode depicts the workings of the Farthest Insertion Techniques.

Algorithm 3.4: The Farthest Insertion Heuristic Pseudocode

Input: set of nodes $\boldsymbol{V}_{\boldsymbol{i}=1,2,\dots,\boldsymbol{n}}$
Output: Path $\boldsymbol{T}$

1. Start the tour from an arbitrary node $\boldsymbol{i}$
2. Add a node $\boldsymbol{j}$ nearest to $\boldsymbol{i}$ to form a partial circuit
$$\boldsymbol{T} = \boldsymbol{i} - \boldsymbol{j} - \boldsymbol{i}$$
3. Find a node $\boldsymbol{k}$ not in the partial circuit for which the distance to any of the nodes in the subtour is longest,
$$\boldsymbol{d}(\boldsymbol{k}, \boldsymbol{T}) = \max_{\boldsymbol{i} \notin \boldsymbol{T}} \boldsymbol{d}(\boldsymbol{i}, \boldsymbol{T})$$
4. Find an edge $[\boldsymbol{i}, \boldsymbol{j}]$ of the partial circuit to insert k, such that $\Delta \boldsymbol{f} = \boldsymbol{c}_{\boldsymbol{ik}} + \boldsymbol{c}_{\boldsymbol{kj}} - \boldsymbol{c}_{\boldsymbol{ij}}$ is minimal and insert $\boldsymbol{k}$.
5. Iterate step 3 until a Hamiltonian cycle is formed.

Figure 3.5 depicts the FIH procedure in a flowchart as follows:

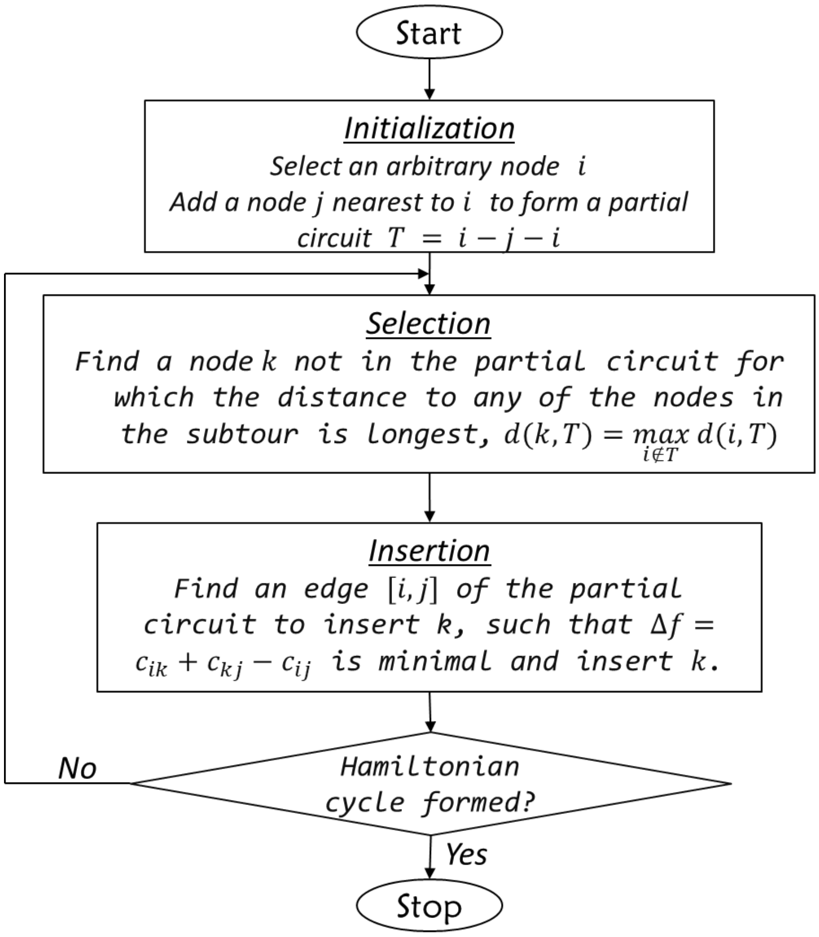

Figure 3.5. Flowchart of the Farthest Insertion Heuristic.

The FIH technique intuitively create an outline of sort, then fills in the details by adding nodes to the subtour. This expertly deals with the problem of outliers bedevilling the NNH method. Analytically, (Rosenkrantz, *et al.,* 1977; Johnson and McGeoch, 2002) had proven that the tours quality of insertion methods is at most twice of an optimal tour, while the approximation ratio/solution quality of a class of high performing Insertion Heuristics (Farthest Insertion Heuristic included) for a TSP instance of nodes n, is at most ${}^{f_s}/_{f_{OPT}} = \lceil \log(n) \rceil + 1$ of the optimal length.

The worst-case complexity if the Farthest Insertion Heuristic is $T(n) = O(n^2)$. In practice, however, the Farthest Insertion Heuristic is the best performing Insertion technique and often produce quality that are between 13% and 15 % worse than optimal tour (Reinelt, 1994; Johnson and McGeoch, 2002; Babel 2020). Generally, also, insertion techniques require more computational time than the NNH to complete tours.

Huang *et al.*, (2016) argued that although, FIH performs relatively very well, the distance between its circuit and new nodes to be inserted impede its performance in terms of accuracy. Suppose that a new node x is to be inserted into a partial tour p_tour, the closer x is to the edge (x_i, x_j), the lesser the likelihood of it introducing error. Suppose that nodes x_1 and x_2 are to be inserted into the same edge (x_i, x_j) of the partial tour p_tour to produce partial tours p_tour_1 and p_tour_2 respectively.

Suppose that cost function

$$c(x_i, x_j, x_1) < c(x_i, x_j, x_2) \qquad (3.9)$$

$$\text{and } d(p_{tour_1}) \leq d(p_{tour_2}) \qquad (3.10)$$

then the upper bounds of error rate for the two tours are

$$\frac{d(p_{tour_1})}{d(ptour)} = \frac{d(p_{tour}) + c(x_i, x_j, x_1)}{d(ptour)} \qquad (3.11)$$

$$\text{and } \frac{d(p_{tour_2})}{d(ptour)} = \frac{d(p_{tour}) + c(x_i, x_j, x_2)}{d(ptour)} \qquad (3.12)$$

$$\text{such that } \frac{d(p_{tour_1})}{d(ptour)} < \frac{d(p_{tour_2})}{d(ptour)} \qquad (3.13).$$

Thus, the performance of FIH still leaves much to be desired in terms of solution quality. If inserting nearest nodes to the circuit leads to outliers and the performance if

FIH is impeded by longer distance, perhaps a half max insertion may yield better solution.

3.4.1.3.The Proposed Half Max Insertion Heuristic (HMIH)

The proposed technique is an insertion method referred to in this study as the Half Max Insertion Heuristic (HMIH). The motivation was to explore some techniques with the possibilities of improving the accuracy of the Farthest Insertion Heuristic. The design of the HMIH was motivated by two observations in literature: One, the superior solution quality of Convex-hull based insertion techniques based on the use of polygons as initial tour (Huang *et al.,* 2016; Huang and Yu, 2017; Víctor *et al.,* 2020) and secondly, the limitation of the FIH's accuracy due to the distance between its initial circuits and the next node to be inserted (Huang *et al.,* 2016).

The insertion heuristics randomly pick one node from Q by $init(Q)$ and creates a partial circuit which is expanded with every iteration. The partial circuit is made up of the points u, v, w to form a minimum polygon.

Let T_i be the partial circuit over nodes of size i such that

$$T_i = (\pi_1, \pi_2, \dots \pi_i, \pi_1) \qquad (3.14)$$

In the $(i + 1)th$ iteration, the insertion heuristics attempt to add one node into the current circuit by minimizing the increment of the total distance of the circuit. The objective is to: determine how to select a node, x, from $Q \backslash T_i$ and determine how to insert x into T_i to obtain T_{i+1}.

Consider an insertion of a node $x(\notin T_i)$ between $u, v\ and\ w\ in\ T_i$:

The method first determines the longest distance d_{max} of any node from either of $u\ or\ v$ and compute ${}^{1}/_{2}\, d_{max}$. Then find a node w not in the subtour whose distance from either $u\, or\, v\ \approx {}^{1}/_{2}\, d_{max}$. Determine an edge (u, v) of the subtour to which the insertion of w gives the smallest increase of length, that is for which

$$\Delta f = c_{ux} + c_{xv} + c_{wx} - c_{uvw}\ is\ smallest \qquad (3.15)$$

Insert x between $u, v\ and\ w$. This process is iterated until a Hamiltonian cycle is formed.

The procedure is as follow:

Algorithm 3.5: The Novel Half Max Insertion Heuristic Algorithm

Input: set of nodes $\boldsymbol{V_{i=1,2,\dots,n}}$
Output: Path $\boldsymbol{T}$

1. **Start** with a sub-graph consisting of node $\boldsymbol{u}$ only.
2. **Find** nodes $\boldsymbol{v}$ and $\boldsymbol{w}$ randomly to form sub-tour $\boldsymbol{u - v - w - u}$.
3. **Compute** the length of the farthest node $\boldsymbol{d_{max}}$ from the subtour and compute $\boldsymbol{{}^{1}/_{2}\, d_{max}}$
4. **Find** a node $\boldsymbol{w}$ not in the subtour whose distance from any node in the subtour $\approx \boldsymbol{{}^{1}/_{2}\, d_{max}}$
5. **Find** the $\boldsymbol{arc}$ $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ in the sub-tour which minimizes $\boldsymbol{c_{ux} + c_{xv} + c_{wx} - c_{uvw}}$. Insert $\boldsymbol{x}$ between $\boldsymbol{u\ v, and\ w}$.
6. **Iterate** step 3 until a Hamiltonian cycle is formed

The HMIH searches require $O(n)$ time, therefore, the time complexity of the algorithm is $O(n^2)$. The procedure is further depicted in the following flowchart in Figure 3.6.

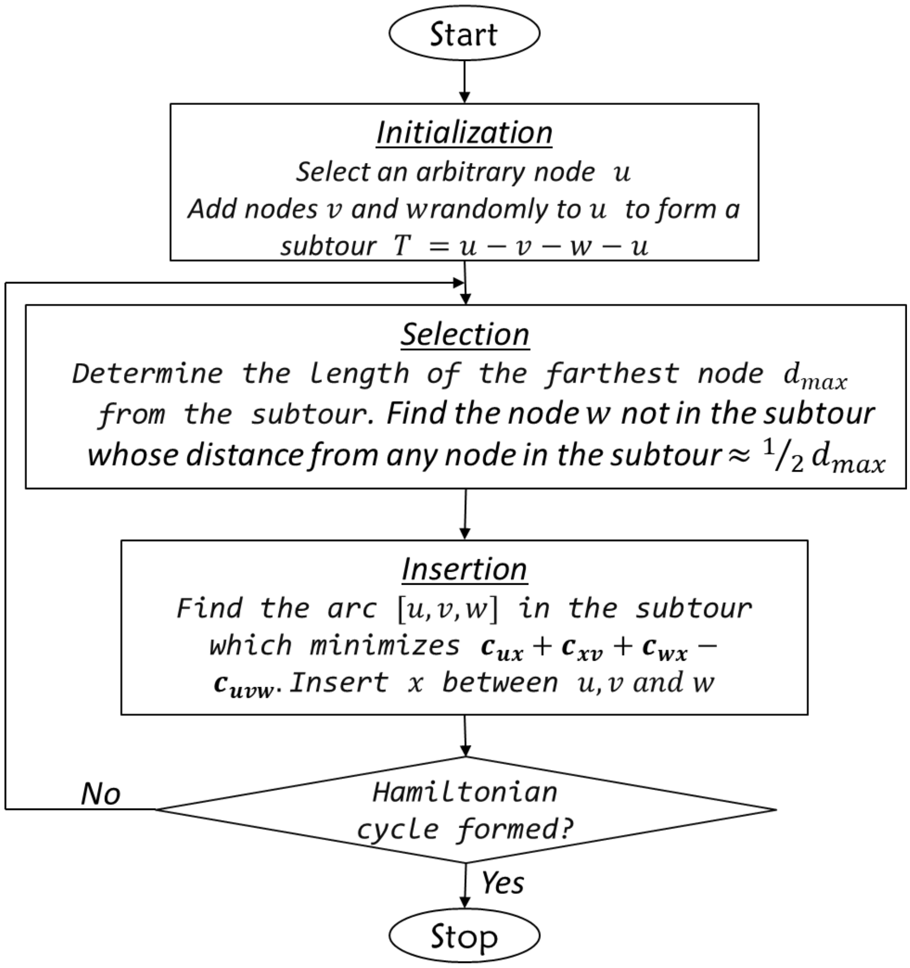

Figure 3.6. Flowchart of the Half Max Insertion Heuristic

In implementing the three heuristics, the development tools used were as follows:

a. JAVA programming language running on version 13.0.1.

b. GNUplot 5.2, patchlevel 8 was used to represent the tour graphically.

The heuristics were implemented using Java Programming Language running on Intel Pentium Core i7 3GHz, Windows 10 (64bit).

CHAPTER FOUR

4.0. RESULTS AND DISCUSSSION OF FINDINGS

4.1. Results

The heuristics were implemented on ten publicly available benchmark instances from TSPLIB made available by Heidelberg University on http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.

"TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types".

The TSPLIB repository was chosen because of the wide range of test cases available on it (for example, the datasets contain instances of pathfinding problem, drilling problems, programmed logic array and so on) and because the optimal costs of each of the instances have been computed and made available, thereby creating a basis for comparison of solution quality against the optimal cost. A list of benchmark instances and their optimal tour costs can be viewed at the following url:

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

TSPLIB contains instances of problems such as Symmetric Travelling Salesman Problem (TSP), Hamiltonian Cycle Problem (HCP), Asymmetric Travelling Salesman Problem (ATSP), Sequential Ordering Problem (SOP), Capacitated Vehicle Routing Problem (CVRP).

Three (3) groups of instances were tested; Group one: instances whose nodes are less than 100. Group two: instances whose nodes are more than 100 but less than 1000. Group three: instances whose nodes are equal to, or more than 1000. The instances considered, their number of nodes and optimal tour length are presented in Table 4.1.

Table 4.1. Ten benchmark instances and their optimal tour length (Km).

S/N	Instances	No of Nodes	OPT
1	*att48*	48	33523
2	*eil51*	51	426
3	*eil101*	101	629
4	*ch130*	130	6110
5	*ch150*	150	6528
5	*pr439*	439	107217
6	*rat 783*	783	8806
7	*dsj1000*	1000	18659688
8	*u2319*	2319	234256
9	*pcb3038*	3038	565530

The datasets which were generated in the EUC_2D format was reformatted to FULLMATRIX form using a conversion module. Figure 4.1(a and b) shows the *ulysses16* sample set as EUC_2D and as FULLMATRIX after conversion.

```
NAME: ulysses16.tsp
TYPE: TSP
COMMENT: Odyssey of Ulysses (Groetschel/Padberg)
DIMENSION: 16
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY
NODE_COORD_SECTION
 1 38.24 20.42
 2 39.57 26.15
 3 40.56 25.32
 4 36.26 23.12
 5 33.48 10.54
 6 37.56 12.19
 7 38.42 13.11
 8 37.52 20.44
 9 41.23 9.10
 10 41.17 13.05
 11 36.08 -5.21
 12 38.47 15.13
 13 38.15 15.35
 14 37.51 15.17
 15 35.49 14.32
 16 39.36 19.56
 EOF
```

Figure 4.1a: *Ulysses 16* in EUD_2D format.

Distance of (city#/city#)

1	509	501	312	1019	736	656	60	1039	726	2314	479	448	479	619	150
509	1	126	474	1526	1226	1133	532	1449	1122	2789	958	941	978	1127	542
501	126	1	541	1516	1184	1084	536	1371	1045	2728	913	904	946	1115	499
312	474	541	1	1157	980	919	271	1333	1029	2553	751	704	720	783	455
1019	1526	1516	1157	1	478	583	996	858	855	1504	677	651	600	401	1033
736	1226	1184	980	478	1	115	740	470	379	1581	271	289	261	308	687
656	1133	1084	919	583	115	1	667	455	288	1661	177	216	207	343	592
60	532	536	271	996	740	667	1	1066	759	2320	493	454	479	598	206
1039	1449	1371	1333	858	470	455	1066	1	328	1387	591	650	656	776	933
726	1122	1045	1029	855	379	288	759	328	1	1697	333	400	427	622	610
2314	2789	2728	2553	1504	1581	1661	2320	1387	1697	1	1838	1868	1841	1789	2248
479	958	913	751	677	271	177	493	591	333	1838	1	68	105	336	417
448	941	904	704	651	289	216	454	650	400	1868	68	1	52	287	406
479	978	946	720	600	261	207	479	656	427	1841	105	52	1	237	449
619	1127	1115	783	401	308	343	598	776	622	1789	336	287	237	1	636
150	542	499	455	1033	687	592	206	933	610	2248	417	406	449	636	1

Figure 4.1b: *Ulysses 16* in FULLMATRIX format

The implementation module generates three outputs. The first is the computation time in nano seconds (μs). Nano second is one billionth of a second. Evidently, the accuracy of time is improved at that level of granularity. The second output is the tour cost, that is the distance taken to generate the tour. This is necessary for the performance evaluation of the heuristic. The third output is the tour path, that is the order in which the nodes join the tour. The tour path is an input to GNUplot which is used to generate the tour. Table 4.2 highlights the three heuristics and their computational speed in solving the TSP instances considered while table 4.3 shows the tour cost of each of the three heuristics on the TSP instances.

