An Improved Approximation Algorithm for Symmetric & Asymmetric TSP Problems

By
Fakhar uddin
00000170486

Supervisor
Dr. Naveed Riaz
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of MS(CS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(September 2019)

Approval

It is certified that the contents and form of the thesis entitled "**An Improved Approximation Algorithm for Symmetric & Asymmetric TSP Problems**" submitted by **Fakhar uddin** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Naveed Riaz**

Signature: _____
Date: _____

Committee Member 1: **Dr. Hasan Tahir**

Signature: _____
Date: _____

Committee Member 2: **Dr. Imran mahmood**

Signature: _____
Date: _____

Committee Member 3: **Dr. Shahzad saleem**

Signature: _____
Date: _____

Abstract

Travelling salesman problem (TSP) is one of the fundamental and most researched problems in the study of Approximation Algorithms. Generally, the algorithm is provided with a metric(V, d) and the aim of the algorithm is to find a closed tour starting from any random vertex and visiting each and every point in V with the minimal possible cost of d. It is a known NP-Complete Problem and has key applications in the areas of logistics, planning, and circuit design. Although, many successful algorithms have been proposed for the symmetric TSP problems, but generally those fail to provide adequate results for the asymmetric problems. We have proposed a new and improved approximation algorithm for both symmetric and asymmetric TSP problems. For this purpose 2-opt local search technique has been used with some enhancements to generate better results. We have compared our technique with seven other different algorithms of different types and the proposed algorithm outperforms others in error margin, time and convergence tests. To cater complexity and time issues in bigger TSP problems, we have introduced a graph compression technique to ensure the solution of bigger problems in a timely manner. So here in this improved approximation algorithm, we have achieved our major objective of receiving a solution which is quite near to the optimum with minimal comparative computational complexity. In this thesis, we have used a well known library TSPLIB, and compared our results with a ruin and recreate algorithm for time, error and convergence. We have also compared the results of our algorithm with six other well known algorithms used in this field which include the nearest neighbor, genetic algorithm, simulated annealing, tabu search, ant colony and tree physiology optimization. The proposed algorithm clearly outperforms other algorithms for multiple parameters.

Dedicated to my teachers, parents and beloved wife,
Who supported me throughout this project.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Fakhar uddin**

Signature: _____

Acknowledgment

First of all, I would like to thank my research supervisor, Dr. Naveed Riaz for introducing me to the exciting field of algorithms and Dr. Abdul Hannan and Dr. Imran mahmood who provided valuable feedback for improving the proposed algorithm and methodology. I am also very grateful to all my other committee members for their consistent encouragement during the completion of this thesis.

Most importantly, none of this could have been possible without the support of my family. This research thesis stands as a testament to their unconditional love and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The traveling salesman problem is the most researched problems in combinatorial optimization domain [3] due to its importance and usage in different areas of daily life and applications. Lots of researchers are still pursuing this area who are striving for the optimization of TSP problems with respect to time, error and complexity. This problem is usually described as : a salesman starts his/her journey from an origin city, goes through every other city once and comes back to the origin city with minimal possible distance traveled [4–6].

1.1 Motivation

Finding the optimum solution using exact algorithm is very expensive both in terms of time and resources. Complexity in this case can reach up to O(n!) due to its NP-Hard status. To deal with this problem, researchers have devised different heuristics and approximation techniques for finding a fast and near optimal solution [7].

The effort for finding the optimal result of any NP-Hard problems can grow exponentially huge, thus making the re-

trieval of solution impractical for relatively larger scale problem size [3]. Heuristics can minimize the complexity from exponential to polynomial time by sacrificing some accuracy. Minimizing complexity from exponential size to polynomial size provides the advantage of efficient computability and the schemes can be refined for improving the result by reducing the error factor [4]. Majority of such algorithms only cater for the symmetrical problems while only few take into account the asymmetric problems. In this thesis, we have devised an algorithm which can work on both kinds of graphs for the TSP, providing impressive results in polynomial time. Almost all real world problems that rely on an effective result of TSP problem like network optimization, logistics, postal, or any other industry which involves planning, logistics, can greatly benefit from the provided TSP algorithm [8, 9].

1.2 Problem Statement

We have a complete graph G = (V, E) as an input to the algorithm; here nodes are denoted with V and edges are denoted with E. Every edge (u, v) $\in$ E has a non-negative integer cost denoted with c(u, v) associated with each edge (u, v) $\in$ E. The problem is to find a Hamiltonian cycle (tour) of G with minimum cost. The approximation algorithm should solve both types of problems i.e. symmetric and asymmetric with a minimal error margin. It should effectively solve large input problems in reasonable amount of time.

1.3 Solution

An Improved approximation algorithm has been devised using 2-opt technique for TSP problem. Our proposed algorithm uti-

lizes edge swapping technique mixed with some carefully studied steps to generate near optimal results and solve problems of both symmetric and asymmetric instances. Moreover, a graph compression technique has been proposed and utilized to solve huge problems in a limited amount of time without compromising the accuracy.

Here are the steps for the proposed algorithms, every step is further described in detail.

- Compute a basic solution of the input problem.
- Compute active edge matrix using graph compression technique.
- Use 2-opt technique for edge swapping.
- Accept the new solution using acceptance criteria.
- Reshuffle the cycle and repeat until convergence or predefined end of iterations.

1.4 Terminologies

1.4.1 Hamiltonian Cycle

Hamiltonian cycle or Hamiltonian circuit is a cycle in a graph or closed loop within a graph that visits each and every node in the graph only once. The minimum weighted Hamiltonian cycle is the TSP solution.

1.4.2 Shortest Path

Let G is a graph and there exists a path from node X to node Y denoted as S_xy then S_xy will be called shortest path if and only if there does not exist any other path from X to Y which is shorter than S_xy.

1.4.3 Complete Graph

A graph G is called a complete graph if every pair of nodes V is connected to each other, or in other words, every node v $\in$ V is connected to each and every other node in the graph. Examples are provided in Fig 1.1 and 1.2

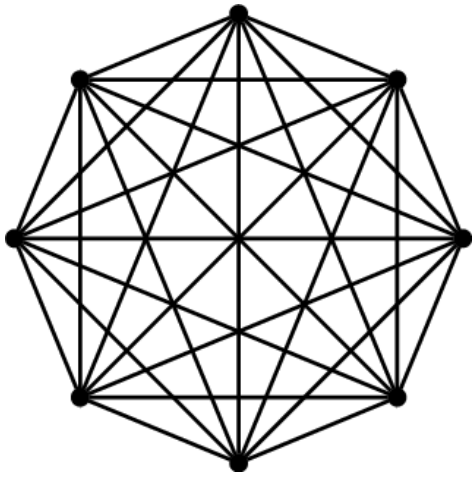

Figure 1.1: 8 nodes undirected complete graph

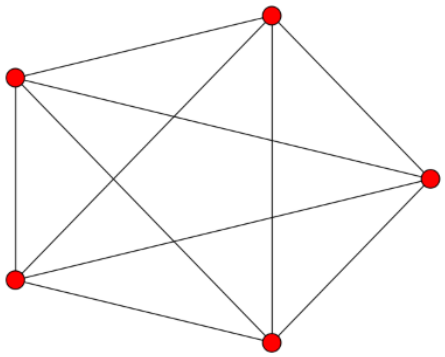

Figure 1.2: 5 nodes undirected complete graph

1.4.4 Asymmetric and Symmetric Graph

In symmetric or un-directed graphs, the distance between any two nodes is always same in either direction. The amount of potential solutions of the problem becomes half due to this symmetry. This is not the case for asymmetric graphs where there are many more possible solution candidates and routes that makes solving TSP more tricky. Traffic clashes, various ways airfare travels and one sided streets are instances of the asymmetrical TSP [10]. Examples of symmetric graphs are provided in Fig 1.3, 1.4 while those of asymmetric graphs are provided in Fig 1.5, and 1.6.

	A_1	A_2	A_3	A_4	A_5
1	0	5	4	6	2
2	5	0	3	5	20
3	4	3	0	9	8
4	6	5	9	0	7
5	2	2	8	7	0

Figure 1.3: Matrix for symmetric complete graph

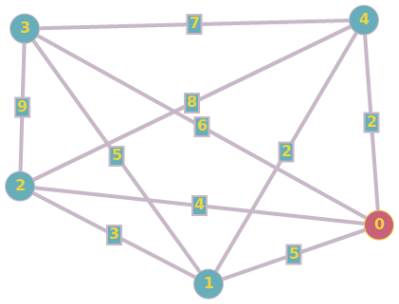

Figure 1.4: Symmetric graph

	A_1	A_2	A_3	A_4	A_5
1	0	5	4	6	2
2	6	0	3	5	20
3	3	9	0	9	8
4	5	4	7	0	7
5	4	1	3	8	0

Figure 1.5: Matrix for asymmetric complete graph

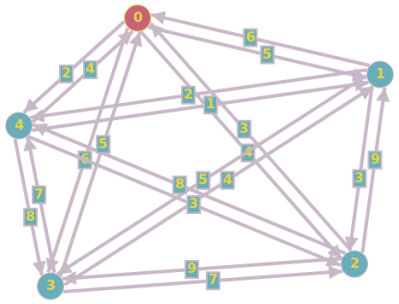

Figure 1.6: Asymmetric graph

1.5 Other Associated problems

There are many other problems associated with the traveling salesman problem or which are the generalized form of TSP [9]. Bottleneck traveling salesman problem is another instance of TSP. Here, the high mass edge of the graph searches the massive graph for Hamiltonian cycle. E.g. the case of large buses for which small streets are not feasible. Another related problem is the Traveling politician problem in which a politician is required to visit each and every city as the voters can visit the nearby city for listening to speech of politician. In this case the pairwise distance between the cities becomes very important. Another generalization is the Vehicle routing problem [11] in which there is a fleet of vehicles and a set of customers that are to be

served in optimal routes. In the Traveling purchaser problem, a buyer has a list of articles to buy and a number of marketplaces to visit and the price of the articles at each marketplace. The objective is to minimize the combined cost of traveling and articles. Other problems include Printing press scheduling problem, School bus routing problem, Crew scheduling problem and Interview scheduling problem.

1.6 Conventional Approaches

Following are the conventional approaches for solving the NP-hard problems like TSP:

- Finding algorithms that always guarantee the optimal solution. Usually the resource demands and computational complexity of such approaches is impractical.
- Finding algorithms that do not guarantee to provide the optimal results but strive for finding solutions that are near optimal. Generally the computational complexity of such techniques is polynomial at worst.

1.7 Exact Algorithms

As the name suggests, exact algorithms produce the optimal result for TSP problem by considering each and every unique solution. An example is the brute force technique that considers every possible combination or comparison resulting in an exponential growth with respect to input size, the solution will become impossible in bigger problems due to its exponential growth nature. Finding the solution becomes unfeasible even for a small input size of 20 as the running time can go up to factorial of the number of cities. There are two types of methods used for exact algorithms: one method utilizes schemes like

branch and bound, interior point, cutting plane and branch and cut. Second method used dynamic programming to solve the problem. A famous example is that of Concorde algorithm that falls under the exact solvers' category which uses a fusion of branch and bound and cutting planes to avoid dead comparisons [10, 12, 13].

- The algorithm utilizes branch and bound technique if the number of cities are between 40 to 60 .
- In case the problem size is over 200, it utilizes evocative method of straight programs using the progressive improvement algorithms.
- It also provides a choice for selection of system for problems with large distances and cities. They have utilized branch-and-bound and problem-specific cut generation for solving a big example consisting of 85,900 cities. [14].

1.8 Non Exact Algorithms

As the name suggests, non-exact algorithms do not always provide the optimal solution but strive to provide a solution that is still usable in practical settings with acceptable and practical computational cost [10]. In other words, there is a trade-off between time, resources and optimal value.

1.8.1 Approximation or Heuristic algorithms

A variety of heuristics and approximation algorithms have been proposed by the theoretical computer science community. It is generally accepted that a near optimal solution that does not deviate from the optimal solution by more than 2 or 3 percent is an acceptable scenario for practical problem sizes. Christofides

is considered a suitable method [15, 16] that has a proven approximation ratio of 3/2 but the method provides near-optimal solution for Metric TSP problem only i.e. the input graph must fulfill the triangular inequality constraints. Other well-known methods include nearest neighbor, Clarke-Wright and further improvement like tabu search, simulated annealing, genetic algorithm, and ant colony optimization. All these algorithms are discussed in detail in chapter 2.

1.9 Thesis Organization

Introduction of the problem, motivation, problem statement and some solution techniques are discussed in first chapter. Famous algorithms like christofieds, nearest neighbor, simulated annealing, tabu search, genetic algorithm and ant colony optimization with some brief problem history are presented in literature review section. Chapter 3 comprises of the proposed methodology, it discusses in detail the proposed algorithm with some complexity analysis. Chapter 4 provides the details of the well-known TSPLIB data set that has been extensively used in the empirical evaluation of our proposed algorithm. We have provided the details of a comparative analysis of our proposed technique with some other well-known and state of the art TSP algorithms in chapter 5. The results are provided in tabular form and are further discussed with the help of some diagrams. Finally in the last chapter, conclusions are drawn and some future directions are identified.

Chapter 2

Literature Review

2.1 Background

The TSP problem made its first appearance in mathematical literature in a 1757 paper by the Leonard Euler. The paper basically looks for an answer to the knights excursion hassle problem in chess, i.e. locating a sequence of knights moves starting from a given square on the chessboard, visiting all different squares precisely once and then returning back to the starting square [8].

Mathematicians W.R. Hamilton, and Thomas Kirkman provided solutions to problems that were pretty similar to the TSP problem in 1800s. A major milestone was achieved with the introduction of Hamiltons Icosian Game by Hamilton, although, the physical game developed on this scheme failed to generate any interest from the general public. The concept was based on the Hamiltonian cycle problem and was a contribution to the graph theory [17].

The first formal description of the problem is associated with Karl Menger. He provided the brute-force version of the algorithm and also highlighted that the nearest neighbor technique does not provide the optimal result for the problem [18].

In 1930 Merrill M. Flood, who was actually looking for the

solution of school bus scheduling, incidentally proposed his idea of solving the TSP problem by doing mathematical calculations. Eventually, the term and name "Traveling Salesperson" was first time formally used by Hassler Whitney in 1959 [19].

In the 1950s and 1960s, the problem gained much wider publicity in the scientific community to the point that RAND Corporation started distributing prizes for step by step solution of the problem. Further notable contributions were made by George Dantzig, Delbert Ray. Fulkerson and Selmer M. Johnson, who were part of the RAND Corporation team and used the cutting plan system technique for finding the solution. This solution proved to be effective with the inclusion of few refinements and variations proposed by Dantzig, Fulkerson and Johnson [18]. They were able to solve an example of 49 cities with 26 different routes. Dantzig, Fulkerson and Johnson were perhaps the first notable researchers who deployed branch and bound algorithm system for the very first time alongwith the cutting plan framework.

Meanwhile, TSP problem also generated considerable interest in other fields of research as well like chemistry, physics, mathematics and other sciences.

In 1962, the problem gained further public attention after a contest which was managed by the company Procter & Gamble in USA to optimally solve a TSP problem for 33 cities. The company offered a lucrative prize to the winners.

In 1972, Richard Karp showed the NP-Completeness of TSP problem by reducing from the Hamiltonian cycle problem.

In 70s and 80s, mathematicians like Grtschel, Padberg, Rinaldi used cutting planes and branch and bound methods for solution estimation of up to 2,392 cities.

In 1991 a library named TSPLIB [20] was formed and published by the Reinelt which is composed of many tsp problems studied over the course of 50 years.

Figure 2.1: Contest to solve 33 cities problem.

In March 2005, Concorde TSP Solver was proposed that provided a TSP solution for 33,810 points in a circuit board. A tour of length 66,048,945 units was found and it was demonstrated that no shorter tour exists. The calculation took roughly 15.7 CPU-years [18].

Further improvements were made in April 2006 when an example of 85,900 points was solved using Concorde TSP Solver, taking more than 136 CPU-years [8]. Detailed history of the TSP is discussed in [10, 19].

TSP serves as a standard for some general heuristics formulated for combinatorial enhancement, e.g. hereditary calculations, recreated toughening, tabu inquiry, subterranean insect settlement streamlining, waterway development elements, and cross-entropy techniques etc.

2.2 Christofides algorithm

In 1976 an algorithm was proposed that was guaranteed to provide a solution within 3/2 factor of the optimal solution. The Christofides algorithm is considered one of the best algorithms due to its ease of understanding, computational complexity, and approximation ratio. This algorithm proposed by Christofides combines the minimum spanning tree with a solution of minimum-weight perfect matching [16, 21].

First, the minimum spanning tree is calculated and then added with minimum weight matching, computed on the vertices of odd degrees. Next, a Euler cycle is created from the combined graph. This Euler cycle is then traversed with shortcuts to neglect already visited nodes.

In 2011, a more improved version of christofides algorithm was proposed for k-depot TSP, which shows a closer approximation of (2- 1/k). If the value of k approaches 2, the approximation bound becomes close to 3/2 (i.e. The original approximation of christofides).

2.3 2-Opt and 3-Opt

Optimizing the problem using smaller moves is also a very popular technique that has yielded promising results. 2-opt and 3-opt algorithms are a branches of local search algorithm which are commonly used by theoretical computer science community for the solution of TSP [22]. 2-opt algorithm removes 2 edges from the graph and then reconstructs the graph to complete the cycle. There is always only one possibility for adding 2 edges in the graph for completion of the cycle. If the new tour length is less than the previous one, it is kept otherwise rejected. Each pair of edges is checked in the graph .

On the other hand, 3-opt removes 3 edges from the tour re-

sulting in creation of 3 sub tours and 8 possibilities for adding new edges to complete the cycle again. Like 2-opt, these steps are performed on all the combination of edges. Time complexity in 3-opt is $O(n^3)$ for a single iteration, which is higher than 2-opt algorithm. Fig 2.2. and 2.3 show the moves of 2-opt and 3-opt algorithms.

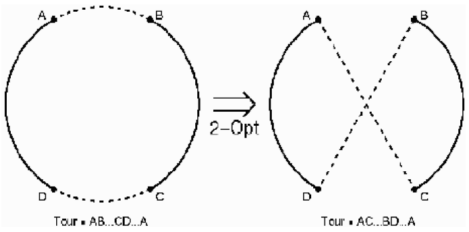

Figure 2.2: 2-opt moves

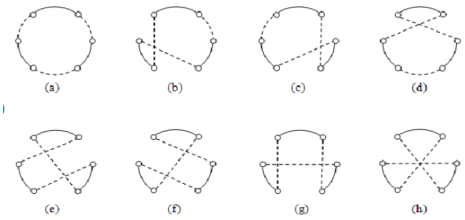

Figure 2.3: 3-opt moves

2.4 Lin-Kernighan

Lin Kernighan is a complex variant of the k-opt algorithm. This algorithm decides at each iteration that which variable k-opt move should be adopted. This variability of choice in each iteration for choosing the value of k in k-opt moves increases the complexity, but generates better results than other k-opt techniques and is considered the most effective technique for generating approximate solutions of TSP. Although it is an approximation algorithm, yet it produces optimum results for majority of the cases. The detail of this technique is provided in [5].

2.5 Branch and Bound

This technique is one of the most widely used technique for solving discrete optimization problems. The basic concept is to split up the viable set into progressive small sub-sets, estimate boundaries on the objective function value across each sub-set,

and then use that objective function to filter out certain subsets in future comparisons. This method is easily applicable on both symmetric and asymmetric problems. A method is studied in paper [23] using branch and bound technique for asymmetric problems.

In branch and bound method, a solution is devised and cost is recorded in upper bound α after traversing a subset. All the branches which exceed the pre-selected bound are never considered. If a new solution is achieved with a better value than the previous one, then α is updated with the new value. The best solution found at the end of this procedure is considered the global optimum. The method can become inefficient in large problems if the sub tree expands too large. To overcome this problem, it can be used in conjunction with some other techniques like minimum spanning tree relaxation for limited branching. Some variants of branch and bound methods are studied in detail in the paper [13].

2.6 Nearest Neighbor

The Nearest Neighbor algorithm (NN) is a straightforward, greedy approximation algorithm. Tour starts with the selection of a random initial city and then incrementally adding the closest unvisited city till all cities are visited.

The steps are simple

- Let G = (V, E) be a complete graph where V are nodes and E are edges.
- Select a random node v $\in$ V and mark it visited.
- Now check for the shortest edge e $\in$ E from this node to the next node.
- Select this next node and mark it as visited.

- Repeat these steps until all the nodes in V are marked as selected or visited.

Although this algorithm is computationally very efficient but generally fails to provide effective results. In paper [7], its empirical results are compared with 5 other algorithms utilizing TSPLIB problems. In this thesis, we have also compared the results of our proposed algorithm with nearest neighbor algorithm results. The comparison highlights that nearest neighbor algorithm is a poor choice for the TSP approximation.