Table 4.2. Computational speed of NNH, FIH and HMIH on ten benchmark instances

S/N	**Instances**	**No of Nodes**	**Computational Speed (μS)**		
			NNH	**FIH**	**HMIH**
1	*att48*	48	14943601	19455700	24837700

2	*eil51*	51	24014300	26145600	28442300
3	*eil101*	101	5127500	83707200	99243300
4	*ch130*	130	28423900	130305001	85040424
5	*ch150*	150	8424700	163042000	217278100
6	*pr439*	439	74732299	4583954300	7078104000
7	*rat 783*	783	284603200	34059530600	72396010300
8	*dsj1000*	1655	487645399	1.3343E+11	2.05499E+11
9	*u2319*	2319	3344218900	2.3164E+12	4.07294E+12
10	*pcb3038*	3038	6527888600	6.57109E+12	1.01185E+13

Table 4.3. Tour cost of NNH, FIH and HMIH on ten benchmark instances

S/N	**Instances**	**No of Nodes**	**OPT**	**Tour Cost**		
				NNH	**FIH**	**HMIH**
1	*att48*	48	33523	40524	35775	35657
2	*eil51*	51	426	510	471	471
3	*eil101*	101	629	811	690	690
4	*ch130*	130	6110	7198	6951	6650
5	*ch150*	150	6528	8191	7542	7211
6	*pr439*	439	107217	139149	122957	124322
7	*rat 783*	783	8806	10779	10828	10434
8	*dsj1000*	1655	18659688	24631468	23563031	20610943
9	*u2319*	2319	234256	281978	272959	256601
10	*pcb3038*	3038	137694	175788	173038	166196

It is evident from table 4.3 that the proposed HMIH has a smaller tour cost and is closer to the optimal tour cost in terms of solution quality than both FIH and NNH. FIH however, compares more favourably with HMIH than NNH.

The tour path which is the order in which the nodes join the tour is fed as an input to GNUplot to generate the path graph. GNUplot is an open-source command-line graphing utility available under the General Public Licence. The path graph on GNUplot is implemented using the following command:

```
plot "att48.tsp" with linespoint
```

"*att48.tsp*" is the filename consisting the tour path output by the program, while "*linespoint*" connects all the point in the right order.

The tour graph of the HMIH, FIH and NNH for some benchmark instances are presented in Figures 4.2, 4.3, 4.4 and 4.5. Figures 4.2 a, b and c show the path graph of NNH, FIH and HMIH respectively for the *att48*. Figures 4.3 a, b and c show the path graph of NNH, FIH and HMIH respectively for the *eil51*. Figures 4.4 a, b and c show the path graph of NNH, FIH and HMIH respectively for the *eil101* and Figures 4.5 a, b and c show the path graph of NNH, FIH and HMIH respectively for the *ch150*.

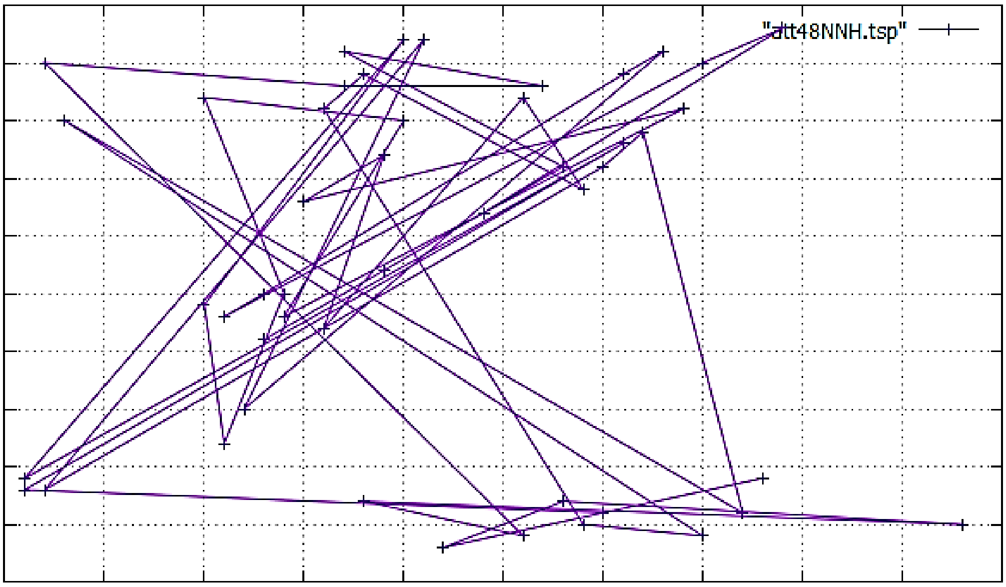

Figure 4.2a: Path graph of NNH for the *att48* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{40524}$)

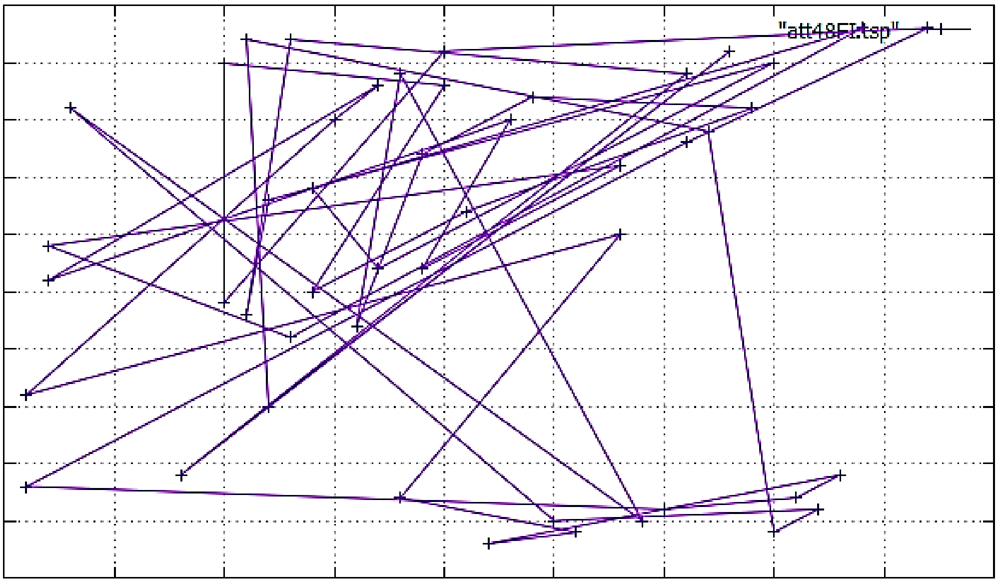

Figure 4.2b: Path graph of FIH for the *att48* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{35774}$)

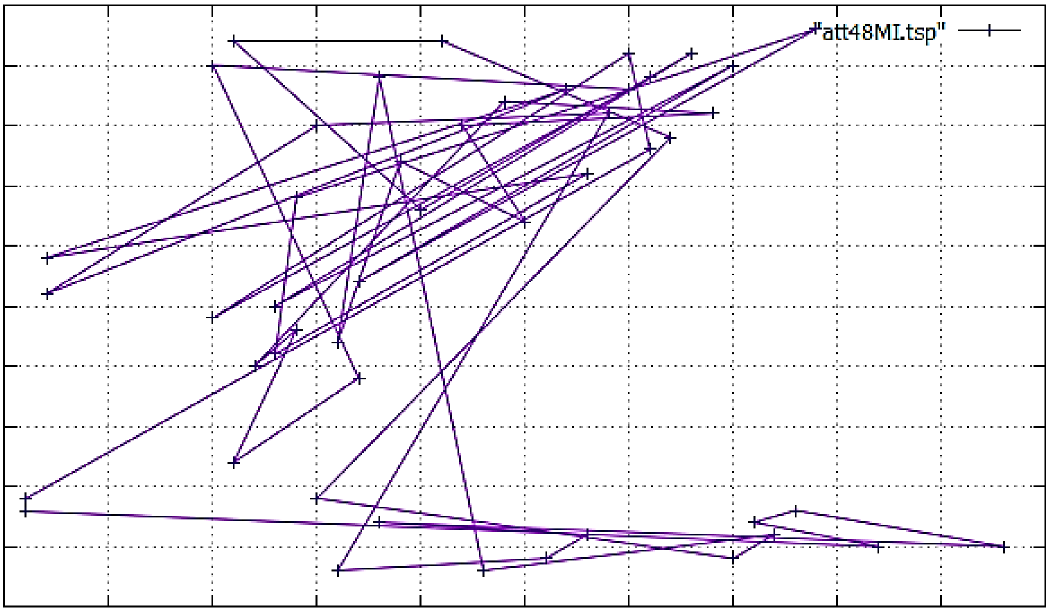

Figure 4.2c: Path graph of HMIH for the *att48* instance **($cost\ (km) = 35657$)**

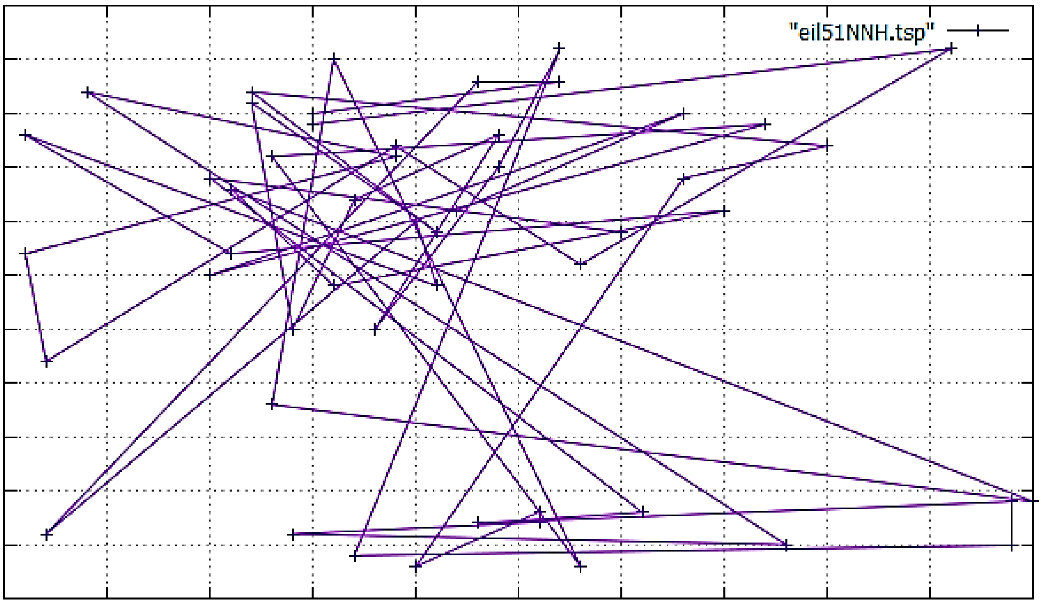

Figure 4.3a: Path graph of NNH for the *eil51* instance ($cost\ (km) = 510$)

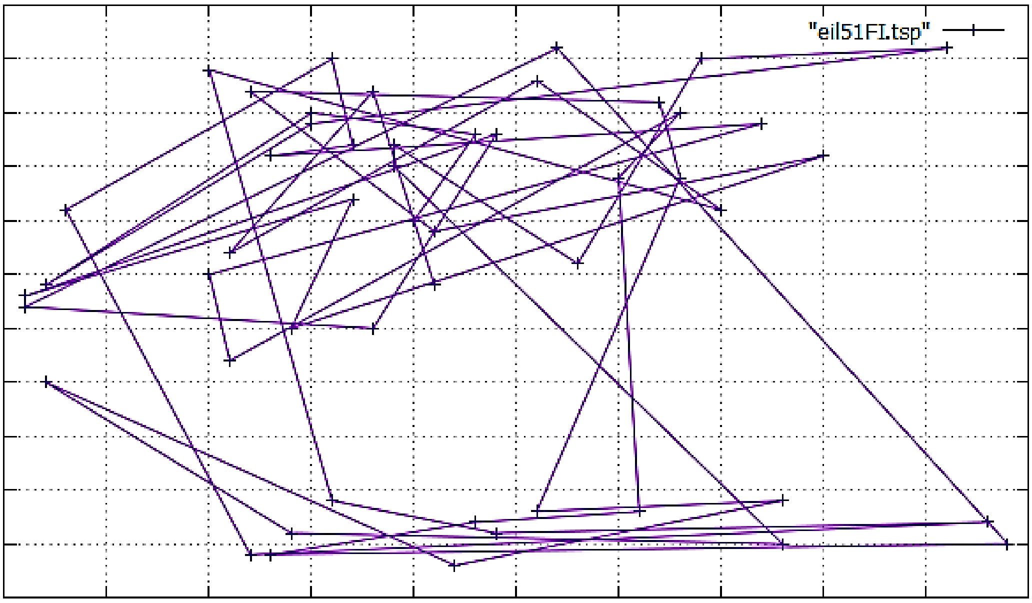

Figure 4.3b: Path graph of FIH for the *eil51* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{471}$)

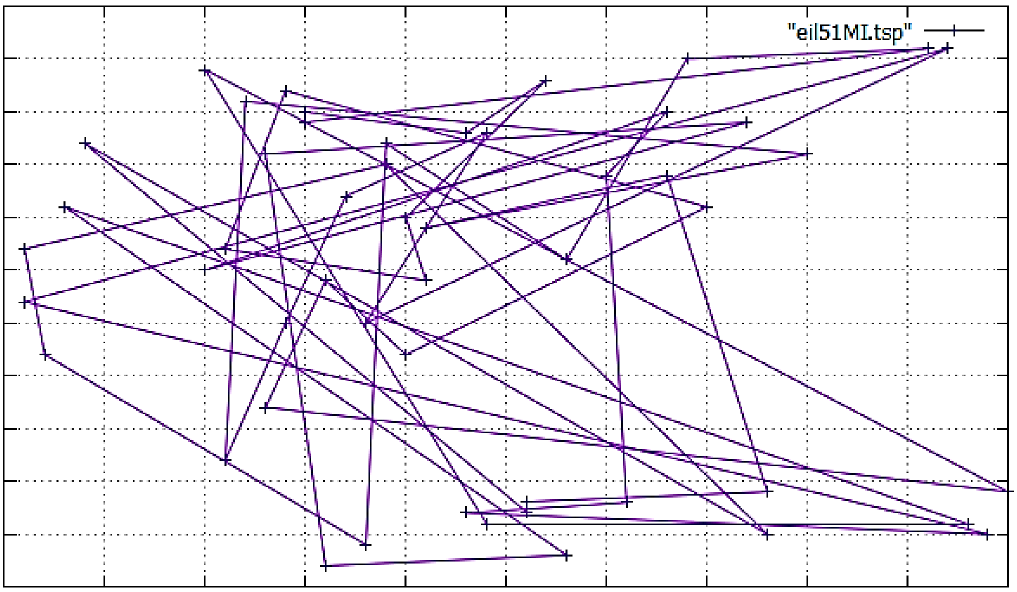

Figure 4.3c: Path graph of HMIH for the *eil51* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{471}$)

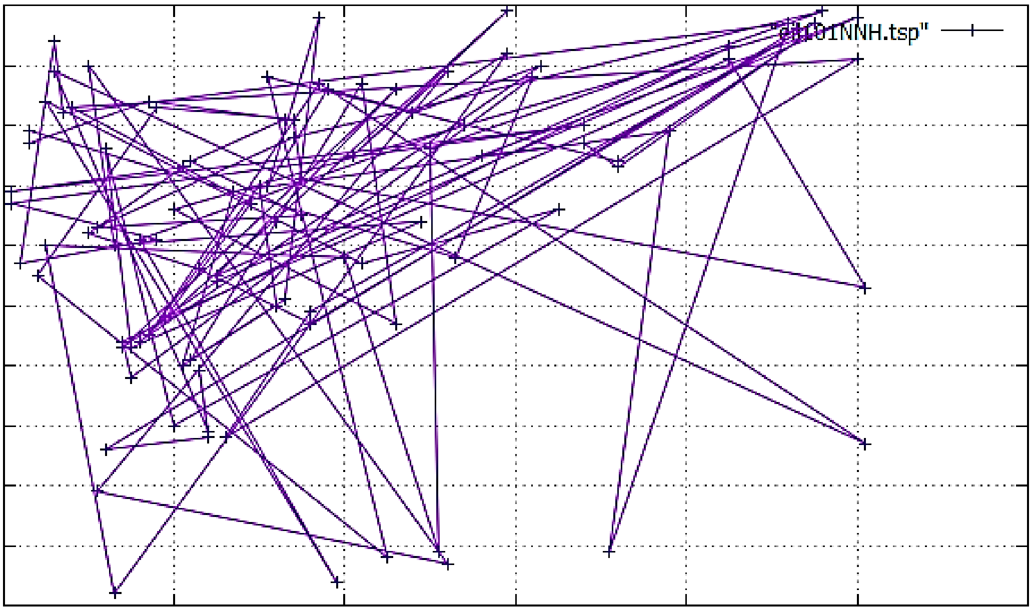

Figure 4.4a: Path graph of NNH for the *eil101* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{811}$)