2.7 Simulated Annealing

In 1983, Kirk patrick, Gelatt and Vecchi introduced a powerful heuristic algorithm known as simulated annealing. SA is a heuristic algorithm based on trajectories inspired by the metal annealing method which slowly freezes into a solid state. SA is a probabilistic algorithm to find the global optimum of the problem from the pool of local optimums. Simulated annealing decides on each step, the probability of keeping the current solution or moving to the next possible one. The probability increases or decreases with the quality of the move. A parameter T is used to measure the probability of the move. When T tends to zero, the probability of selection becomes more unlikely.

$$P(acceptance) \sim 1 - exp(\Delta E/CT))$$

Here C is a constant related to energy or temperature, and the value of T is a control parameter and set very high initially. Simulated annealing allows some bad moves to traverse through the big solution space. The acceptance of the new state is also based on some predefined criteria. This process is repeated until the convergence to the solution [3]. In [24] authors proved that

threshold acceptance is better than simulated annealing.

2.8 Genetic Algorithm

Genetic algorithm has also garnered considerable attention as the candidate for the solutions of various optimization problems. They basically represent a separate class of algorithms that deal with problems of complex nature. Genetic algorithms have practical use and importance in various other fields as well as our social life. Unlike other meta-heuristic methods, GA use natural selection rules, crossover and mutations to make the computation easier and faster. These aspects make it more valuable, better performer, and more efficient algorithm than others [25–27].

The genetic algorithm is inspired by the genetic operators of evolution, i.e. selection, crossover and mutation. GA has been extensively used in literature for TSP and related problems. Mutation is the most effective operator driving the search for a better solution. Swapping, flipping and sliding are the main types of mutations used in GA. The idea behind the genetic algorithm comes from genes where the offspring is created by exchanging the genes of their parents. Full life cycle of GA is described in Fig 2.4.

2.9 Tabu Search

TS was proposed by Fred Glover in 1986 and is also known as an algorithm for neighborhood search. Here the search method is primarily based on memory of search history, denoted as tabu listing. It is an intensive local search algorithm [28]. Tabu search avoids the problem of getting stuck in local optima by allowing moves with negative gains and constructing a tabu list

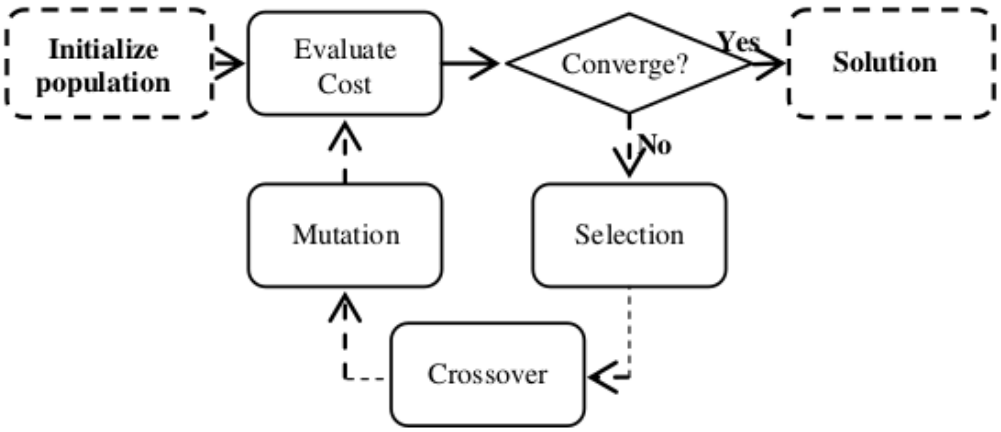

Figure 2.4: A Genetic Algorithm for solving TSP.

to disallow contradictory moves. Whenever it gets stuck in local optima, it searches a solution from the neighborhood stored in memory even if it is worse than the current selected one(negative gain), thus allowing it to discover more feasible options from the solution space. Here a tabu list helps TS to avoid cycling in the tour. Tabu search uses 2-opt moves to enhance the solution. The running time of this algorithm is $O(n^3)$, which makes is slower than other 2-opt local search algorithms.

2.10 Neural network

Simple and yet challenging optimization problems like TSP has inspired the researchers to use new and improved techniques in this domain. Neural network is yet another tool that has been utilized by the researchers seeking better solutions for TSP. Yet, results from this field are not comparable to other heuristics. However, the neural network field is still growing and many improvements are under way. Neural networks is a powerful computation technique which provides parallel computation and reduces a huge amount of time using parallel technology. Hence, after maturity this technology can provide the solution for the

optimization problems at a speed that surpasses the other well-known techniques.

In their work, Rong Li and Junfei Qiao [17] advocated for the use of a modified Hopfield neural network. In this paper two experimental studies were carried out. For experimentation, they selected 10 and 19 cities for modified and traditional algorithms respectively. They proved that the modified Hopfield neural network is more efficient in working as compared to the traditional Hopfield neural network. Morever, they further justified that the modified Hopfield neural network technique was practice able. Other optimization issues can also be solved by the modified Hopfield neural network according to the analogy.

2.11 Ant Colony Optimization

It has become a very common practice to solve complex problems with the help of natural phenomena. Solving TSP with ant behavior, genetic algorithm or tree Physiology are some examples of this practice. Machine learning scientist Marco Dorigo outlined in 1993 a strategy for heuristically solving TSP by deploying a technique of recreation of a subterranean insect province called ACS (subterranean insect settlement framework). It used the analogy that how genuine ants discover short ways between sustainable sources and their home. Ants behavior is depicted in the Fig 2.5

As ants are blind, so they start navigation toward the food source from their colony and deposit the pheromones on their pathway. Every ant searches and follows the path at random. The probability of following a path increases with the increase of pheromones in that path. The ants investigate, saving pheromone on each edge that they cross, until they have all finished a visit. The algorithm uses artificial ant behavior which records their

location and quality of the solution so that this path can be checked for acceptance or rejection in future iterations. The measure of pheromones storage corresponds to the visit length: the shorter the visit, the more it stores [1, 2].

In their work, Leila Eskandari, Ahmad Jafarian [29] argue that ACO is one of the efficient nature motivated meta-heuristic algorithm which has outperformed a considerable set of algorithms in this domain. They have modified and improved the ACO algorithm to devise another strategy for solving TSP problem. Basically they compare both local and global solutions for finding the best possible solution.

In paper [1] authors used tabu listing to avoid repetition of path selection in ant colony optimization. Results from the study show that tabu listing used with ACO considerably improves the overall algorithm time and convergence.

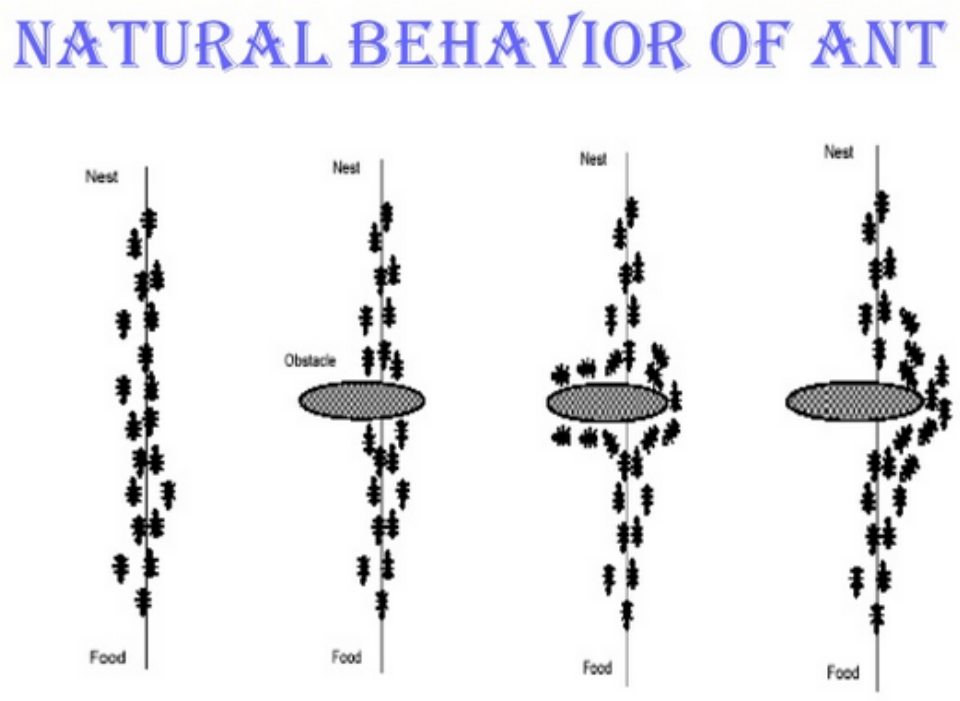

Figure 2.5: Ants behaviour shows they choose shortest path. [1, 2]

2.12 Tree Physiology Optimization

An important property of nature is sustainability and continuous improvements for survival. This unique property reflects the pattern of optimization. There are many meta-heuristic algorithms inspired by the nature, which are producing good results with powerful techniques. Tree physiology optimization is also one of the nature inspired algorithm used to solve optimization problems.

The algorithm is influenced by a tree development scheme that uses the shoot and root feature to achieve optimal survival. The shooting system expands to a light source in ordinary plant growth to capture light and initiate the photosynthesis process. The method of photosynthesis transforms light into carbon with the assistance of water, which is then provided and used by other components of the plant, particularly, the root system uses oxygen to elongate shooting in the opposite direction. It consumes carbon to further elongate inside the floor for water and nutrient searching, which is then provided for shooting extension system. The shoot-root system's connection to ideal development can be converted by a straightforward concept into an optimization algorithm, shoot searches for carbon using root nutrient, and root searches for nutrient using shooting system. In paper [7], TPO results are compared with 5 other algorithms.

2.13 Ruin and recreate

R&R is a simple but powerful meta heuristic to solve combinatorial optimization problems. The ruin and Recreate (R&R) method uses the concepts of simulated annealing or threshold acceptance with massive actions in place of smaller ones. As the name suggests, a big chunk of the problem is ruined and recreated. Complex problems like timetable scheduling or vehicle

routing problems, which are often discontinuous, require large moves to bypass the local optima. R&R algorithm has proved to be an important candidate for finding global optimum and parallelism.

Vehicle routing problem using R&R is discussed in detail in paper [11]. There is a fleet of vehicle which has to serve different number of customers. There is a central depot where the route of each vehicle starts and ends. Vehicles need to serve the customers that have certain demands. The idea is to serve the customers with a minimum route (distance traveled) and within the capacity of every vehicle assigned to it. A time window constraint can also be added to the problem, i.e; every customer can add start and end time to its service. A total of 56 problems have been studied in this paper and results are discussed.

Jsprit is an implementation of the algorithm R&R. We have used that implementation to benchmark the problems from TSPLIB and compared those results with our proposed algorithm. In this study, R&R has performed considerably well in terms of error margin and convergence in some problems.

Chapter 3

Proposed Methodology

Graphs are important type of data structures with many applications. Traveling salesman problem (TSP) is one of the most important graph theory problem that belong to NP complete class of problems [17]. Finding optimal solution for such problems in real time simulation is currently not possible. Therefore one may opt for approximation solutions for these problems. This domain is a hotbed for researchers for past few decades now, and researchers are actively seeking for effective solutions to such problems that can be computed in polynomial time. Research community is consistently looking for novel techniques and algorithms that can outperform the previous well known algorithms. TSP problems can be classified into symmetric and asymmetric TSP problems [10], which are already discussed in chapter 1. Majority of the proposed algorithms and techniques solving TSP mainly focus on either symmetric or asymmetric TSP problems. In this work a new simpler and computationally cheap algorithm is proposed. The algorithm effectively solves both type of TSP problems. One of the key aspect of the proposed algorithm is the introduction of a new technique for graph compression, which reduces computational complexity considerably.

3.1 Proposed Solution

3.1.1 Basis for Solution

The first step in our algorithm is the generation of the Hamiltonian cycle. The Hamiltonian cycle problem works as the basis of our algorithm. Initially, the algorithm requires to be provided with a Hamiltonian cycle as a base input to this algorithm. A random Hamiltonian cycle is quickly computed by using the greedy technique (least cost edge) and then edge swap technique is used to iteratively improve upon the original tour. It is pertinent to mention that although we have computed Hamiltonian cycle in a greedy manner for this work, any method can be utilized to compute the Hamiltonian cycle as the effectiveness of our proposed algorithm would greatly exceed in cases where the initial Hamiltonian cycle is of higher quality in terms of total edge weight. However, it is not entirely dependent on such optimistic scenarios only as the proposed edge swap technique along with graph compression are also keys for proposed algorithm's success.

3.1.2 Graph Compression

A Hamiltonian cycle H is a complete cycle from a graph G =(V,E) where every node is visited only once, here V denotes the nodes in the graph and E denotes the edges.

Formally, Let G = (V, E) Where $\|V\| = n$, n denotes the total number of nodes. A greedy Hamiltonian circuit H is obtained by starting from any random node $v_i \in V$ and then traversing the next node with minimum edge $e_i \in E$. When each and every node has been traversed, a new edge will be added in the graph from the least cost edge is selected from the last node to the initial node to complete the cycle.

The time complexity for finding the Hamiltonian cycle depends on the type of algorithm used. There are many greedy solutions available which guarantee O(n) time complexity. The TSP graph can be expressed in the form of cost matrix. Initially full cost matrix is taken. For smaller problem cost matrix is small, but with increase in matrix size the space cost increases quadratically. Therefore, we modify the cost matrix and use an active edge matrix which only have 100 edges for each node. The complexity of this step is

$$\text{Time complexity} = O(n + k.\log(k)) = O(n)$$

Where n denotes the total number of nodes, k denotes the nodes which are needed to be activated for that particular node. So if we select 100 active nodes that means k = 100.

For each graph, a n * n matrix is initialized with 0s, where n is the number of nodes in the graph. Nearest k points would be identified and those points will be considered active for that node $v_i \in V$ and would be marked 1 in the active edge matrix.

3.1.3 Shuffling

Next step is to shuffle the original Hamiltonian cycle generated in the first step. The algorithm selects a group of three nodes from the original cycle and then processes those in clockwise direction in two steps. We would like to highlight that although it is possible to select variable number of nodes and steps, we have selected this combination of 3 nodes and 2 steps after extensive evaluation with different combination of number of nodes and steps. After this shuffling step, the algorithm mutates and accepts the new Hamiltonian cycle if it provides some gain over the previous solution. Shuffling is an iterative process that is repeated a fixed number of times.

Time complexity of shuffling is O(n/l), where l is the number of sub elements to shift clockwise or anticlockwise.

3.1.4 Mutation

Mutation is an integral part of our proposed algorithm. Mutation refers to the on-going modifications to the current solution. Different techniques have been used for mutation in different algorithms and Lin-kernighan is a famous one which removes two, three, four or five connections from the graph and then selects a new solution from 2, 4, 25 or 208 different possible scenarios [5]. Here in this thesis we have applied a simple mutation; we remove two edges from the complete Hamiltonian cycle H and revert them and reconnect them again to create a new solution. This is the same technique used in lin-2-opt method [22].

Suppose we have two edges AD and EB, shown in the Fig 3.1. We will remove these two edges from the graph and add two more edges as AB and ED to complete the graph.

The time complexity of this step would be constant as only two edges are to be removed and two new edges have to be added.

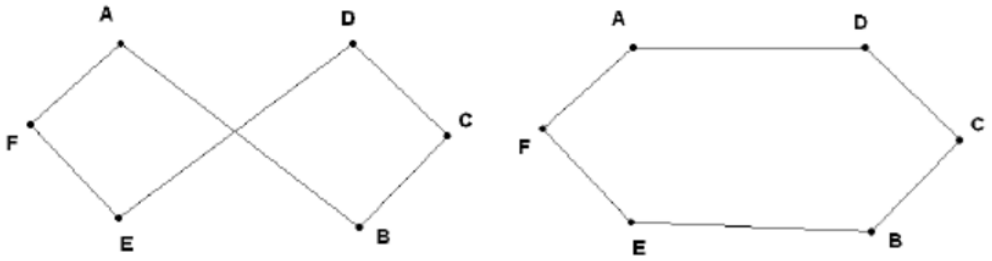

Figure 3.1: Reverting an edge in 2-opt algorithm

3.1.5 Gain Computation

The total weight of this newly computed graph will be calculated again and if the new sum of weight is lesser than the previous one, then distance is minimized from the previous value, new edges will be accepted, otherwise rejected. This step will be repeated for every edge in the graph of respective node.
Let x be the distance from node A to node D from the Fig 3.1 and $\overline{x}$ be the distance from node E to node B. In case of symmetric graph, distance from A to D and D to A will be identical. Let y be the distance from node A to node B and $\overline{y}$ be the distance from node E to node D. Then we will have gain as

$$\text{Gain} = R_p \text{ - } R_n$$

Here R_p is the sum of edge distance which is to be removed i.e. $R_p = x + \overline{x}$, while R_n be the sum of edge distance which is to be added in newly graph i.e. $R_n = y + \overline{y}$.

We would like to highlight that in the case of symmetric graphs, we need to calculate gain from two edges that have been replaced in the graph, while the rest of the graph remains the same. However, in the case of asymmetric graphs, whole graph would be computed again, as reverted edges can have different values. So, the whole graph will get changed when two edges are reverted and added in the cycle. Time complexity in the case of symmetric graph will remain constant while in case of asymmetric graph it will be O(n).

3.1.6 Selection Criteria

Opting for more number of steps and removing more than two edges will add more complexity to the problem, that's why we have kept smaller number of steps and have tried to gradually improve the solution in this algorithm. Reverting the edges

and then testing for improved solution is a simple yet powerful method. But, there is an overhead in this solution. As we are checking for every suitable solution and then adapting it in a greedily manner raises the possibility of missing a better solution. This point can be improved in the future by defining some constraints for the possible acceptance of a solution e.g. simulated annealing accepts every better solution with a certain probability measure [3], and in the case of threshold acceptance, results are accepted if they satisfy certain threshold values [24].

If gain in the previous step is greater than 0, we will keep this new solution. The time complexity of this step is constant.

3.2 Complexity Analysis

The time complexity of the proposed algorithm is O(n) in first step. In second step, it is O(n) for graph compression or active edge matrix. In the next steps we have

$$\mathrm{T} = \mathrm{O(n)} + \mathrm{O(n)} + \mathrm{C.O(n)} * \mathrm{T.O(n.k)} = \mathrm{O}(n^2)$$

Where C * T corresponds to the number of iterations in the algorithm. A loop has been used for shuffling and is denoted here with C. Another loop has been used for mutations and swapping, computing gain and accepting the new Hamiltonian cycle, which is denoted here by T. k is the number of active edges used in graph compression. It is evident that complexity of the proposed algorithm is polynomial.

Algorithm 3.1 An Improved Approximation Algorithm for Symmetric & Asymmetric TSP Problems

Input: A file containing three columns, row number, x point, y point
Output: Best tour, Error percentage, time, and complete graph
1- Select a random Hamiltonian cycle *H*, constructed from a directed sequence $S = p_1,p_2,p_3...p_n$, such that p_n is connected to p_1. Initially *S* can be selected using a low cost algorithm e.g, greedy algorithm.
1.1- Select AEM matrix through graph compression technique.
2- Initialize set A = S
3- Select a point $p_q \in S$
4- Select three points p_s, p_t, $p_r \in S$, such that p_s is the point next to p_q, and p_r is a point next to p_t, and none of p_q, p_s, p_t and p_r are the same.
5- check if p_s is an active edge for p_q, then break this edge to form a path $p_1 = p_r,,p_s$
6- Break second edge (p_r, p_t) of H to form second path $p_2 = p_t,,p_q$
7- Reconnect p_1 with inverted p_2 sequence to form a new Hamiltonian cycle I = p_r ,p_s, p_q, ,p_t, I has two new edges (p_s,p_t) and (p_q,p_r). The original p_2 and inverted p_2 does not differ in length for an undirected (symmetric) graph.
8- Find the difference $\Delta d = (d_{sq} + d_{tr})$ –$(d_{st} + d_{qr})$, where d_{sq} is the distance between point p_s and p_q. Here (p_s,p_q) and (p_t,p_r) are discarded edges and (p_s, p_t) and (p_q, p_r) are new edges.
9- If the $\Delta d > 0$ replace H with I.
10- Repeat steps 4 to 7 until all the $p_t \in S$ are selected, where p_t and p_q are different.
11- Remove current p_q from A and select a new $p_q \in A$.
12- Repeat step 3 to step 11 until $A = \phi$
13- Repeat step 2 to step 12 for K iterations to achieve convergence.