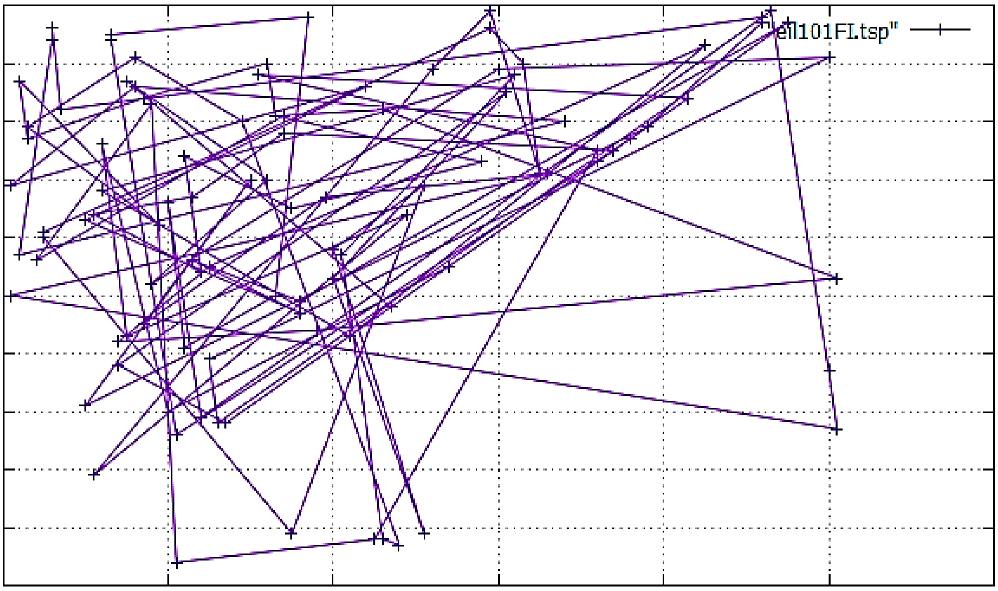

Figure 4.4b: Path graph of FIH for the *eil101* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{690}$)

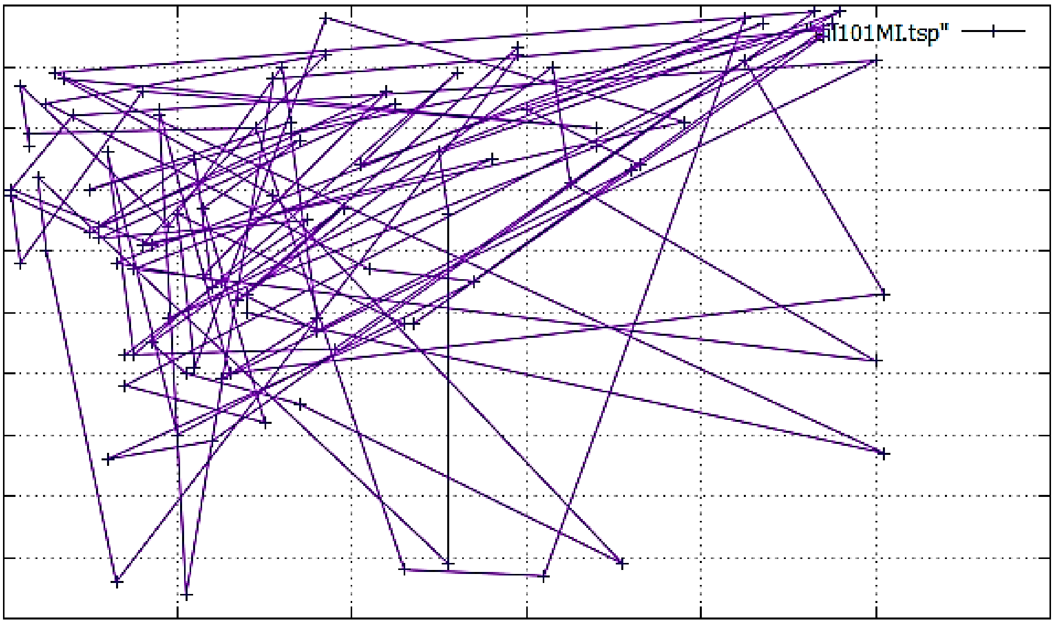

Figure 4.4c: Path graph of HMIH for the *eil101* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{690}$)

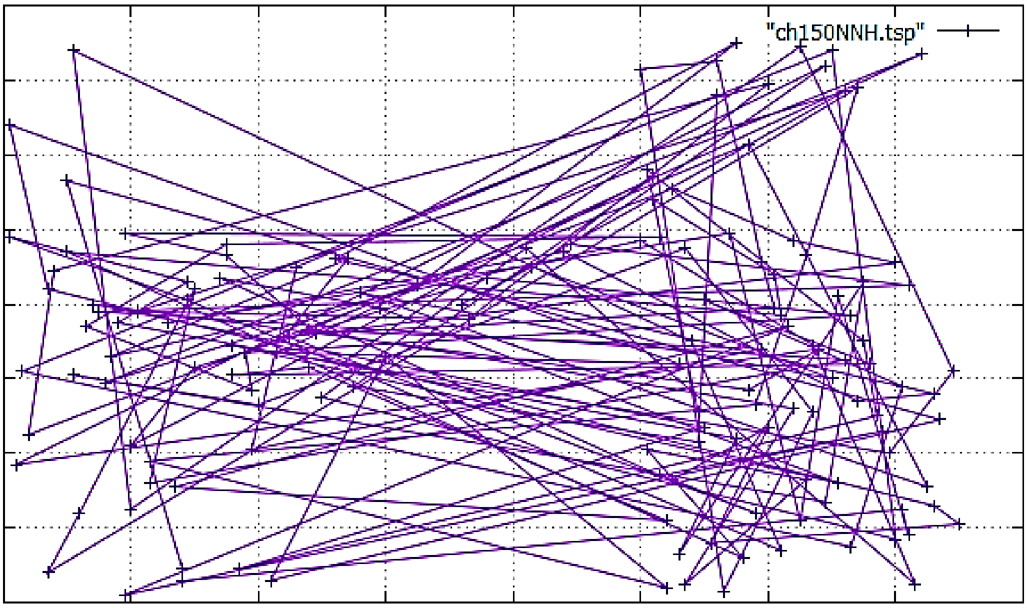

Figure 4.5a: Path graph of NNH for the *ch150* instance ($\boldsymbol{cost}\ (\boldsymbol{km}) = \mathbf{8191}$)

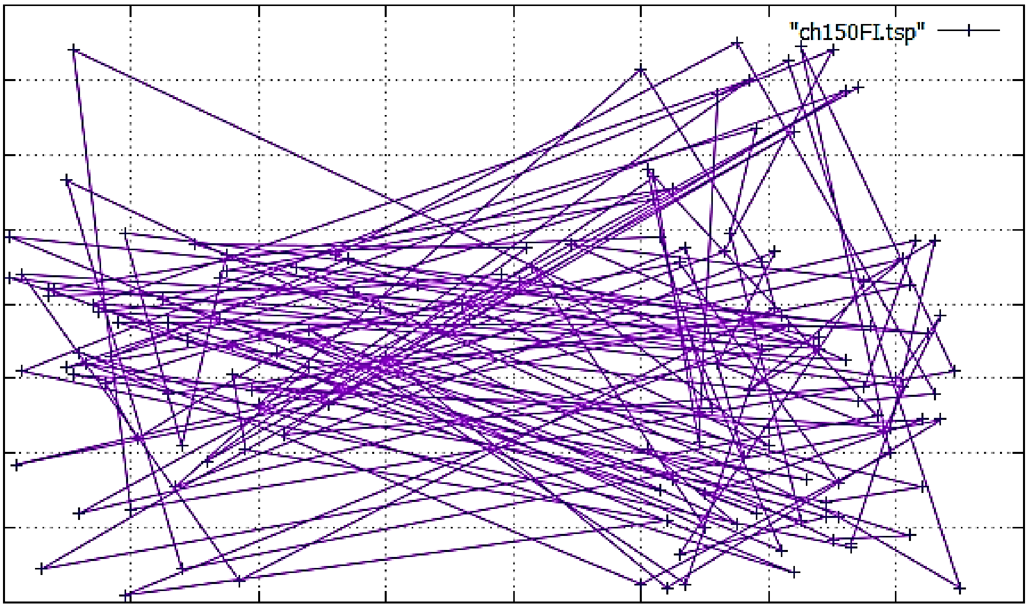

Figure 4.5b: Path graph of FIH for the *ch150* instance ($\boldsymbol{cost}$ ($\boldsymbol{km}$) = $\mathbf{7542}$)

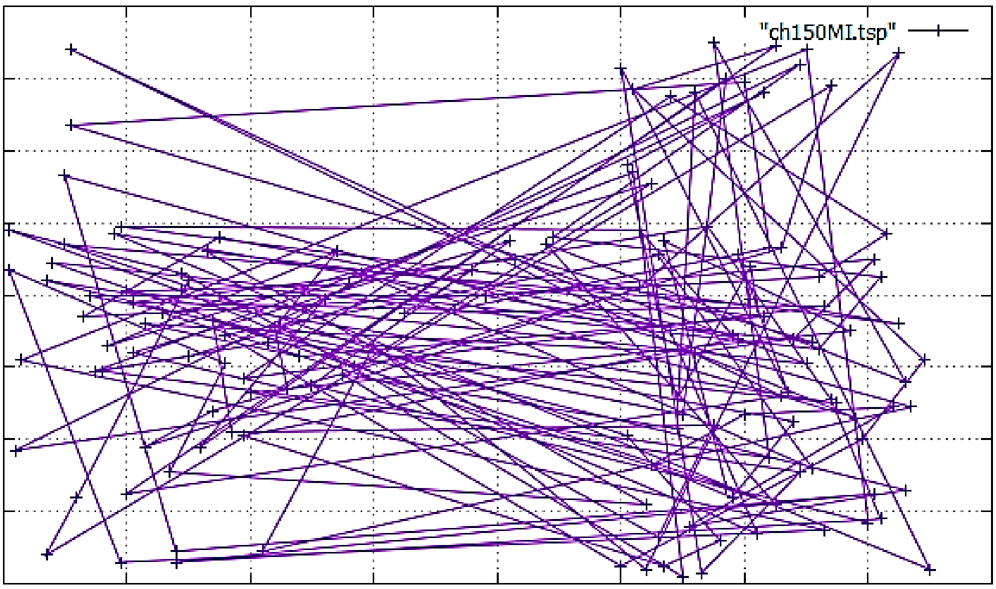

Figure 4.5c: Path graph of HMIH for the *ch150* instance ($\boldsymbol{cost}$ ($\boldsymbol{km}$) = $\mathbf{7211}$)

4.2. Performance Evaluation and Discussion

4.2.1. Comparative Evaluation of the Heuristics' Computational Speed

Table 4.2 reveals that the NNH had the fastest computational speed, followed by the FIH and then the proposed HMIH technique in all the instances. It should be noted that the proposed HMIH compared favourably with the FIH in this regard. This is consistent with literature findings that insertion techniques require more computational time than the NNH to complete tours (Reinelt, 1994; Johnson and McGeoch, 2002; Laha *et al.*, 2016; Babel 2020). Additionally, the increased computational time of the proposed HMIH can be attributed to the additional computation of the *half max* insertion criteria. This is consistent with works by Reinert, (1994), Laha *et al.*, (2016), Lity *et al.*, (2017) and Babel (2020) which suggest that computational speed is affected by the insertion criteria computations.

4.2.2. Comparative Evaluation of the Heuristics' Solution Quality

In evaluating the solution quality of the heuristics, the following parameters were deployed:

Percentage Error (δ): the percentage error of the heuristics' solution quality is the percentage deviation of the solution from the optimal tour solution. This is computed as:

$$\delta = \frac{sol\eta - opt}{opt} \times 100\% \qquad (4.1)$$

where $sol\eta$ is the solution cost obtained by each heuristic, and opt is the optimal solution cost. This is the same thing as the performance ratio for non-optimal heuristics.

***Quality impr.* (Σ):** this the improvement of the HMIH method's solution quality with respect to NNH and FIH. This is computed by:

$$\Sigma = \mathrm{E}_{NNH/FIH} - \mathrm{E}_{HMIH} \qquad (4.2)$$

where $\mathrm{E}_{NNH/FIH}$ is the error in percentage of the NNH or FIH and E_{HMIH} is the error in percentage of the HMIH.

***Goodness Value* (g):** this is also referred to as the accuracy. This is the inverse of error and is computed as

$$g = \left(1 - \frac{soln - opt}{opt}\right) 100\% \qquad (4.3)$$

Table 4.4 displays the $percentage\ error, quality\ impr\ and\ goodness\ value$ for all the heuristics on the ten benchmark instances.

Table 4.4. percentage error, $quality\ impr$ and goodness value for all the heuristics on the ten benchmark instances

S/N	Instances	No of Nodes	HMIH (%)				FIH(%)		NNH(%)	
			δ	Σ_{NNH}	Σ_{FIH}	g	δ	g	δ	g
1	*att48*	48	6.3	14.6	0.4	93.7	6.7	93.3	20.9	79.1
2	*eil51*	51	10.6	9.1	0	89.4	10.6	89.4	19.7	80.3
3	*eil101*	101	9.7	19.2	0	90.3	9.7	90.3	28.9	71.1
4	*ch130*	130	8.8	9.0	5.0	91.2	13.8	86.2	17.8	82.2
5	*ch150*	150	10.5	15	5	89.5	15.5	84.5	25.5	74.5
6	*pr439*	439	15.9	13.9	-1.2	84.1	14.7	85.3	29.8	70.2
7	*rat 783*	783	18.5	4.9	4.4	81.5	22.9	77.1	22.4	77.6
8	*dsj1000*	1655	10.5	21.5	15.8	89.5	26.3	73.7	32	68
9	*u2319*	2319	9.5	10.9	7	90.5	16.5	83.5	20.4	79.6
10	*pcb3038*	3038	20.7	7	5	79.3	25.7	74.3	27.7	72.3

The lower the value of the percentage error (δ) of the technique, the closer it is to optimal cost and thus the better the technique. Conversely, techniques with higher goodness value (g) are adjudged to be better the technique than those whose goodness value are lower. From Table 4.4, it can be deduced that the HMIH performed better than both the NNH in all instances and in all, but three instances (*pr439, eil51 and eil101*) for FIH. This is because, the HMIH has smaller percentage error (δ) and higher accuracy (g) compared to the NNH in all instances. In the case of FIH, the HMIH did better in terms of percentage error and accuracy except in the case of instances *pr439, eil51 and eil101*. FIH outperformed HMIH for pr439, while FIH and HMIH had equal percentage error and accuracy for *eil51 and eil101*. This is depicted graphically in Figure 4.6 and 4.9 respectively.

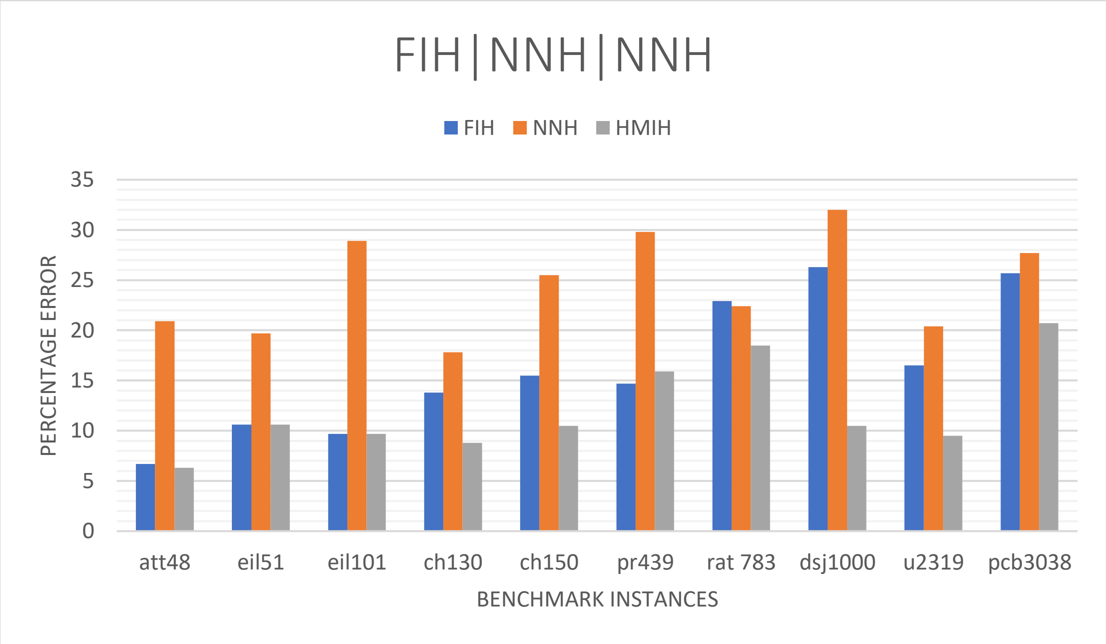

Figure 4.6. Percentage error value for FIH, NNH and HMIH

The lower the value of the percentage error (δ) of the technique, the closer it is to optimal cost and thus the better the technique

On the average, the NNH tour quality was 24.51% worse than the optimal tour. Additionally, the FIH average performance for the instances considered was 16.24% of the Held-Karp lower bound. The NNH reached a peak of 32% and a base value of 17.8%. The FIH reached a peak of 26.3% and a base value of 6.7%. These performances are consistent with documented findings about NNH and FIH in literature (Reinelt, 1994; Johnson and McGeoch, 2002; Babel 2020). On the other hand, the performance of HMIH was 12.1% worse than the optimal tour length. On the average, the proposed HMIH has a 4.14%-point quality improvement over FIH. Figure 4.6 shows a chart of the percentage deviation/error of NNH, FIH and HMIH from the optimal tour length.