Chapter 4

Materials

Only a limited number of test problems are publicly available to benchmark the well known combinatorial optimization problems like TSP. TSPLIB provides a nice collection of examples that have been recorded over the course of more than 50 years for this purpose. In this research work, we have also used this test suite for benchmarking the results [20]. The examples in the library are of following natures.

- Problems related to Symmetric TSP (TSP).
- Problems related to Asymmetric TSP (ATSP).
- Problems related to Hamiltonian cycle (HCP).
- Problems related to Sequential ordering (SOP).
- Problems related to Capacitated vehicle routing (CVRP).

4.1 TSPLIB

This library contains multiple problems of different categories ranging from 10 nodes to 85900 nodes. File format and types are explained below.

4.1.1 File Format

Each file has two parts, specification part and a data part. Information about content is provided in specifications part in the form of keyword value pair, while explicit data is present in the data part. Here we provide some in depth intrinsic detail of a TSPLIB file.

4.1.2 Type: Describes the type of the data

In this thesis work, we have only considered the symmetric and asymmetric problems. Below tables describe data types in the TSPLIB library 4.1.

Table 4.1: Types of data presented in TSPLIB library.

Type	**Description**
TSP	Symmetric TSP data
SOP	Sequential ordering problem data
ATSP	Asymmetric TSP data
HCP	Data for Hamiltonian cycle problem
TOUR	A collection of tours
CVRP	Data for Capacitated vehicle routing problem

4.1.3 Edge Weight Type: Describes the edge weights in the file

For symmetric problems only EUC_2D and CEIL_2D weights are considered and benchmarked, while for asymmetric problems the complete matrix is considered as input. Edge weight types are defined in table 4.2.

Table 4.2: Types of edge weights presented in TSPLIB library file.

Type	Description
FUNCTION	A function describes the weights
FULL MATRIX	Full matrix with distances
UPPER ROW	Row wise upper triangle, no diagonal entries
LOWER ROW	Row wise lower triangle, no diagonal entries
UPPER DIAG ROW	Upper row with diagonal entries
LOWER DIAG ROW	Lower row with diagonal entries
UPPER COL	Column wise upper triangle, no diagonal entries
LOWER COL	Column wise lower triangle, no diagonal entries
UPPER DIAG COL	Upper column with diagonal entries
LOWER DIAG COL	Lower column with diagonal entries

4.1.4 Distance Formula

As discussed already, we have used EUC_2D and CEIL_2D formats in our algorithm, and in this section we have described the formula of calculating the distance from one node to the other. A floating point coordinates are provided in each file for each node. e.g. Let

$$X_i, Y_i$$

denotes the two coordinates X and Y of a node i. Then in case of EUC_2D the distance between two points i and j would is calculated as

$$X_d = X_i - X_j$$

$$Y_d = Y_i - Y_j$$

$$D_ij = \sqrt{X_d^2 + Y_d^2}$$

The edge weight type CEIL_2D requires that the two-dimensional Euclidean distances are rounded up to the next integer value.

4.2 Symmetric problems

We have selected 78 different problems for symmetric TSP from the library in EUC_2D and CEIL_2D formats. Selected problems range from 50 nodes to 85900 nodes. We have divided them into 3 main categories i.e. small, medium and large. In small category, we have selected 42 problems ranging from 50 to 500 nodes. In the medium category, we have selected 28 problems ranging from 500 to 5000 nodes and in the large category, we have selected 8 problems which have a range of 5000 to 85900 number of nodes.

Small category problems are listed in table 4.3. Medium category problems are listed in table 4.4, while large problems are listed in the table 4.5. The name of the problems, number of nodes and optimum tour values are provided in these tables that have been used for comparison with different algorithms in the comparison section.

4.2.1 Input file format

For all problems belonging to symmetric category, the input file is a text file. The first row contains the optimal value with origin (0,0). Each file has 3 columns; first column denotes the serial number, second column the x-axis value and the third column provides the y-axis value. For example, if we have a problem of 50 nodes, then the text file will have 51 rows, where first row will have optimum tour value and 0,0 values for x and y-axis, while another 50 rows will have node number, x-axis value and y-axis value respectively.

Table 4.3: TSP Small Symmetric Problems having 50-500 Nodes.

#	Symmetric	Opt.	# of Nodes	#	Symmetric	Opt.	# of Nodes
1	a280	2579	280	**23**	lin318	42029	318
2	berlin52	7542	52	**24**	u159	42080	159
3	bier127	118282	127	**25**	pcb442	50778	442
4	ch130	6110	130	**26**	pr107	44303	107
5	ch150	6528	150	**27**	pr124	59030	124
6	d198	15780	198	**28**	pr136	96772	136
7	d493	35002	493	**29**	pr144	58537	144
8	eil101	629	101	**30**	pr152	73682	152
9	eil51	426	51	**31**	pr226	80369	226
10	eil76	538	76	**32**	pr264	49135	264
11	fl417	11861	417	**33**	pr299	48191	299
12	gil262	2378	262	**34**	pr439	107217	439
13	kroa100	21282	100	**35**	pr76	108159	76
14	kroa150	26524	150	**36**	rat195	2323	195
15	kroa200	29368	200	**37**	rat99	1211	99
16	krob100	22141	100	**38**	rd100	7910	100
17	krob150	26130	150	**39**	rd400	15281	400
18	krob200	29437	200	**40**	st70	675	70
19	kroc100	20749	100	**41**	ts225	126643	225
20	krod100	21294	100	**42**	tsp225	3916	225
21	kroe100	22068	100				
22	lin105	14379	105				

Table 4.4: TSP Medium Symmetric Problems having 500-5000 Nodes

#	Symmetric	Opt.	# of Nodes	#	Symmetric	Opt.	# of Nodes
1	p654	34643	654	**16**	u1432	152970	1432
2	d1291	50801	1291	**17**	u1817	57201	1817
3	d1655	62128	1655	**18**	u574	36905	574
4	d657	48912	657	**19**	u724	41910	724
5	fl1400	20127	1400	**20**	vm1084	239297	1084
6	fl1577	22249	1577	**21**	vm1748	336556	1748
7	nrw1379	56638	1379	**22**	fl3795	28772	3795
8	pcb1173	56892	1173	**23**	d2103	80450	2103
9	pr1002	259045	1002	**24**	u2319	234256	2319
10	rat575	6773	575	**25**	fnl4461	182566	4461
11	rat783	8806	783	**26**	pcb3038	137694	3038
12	rl1304	252948	1304	**27**	pr2392	378032	2392
13	rl1323	270199	1323	**28**	u2152	64253	2152
14	rl1889	316536	1889				
15	u1060	224094	1060				

Table 4.5: TSP Big Symmetric Problems having more than 5000 Nodes.

#	Symmetric	Opt.	# of Nodes
1	rl5934	556045	5934
2	rl5915	565530	5915
3	rl11849	923288	11849
4	usa13509	19982859	13509
5	brd14051	469385	14051
6	d18512	645238	18512
7	pla33810	66048945	33810
8	pla85900	142382641	85900

4.3 Asymmetric problems

We have selected 17 problems in the asymmetric category from the library. All these problems are listed below in the table 4.6. The second column provides the name of the problem followed by the optimal value and the total number of nodes in that problem.

Table 4.6: Asymmetric Problems.

#	Asymmetric	Opt.	# of Nodes
1	br17	39	17
2	ft53	6905	53
3	ft70	38673	70
4	ftv33	1286	34
5	ftv35	1473	36
6	ftv38	1530	39
7	ftv44	1613	45
8	ftv47	1776	48
9	ftv55	1608	56
10	ftv64	1839	65
11	ftv70	1950	71
12	ftv170	2755	171
13	kro124	36230	100
14	p43	5620	43
15	rgb323	1326	323
16	rgb358	1163	358
17	ry48	14422	48

4.3.1 Input file format

The input for the asymmetric TSP problem is again a text file consisting of a matrix of complete graphs. Due to asymmetric nature, we can not have only upper diagonal or lower diagonal matrix as input, because distance from point a to b and distance from point b to a are not not necessarily the same in asymmetric problems, so missing matrix cannot be computed at its own.

Chapter 5

Experimental Results & Analysis

In this section, we present the detailed evaluation of our proposed algorithm using multiple resources and also provide a performance comparison with some other well-known algorithms selected from the literature. We have mainly benchmarked TSPLIB examples to test the efficiency of our algorithm and have compared it with the $R\&R$ algorithm, and some other well known approximation algorithms selected from the existing literature. As our algorithm is able to cater both symmetric and Asymmetric problems, we have picked up 78 symmetric and 17 Asymmetric problems from well-known TSPLIB library. The details of the problems that have been used for empirical evaluation are provided in Chapter 4.

In existing literature, a large number of algorithms have been proposed as a possible solution for TSP problem and we wanted to highlight the efficiency and effectiveness of the empirical results received by the deployment of proposed algorithm. We have rigorously evaluated the proposed algorithm and present our findings in this chapter with a detailed comparison with the $R\&R$ and 6 other approximation algorithms.

Jsprit is an implementation of the ruin and recreate algo-

rithm [11]. We have empirically tested the $R\&R$ algorithm by using TSPLIB problems, and the results received are presented in table 5.1 along with the results of proposed algorithm. The comprehensive comparative analysis of the proposed algorithm with existing algorithms i.e. Nearest Neighbor (NN), Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO) and Tree Physiology Optimization (TPO) are provided in the Table 5.2.

All the algorithms have been evaluated for the same TSPLIB problems selected in the initial stage of study. It is pertinent to mention that the original authors of the paper [7] only targeted the symmetrical problems and presented the comparative analysis on the basis of size of the problem and number of iteration run by each algorithm. The problems have been divided into 3 categories depending upon the number of nodes, i.e.; small, medium and large. The results for these comparisons are provided in the Table 5.4, 5.5, 5.6.

5.1 Parameter settings and machine configuration

In proposed algorithm, there are two main variables, niter and kiter. niter is set for shuffling the Hamiltonian cycle after every mutation cycle, while kiter is set for number of mutation cycles. So basically

Total Number of Iterations = niter * kiter

We have divided the benchmark problems into 3 categories from the library i.e. small, medium and large. Small problems are in the range of ($50 < n \leq 500$), medium benchmark problems have a range of ($500 < n \leq 5000$) and large problems ranges (n

$>$ 5000). We have defined 2000 iterations for small category and asymmetric problems, 500 iterations for medium and 50 iterations for problems belonging to big category.
R&R results have also been generated by running the algorithm on 2000, 500, and 50 iterations for small, medium, and big sized problems for the sake of fairnes. The Authors that have published the results for other algorithms i.e. NN, GA, SA, TS, ACO and TPO algorithms have opted for a minimum of 10000 iterations for each algorithm [7]. For convergence graph, every problem has been iterated 10, 100, 500, 1000 and 2000 times and the corresponding result was recorded.
All the empirical results that have been received by running the proposed algorithm and *R&R* on TSPLIB library have been computed using a core i7, 6600U CPU @ 2.60 GHz * 2 machine having 16GB RAM and running 64 bit operating system.

5.2 Experiment with *R&R*

In this section, we have presented the experimental results of the proposed algorithm along with the results of the well-known Ruin and Recreate algorithm implemented as Jsprit using similar settings. Each test case goes through 10, 100, 500, 1000 and 2000 iterations to draw the graph for convergence. We ran both algorithms for fixed number of times for every TSP problem. Moreover, we also recorded the time taken by each algorithm to provide the results for each problem.

In case of proposed algorithm, for small problems comprising of 50 to 500 nodes, we decided to run the algorithm on 2000 iterations and with full active matrix, which means that each and every node is active in the graph. In the Table 5.1, we have provided the results of conducting the experiment on both algorithms with the details of individual TSP problem. The col-

umn optimum, provides the value of optimal result for each TSP problem provided in TSPLIB library. In column Time, we have listed the actual wall clock time consumed by individual algorithm for particular TSP Problem. The Error column provides the actual difference between the received result and the optimal result on the basis of basic formula. The formula is defined as

$$\text{Error} = (\text{ result - opt }) / \text{ opt } * 100$$

In 30 out of 41 selected bench marked problems, belonging to small category, the proposed algorithm performed better than $R\&R$. In 11 problems $R\&R$ outperformed the proposed algorithm. However, a key highlight of our proposed algorithm in comparison to the $R\&R$ is that for the proposed algorithm, the error variance never increased more than 7%, while the $R\&R$ algorithm produced a below par 42.7% error margin for the problem pr107. It is also evident from the results that there are many cases where the error percentage of the results produced by the $R\&R$ is more than 18 percent. Moreover, if we look at the execution time factor as well, proposed algorithm beats the $R\&R$ by a fair margin while still producing notably better results for all the tested problems. It is also worth mentioning that after applying graph compression e.g. for up to 100 nodes, the time taken by proposed algorithm improved even further. It is also evident from the graph 5.2, that $R\&R$ algorithm performs so poorly in terms of computation time for the bigger problems that we had to abort the computation of results for bigger problems for comparison sake. However, we have recorded the results from proposed algorithm for all categories and have presented those in tables 5.4, 5.5, 5.6, for future references.

Table 5.1: Comparison of proposed algorithm with $R\&R$ in time and error

Name	**Opt.**	**R&R**			**Proposed Algorithm**		
		Best	**Error %**	**Time (sec)**	**Best**	**Error %**	**Time (sec)**
berlin52	7542	7,820.80	3.70	5.7	7,544.26	0.03	0.12
bier127	118282	127,550.90	7.84	30.2	119,228.26	0.8	0.58
ch130	6110	6,170.64	0.99	36.3	6,172.32	1.02	0.65
ch150	6528	6,712.87	2.83	52.4	6,683.37	2.38	1.00
d198	15780	15,937.39	1.00	74	16,160.30	2.41	1.85
d493	35002	35,923.41	2.63	258	36,479.08	4.22	14.21
eil101	629	662.89	5.39	15.2	666.93	6.03	0.31
eil51	426	444.42	4.32	2.5	432.35	1.49	0.08
eil76	538	573.28	6.56	6.6	553.01	2.79	0.17
fl417	11861	13,094.38	10.40	207	12,061.45	1.69	8.46
gil262	2378	2,422.93	1.89	108	2,462.66	3.56	2.63
kroa100	21282	21,559.42	1.30	15.8	21,522.49	1.13	0.31
kroa150	26524	26,952.53	1.62	46.8	26,635.40	0.42	0.77
kroa200	29368	30,231.95	2.94	73	30,031.72	2.26	1.39
krob100	22141	22,746.89	2.74	15	22,295.99	0.7	0.33
krob150	26130	26,237.26	0.41	48	26,568.98	1.68	0.76
krob200	29437	29,918.26	1.63	81	30,052.23	2.09	1.71
kroc100	20749	21,154.50	1.95	15	20,815.40	0.32	0.36
krod100	21294	21,842.21	2.57	17.5	21,813.57	2.44	0.31
kroe100	22068	22,413.32	1.56	14.8	22,299.71	1.05	0.34
lin105	14379	14,564.71	1.29	17	15,207.23	5.76	0.46
lin318	42029	42,822.78	1.89	133	44,306.97	5.42	5.76
pcb442	50778	52,910.40	4.20	204	52800	3.98	11.60
pr107	44303	63,228.97	42.72	19	46,066.26	1.08	0.41
pr124	59030	70,108.17	18.77	26	59,667.52	1.24	0.71
pr136	96772	104,690.06	8.18	36	97,971.97	2.25	0.84
pr144	58537	69,368.47	18.50	42	59,854.08	0.46	0.84
pr152	73682	79,805.00	8.31	46	74,020.94	2.11	0.96
pr226	80369	88,804.29	10.50	99	82,064.79	0.82	2.28
pr264	49135	58,608.78	19.28	105	49,537.91	3.78	3.75
pr299	48191	53,950.30	11.95	116	50,012.62	3.27	4.71
pr439	107217	120,717.01	12.59	188	110,723.00	1.83	11.63
pr76	108159	110,467.08	2.13	6.5	110,138.31	0.94	0.23
rat195	2323	2,378.49	2.39	67	2,344.84	6.68	1.80
rat99	1211	1,231.29	1.68	13	1,291.89	1.65	0.40
rd100	7910	8,046.29	1.72	14.8	8,040.52	1.12	0.38
rd400	15281	15,725.07	2.91	177	15,452.15	5.26	10.18
st70	675	693.37	2.72	5.2	710.51	1.96	0.21
ts225	126643	142,681.07	12.66	82.5	129,125.20	1.38	2.28
tsp225	3916	4,371.03	11.62	83.2	3,970.04	3.52	2.27
u159	42080	49,734.95	18.19	47	43,561.22	2.51	0.93

5.3 Performance Comparison

The graph of Fig 5.1 depicts the error margin in the three algorithms, i.e; proposed, NN, and $R\&R$ algorithm. It is evident that the proposed algorithm outperforms the two other algorithms most of the time, and $R\&R$ performed comparatively better than NN. But it is also worth noting that in some cases, the $R\&R$ algorithm yielded results that were even worst than NN plus the variance between error margin from $R\&R$ is also considerably higher. On the other hand, the proposed algorithm produced better and consistent results and never exceeded the acceptable margin of 7% for problems belonging to small category.
Fig 5.2 provides a graphical comparison of time consumed by the $R\&R$ and proposed algorithm. The results advocate for the supremacy of our proposed algorithm in terms of computational time. It is pertinent to mention that the computational time consumed by the proposed algorithm is impressive even for the considerably larger problems. The computational time performance improved further after the introduction of graph compression technique. The graph clearly shows the abrupt increase in time taken by $R\&R$ algorithm for problems with increasing number of nodes.

5.4 Comparison with Other Approximation Algorithms

In this section we elaborate and compare the results of six other well-known approximation algorithms with our proposed algorithm and highlight the key findings. In the original comparitive study [7], the author of the study selected some parameters like number of iterations and number of experiments on each algorithm. The minimum number of iterations was selected as

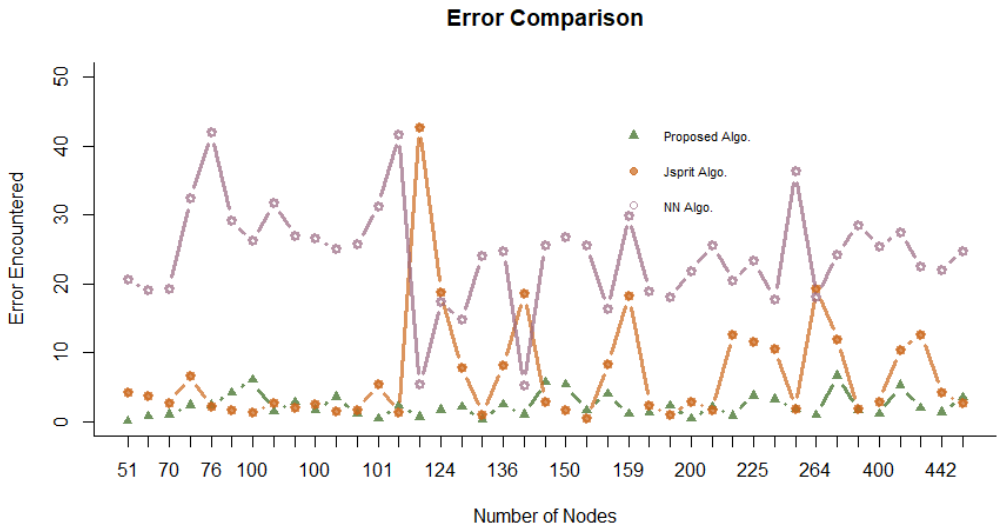

Figure 5.1: Error Comparison of Proposed algorithm, *R&R* and NN.

10,000. Each algorithm continued its execution until either they reached the already published results or completed the defined number of iterations.