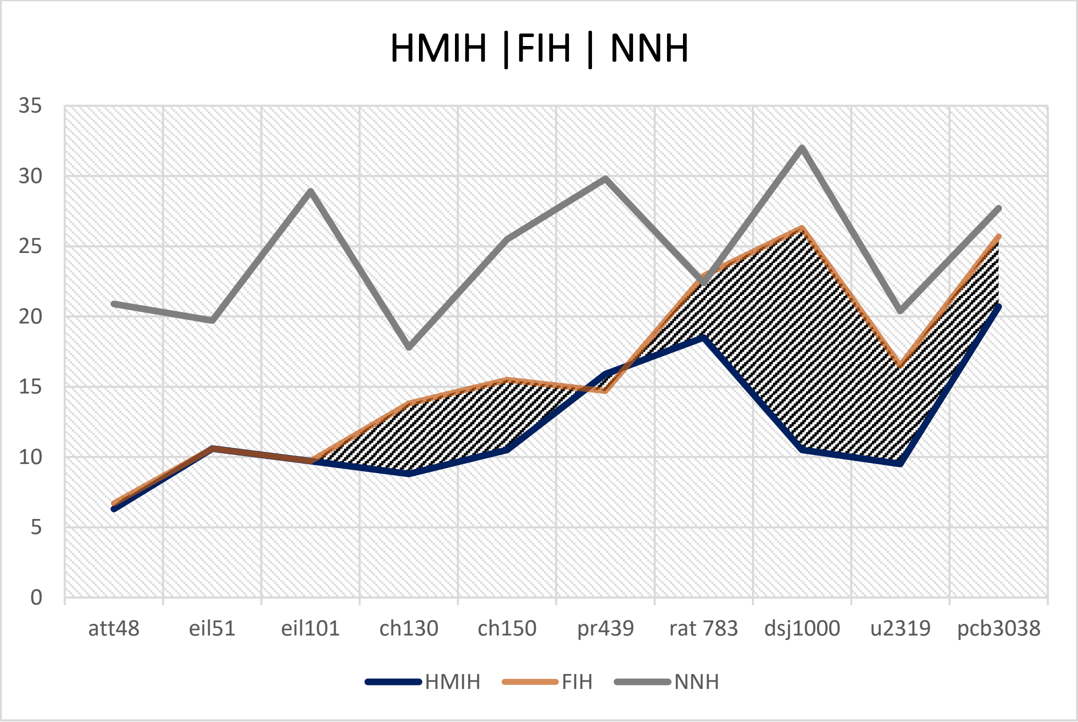

Figure 4.7. Percentage error of NNH, FIH and HMIH on the ten benchmark instances depicting the quality improvement of the HMIH over NNH and FIH

The shaded area of the chart denotes the quality improvement of the HMIH over the FIH.

4.3. Findings

The proposed HMIH consistently outperformed the FIH across a wide spectrum of benchmark instances with statistical significance of as much as 16% at some point as highlighted by the shaded area of quality improvement in Figure 4.6. The average goodness value of the proposed HMIH was 86.9% compared to 81.7% for the FIH and 74.5% for the NNH. This means that the proposed HMIH has a higher accuracy than FIH and NNH (see Figure 4.7). It is worthy of note that the FIH is considered the best performing Insertion techniques and other lower-order complexity heuristics (Reinelt, 1994; Johnson and McGeoch, 2002; Laha *et al.*, 2016; Ursani *et al.*, 2016; Babel 2020).

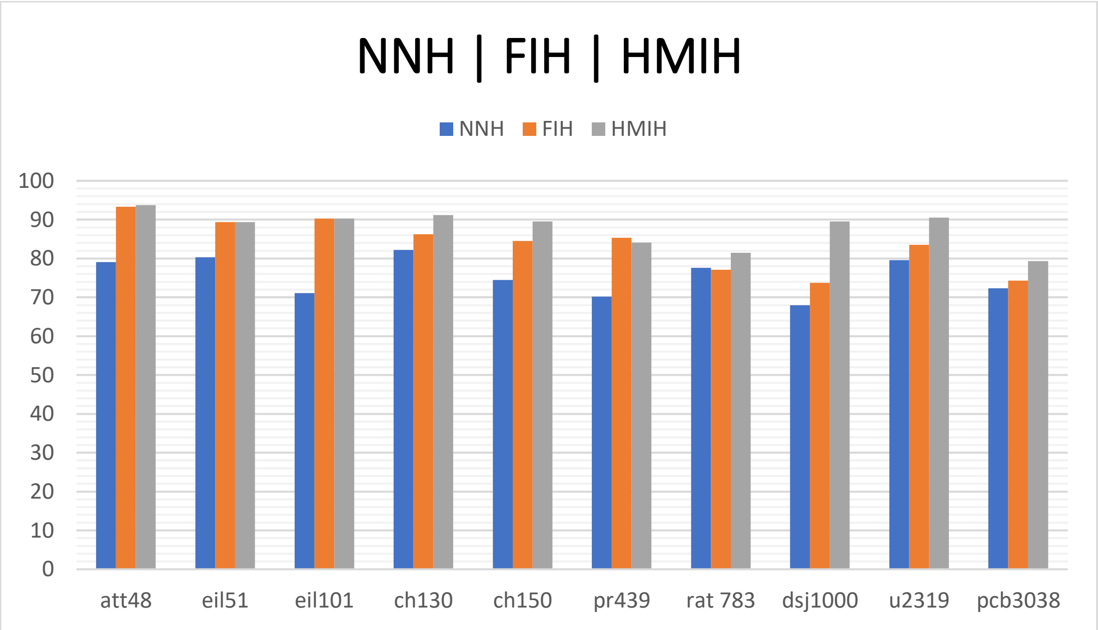

Figure 4.8. Measure of goodness value of HMIH, FIH and NNH

Additionally, while the FIH is faster, the computation speed of the proposed HMIH is within the same range, and since the HMIH searches were conducted $O(n)$ times, HMIH has the same complexity of $O(n^2)$ as the FIH and NNH. The computational speed performance of HMIH appears to follow a trend among lower order complexity heuristics where high performing method tends to take longer computation time, perhaps owing to more intricate process involved in getting better performance. With the exception of Random Insertion which requires no computation effort to add new nodes, the better the performance, the longer the time of computation tend to be (Reinelt, 1994; Johnson and McGeoch, 2002; Laha *et al.*, 2016; Ursani and Corne, 2016; Babel, 2020).

CHAPTER FIVE

5.0. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

5.1. Summary

In this work, the Travelling Salesman Problem was studied as a classic Combinatorial Optimization Problem. Combinatorial Optimization Problems deal with finding the best solutions that help optimise cost functions within the constraint of limited resources which may be time, space, energy and so forth. While there are numerous formulated Combinatorial Optimization Problems, such as Satisfiability Problems (SAT), Graph Colouring Problems (GCP), Cutting Stock Problem (CSP), Minimum Spanning Tree (MST), Constraint Satisfaction Problem (CSP), Bin Parking Problem (BPP) and so on, spanning the fields of Bioinformatics, Artificial Intelligence, Mathematics, Operations Research, Computer Science, the TSP is perhaps the most central to the field of combinatorics. Work on the TSP has been a driving force for the emergence and advancement of many important research areas, such as stochastic local search or integer programming, as well as for the development of complexity theory. Additionally, the TSP has also become a standard testbed for new algorithmic ideas; many of the most important techniques for solving combinatorial optimisation problems such as cutting plane techniques, branch and cut, simulated annealing, Ant colony, Branch-and-bound, and so on were developed using the TSP as an example application.

In solving the Travelling Salesman Problem, two popular tour construction heuristics were examined, namely the Nearest Neighbour Heuristic and the Farthest Insertion Heuristic. Obtaining high performing tour construction heuristics is a pressing research concern because they do not only generate good results, but they equally serve as seed for the development of other classes of heuristics and can be used to build initial solutions for high performing techniques. The NNH is fast, flexible, and simple to implement. It however solves the Travelling Salesman Problem using a greedy approach and suffers immensely from "*curse of dimensionality*". The FIH on the other hand is considered as the best performing insertion heuristic and best among lower order complexity heuristics. However, its performance is impeded by the distance between the partial circuit and the new node to be inserted. Thus, if inserting nearest nodes to the circuit leads to outliers and the performance of FIH is impeded by longer distance, perhaps a half max insertion may yield better solution. Thus, the NNH and the FIH were studied and a new insertion technique referred to as HMIH was formulated and experimented in order to generate better quality output, in reasonable time.

The three techniques (NNH, FIH and the derived HMIH) were implemented using Java Programming Language on ten TSPLIB benchmark instances. The experimental result generated showed that the speed of computation of the new method was poorer than that of NNH and FIH. However, this was compensated for with the solution quality.

5.2. Conclusion

In this study, a new Insertion heuristic was formulated and experimented on ten publicly available benchmark instances, alongside the NNH and FIH. The benchmark instances sizes were varied into three groups. Group one consisted two instances with less than a hundred (100) nodes, the second group had five instances whose nodes varied between one hundred and one (101) and nine hundred and ninety-nine (999), while the third group had three instances with one thousand (1000) nodes and above. The experimental results were displayed using tables and graph, and compared on the basis of parameters such as computational speed, percentage deviation from the optimal result, quality improvement and measure of goodness value. The results presented were able to address the research questions posed in the introductory section of the work. It was experimentally ascertained that the new improved heuristic obtained better solution qualities yet within the bracket of computational time as the FIH. Thus, it is safe to argue that it retained the same complexity of $O(n^2)$ as the FIH, and yet produces better solution quality.

The objectives of the study were achieved as the instances were simulated as a TSP problem first by converting the datasets to distance matrix and then implementing on varying sizes of benchmark instances.

5.3. Limitations

Based on the scope of this work, the implementation environment was limited to only the Object-Oriented Programming paradigm, as JAVA programming language was used to implement the heuristics. Additionally, no complexity curtailing technique was applied to the new formulated heuristic. Finally, the datasets were limited to ten instances.

5.4 Recommendations for Future Research

The following propositions are recommended to further this research:

i. The heuristics may be implemented using more than one programming paradigm. It will be interesting to simulate the behavior of the different programming paradigms on given instances using these techniques.

ii. Complexities curtailing techniques may be studied and applied to the proposed HMI technique to further improve its performance in terms of computational time.

Improving the computational time of the Half Max Insertion Heuristic is also a candidate for future research. Also, a future researcher may like to integrate this new heuristic with one or more of the existing state-of-the-art techniques with a view to examining the behavior of the resulting heuristic vis-à-vis each of the existing ones.

5.5 Contributions to Knowledge

Arising from the critical investigation of the NNH and FIH, the main contribution of this study to knowledge is the invention, implementation, and simulation of a new heuristic referred to in this study as Half Max Insertion Heuristic (HMIH). The HMIH overcomes the limitations of both the FIH and NNH; it performs better than both in terms of optimality. The study has therefore provided us with a new and superior computational method for solving NP-Hard problems.

REFERENCES

Abdel-Basset M., Abdel-Fatah L. & Sangaiah, A.K., (2018). Metaheuristic Algorithms: A Comprehensive Review. In A.K. Sangaiah, M. Sheng & Z. Zhang (Eds.), *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications* 185–231. Academic Press. https://doi.org/10.1016/B978-0-12-813314-9.00010-4.

Abdi S., Pourkarimi L., Ahmadi M. & Zargari F., (2017). Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds. *Future Generation Computer Systems*, *71*, 113-128.

Abdulkarim H.A. & Alshammari I.F., (2015). Comparison of Algorithms for Solving the Travelling Salesman Problem. *International Journal of Engineering and Advanced Technology 4(6)*, 76-78.

Abdul-Rahman S., Benjamin A.M., Omar M.F., Ramli R., Ku-Mahamud K.-R., & Abduljabbar W.K., (2017). Designing and implementation a web-based architecture for an examination timetabling system. *Journal of Engineering and Applied Sciences 12(23)*, 7299–7305.

Abeledo H., Fukasawa R., Pessoa A. & Uchoa E. (2010). The Time Dependent Travelling Salesman Problem: Polyhedra and Branch-Cut-and-Price Algorithm. In: Festa P. (Eds.), *Experimental Algorithms. SEA 2010. Lecture Notes in Computer Science*, *6049*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-13193-6_18

Abid M.M. & Muhammad I., (2015). Heuristic Approaches to Solve the Travelling Salesman Problem. *TELKOMNIKA Indonesian Journal of Electrical Engineering 5(2),* 390-396.

Aguayo M.M., Sarin S.C. & Sherali H.D., (2016). Solving the single and multiple asymmetric Travelling Salesmen Problems by generating subtour elimination constraints from integer solutions. *IISE Transactions 5(1),* 45-53. https://doi.org/10.1080/24725854.2017.1374580.

Ahmed Z.H., (2011). A Data-Guided Lexisearch Algorithm for the Asymmetric Travelling Salesman Problem. *Mathematical Problems in Engineering 2011*, 1-18. https://doi.org/10.1155/2011/750968.

Ahmed Z.H., (2011). A Data-Guided Lexisearch Algorithm for the Bottleneck Travelling Salesman Problem. *International. Journal of Operational Research 12(1),* pp 20-33.

Ahmed Z.H., (2013). A hybrid genetic algorithm for the bottleneck travelling salesman problem. *ACM Trannsaction on Embedded Computing Systems 12(1),* 1-10.

Ahmed Z.H., (2014). The Ordered Clustered Travelling Salesman Problem: A Hybrid Genetic Algorithm. *The Scientific World Journal 2014,* 1-14. https://doi.org/10.1155/2014/258207.

Ajaz A.K. & Himani A (2016). Determining the Shortest Path for Travelling Salesman Problem using Nearest Neighbor Algorithm. *International Journal for Scientific Research & Development 3(12),* 856-859.

Alamdari S., Fata E. & Smith S.L., (2013). Min-Max Latency Walks: Approximation Algorithms for Monitoring Vertex-Weighted Graphs. In: E. Frazzoli, T. Lozano-

Perez, N. Roy & D. Rus (Eds.), *Algorithmic Foundations of Robotics X. Springer Tracts in Advanced Robotics, 86*. 139-155. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-36279-8_9.

Ali Z.A. (2016). Concentric Tabu Search Algorithm for Solving Travelling Salesman Problem (Eastern Mediterranean University January-North Cyprus). *Master of Science in Computer Engineering Thesis.* Retrieved February 13, 2020, from http://i-rep.emu.edu.tr:8080/xmlui/handle/11129/2933.

Allaoua H., (2017). Combination of Genetic Algorithm with Dynamic Programming for Solving TSP. *International Journal of Advances in Soft Computing and its Application 9(2)*, 31-44.

AlSalibi B.A., Jelodar M.B. & Venkat I., (2013). A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Travelling Salesman Problem. *International Journal of Computer Science and Electronics Engineering 1(1)*, 34-38.

An H.-C., Kleinberg R. & Shmoys D.B., (2012). Improving Christofides' algorithm for the s-t path TSP. *Proceeding of the 44th Annual ACM Symposium on Theory of Computing (STOC'12)*, 875--886.

Anbuudayasankar S.P., Ganesh K. & Mohapatra S. (2014). Survey of Methodologies for TSP and VRP. In: *Models for Practical Routing Problems in Logistics*, 11-42. Springer, Cham. https://doi.org/10.1007/978-3-319-05035-5_2.

Applegate D., Bixby R., Chvátal V. & Cook W., (2001). TSP Cuts Which Do Not Conform to the Template Paradigm. In: M. Jünger & D. Naddef (Eds.), *Computational Combinatorial Optimization. Lecture Notes in Computer Science*

2241, 261-303. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45586-8_7.

Arash Asadpour, Michel X. Goemans, Aleksander Madry, Shayan Oveis Gharan, & Amin Saberi., (2010). An O(log n/ log log n)-approximation Algorithm for the Asymmetric Travelling Salesman Problem. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, 379–389.

Arigliano A., Ghiani G., Grieco A., Guerriero E. & Plana I (2018) Time-dependent asymmetric Travelling salesman problem with time windows: properties and an exact algorithm. *Discrete Applied Mathematics 261,* 28-39. https://doi.org/10.1016/j.dam.2018.09.017.

Arthanari T. & Qian K. (2018) Symmetric Travelling Salesman Problem. In: S. Neogy, R. Bapat & Dubey D. (Eds.), *Mathematical Programming and Game Theory*. Indian Statistical Institute Series. Springer, Singapore

Arthanari, T.S., (1983). On the Travelling salesman problem. *Mathematical Programming - The State of the Art.* Springer, Berlin

Arthanari, T.S. & Usha, M., (2000). An alternate formulation of the symmetric Travelling salesman problem and its properties. *Discrete Applied Mathematics 98(3),* 173–190.

Ascheuer N., Junger M. & Reinelt G., (1999). A Branch & Cut Algorithm For The Asymmetric Travelling Salesman Problem With Precedence Constraints. *Computational Optimization and Applications 17,* 61–84. https://doi.org/10.1023/A:1008779125567

Assaf M. and Ndiaye M., (2017). A transformation for multiple depot multiple Travelling salesman problem. *Proceedings of International Conference on Engineering & MIS (ICEMIS), Monastir, 2017,* 1-5. https://doi.org/10.1109/ICEMIS.2017.8273004.

Assaf M. and Ndiaye M., (2017). Multi travelling salesman problem formulation. *Proceedings of the 4th International Conference on Industrial Engineering and Applications (ICIEA), Nagoya, 2017,* 292-295. https://doi.org/10.1109/IEA.2017.7939224.

Ayorkor, M.G., Stentz, A., & Bernardine, D.M. (2007). The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, CMU-RI-TR-07-27. Retrieved April 4, 2019 from http://www.cs.cmu.edu/~gertrude/dyn_assign_techreport.pdf.

Babel L., (2020). New heuristic algorithms for the Dubins Travelling salesman problem. *Journal of Heuristics 26*, 503-530. https://doi.org/10.1007/s10732-020-09440-2.

Baidoo E. & Oppong S.O., (2016). Solving the TSP using Traditional Computing Approach. *International Journal of Computer Applications 152(8)*, pp 13-19.

Balbal S., Laalaoui Y. & Benyettou M., (2015). Local search heuristic for Multiple Knapsack Problem. *International Journal of Intelligent Information Systems 4(3),* pp 5–9.

Balseiro S.R., Loiseau I. & Ramonet J. (2011). An Ant Colony algorithm hybridized with insertion heuristics for the Time Dependent Vehicle Routing Problem with

Time Windows. *Computers & Operations Research 38*, 954–966. https://doi.org/10.1016/j.cor.2010.10.011.

Bansal J.C. & Deep K., (2012). A Modified Binary Particle Swarm Optimization for Knapsack Problems. *Applied Mathematics and Computation 218(22)* 11042-11061. https://doi.org/10.1016/j.amc.2012.05.001

Baranwal M., Parekh P., Marla L., Salapaka S.M. & Beck C., (2016). Vehicle routing problem with time windows: A deterministic annealing approach. *Proceedings of the American Control Conference (ACC),* 790-795.

Baranwal M., Roehl B. & Salapaka S.M., (2017). Multiple Travelling salesmen and related problems: A maximum-entropy principle based approach. *Proceedings of American Control Conference (ACC), Seattle, WA, 2017,* 3944-3949, doi: 10.23919/ACC.2017.7963559.

Barketau M. & Pesch E., (2016). An approximation algorithm for a special case of the asymmetric travelling salesman problem. *International Journal of Production Research 54(14)*, 4205-4212. http://doi.org/10.1080/00207543.2015.1113327.

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M & Vance P (1998) Branch-and-price: Column generation for solving huge integer programs. *Operations Research 46*, 316–329.

Barto L., (2015). The Constraint Satisfaction Problem and Universal Algebra. *The Bulletin of Symbolic Logic 21(3),* 319-337.

Barvinok A., Gimadi E.K. & Serdyukov A.I., (2007). The maximum TSP. In: G. Gutin & A.P. Punnen (Eds.), The Travelling Salesman Problem and Its Variations.

Combinatorial Optimization 12, 585-607. Springer, Boston, MA. https://doi.org/10.1007/0-306-48213-4_12.

Basu S, Sharma M. & Sarathi P.G. (2017) Efficient preprocessing methods for tabu search: an application on asymmetric travelling salesman problem, *Information Systems and Operational Research 55(2),* 134-158. DOI: 10.1080/03155986.2017.1279897

Basu S., (2012). Tabu Search Implementation on Travelling Salesman Problem and Its Variations: A Literature Survey. *American Journal of Operations Research 2,* 163-173. DOI:10.4236/ajor.2012.22019.

Battarra M., Pessoa A.A, Subramanian A. & Uchoa E., (2014). Exact algorithms for the Travelling Salesman Problem with Draft Limits. European *Journal of Operational Research 235 (1),* 115-128. https://doi.org/10.1016/j.ejor.2013.10.042

Bazylevych R., Kutelmakh R., Dupas R. & Bazylevych L., (2007). Decomposition Algorithms for Large-scale Clustered TSP (2007). *Proceedings of the third Indian International Conference on Artificial Intelligence (IICAI-07),* 256 – 267.

Becker H. & Buriol L.S., (2019). An empirical analysis of exact algorithms for the unbounded knapsack problem. European journal of operational research Vol. 277(16), pp. 84-99 https://doi.org/10.1016/j.ejor.2019.02.011

Bednarczuk, E.M., Miroforidis, J. & Pyzel, P., (2018). A multi-criteria approach to approximate solution of multiple-choice knapsack problem. *Computational Optimization and Applications 70,* 889–910. https://doi.org/10.1007/s10589-018-9988-z

Bellman, R.E., (1962). Dynamic programming treatment of the Travelling salesman problem. *Journal of the ACM 9(1),* 61–63.

Bentley J.L. (1992). Fast Algorithms for Geometric Travelling Salesman Problems. *ORSA Journal of Computing 4(4),* 387-411.

Bernardino R. & Paias A., (2018). Solving the family Travelling salesman problem. *European Journal of Operational Research 267,* 453–466. https://doi.org/10.1016/j.ejor.2017.11.063.

Bouazzi K., Hammami M. & Bouamama S., (2019). Hybrid Genetic Algorithm for CSOP to Find the Lowest Hamiltonian Circuit in a Superimposed Graph. In: L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz & J. Zurada (Eds.), *Artificial Intelligence and Soft Computing. ICAISC 2019. Lecture Notes in Computer Science 11509,* 512-525. Springer, Cham. https://doi.org/10.1007/978-3-030-20915-5_46

Bouman P., Agatz N. & Schmidt M., (2018) Dynamic programming approaches for the Travelling salesman problem with a drone. *Networks 72*, 528–542. https://doi.org/10.1002/net.21864

BUI Q.-T., (2015). Modelling and solving complex combinatorial optimization problems: quorumcast routing, elementary shortest path, elementary longest path and agricultural land allocation. Ph.D Thesis in Information Technology, Université catholique de Louvain. Retrieved February 12, 2020 from https://www.info.ucl.ac.be/~yde/Papers/thesis_Trung2015.pdf.

Burke E.K., Kendall G., Misir M. & Ozcan E., (2012). Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research 196*, 73–90, 2012.

Campuzano G., Obreque C. & Aguayo M.M., (2020). Accelerating the Miller–Tucker–Zemlin model for the asymmetric Travelling salesman problem. *Expert Systems with Applications 148,* 113-229.

Chatting M., (2018). A Comparison of Exact and Heuristic Algorithms to Solve the Travelling Salesman Problem. The Plymouth Student Scientist, vol. 11(2), pp. 53-91.

Chauhan C., Gupta R. & Pathak K., (2012). Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach. *International Journal of Combinatorial Optimization: Computer Applications 52(4),* 12-19.

Chen G.H. & Shah D., (2018). Explaining the Success of Nearest Neighbor Methods in Prediction. *Foundations and Trends R in Machine Learning 10(5),* 337–588. DOI: 10.1561/2200000064.

Christian, B., & Cremaschi, S. (2018). Planning pharmaceutical clinical trials under outcome uncertainty. *Computer-Aided Chemical Engineering 41,* 517–550. doi:10.1016/b978-0-444-63963-9.00021-x

Christiansen M., Fagerholt K., Nygreen B. & Ronen D., (2013). Ship routing and scheduling in the new millennium. *European Journal of Operational Research 228(3),* 467-483.

Cruz R. C., Silva T. C. B., Souza M. J. F., Coelho V. N., Mine M. T. & Martins A. X. (2012). GENVNS-TS-CL-PR: A heuristic approach for solving the vehicle routing problem with simultaneous pickup and delivery. *Electronic Notes in Discrete Mathematics 39,* 217–224. https://doi.org/10.1016/j.endm.2012.10.029.

Cuevas A.M.C., Martínez J.A.S. & Saucedo J.A.M., (2020) A Two Stage Method for the Multiple Travelling Salesman Problem. *International Journal of Applied Metaheuristic Computing 11(3),* 79-91.

Daamen R. & Phillipson F., (2015). Comparison of heuristic methods for the design of edge disjoint circuits. *Computer Communications 61*, 90–102

Damghanijazi E. & Mazidi A., (2017). Meta-Heuristic Approaches for Solving the Travelling Salesman Problem. *International Journal of Advanced Research in Computer Science 8(5),* 18-23.

Dantzig, G.B., Fulkerson, D.R. & Johnson, S.M., (1954). Solution of a large-scale Travelling-saesman problem. *Operations Research 2(4),* 393–410

Date K. & Nagi R., (2016). GPU-accelerated Hungarian algorithms for the Linear Assignment Problem. *Parallel Computing 57,* 52-72. https://doi.org/10.1016/j.parco.2016.05.012

Demez H., (2013). Solution Methods of Travelling Salesman Problem. Masters of Science Thesis, Eastern Mediterranean University. Retrieved July 4, 2020 from http://i-rep.emu.edu.tr:8080/xmlui/bitstream/handle/11129/654/Demez.pdf?sequence=1.

Desrochers M. & Laporte G. (1990) Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints. *Operations Research Letters 10(1),* 27–36

Deudon M., Cournut P., Lacoste A., Adulyasak Y. & Rousseau LM. (2018) Learning Heuristics for the TSP by Policy Gradient. In: W.J. Van Hoeve (Eds.), *Integration* *of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2018. Lecture Notes in Computer Science 10848.* Springer, Cham

Dijck E.V., 2018. A Branch-and-Cut Algorithm for the Travelling Salesman Problem with Drone. Master Thesis Operations Research & Quantitative Logistics. Erasmus University Rotterdam. Retrieves July 4, 2020 from https://thesis.eur.nl/pub/44107/Dijck-van.pdf.

Dong W., Dong X. & Wang Y., (2017). The Improved Genetic Algorithm for Multiple Maximum Scatter Tevelling Salesperson Problems. In: Li J. *et al.* (Eds.), *Wireless Sensor Networks. CWSN 2017. Communications in Computer and Information Science 812*, 155-164. Springer, Singapore. https://doi.org/10.1007/978-981-10-8123-1_14

Dorigo M., Di Caro G. & Gambardella L.M., (1999). Ant algorithms for discrete optimization. *Artificial Life 5(2),* 137–172.

Dowlatshahi M.B., Nezamabadi-Pour H. & Mashinchi M., (2014). A discrete gravitational search algorithm for solving combinatorial optimization problems. *Information Sciences 258,* 94-107

Droste I., (2017). Algorithms for the Travelling salesman problem. Bachelor Thesis in Mathematics, Faculteit B_etawetenschappen, University of Utrecht. Retrieved April 15, 2020 from https://dspace.library.uu.nl/bitstream/handle/1874/366424/BachelorthesisIsabelDroste.pdf?sequence=2.

Dudycz S., Marcinkowski J., Paluch K. & Rybicki B. (2017) A 4/5 - Approximation Algorithm for the Maximum Travelling Salesman Problem. In: F. Eisenbrand. & J. Koenemann (Eds.), *Integer Programming and Combinatorial Optimization. IPCO*

2017. Lecture Notes in Computer Science, 10328. Springer, Cham. https://doi.org/10.1007/978-3-319-59250-3_15

Dumitrescu I., Ropke S., Cordeau J. & Laporte G., (2010). The Travelling Salesman Problem with Pickup and Delivery: Polyhedral Results and a Branch-and-Cut Algorithm. *Mathematical Programming 121(2),* 269-305. https://doi.org/10.1007/s10107-008-0234-9

Ejim S., (2016). Implementation of Greedy Algorithm in Travel Salesman Problem. Final Year Mini - Project for the Course: Design and Analysis of Algorithm, pp 1-8. Retrieved May 12, 2020 from https://www.researchgate.net/publication/307856959_Implementation_of_Greedy_Algorithm_in_Travel_Salesman_Problem?channel=doi&linkId=57cf068508ae582e06938947&showFulltext=true. 10.13140/RG.2.2.23921.48485.

Englert, M., Röglin, H. & Vöcking, B. Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP. *Algorithmica 68,* 190–264. https://doi.org/10.1007/s00453-013-9801-4

Erdogan G., Battarra M., Laporte G. & Vigo D., (2012). Metaheuristics for the Travelling salesman problem with pickups, deliveries, and handling costs. *Computer and Operations Research 39*, 1074 -1086.

Fachini R.F. & Armentano V.A., (2018). Exact and heuristic dynamic programming algorithms for the Travelling salesman problem with flexible time windows. *Optimization Letters 14*, 579–609. https://doi.org/10.1007/s11590-018-1342-y

Fan J., (2011). The Vehicle Routing Problem with Simultaneous Pickup and Delivery Based on Customer Satisfaction. *Procedia Engineering 15,* 5284 – 5289. https://doi.org/10.1016/j.proeng.2011.08.979.

Faudzi S., Abdul-Rahman S. & Abd Rahman R., (2018). An Assignment Problem and Its Application in Education Domain: A Review and Potential Path. *Advances in Operations Research 2018* 1-19. https://doi.org/10.1155/2018/8958393

Feillet D., Gendreau M., Medaglia A.L. & Walteros J.L., (2010). A note on branch-and-cut-and-price. *Operations Research Letters 38(5),* 346-353. https://doi.org/10.1016/j.orl.2010.06.002.

Fischer A., Fischer F., Jäger G., Keilwagend J., Molitor P. & Grosse I., (2014). Exact algorithms and heuristics for the Quadratic Travelling Salesman Problem with an application in bioinformatics. *Discrete Applied Mathematics 166, 97-114.* http://dx.doi.org/10.1016/j.dam.2013.09.011.

Fomeni F.D., Kaparis K. & Letchford A.N. (2020). A cut-and-branch algorithm for the Quadratic Knapsack Problem. *Discrete Optimization. In Press.* https://doi.org/10.1016/j.disopt.2020.100579

Fontaine P., Taube F. & Minner S., (2020). Human solution strategies for the vehicle routing problem: Experimental findings and a choice-based theory. *Computers and Operations Research 120 2020*, 1-16. https://doi.org/10.1016/j.cor.2020.104962.

Fosin J., Davidović D. & Carić T., (2013). A GPU Implementation of Local Search Operators for Symmetric Travelling Salesman Problem. *PROMET 25(3),* 225-234. https://doi.org/10.7307/ptt.v25i3.300

Frenkel E., Nikolaev A. & Ushakov A. (2016). Knapsack problems in products of groups. *Journal of Symbolic Computation, 74,* 96-108. https://doi.org/10.1016/j.jsc.2015.05.006

Fukunaga A., (2011). A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research 184*, 97–119.

Gabrel V., Manouvrier M. & Murat C. (2014) Optimal and Automatic Transactional Web Service Composition with Dependency Graph and 0-1 Linear Programming. In: X. Franch, A.K. Ghose, G.A. Lewis & S. Bhiri (Eds.), *Service-Oriented Computing. ICSOC 2014. Lecture Notes in Computer Science, Vol. 8831. Springer, Berlin, Heidelberg*. https://doi.org/10.1007/978-3-662-45391-9_8

Gahir D., (2014). A fully Polynomial Time Approximation Scheme for Weight Constrained BTSP with Two Linear Constraints on Halin Graph. *International Journal of Science and Research 3(6)*, 315-317.

Gavish B. & Graves S (1978) The travelling salesman problem and related problems. Working Paper GR078-78. Operations Research Center, Massachusetts Institute of Technology, Cambridge. Retrieved April 25, 2020. http://hdl.handle.net/1721.1/5363

Gendreau M., Jabali O. & Rei W., (2014). Stochastic Vehicle Routing Problems. In: P. Toth & D. Vigo, (Eds.), *Vehicle routing: problems, methods, and applications, SIAM, New York, USA (2014).* 213-240. https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594.fm

Genova K. & Williamson D.P., (2017). An Experimental Evaluation of the Best-of-Many Christofides’ Algorithm for the Travelling Salesman Problem. *Algorithmica 78*, 1109–1130. https://doi.org/10.1007/s00453-017-0293-5

Ginting H.N., Osmond A.B. & Aditsania A., (2019). Item Delivery Simulation Using Dijkstra Algorithm for Solving the Travelling Salesman Problem. *International*

Conference on Electronics Representation and Algorithm (ICERA 2019), J. Phys.: Conf. Ser. 1201 012068, 1-9. https://doi.org/10.1088/1742-6596/1201/1/012068

Giovanni L. De (2017). Methods and Models for Combinatorial Optimization: Heuristics for Combinatorial Optimization. Retrieved April 25, 2019 from https://www.math.unipd.it/~luigi/courses/metmodoc1718/m02.meta.en.partial01.pdf

Godinho M.T., Gouveia L. & Pesneau P., (2014). Natural and Extended formulations for the Time-Dependent Travelling Salesman Problem. *Discrete Applied Mathematics, 164,* 138-153. https://doi.org/ff10.1016/j.dam.2011.11.019ff. ffhal-00648451f.

Goetschalckx M. (2011) Single Vehicle Round-trip Routing. In: Supply Chain Engineering. *International Series in Operations Research & Management Science,* 161. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-6512-7_8

Gorenstein S., (1970). Printing press scheduling for multi-edition periodicals. *Management Science* 16 (6), 373-383.

Gouveia L. & Pires J. (1999) The asymmetric travelling salesman problem and a reformulation of the Miller–Tucker–Zemlin constraints. *European Journal of Operations Research 112,* 134–146

Gupta D., (2013). Solving tsp using various meta-heuristic algorithms. *International Journal of Recent Contributions from Engineering, Science & IT 1(2)*, 19–26.

Gupta S. & Kakkar M., (2012). Techniques For Solving the Travelling Sales Man Problem. *International Journal Of Engineering Science & Advanced Technology 2(5),* 1357 – 1360.

Gupta S., Batra D. & Verma P., (2014). Greedy Estimation of Distributed Algorithm to Solve Bounded knapsack Problem. *International Journal of Computer Science and Information Technologies 5(3),* 4313-4316.

Hassin, R., & Rubinstein, S. (2000). Better approximations for max TSP. *Information Processing Letters 75(4),* 181-186.