Moreover, the authors of the original article [7] have actually evaluated the algorithms for multiple facets like worst time consumption etc. but we were more interested in the resultant error rate of each algorithm. In table 5.2, we have presented the resultant error margin of the six selected algorithms alongwith the results of the proposed algorithm. Each problem was evaluated with 2000 iterations of the proposed algorithm. The Table 5.2 provides the error percentage of all the algorithm using simple basic formula of error percentage i.e; (best - opt / opt) * 100. It is clearly evident that the results of our proposed algorithm are much superior and effective than the other approximation algorithms in terms of error percentage.
Out of the 14 problems, our proposed algorithm yielded the best results for 11 problems and even for the other 3 problems,

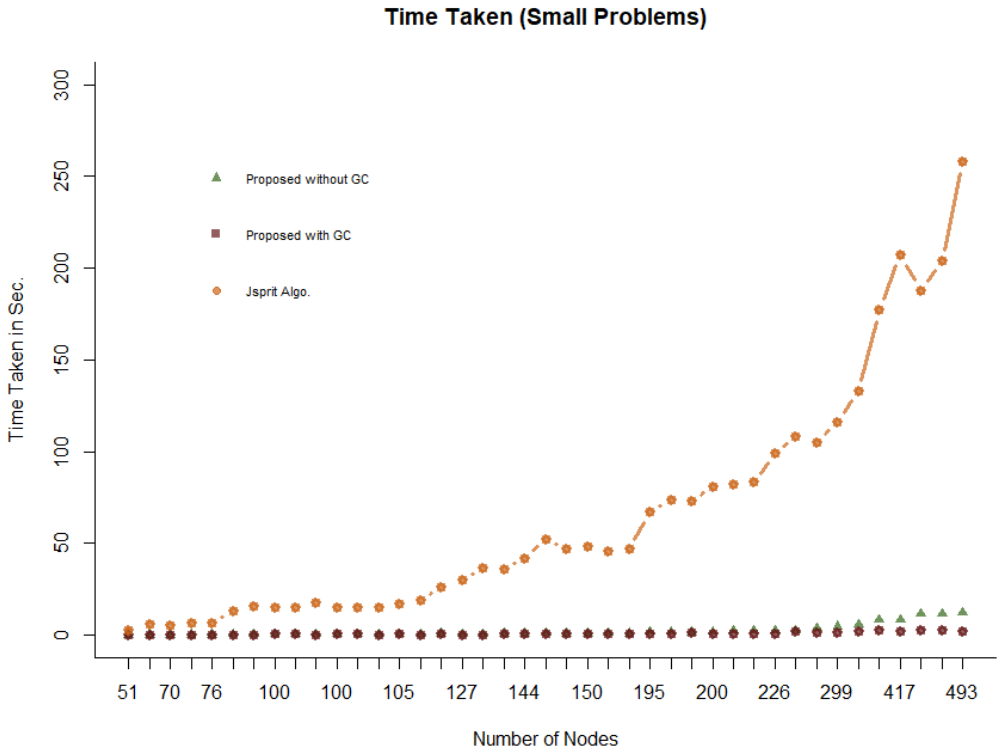

Figure 5.2: Time comparison of $R\&R$ and proposed algorithm with and without applying graph compression.

the difference is minimal. In problem rat99, The performance of Tabu search provided better solution with an error percentage of 2.68% while our proposed algorithm yielded a 2.79% error rate. However, when we regenerated our results with a graph compression technique of using only 40 active edges, our algorithm produced a much improved error rate of 2.26%, even outperforming the initial best performing tabu search. For the problem eil101, tabu search provided a 6.14% error rate while our proposed algorithm showed 6.27% error rate. However, again when we applied graph compression technique, the proposed algorithm was able to beat the tabu search with an error rate of 5.63%. For the problem d198, tabu search showed an error margin of 1.92% and our proposed algorithm produced 2.74% error.

And again after reproducing the results with graph compression i.e. setting active edges to 40, we were able to produce 1.49% error rate.
It is clearly visible that our proposed algorithm performed well for more than 80% of the cases in normal execution, and even for the rest of the cases, our algorithm was able to outperform the others in terms of error percentage with the introduction of graph compression technique.
The Fig 5.3, depicts the graphical comparison of error margins of the 7 different algorithms. The graph clearly shows the impressive performance of the proposed algorithm.

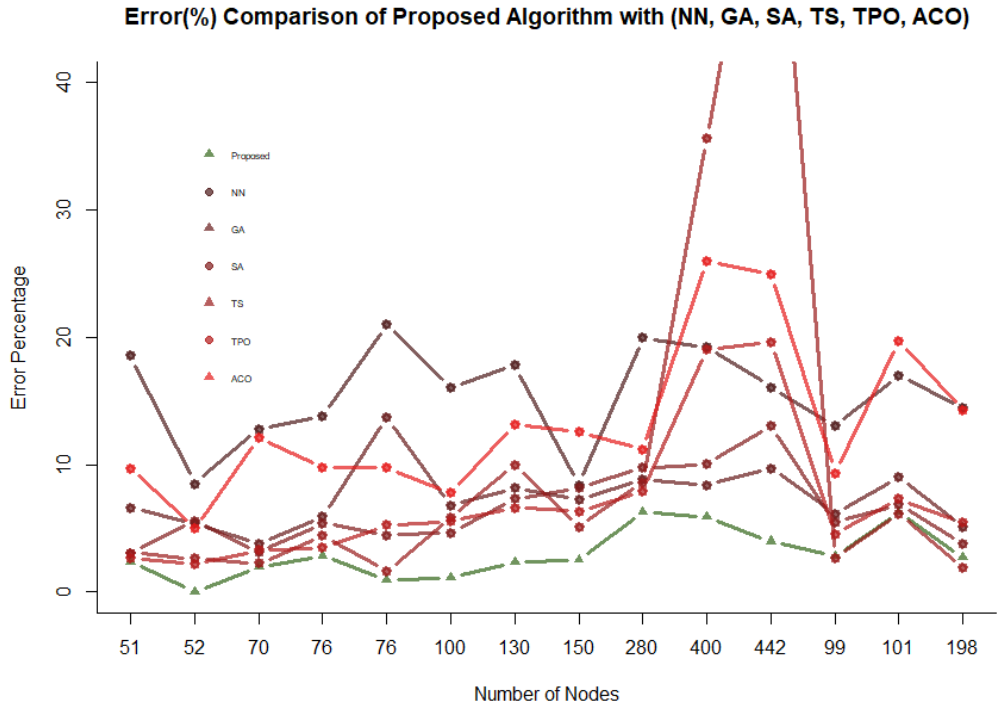

Figure 5.3: Error comparison between proposed algorithm and NN,GA,SA,TS,TPO and ACO.

Table 5.2: Comparison of proposed algorithm with existing algorithms

#	TSP Problem	Existing Literature		Proposed Algorithm
		Algorithm	**Error %**	**Error %**
1	eil51	TPO	2.64	**2.37**
		NN	18.56	
		GA	6.60	
		SA	3.08	
		TS	3.08	
		ACO	9.73	
2	berlin52	TPO	2.17	**0.03**
		NN	8.50	
		GA	5.36	
		SA	5.55	
		TS	2.63	
		ACO	5.04	
3	st70	TS	2.26	**1.96**
		NN	12.82	
		GA	3.81	
		SA	3.16	
		TPO	3.28	
		ACO	12.08	
4	eil76	TPO	3.49	**2.87**
		NN	13.80	
		GA	5.95	
		SA	5.42	
		TS	4.41	
		ACO	9.78	
5	pr76	TS	1.64	**0.94**
		NN	21.05	
		GA	13.70	
		SA	4.48	
		TPO	5.26	
		ACO	9.78	
6	kroA100	SA	4.68	**1.13**
		NN	16.05	
		GA	6.79	
		TS	5.82	
		TPO	5.55	
		ACO	7.80	
7	ch130	NN	17.82	**2.34**
		SA	7.34	
		GA	8.20	
		TS	9.94	
		TPO	6.63	
		ACO	13.16	

#	TSP Problem	Existing Literature		Proposed Algorithm
		Algorithm	**Error %**	**Error %**
8	ch150	TS	5.12	**2.54**
		NN	8.42	
		GA	7.30	
		SA	8.18	
		TPO	6.35	
		ACO	12.60	
9	a280	GA	8.82	**6.29**
		SA	9.74	
		NN	19.98	
		TS	8.60	
		TPO	7.89	
		ACO	11.20	
10	rd400	GA	8.42	**5.89**
		SA	10.05	
		NN	19.23	
		TS	35.62	
		TPO	19.04	
		ACO	26.03	
11	pcb442	GA	9.73	**3.98**
		SA	13.08	
		NN	16.10	
		TS	63.70	
		TPO	19.64	
		ACO	24.93	
12	rat99	TS	**2.68**	2.79
		NN	13.03	
		GA	6.16	
		SA	5.48	
		TPO	4.53	
		ACO	9.36	
13	eil101	TS	**6.14**	6.27
		NN	17.01	
		GA	9.04	
		SA	6.86	
		TPO	7.31	
		ACO	19.70	
14	d198	TS	**1.92**	2.74
		NN	14.46	
		GA	5.09	
		SA	3.81	
		TPO	5.49	
		ACO	14.27	

5.5 Asymmetric Results

We have computed the results of 17 problems from the category of asymmetric problems. Results have been presented in table 5.3. It is worth noting that our proposed algorithm was able to achieve the optimal results for 2 of the problems. Furthermore, the received error margin for 6 problems is under 10%. However, we still feel that there is lots of room for refinement in the algorithm if we have to tackle the asymmetric problems as vigilantly as the symmetric TSP problem . The possible improvements are also highlighted in the chapter of Future Work. Hopefully with the improvements mentioned in future work, we will be able to achieve better performance for asymmetric TSP problem as well.

Table 5.3: Asymmetric results from proposed algorithm and Nearest neighbour

Asymmetric	**Opt**	**Proposed Algorithm**		**nearest neighbour**	
		Best	Error (%)	Best	Error (%)
br17	39	39	0	92	135.90
ft53	6905	8585	24.33	13062	89.17
ft70	38673	44904	16.11	56905	47.14
ftv170	2755	5205	88.93	9274	236.62
ftv33	1286	1354	5.29	2117	64.62
ftv35	1473	1680	14.05	2470	67.68
ftv38	1530	1677	9.61	2550	66.67
ftv44	1613	1915	18.72	2853	76.88
ftv47	1776	2164	21.85	3875	118.19
ftv55	1608	1901	18.22	3571	122.08
ftv64	1839	2391	30.02	4235	130.29
ftv70	1950	2726	39.79	4760	144.10
kro124	36230	37538	3.61	54881	51.48
p43	5620	5620	0	5768	2.63
rbg323	1326	3181	139.89	5884	343.74
rbg358	1163	3040	161.39	5890	406.45
ry48	14422	14940	3.59	18237	26.45

5.6 Qualitative Results

In this section, we present the received results of our proposed algorithm by deploying the graph compression technique for all 3 categories of problems that is small, medium, and big. The results of small category problems are provided in the table 5.4, and the resultant comparison graph is depicted in fig 5.4. For comparison sake, we have also implemented and presented the results of nearest neighbor algorithm along with the results of our proposed algorithm. An important point to be noted is that for all small category problems, our proposed algorithm produced the error margin percentage of below 7%. We would like to mention that only one of problems that is p264 produced a below par 13% error margin rate. But once we added the number of active edges, it provided a much improved error rate of 3.78%. That means all the problems were successfully managed under error margin of 7%. In graphs 5.5, 5.6, 5.4, we can clearly see that the variance of error is less in proposed algorithm as compare to nearest neighbor.