Hassin, R., Levin, A. & Rubinstein, S., (2009). Approximation algorithms for maximum latency and partial cycle cover. *Discrete Optimization 6(2),* 197–205.

Hazra T.K. & Hore A., (2016). A comparative study of Travelling Salesman Problem and solution using different algorithm design techniques. *Proceedings of the IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, 2016,* 1-7, doi: 10.1109/IEMCON.2016.7746316.

Held, M. & Karp, R.M., (1970). The travelling salesman problem and minimum spanning trees. *Operations Research 18(6),* 1138–1162

Helsgaun K., (2009). General k-opt sub moves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation* 1, 119–163. https://doi.org/10.1007/s12532-009-0004-6

Helsgaun K., (2014). Solving The Bottleneck Travelling Salesman Problem Using The Lin-Kernighan-Helsgaun Algorithm. Technical Report, Roskilde University, Roskilde, Denmark, 1-45. Retrieved May 2, 2019 from https://www.researchgate.net/publication/266382000_Solving_the_Bottleneck_Travelling_Salesman_Problem_Using_the_Lin-Kernighan-Helsgaun_Algorithm

Hifi M., (2014). An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization 46(8),* 1109-1122, DOI: 10.1080/0305215X.2013.819096

Hoffman I., (2016). The Maximum Scatter TSP on a Regular Grid – How to Avoid Heat Peaks in Additive Manufacturing. Doctoral Thesis, University of Beyreuth. Retrieved September 30, 2019 from https://epub.uni-bayreuth.de/3182/1/Stock_Isabella_The_Maximum_Scatter_TSP.pdf.

Hoffmann I., Kurz S. & Rambau J. (2017). The Maximum Scatter TSP on a Regular Grid. In: K. Dörner, I. Ljubic, G. Pflug & G. Tragler (Eds,), *Operations Research Proceedings 2015. Operations Research Proceedings (GOR (Gesellschaft für Operations Research e.V.)).* Springer, Cham. https://doi.org/10.1007/978-3-319-42902-1_9

Huang W. & Yu J.X. (2017). Investigating TSP Heuristics for Location-Based Services. *Data Science and Engineering 2*, 71–93. DOI 10.1007/s41019-016-0030-0.

Huang W., Yu J.X. & Shang Z. (2016) A Sketch-First Approach for Finding TSP. In: M. Cheema, W. Zhang & L. Chang (Eds.), *Databases Theory and Applications. ADC 2016. Lecture Notes in Computer Science* 9877. Springer, Cham. https://doi.org/10.1007/978-3-319-46922-5_10

Hussain A., Muhammad Y.S., Sajid M.N., Hussain I., Shoukry A.M. & Gani S., (2017). Genetic Algorithm for Travelling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience* 2017, 1-7. https://doi.org/10.1155/2017/7430125

Hyung-Chan An, Robert D. Kleinberg, & David B. Shmoys, (2010). Approximation Algorithms for the Bottleneck Asymmetric Travelling Salesman Problem. In: M.

Serna, R. Shaltiel, K. Jansen, & J. Rolim (Eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. RANDOM 2010, APPROX 2010. Lecture Notes in Computer Science,* 6302. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15369-3_1

Jain E., Jain A. & Mankad S.H., (2014). A new approach to address Subset Sum problem. Proceedings of the fifth International Conference - Confluence The Next Generation Information Technology Summit (Confluence), Noida, 2014, 953-956, doi: 10.1109/CONFLUENCE.2014.6949229.

Jain V. & Prasad J.S., (2017). Solving Travelling Salesman Problem Using Greedy Genetic Algorithm GGA. *International Journal of Engineering and Technology 9(2)* 1148-1154. DOI: 10.21817/ijet/2017/v9i2/170902188

Jamal Ouenniche, Prasanna K. Ramaswamy & Michel Gendreau (2017). A dual local search framework for combinatorial optimization problems with TSP application. Journal of the Operational Research Society 68, 1377–1398 (2017). https://doi.org/10.1057/s41274-016-0173-4.

Jawaid S.T. & Smith S.L., (2013). The maximum Travelling salesman problem with submodular rewards. In proceedings of the 2013 American Control Conference. 3997-4002, doi: 10.1109/ACC.2013.6580451.

Jawaid S.T. & Smith S.L., (2015). Informative path planning as a maximum Travelling salesman problem with submodular rewards. *Discrete Applied Mathematics 186,* 112-127. https://doi.org/10.1016/j.dam.2015.01.004

Jepsen, M. K. (2011). Branch-and-cut and Branch-and-Cut-and-Price Algorithms for Solving Vehicle Routing Problems. Phd Thesis, The Technical University of

Denmark. Retrieved March 13 2020 from url: https://backend.orbit.dtu.dk/ws/portalfiles/portal/6317942/jepsen08072011.pdf

Jeřábek K., Majercak P., Kliestik T. & Valaskova K., (2016). Application of Clark and Wright´s Savings Algorithm Model to Solve Routing Problem. *Supply Logistics. Preliminary communication, 63(3),* 115-119. DOI 10.17818/NM/2016/SI7

Johnson, D.S. and McGeoch, L.A., (2002). Experimental analysis of heuristics for the STSP. In: G. Gutin & A.P. Punnen (Eds.), *The Travelling Salesman Problem and Its Variants,* 369–443, Kluwer, Dordrecht (2002).

Kabadi S.N. & Punnen A.P., (2007). The Bottleneck TSP. In: G. Gutin, A.P. Punnen (Eds.), *The Travelling Salesman Problem and Its Variations. Combinatorial Optimization*, 12, 697-735. Springer, Boston, MA, https://doi.org/10.1007/0-306-48213-4_15

Kadri R.L. & Boctor F.F., (2018). An efficient genetic algorithm to solve the resource-constrained project scheduling with transfer times. *European Journal of Operational Research 265(2),* 454-462. https://doi.org/10.1016/j.ejor.2017.07.027.

Kahar M.N.M. & Kendall G., (2010). The examination timetabling problem at Universiti Malaysia Pahang: comparison of a constructive heuristic with an existing sofware solution. *European Journal of Operational Research 207(2)*, 557–565.

Kampf R., Stopka O., Bartuska L. & Zeman K., (2015). Circulation of vehicles as an important parameter of public transport efficiency. Proceedings of the 19th International Scientific Conference on Transport Means. Kaunas (Lithuania): Kaunas University of Technology, 2015, 143-146.

Kao M.-Y. & Sanghi M., (2009). An approximation algorithm for a bottleneck Travelling salesman problem. *Journal of Discrete Algorithms 7(3)*, 315-326. https://doi.org/10.1016/j.jda.2008.11.007

Kellerer H., Pferschy U. & Pisinger D. (2004). Introduction. In: H. Kellerer, U. Pferschy & D. Pisinger (Eds.), *Knapsack Problems. Springer, Berlin, Heidelberg,* 1-14. https://doi.org/10.1007/978-3-540-24777-7_1

Keskin M., Laporte G. & Çataya B., (2019). Electric Vehicle Routing Problem with Time-Dependent Waiting Times at Recharging Stations. *Computers & Operations Research 107,* 77-94. https://doi.org/10.1016/j.cor.2019.02.014

Kitjacharoenchaia P., Mario Ventrescaa, Mohammad Moshref-Javadib, Seokcheon Leea, Jose M.A. Tanchocoa, & Patrick A. Brunesea (2019). Multiple Travelling salesman problem with drones: Mathematical model and heuristic approach. *Computers & Industrial Engineering 129,* 14–30. https://doi.org/10.1016/j.cie.2019.01.020

Kızılateş G., Nuriyeva F. & Kutucu H., (2015). A tour extending hyper-heuristic algorithm for the Travelling salesman problem. *Proceedings of the IAM 4(1),* 8-15.

Kovács L., Iantovics L.B. & Iakovidis D.K. (2018). IntraClusTSP—An Incremental Intra-Cluster Refinement Heuristic Algorithm for Symmetric Travelling Salesman Problem. *Symmetry. 10(12)*, 1-31 https://doi.org/10.3390/sym10120663

Kowalik, L. & Mucha, M., (2007). 35/44-approximation for asymmetric maximum TSP with triangle inequality. In: F. Dehne, J.-R. Sack & N. Zeh, (Eds.), *WADS 2007. LNCS, 4619,* 589–600. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73951-7 51

Kowalik, L. & Mucha, M., (2008). Deterministic 7/8-approximation for the metric maximum TSP. In: A. Goel, K. Jansen, J.D.P. Rolim & R. Rubinfeld, (Eds.), *APPROX/RANDOM -2008. LNCS, 5171,* 132–145. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85363-3 11

Kozanidis G., (2018) Branch and price for covering shipments in a logistic distribution network with a fleet of aircraft. *Optimization Methods and Software 33(2),* 221-248. DOI: 10.1080/10556788.2017.1281923

Kozma L. & Momke T., (2016). A PTAS for Euclidean Maximum Scatter TSP. Proceedings of the thirty second European Workshop on Computational Geometry, 2016.

Kozma L. & Mömke T., (2017). Maximum scatter TSP in doubling metrics. Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms January 2017, 143–153.

Kyritsis M., Gulliver S. R., Feredoes E. & Ud Din S. (2018). Human behaviour in the Euclidean Travelling Salesperson Problem: Computational modelling of heuristics and figural effects. *Cognitive Systems Research 52,* 387-399.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B. (2019). An improved sexual genetic algorithm for solving 0/1 multidimensional knapsack problem. *Engineering Computations 36(7),* 2260-2292. https://doi.org/10.1108/EC-01-2019-0021

Labadie, N., Melechovsky, J., & Prins, C. (2014). An evolutionary algorithm with path relinking for a bi-objective multiple Travelling salesman problem with profits. In Applications of Multi-Criteria and Game Theory Approaches, pp. 195–223. London: Springer.

Laha D. & Gupta J.N.D., (2016). A Hungarian penalty-based construction algorithm to minimize makespan and total flow time in no-wait flow shops. *Computers & Industrial Engineering 98,* 373–383

LaRusic J., (2010). The bottleneck Travelling salesman problem and some variations M.Sc. Thesis, Department of Mathematics, Simon Fraser University. Rtrieved March 13, 2020 from https://summit.sfu.ca/item/9936.

Lau S.K. (2002). Solving Travelling salesman problem with a heuristic learning approach, Doctor of Philosophy thesis, Department of Information Systems, University of Wollongong. Retrieved January 22, 2020 from http://ro.uow.edu.au/theses/1455.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. & Shmoys, D.B. (1985). The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, New York.

Leão A.A.S., Cherri L.H. & Arenales M.N., (2014). Determining the K-best solutions of knapsack problems. *Computers & Operations Research 49,* 71-82. https://doi.org/10.1016/j.cor.2014.03.008

Lesca, J., Minoux, M. & Perny, P., (2019). The Fair OWA One-to-One Assignment Problem: NP-Hardness and Polynomial Time Special Cases. *Algorithmica 81,* 98–123. https://doi.org/10.1007/s00453-018-0434-5

Lim Y.-F., Hong P.-Y., Ramli R., Khalid R. & Baten Md. A., (2016). Performance Evaluation of Heuristic Methods in Solving Symmetric Travelling Salesman Problems. *Journal of Artificial Intelligence 9,* 12-22.

Lingling Du & Ruhan Heb (2012). Combining Nearest Neighbor Search with Tabu Search for Large-Scale Vehicle Routing Problem. *Physics Procedia 25,* 1536 – 1546. doi: 10.1016/j.phpro.2012.03.273

Lity S., Al-Hajjaji M., Thüm T. & Schaefer I, (2017). Optimizing product orders using graph algorithms for improving incremental product-line analysis. VAMOS '17: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, (2017), pp 60–67. https://doi.org/10.1145/3023956.3023961.

Liu, M., & Zhang, P (2014). New hybrid genetic algorithm for solving the multiple Travelling salesman problem: An example of distribution of emergence materials. *Journal of Systems & Management 23(2),* 247–254.

Lorterapong P. & Ussavadilokrit M., (2013). Construction Scheduling Using the Constraint Satisfaction Problem Method. *Journal of Construction Engineering and Management 139(4),* 414-422.

Mackworth A.K. (1977). Consistency in Network of Relations. *Artificial Intelligence 8,* 99-118,

Malawski M., Figiela K. & Nabrzyski J., (2013). Cost minimization for computational applications on hybrid cloud infrastructures. *Future Generation Computer Systems 29(7),* 1786-1794.

Mário Mestria (2018). New hybrid heuristic algorithm for the clustered Travelling salesman problem. *Computers & Industrial Engineering 116*__1–12. https://doi.org/10.1016/j.cie.2017.12.018.

Martello S. & Monaci M. (2020). Algorithmic approaches to the multiple knapsack assignment problem. *Omega 90*, 1-11 https://doi.org/10.1016/j.omega.2018.11.013

Martello S. & Toth P. (1990). KNAPSACK PROBLEMS Algorithms and Computer Implementations. John Wiley & Sons Ltd. West Sussex, England. ISBN: 0471 924202.

Marti R & Reinelt G. (2011). Heuristic Methods. The linear Ordering Problem Exact and Heuristic Methods in Combinatorial Optimisation. Springer, -Verlag Berlin Heidelberg. ISBN: 978-3-64-16728-7. 17-40. DOI: 10.1007/978-3-642-16729-4 2

Matai R., Singh S.P. & Mittal M.L., (2010). Travelling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches. In: D. Davendra (Eds.), *Travelling Salesman Problem, Theory and Applications,* 1-19. DOI: 10.5772/12909

Mennell W.K., (2009). Heuristics for solving three routing problems: Close-enough Travelling salesman problem close-enough vehicle routing problem sequence-dependent team orienteering problem. PhD Thesis, University of Maryland. Retrieved October 15, 2019 from https://drum.lib.umd.edu/handle/1903/9822

Misevičius A., Smolinskas J. & Tomkevičius A., (2015). Iterated Tabu Search For The Travelling Salesman Problem: New Results. *Information Technology And Control 34(4)*, 327-336.

Mitchell J.E., (2008). Integer Programming: Cutting Plane Algorithms. In: Floudas C., Pardalos P. (eds) Encyclopedia of Optimization. Springer, Boston, MA, 1650-1657.

Monnot, J., (2005). Approximation algorithms for the maximum hamiltonian path problem with specified endpoint(s). *European Journal of Operations Research 161(3)*, 721–735

Morais V.W.C., Mateus G.R. & Noronha T.F., (2014). Iterated local search heuristics for the Vehicle Routing Problem with Cross-Docking. *Expert Systems with Applications 41*, 7495–7506. http://dx.doi.org/10.1016/j.eswa.2014.06.010.

Mostafa, H., Müller, L. & Indiveri, G., (2015). An event-based architecture for solving constraint satisfaction problems. *Nature Communications 6*, 8941 (2015). https://doi.org/10.1038/ncomms9941

Muklason A., Parkes A.J., Ozcan E., McCollum B. & McMullan P., (2017). Fairness in examination timetabling: Student preferences and extended formulations. *Applied Soft Computing 55,* 302–318, 2017.

Myasnikov A.G., Nikolaev A., Ushakov A. (2015). Knapsack problems in groups. *Mathematical Computing 84(292),* 987-1016

Nath R., (2016). Approximation solution of Travelling Salesman Problem using Dijkstra and Bitonic algorithms. *International Journal of Innovations & Advancement in Computer Science* 5(2), 102-108.

Necula, R., Breaban, M., & Raschip, M. (2015). Tackling the bi-criteria facet of multiple Travelling salesman problem with ant colony systems. Procedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), 873–880. Vietri sul Mare, Italy: IEEE.

Neelima S., Satyanarayana N. & Murthy P.K. (2016). A Comprehensive Survey on Variants in Artificial Bee Colony. *International Journal of Computer Science and Information Technologies 7(4),* 1684–89

Neissi N.A. & Mazloom M., (2009). GLS Optimization Algorithm for Solving the Travelling Salesman Problem. Proceeding of the Second International Conference on Computer and Electrical Engineering, 291-294. doi:10.1109/iccee.2009.102

Neos (2018). Combinatorial Optimization. Retrieved February 13, 2019 from https://neos-guide.org/content/combinatorial-optimization.

Nikolas Klug, Alok Chauhan, Vijayakumar & Ramesh Ragala, (2019). k -RNN: Extending NN-heuristics for the TSP. *Mobile Networks and Applications 24,* 1210–1213.

Nima Anari & Shayan Oveis Gharan, (2015). Effective-Resistance-Reducing Flows, Spectrally Thin Trees, and Asymmetric TSP. Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), 20–39.

Oberlin P., Rathinam S. & Darbha S., (2009). A Transformation for a Multiple Depot, Multiple Travelling Salesman Problem. Proceedings of the American Control Conference, St. Louis, MO, 2009, 2636-2641, doi: 10.1109/ACC.2009.5160665.

Oberlin, P., RathinamS. & Darbha S.. (2009). A transformation for a heterogeneous, multi-depot, multiple Travelling salesman problem. Proceedings of the American Control Conference, 1292-1297, St. Louis, June 10 - 12, 2009.

Oliveira J.F. & Carravilla M.A., (2009). Heuristics and Local Search. Retrieved Fevruary 13, 2019 from https://paginas.fe.up.pt/~mac/ensino/docs/OR/CombinatorialOptimizationHeuristicsLocalSearch.pdf.

Oliver Lum, Rui Zhang, Bruce Golden & Edward Wasil (2017). A hybrid heuristic procedure for the Windy Rural Postman Problem with Zigzag Time Windows. *Computers and Operations Research 88,* 247–257. http://dx.doi.org/10.1016/j.cor.2017.07.007.

Padberg M. & Rinaldi G., (1991). A Branch-And-Cut Algorithm For The Resolution Of Large-Scale Symmetric Travelling Salesman Problems. *SIAM Review 33(1),* 60–100. https://doi.org/10.1137/1033004

Papadimitriou C.H., (1992). The Complexity of the Lin–Kernighan Heuristic for the Travelling Salesman Problem. *SIAM Journal on Computing 21(3),* 450–465. doi:10.1137/0221030

Patel V. & Baggar C., (2014). A survey paper of the Bellman-ford algorithm and Dijkstra algorithm for finding the shortest path in a GIS application. *International Journal of P2P Network Trends and Technology (IJPTT) 4(1),* 21-23.

Pelaez V., Campos A., Garcia DF. & Entrialgo J., (2016). Autonomic scheduling of deadline-constrained bag of tasks in hybrid clouds. In: International Symposium on Performance Evaluation of Computer and Telecommunication Systems, Montreal, QC, Canada; 2016, 1-8.

Peng Y., Lu D. & Chen Y. (2014). A Constraint Programming Method for Advanced Planning and Scheduling System with Multilevel Structured Products. *Discrete Dynamics in Nature and Society* 2014, 1-8. https://doi.org/10.1155/2014/917685

Pichpibul T. & Kawtummachai R., (2012). New Enhancement for Clarke-Wright Savings Algorithm to Optimize the Capacitated Vehicle Routing Problem. *European Journal of Scientific Research 78(1),* 119-134.

Potvin J.Y. & Guertin F., (1996). The Clustered Travelling Salesman Problem: A Genetic Approach. In: I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics. Springer, Boston, MA

Puchinger J., Raidl G. & Pferschy U., (2010). The Multidimensional Knapsack Problem: Structure and Algorithms. *INFORMS Journal on Computing, Institute for Operations Research and the Management Sciences (INFORMS) 22(2),* 250 - 265. ff10.1287/ijoc.1090.0344ff. ffhal01224914f

Punnen A.P. (2007) The Travelling Salesman Problem: Applications, Formulations and Variations. In: G. Gutin G. A.P. Punnen (Eds.), *The Travelling Salesman Problem and Its Variations. Combinatorial Optimization,* 12. Springer, Boston, MA. 1-28.

Qing, N., Kang, F., & Marine, S. O. (2015). Application of a new acceleration particle swarm optimization for solving multiple Travelling salesman problems. *Journal of Shaanxi Normal University, 43(6),* 7.

Qu R., He F. & Burke E.K., (2009). Hybridizing Integer Programming Models with an Adaptive Decomposition Approach for Exam Timetabling Problems. In Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications, pp. 435–446.

Ramshaw L. & Tarjan R.E., (2012). On Minimum-Cost Assignments in Unbalanced Bipartite Graphs. HP Laboratories. HPL-2012-40R1. Retrieved July 3, 2020 from https://www.hpl.hp.com/techreports/2012/HPL-2012-40R1.pdf.

RAO W. & JIN C, (2010). A New Hybrid Algorithm for Solving TSP. Proceedings of ICLEM 2010: Logistics for Sustained Economic Development, (2010), 3327-3334.

Ratnasari A., Ardiani F. & Nurvita F., (2013). Penentuan Jarak Terpendek dan Jarak Terpendek Alternatif menggunakan Algoritma Dijkstra serta Estimasi Waktu Tempuh. Universitas Islam Indonesia. ISBN: 979-26-0266-6. Yogyakarta.

Rego C., Gamboa D., Glover F. & Osterman C., (2011). Travelling salesman problem heuristics: leading methods, implementations, and latest advances. *European Journal of Operational Research, 211(3),* 427-441

Reinelt, G., (1994). The Travelling Salesman: Computational Solutions for TSP Applications. Springer-Verlag Berlin Heidelberg. DOI_10.1007/3-540-48661-5

Roberti, R. & Toth, P., (2012). Models and algorithms for the Asymmetric Travelling Salesman Problem: an experimental comparison. *European Journal of Transportation and Logistics 1,* 113–133. https://doi.org/10.1007/s13676-012-0010-0

Roldán, E., Negny, S., Marc Le Lann, J., & Cortés, G. (2011). Modified Case Based Reasoning cycle for Expert Knowledge Acquisition during Process design. 21st European Symposium on Computer Aided Process Engineering, 296–300. doi:10.1016/b978-0-444-53711-9.50060-2

Rosenkrantz D.J., Stearns R.E. & Lewis II, P.M., (1977). An analysis of several heuristics for the Travelling salesman problem. *SIAM Journal of Computing 6(3),* 563–581.

Rutishauser U., Slotine J.-J. & Douglas R.J. (2018). Solving constraint-satisfaction problems with distributed neocortical-like neuronal networks. *Neural Computation 30(5),* 1359–1393. doi: 10.1162/NECO_a_01074

Sabar N.R., Ayob M., Qu R. & Kendall G., (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence, 37(1),* 1–11.

Sahu M., Singh A.V. & Khatri S.K., (2019). A Classical Constraint Satisfaction Problem and its Solution using Artificial Intelligence. Proceedings of the 2019 Amity International Conference on Artificial Intelligence (AICAI), Dubai, United Arab Emirates, 429-433, doi: 10.1109/AICAI.2019.8701325.

Saiyed A.R., (2012). The Travelling Salesman problem. Indiana State University Terre Haute, IN 47809, USA. Retrieved April 13, 2020 from http://cs.indstate.edu/~zeeshan/aman.pdf.

Salehi, K. (2014). An approach for solving multi-objective assignment problem with interval parameters. *Management Science Letters 4(9),* 2155-2160.

Savelsbergh M.W.P. (2001). Branch and Price: Integer Programming with Column Generation. In: C.A. Floudas, P.M. Pardalos (Eds.), Encyclopedia of Optimization. Springer, Boston, MA. https://doi.org/10.1007/0-306-48332-7.

Schulze B., Stiglmayr M. & Paquete, L., (2020). On the rectangular knapsack problem: approximation of a specific quadratic knapsack problem. *Mathematical Methods of Operations Researh 92,* 107-132. https://doi.org/10.1007/s00186-020-00702-0

Sengupta A., (2017). Assignment Problem: Meaning, Methods and Variations. Retrieved July 3, 2020 from https://www.engineeringenotes.com/project-management-2/operationsresearch/assignment-problem-meaning-methods-and-variations-operations-research/15652.

Sergeev S.I., (2014). Maximum travelling salesman problem. *Automation and Remote Control 75(12)*, 2170–2189. https://doi.org/10.1134/S0005117914120078

Shah K., Reddy P. & Vairamuthu S. (2015) Improvement in Hungarian Algorithm for Assignment Problem. In: L. Suresh, S. Dash & B. Panigrahi (Eds.), *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems. Advances in Intelligent Systems and Computing,* 324. Springer, New Delhi. https://doi.org/10.1007/978-81-322-2126-5_1.

Sherali H, Driscoll P (2002) On tightening the relaxations of Miller–Tucker–Zemlin formulations for asymmetric Travelling salesman problems. *Operations Research* 50(6), pp 656–669

Shim, V. A., Tan, K. C., & Tan, K. K. (2012). A hybrid estimation of distribution algorithm for solving the multi-objective multiple Travelling salesman problem. IEEE congress on evolutionary computation, 1–8. Brisbane, QLD, Australia: IEEE

Shuai Y., Yunfeng S. & Kai Z., (2019). An effective method for solving multiple travelling salesman problem based on NSGA-II, Systems Science & *Control Engineering 7(2),* 108-116, DOI: 10.1080/21642583.2019.1674220

Singh S., (2012). A Comparative Analysis of Assignment Problem. IOSR *Journal of Engineering 2(8),* 1–15.

Sitek P. & Wikarek J., (2016). A Hybrid Programming Framework for Modeling and Solving Constraint Satisfaction and Optimization Problems. Scientific Programming 2016, 1-14. https://doi.org/10.1155/2016/5102616

Štencek J., (2013). Travelling salesman problem. Bachelor’s Thesis, Degree Programme in Software engineering, JAMK University of Applied Sciences, May 2013. Retrieved Februry 12, 2020 from https://www.theseus.fi/bitstream/handle/10024/59942/Jakub_Stencek_TSP_Bachelor_Thesis.pdf?sequence=1&isAllowed=y.

Stratopoulos C., (2017). Primal Cutting Plane Methods for the Travelling Salesman Problem. Masters Thesis Mathematics in Combinatorics and Optimization, University of Waterloo, Ontario, Canada, 2017. Retrieved February 15, from https://uwspace.uwaterloo.ca/bitstream/handle/10012/11755/Stratopoulos_Christos.pdf?sequence=1&isAllowed=y.

Sundar K. & Rathinam S., (2017). Algorithms for Heterogeneous, Multiple Depot, Multiple Unmanned Vehicle Path Planning Problems. *Journal of Intelligent Robotic Systems 88*, 513–526. https://doi.org/10.1007/s10846-016-0458-5

Svensson O. Tarnawski J. & Végh L.A. (2018). A constant-factor approximation algorithm for the asymmetric Travelling salesman problem. STOC 2018: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, 204–213. https://doi.org/10.1145/3188745.3188824

Syahputra M.F.A., Devita R.N., Siregar S.A. & Kirana K.C., (2016). Implementation of Travelling Salesman Problem (TSP) based on Dijkstra's Algorithm in Logistics System. JAVA, *International Journal of Electrical and Electronics Engineering 4(1),* 39-44.

Tang L., Liu J., Rong A. & Yang Z., (2000). A multiple Travelling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. *European Journal of Operational Research 124(2),* 267-282.

Thenepalle J.K. & Singamsetty P., (2019). An open close multiple travelling salesman problem with single depot. *Decision Science Letters 8(2),* 121-136. DOI: 10.5267/j.dsl.2018.8.002.

Tutte W.T., (2012). Graph Theory as I Have Known It (Oxford Lecture Series in Mathematics and Its Applications). Clarenford Press, Oxford. ISBN 978-0-19-966055-1.

Universität Heidelberg, (2007)._Optimal solutions for symmetric TSPs. Retrieved December 17, 2019 from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html.

Ursani Z. & Corne D.W., (2016). Introducing Complexity Curtailing Techniques for the Tour Construction Heuristics for the Travelling Salesperson Problem. *Journal of Optimization 2016,* 1-16. https://doi.org/10.1155/2016/4786268

Vaishnav P., Choudhary N. & Jain K., (2017). Travelling Salesman Problem Using Genetic Algorithm : A Survey. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology 2(3),* 105–108.

Valenzuela C.L. & Jones A.J., (1997). Estimating the Held-Karp lower bound for the geometric TSP. *European Journal of Operational Research 102(1),* 157-175.

Van den Bossche R., Vanmechelen K. & Broeckhove J., (2013). Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. *Future Generation Computing Systems 29(4),* 973-985.

Venkatesh P., Singh A. & Mallipeddi R., (2019). A Multi-Start Iterated Local Search Algorithm for the Maximum Scatter Travelling Salesman Problem. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 2019, 1390-1397, doi: 10.1109/CEC.2019.8790018.

Víctor Pacheco-Valencia, José Alberto Hernández, José María Sigarreta & Nodari Vakhania, (2020). Simple Constructive, Insertion, and Improvement Heuristics Based on the Girding Polygon for the Euclidean Travelling Salesman Problem. *Algorithms 1*3 (5), 1-30. doi:10.3390/a13010005.

Walrand J. & Varaiya P., (2000). The Internet and TCP/IP Networks. In: Jean Walrand, Pravin Varaiya, (eds) High-Performance Communication Networks (Second Edition), Morgan Kaufmann, 2000, 155-203. https://doi.org/10.1016/B978-0-08-050803-0.50013-7

Waltz D.L., (1972). Generating semantic descriptions from drawings of scenes with shadows". Technical Report. AI-TR-271, MIT, Cambridge, M A, 1972. Retrieved December 10, 2019 from https://dspace.mit.edu/handle/1721.1/41205.

Wang B., Song Y., Sun Y. & Liu J., (2016). Managing deadline-constrained bag-of-tasks jobs on hybrid clouds. In: High Performance Computing Symposium, Pasadena,CA,USA; 2016:22.

Wang C., Lin M. & Zhong Y. (2016). Swarm simulated annealing algorithm with knowledge-based sampling for travelling salesman problem. *International Journal of Intelligent Systems Technologies and Applications 15(1)*, 74–94.

Wang H.-F. & Chen Y.-Y., (2012). A genetic algorithm for the simultaneous delivery and pickup problems with time window. *Computers & Industrial Engineering 62,* 84–95. https://doi.org/10.1016/j.cie.2011.08.018

Wang H.-F. & Chen Y.-Y., (2013). A coevolutionary algorithm for the flexible delivery and pick up problem with time windows. *International Journal of Production Economics 141,* 4–13. http://dx.doi.org/10.1016/j.ijpe.2012.04.011.

Weiler C., Biesinger B., Hu B. & Raid G.R. (2015). Heuristic Approaches for the Probabilistic Travelling Salesman Problem. R. Moreno-D´ıaz et al. (Eds.): EUROCAST 2015, LNCS 9520, 342–349, 2015. DOI: 10.1007/978-3-319-27340-2 43

Woeginger G.J., (2003). Exact algorithms for NP-hard problems: a survey. Combinatorial optimization - Eureka, you shrink! Springer-Verlag New York, Inc. New York, NY, USA. ISBN: 3-540-00580-3, 185 – 207.

Woumans G., De Boeck L., Belien J. & Creemers S., (2016). A column generation approach for solving the examination-timetabling problem. *European Journal of Operational Research 253(1),* 178–194.

Xu X.-Y., Huang X.-L., Li Z.-M, Gao J., Jiao Z.-Q., Wang Y., Ren R.-J., Zhang H.P. & Jin X.-M., (2020). A scalable photonic computer solving the subset sum problem. *Science Advances* 6(5), 1-7. DOI: 10.1126/sciadv.aay5853

Xu Z. & Rodrigues B.C., (2017). An extension of the Christofides heuristic for the generalized multiple depot multiple Travelling salesmen problem. European *Journal of Operational Research 257(3),* 735-745. DOI: 10.1016/j.ejor.2016.08.054

Xu Z., Xu L. & Rodrigues B.C., (2011). An Analysis of the Extended Christofides Heuristic for the k-depot Travelling Salesman Problem. *Operations Research Letters 39(3),* 218-223. https://doi.org/10.1016/j.orl.2011.03.002

Yang J., Shi X., Marchese M. & Liang Y., (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science 18* 1417-1422.

Yu T & LinK.-J., (2004). Service selection algorithms for Web services with end-to-end QoS constraints. Proceedings of the IEEE International Conference on e-Commerce Technology, 2004. CEC 2004., San Diego, CA, USA, 2004, 129-136, doi: 10.1109/ICECT.2004.1319726.

Yuan X., (1999). On the extended Bellman-Ford algorithm to solve two-constrained quality of service routing problems. Proceedings of the Eighth International Conference on Computer Communications and Networks (Cat. No.99EX370), Boston, MA, USA, 1999, 304-310.

Zamani R., (2013). A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. *European Journal of Operational Research 229(2),* 552-559. https://doi.org/10.1016/j.ejor.2013.03.005

Zhang Y. & Sun J., (2017). Novel efficient particle swarm optimization algorithms for solving QoS-demanded bag-of-tasks scheduling problems with profit maximization on hybrid clouds. *Concurrency and Computation Practice and Experience* 29(21), 42-49. https://doi.org/10.1002/cpe.4249

Zhong T. & Young R. (2010). Multiple Choice Knapsack Problem: Example of planning choice in transportation. *Evaluation and Program Planning 33(2),* 128-137. https://doi.org/10.1016/j.evalprogplan.2009.06.007

APPENDICES

APPENDIX I: DATASET CONVERSION MODULE

```
import tsplib95

import networkx

import numpy as np

arr = ['ulysses16.tsp']

def iterate(arr):

    for i in range(len(arr)):

        name = arr[i].split(".")

        problem = tsplib95.load_problem(arr[i])

        graph = problem.get_graph()

        distance_matrix = networkx.to_numpy_matrix(graph)

        distance_matrix = np.array(distance_matrix)

        fo = open("./output/" + name[0] + ".txt", 'w')

        for i in range(len(distance_matrix)):

            for j in range(len(distance_matrix[i])):

                fo.write(str(distance_matrix[i][j]))

                if(j != len(distance_matrix) - 1):

                    fo.write(" ")
```

```
            if(i != len(distance_matrix) - 1):
                fo.write("\n")
        fo.close()


if '__main__' == __name__:
    iterate(arr)
```

APPENDIX II: CONSTRUCTOR – STRATEGY MODULE

```
package tsp;


/**
 * The {@code Strategy} class represent a specific strategy to
solve a given TSP
 * problem
 *
 * @author Nathaniel
 */
public abstract class Strategy {

     /**
      * A {@code RoadMap} object that this strategy works on
      */
     protected RoadMap rm;

     /**
      * The only constructor.
      *
```

```
	 * @param rm A {@code Strategy} object to be directly assigned to the RoadMap rm
	 *           attribute
	 */
	public Strategy(RoadMap rm) {
		this.rm = rm;
	}

	/**
	 * Every child class must provide an implementation to solve this TSP problem
	 *
	 * @return A {@code Tour} object that represents the solution of this strategy
	 */
	public abstract Tour solve();

	/**
	 * Get a built-in strategy to solve this TSP using brute force
```

```
     *
     * @param rm A {@code RoadMap} object that this strategy works on
     * @return A {@code Tour} object that represents the solution of this strategy
     */
    public static Strategy bruteForce(RoadMap rm) {
        return new BruteForceStrategy(rm);
    }

    /**
     * Get a built-in strategy to solve this TSP using Nearest Neighbor Heuristic
     *
     * @param rm    A {@code RoadMap} object that this strategy works on
     * @param start The city this strategy starts with
     * @return A {@code Tour} object that represents the solution of this strategy
     */
```

```
	public static Strategy nearestNeighbor(RoadMap rm, String start) {
		return new NearestNeighborStrategy(rm, start);
	}

	/**
	 * Get a built-in strategy to solve this TSP using Farthest Insertion Heuristic
	 *
	 * @param rm A {@code RoadMap} object that this strategy works on
	 * @param a  One of the three cities to form a triangle that this strategy
	 *           starts with
	 * @param b  One of the three cities to form a triangle that this strategy
	 *           starts with
	 * @param c  One of the three cities to form a triangle that this strategy
	 *           starts with
	 * @return A {@code Tour} object that represents the solution of this strategy
```

```
	 */
	public static Strategy farthestInsertion(RoadMap rm, String a, String b, String c) {
		return new FarthestInsertionStrategy(rm, a, b, c);
	}
	/**
	 * Get a built-in strategy to solve this TSP using Nearest Insertion Heuristic
	 *
	 * @param rm A {@code RoadMap} object that this strategy works on
	 * @param a  One of the three cities to form a triangle that this strategy
	 *           starts with
	 * @param b  One of the three cities to form a triangle that this strategy
	 *           starts with
	 * @param c  One of the three cities to form a triangle that this strategy
	 *           starts with
```

```
     * @return A {@code Tour} object that represents the solution of this strategy
     */
    public static Strategy nearestInsertion(RoadMap rm, String a, String b, String c) {
        return new NearestInsertionStrategy(rm, a, b, c);
    }
    /**
     * Get a built-in strategy to solve this TSP using Min Max Insertion Heuristic
     *
     * @param rm A {@code RoadMap} object that this strategy works on
     * @param a  One of the three cities to form a triangle that this strategy
     *           starts with
     * @param b  One of the three cities to form a triangle that this strategy
     *           starts with
     * @param c  One of the three cities to form a triangle that this strategy
```

```
     *           starts with
     * @return A {@code Tour} object that represents the solution of this strategy
     */
    public static Strategy MinMaxInsertion(RoadMap rm, String a, String b, String c) {
        return new MinMaxStrategy(rm, a, b, c);
    }
    /**
     * Get a built-in strategy to solve this TSP using Farthest Insertion Heuristic
     *
     * @param rm A {@code RoadMap} object that this strategy works on
     * @param a  One of the three cities to form a triangle that this strategy
     *           starts with
     * @param b  One of the three cities to form a triangle that this strategy
     *           starts with
```

```
         * @param c  One of the three cities to form a triangle
that this strategy
         *           starts with
         * @return A {@code Tour} object that represents the
solution of this strategy
         */
        public static Strategy MidpointinsertionStrategy(RoadMap
rm, String a, String b, String c) {
                return new MidpointinsertionStrategy(rm, a, b, c);
        }
        //              public              static              double
CheapestInsertionStrategy(double[][] rm, int start) {
        // return new CheapestInsertionStrategy(rm, start);
        // }
```

APPENDIX III: NEAREST NEIGHBOUR HEURISTIC JAVA CODE

```
package tsp;

import java.util.LinkedList;

class NearestNeighborStrategy extends Strategy {

	private String start;

	protected NearestNeighborStrategy(RoadMap rm, String start) {

		super(rm);

		this.rm.checkCity(start);

		this.start = start;

	}

	private String findNearestNeighbor(String city, LinkedList<String> unvisited) {

		double shortest = Double.MAX_VALUE;

		String nearest = null;

		for (String s : unvisited) {

			double current = this.rm.getDistance(city, s);

			if (Double.compare(current, shortest) < 0) {

				nearest = s;
```

```
                    shortest = current;
                }
            }
            return nearest;
        }

        @Override
        public Tour solve() {
            long start_time = System.nanoTime();
            Tour.Builder tb = new Tour.Builder(this.rm);
            String current = this.start;
            LinkedList<String>       unvisited       =       new
LinkedList<String>(this.rm.getCitySet());
            unvisited.remove(current);
            while (!unvisited.isEmpty()) {
                String                nearest                =
this.findNearestNeighbor(current, unvisited);
                tb.addPair(current, nearest);
                current = nearest;
                unvisited.remove(current);
```

```
        }
        tb.addPair(current, start);
        long end_time = System.nanoTime();
        System.out.printf("Time Taken : %d\n", end_time -
start_time);
        return tb.build();
    }

}
```

APPENDIX IV: FARTHEST INSERTION HEURISTIC JAVA CODE

```
package tsp;

class FarthestInsertionStrategy extends Strategy {

	private String a;

	private String b;

	private String c;

	protected FarthestInsertionStrategy(RoadMap rm, String a, 
String b, String c) {

		super(rm);

		rm.checkCity(a);

		rm.checkCity(b);

		rm.checkCity(c);

		if(a.equals(b) || a.equals(c) || b.equals(c)){

			throw new RuntimeException(a + ", " + b + ", " 
+ c + " cannot form a triangle");

		}

		this.a = a;

		this.b = b;

		this.c = c;
```

```
    }

        private double distanceFrom(String city, Tour.Builder
tb){

        double max = 0;

        for(String s : tb.getCities()){

            double current = this.rm.getDistance(city, s);

            if(current > max){

                max = current;

            }

        }

        return max;

    }


    private String findFarthestCity(Tour.Builder tb){

        String farthest = "";

        double maxDist = 0;

        for(String city : this.rm.getCitySet()){

            if(!tb.covers(city)){

                double          currentDist          =
this.distanceFrom(city, tb);
```

```
                    if(currentDist > maxDist){
                        maxDist = currentDist;
                        farthest = city;
                    }
                }
            }
            return farthest;
        }

        private void insertCity(String city, Tour.Builder tb){
            Pair target = null;
            double minIncr = Double.MAX_VALUE;
            for(Pair p : tb.getPairs()){
                String a = p.getSmaller();
                String b = p.getLarger();
                double  incr  =  this.rm.getDistance(city,  a)  +
    this.rm.getDistance(city, b) - this.rm.getDistance(p);
                if(Double.compare(incr, minIncr) < 0){
                    target = p;
                    minIncr = incr;
```

```
            }
        }
        tb.removePair(target);
        tb.addPair(target.getSmaller(), city);
        tb.addPair(target.getLarger(), city);
    }
    @Override
    public Tour solve() {
        Tour.Builder tb  = new Tour.Builder(this.rm);
        tb.addPair(this.a, this.b);
        tb.addPair(this.a, this.c);
        tb.addPair(this.b, this.c);
        while(tb.size() < this.rm.size()){
            String farthest = this.findFarthestCity(tb);
            this.insertCity(farthest, tb);
        }
        return tb.build();
    }
    }
```

APPENDIX V: HALF MAX INSERTION HEURISTIC JAVA CODE

```
package tsp;

import java.util.Arrays;

import java.util.ArrayList;

import java.util.List;

import java.util.Collections;

class MidpointinsertionStrategy extends Strategy {

	private String a;

	private String b;

	private String c;

	protected MidpointinsertionStrategy(RoadMap rm, String a, 
String b, String c) {

		super(rm);

		rm.checkCity(a);

		rm.checkCity(b);

		rm.checkCity(c);

		if(a.equals(b) || a.equals(c) || b.equals(c)){

			throw new RuntimeException(a + ", " + b + ", " 
+ c + " cannot form a triangle");

		}
```

```
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public static double findMax(double[] a, int total){
        double temp;
        for (int i = 0; i < total; i++){
              for (int j = i + 1; j < total; j++)
              {
                  if (a[i] > a[j])
                  {
                      temp = a[i];
                      a[i] = a[j];
                      a[j] = temp;
                  }
              }
          }
         return a[total-1];
    }
```

```
// public double findClosest(double myNumber, double[]
numbers){
//    double distance = Math.abs(numbers[0] - myNumber);
//    int idx = 0;
//    for(int c = 1; c < numbers.length; c++){
//        double cdistance = Math.abs(numbers[c] -
myNumber);
//        int closestSoFar = abs(numbers[i] - myNumber);
//        if (abs(numbers[i] - myNumber) < abs(closestSoFar
- myNumber)) {
//        // if(cdistance < distance){
//            idx = c;
//            distance = cdistance;
//        }
//    }
//    double theNumber = numbers[idx];
//    return theNumber;
// }
```

```
private static void removeDuplicates(String[] array) {

    int[] occurence = new int[array.length];

    for (int i = 0; i < array.length; i++) {

        for(int j=i+1;j<array.length;j++){

            if(array[i]==array[j]){

                occurence[j]=j;

            }

        }

    }

    int resultLength=0;

    for(int i=0;i<occurence.length;i++){

        if(occurence[i]==0){

            resultLength++;

        }

    }

    String[] result=new String[resultLength];

    int index=0;int j=0;

    for(int i=0;i<occurence.length;i++){

        index = occurence[i];
```

```
            if(index==0){
                result[j]= array[i];
                j++;
            }
        }
        for(String eachString : result){
            // System.out.println(eachString);
        }
    }
    public double findClosest(String city, double targetNumber, Tour.Builder tb){
        double closestDifference = targetNumber;
        double closestNumber= 0;
        // for (int i = 0; i < numbers.length; i++){
        int i = 0;
        for(String s : tb.getCities()){
            System.out.println(s);
            if(closestDifference > java.lang.Math.abs(this.rm.getDistance(city, s)-targetNumber)){
```

```
                    closestDifference                    =
java.lang.Math.abs(this.rm.getDistance(city, s)-targetNumber);
                    closestNumber=  this.rm.getDistance(city,
s);
                }
                i++;
            }
            return closestNumber;
        }
        public  double  calcDistanceFrom(String  city,  Tour.Builder
tb){
            double max = 0;
            double mid = 0;
            double min = 0;
            for(String s : tb.getCities()){
                max = this.rm.getDistance(city, s);
                mid = this.rm.getDistance(city, s);
                double current = this.rm.getDistance(city, s);
                if(current < max && current > min){
                    max = max;
```

```
}

            min = min;

            mid = current;

        }else if(current > max){

            mid = max;

            max = current;

            min = min;

        }else if(min > max){

            max = min;

            mid = mid;

            max = max;

        }else{

            max = max;

            min = min;

            mid = mid;

        }

    }

    return mid;

}
```

```
	private double distanceFrom(String city, Tour.Builder tb){
		double max = 0;
		for(String s : tb.getCities()){
			double current = this.rm.getDistance(city, s);
			if(current > max){
				max = current;
			}
		}
		return max;
	}
	private    String    findMiddleCity(Tour.Builder    tb,    int
rmsize){
		String middle = "";
		double maxDist = 0;
		int i = 0;

		for(String city : this.rm.getCitySet()){
			if(!tb.covers(city)){
				//        double        currentDist        =
findClosest(findMax(allcitties, rmsize), allcitties);
```

```
                    //       double       currentDist        =
this.distanceFrom(city, tb);
                    double           currentDist             =
calcDistanceFrom(city, tb);
                    middle = city;
                    // if(currentDist > maxDist){
                    //   maxDist = currentDist;
                    //   middle = city;
                    // }else{
                    //   middle = city;
                    // }
               }
               i++;
          }
          return middle;
     }
     private void insertCity(String city, Tour.Builder tb){
          Pair target = null;
          double minIncr = Double.MAX_VALUE;
          for(Pair p : tb.getPairs()){
```

```
                String a = p.getSmaller();

                String b = p.getLarger();

                double  incr  =  this.rm.getDistance(city,  a)  +
this.rm.getDistance(city, b) - this.rm.getDistance(p);

                if(Double.compare(incr, minIncr) < 0){

                     target = p;

                     minIncr = incr;

                }

           }

           tb.removePair(target);

           tb.addPair(target.getSmaller(), city);

           tb.addPair(target.getLarger(), city);

     }

     @Override

     public Tour solve() {

           Tour.Builder tb  = new Tour.Builder(this.rm);

           tb.addPair(this.a, this.b);

           tb.addPair(this.a, this.c);

           tb.addPair(this.b, this.c);

           while(tb.size() < this.rm.size()){
```

```
                String    middle    =    this.findMiddleCity(tb,
this.rm.size());
                this.insertCity(middle, tb);
            }
            return tb.build();
        }
        }
```

APPENDIX VI: IMPLEMENTATION MODULE – MAIN CLASS

```
import tsp.FileProcessor;

import tsp.RoadMap;

import tsp.Strategy;

import util.Printer;



// import tsp.CheapestInsertion;

// import tsp.ConstructionHeuristic;

public class TSPDemo {

    public static final String bruteforce = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/bruteforce.txt";

    public static final String german = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/german.txt";

    public static final String q1 = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/q1.txt";

    public static final String q1seq = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/q1seq.txt";
```

```
	public static final String seq = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/seq.txt";
	public static final String matric = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/matric.txt";
	public static final String tut8q1 = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/tut8q1.txt";
	public static final String tut8q2 = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/tut8q2.txt";
	public static final String tut8q2seq = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/tut8q2seq.txt";
	public static final String q6 = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/q6.txt";
	public static final String q9 = "/Users/apple/Documents/java/Travelling-Salesman-Problem-Solver/src/samples/q9.txt";
	// our own import
```

```
public static final String burma14 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/burma14.txt";
public static final String att48 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/att48.txt";
public static final String ulysses22 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/ulysses22.txt";
public static final String ulysses16 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/ulysses16.txt";
public static final String bays29 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/bays29.txt";
public static final String berlin52 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
```

```
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/berlin52.txt";
    public static final String brazil58 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/brazil58.txt";
    public static final String eil51 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/eil51.txt";
    public static final String eil76 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/eil76.txt";
    public static final String rat99 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/rat99.txt";
    public static final String bier127 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/bier127.txt";
```

```
public static final String d657 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/d657.txt";
public static final String eil101 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/eil101.txt";
public static final String gr229 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/gr229.txt";
public static final String lin318 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/lin318.txt";
public static final String pr439 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/pr439.txt";
public static final String rat195 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-
```

```
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/rat195.txt";
      public static final String rat575 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/rat575.txt";
      public static final String u724 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/u724.txt";
      public static final String ch130 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/ch130.txt";
      public static final String ch150 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/ch150.txt";
      public static final String rat783 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/rat783.txt";
```

```
public static final String ali535 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/ali535.txt";
public static final String dsj1000 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/dsj1000.txt";
public static final String u2319 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/u2319.txt";
public static final String pcb3038 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/pcb3038.txt";
public static final String rl5915 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/rl5915.txt";
public static final String fl3795 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-
```

```
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/fl3795.txt";
    public static final String d1655 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/d1655.txt";
    public static final String d2103 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/d2103.txt";
    public static final String pr2392 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/pr2392.txt";
    public static final String rl1889 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/rl1889.txt";
    public static final String u1817 =
"C:/Users/asani/Documents/Travelling-Salesman-Problem-
Solver/Travelling-Salesman-Problem-
Solver/src/samples/output/u1817.txt";
```

```
	public static final String u2152 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/u2152.txt";

	public static final String vm1748 = "C:/Users/asani/Documents/Travelling-Salesman-Problem-Solver/Travelling-Salesman-Problem-Solver/src/samples/output/vm1748.txt";

	public static void main(String[] args) {

		FileProcessor fpdt = FileProcessor.DISTANCE_MATRIX;

		// RoadMap rm = fpdt.read(matric);

		RoadMap rm = fpdt.read(eil51);

		// RoadMap rm = fpdt.read(dsj1000);

		// System.out.println(Strategy.bruteForce(rm).solve());

		System.out.println(Strategy.nearestNeighbor(rm, "C1").solve());


		//long startTime1 = System.nanoTime();

		//System.out.println(Strategy.farthestInsertion(rm, "C3", "C4", "C5").solve());
```

```
//long endTime1   = System.nanoTime();

//long totalTime1 = endTime1 - startTime1;

//System.out.println(totalTime1);

//System.out.println("/////////////////////////");


//long startTime = System.nanoTime();

//System.out.println(Strategy.nearestInsertion(rm,
"C3", "C4", "C5").solve());

//long endTime = System.nanoTime();

//long totalTime = endTime - startTime;

//System.out.println(totalTime);


//long startTime = System.nanoTime();


//System.out.println(Strategy.MidpointinsertionStrategy(r
m,"C3", "C4", "C5").solve());

//long endTime   = System.nanoTime();

//long totalTime = endTime - startTime;

//System.out.println(totalTime);
```

```
            // double[][] distances = new double[][] {{0, 8, 4,
9, 9},
            //
{8, 0, 6, 7, 10},
            //
{4, 6, 0, 5, 6},
            //
{9, 7, 5, 0, 4},
            //
{9, 10, 6, 4, 0}};
            // System.out.println("/////////////////");
            // heuristic = new CheapestInsertion(distances, 0);
            // Printer.printArray(heuristic.getTour());
            // long startTime1 = System.nanoTime();
            //   System.out.println(Strategy.MinMaxInsertion(rm,
"C3", "C4", "C5").solve());
            // long endTime1   = System.nanoTime();
            // long totalTime1 = endTime1 - startTime1;
            // System.out.println(totalTime1);
      }
}
```