5.6.1 Statistical Analysis

In this section, we have provided the graphs for convergence of both proposed and $R\&R$ algorithms. For this purpose, we ran and executed the selected examples with 10, 100, 500, 1000 and 2000 iterations. We have plotted the results for 8 different problems. From the diagrams, it is clearly visible that that in general, the proposed algorithm converged faster than the $R\&R$. Only for one problem i.e. eil101, although the proposed algorithm performed well in the beginning, but in the end $R\&R$ showed better results in convergence.

Table 5.4: Comparison of proposed algorithm with nearest neighbour drawn by graph compression technique in small category

#	Symmetric	opt	Proposed Algorithm	Nearest Neighbor
1	a280.txt	2579	6.0256	22.0671
2	berlin52.txt	7542	0.03137	19.0787
3	bier127.txt	118282	0.86683	14.7696
4	ch130.txt	6110	2.5507	23.9818
5	ch150.txt	6528	2.6766	25.5302
6	d198.txt	15780	2.2228	17.9979
7	d493.txt	35002	4.6442	24.6968
8	eil101.txt	629	6.1919	31.1991
9	eil51.txt	426	2.6464	20.5657
10	eil76.txt	538	5.299	32.3408
11	fl417.txt	11861	1.6936	27.427
12	gil262.txt	2378	3.2634	36.3106
13	kroa100.txt	21282	1.1341	26.193
14	kroa150.txt	26524	2.9467	26.7149
15	kroa200.txt	29368	3.7162	21.896
16	krob100.txt	22141	0.70763	31.679
17	krob150.txt	26130	1.6714	25.6248
18	krob200.txt	29437	2.2473	25.6296
19	kroc100.txt	20749	0.32471	26.885
20	krod100.txt	21294	2.4403	26.5636
21	kroe100.txt	22068	1.1458	25.0099
22	lin105.txt	14379	1.3808	41.6146
23	lin318.txt	42029	3.1604	28.5626
24	pcb442.txt	50778	5.2409	22.0687
25	pr107.txt	44303	0.89112	5.3612
26	pr124.txt	59030	1.0717	17.397
27	pr136.txt	96772	2.3174	24.8066
28	pr144.txt	58537	0.52838	5.3192
29	pr152.txt	73682	4.3161	16.3146
30	pr226.txt	80369	0.66134	17.8134
31	pr264.txt	49135	13.0335	18.0886
32	pr299.txt	48191	2.0508	24.295
33	pr439.txt	107217	6.7901	22.4452
34	pr76.txt	108159	0.56165	41.8855
35	rat195.txt	2323	6.9404	18.8961
36	rat99.txt	1211	2.9929	29.2093
37	rd100.txt	7910	1.1272	25.6784
38	rd400.txt	15281	6.3221	25.4372
39	st70.txt	675	3.0801	19.338
40	ts225.txt	126643	2.2892	20.4121
41	tsp225.txt	3916	3.2468	23.3145
42	u159.txt	42080	3.3684	29.9169

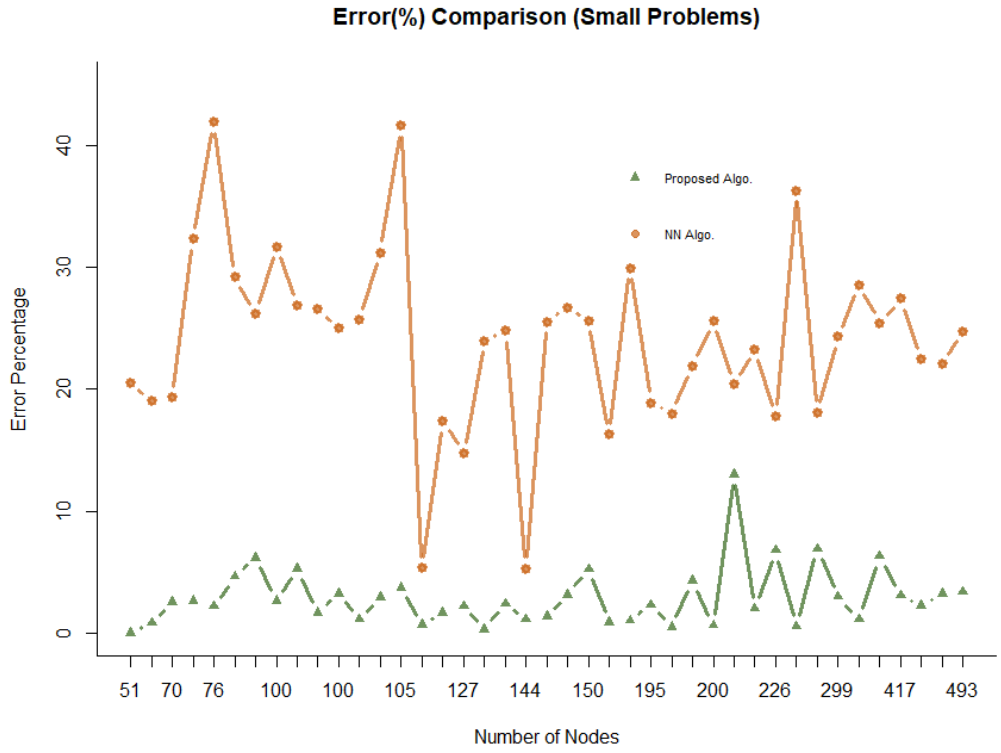

Figure 5.4: Error comparison of Proposed algorithm with NN in small category.

5.7 pla85900 Solution

This is a special problem in TSPLIB library. It consists of 85900 nodes. It is the largest optimally solved problem for TSP. It was solved in 2005/06 with the Concorde algorithm [8]. Solving this problem with the proposed methodology was a big challenge for us as it demanded huge memory resources. For representing this problem as a matrix, atleast 55GB of memory was required. The solution graph and a zoomed view of that graph are shown in figures 5.15, and 5.16.

To cater for this huge memory requirement problem, we divided the big problem into 4 smaller problems. We solved those small

Table 5.5: Comparison of proposed algorithm with nearest neighbour drawn by graph compression technique in medium category

#	Symmetric	opt	Proposed Algorithm	Nearest Neighbor
1	d1291.txt	50801	7.3098	17.9922
2	d1655.txt	62128	8.1439	20.5469
3	d2103.txt	80450	5.2668	8.7241
4	d657.txt	48912	5.0073	27.1189
5	fl1400.txt	20127	4.1048	34.0085
6	fl1577.txt	22249	3.7165	25.5828
7	fl3795.txt	28772	6.2056	18.9544
8	fnl4461.txt	182566	10.3626	24.4244
9	nrw1379.txt	56638	9.1572	23.6192
10	p654.txt	34643	3.4548	25.3112
11	pcb1173.txt	56892	7.3696	23.5287
12	pcb3038.txt	137694	10.0527	27.5094
13	pr1002.txt	259045	5.3415	21.8308
14	pr2392.txt	378032	7.7408	22.0022
15	rat575.txt	6773	8.0157	24.75
16	rat783.txt	8806	7.9636	27.8114
17	rl1304.txt	252948	4.94	34.3349
18	rl1323.txt	270199	5.4118	22.9075
19	rl1889.txt	316536	4.4935	26.5842
20	u1060.txt	224094	5.0191	25.6775
21	u1432.txt	152970	8.0601	23.4327
22	u1817.txt	57201	7.9166	24.3038
23	u2152.txt	64253	10.2833	24.7878
24	u2319.txt	234256	8.439	19.008
25	u574.txt	36905	4.6943	27.0339
26	u724.txt	41910	8.0203	31.7662
27	vm1084.txt	239297	5.493	25.9812
28	vm1748.txt	336556	6.4284	21.2545

problems and at the end merged them together to get the approximate solution. To break one big problem into 4 small problems, we took the average of x-axis points and y-axis points. All those points on x-axis which were less than the average value were placed below the virtual line and all those points on x-axis

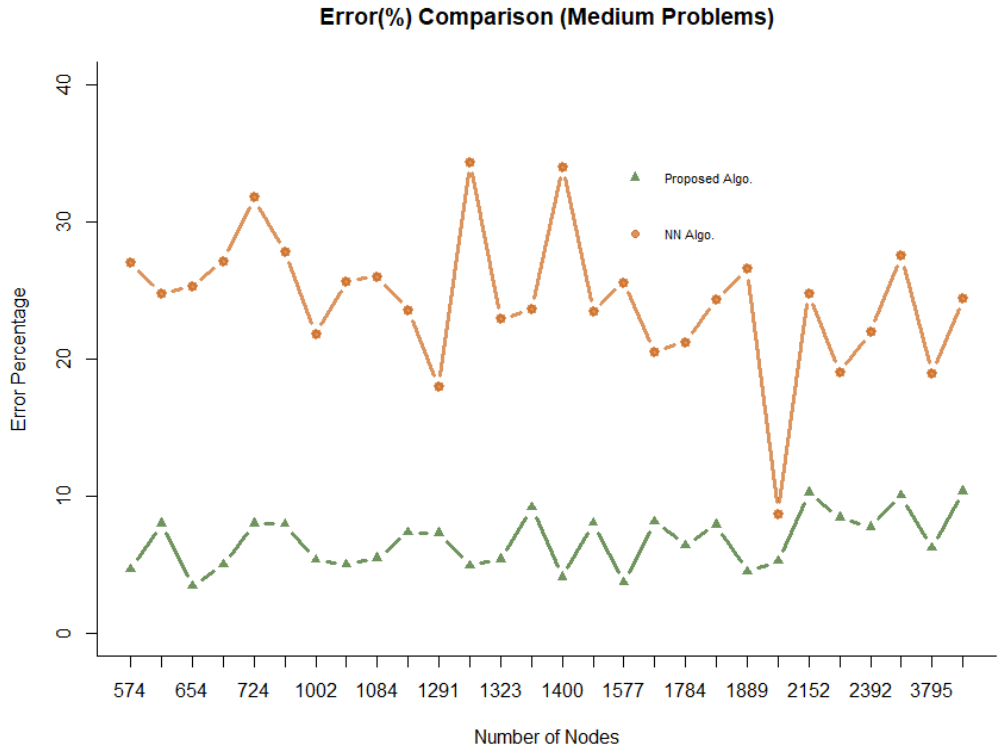

Figure 5.5: Error comparison of Proposed algorithm with NN in Medium category.

Table 5.6: Comparison of proposed algorithm with nearest neighbour drawn by graph compression technique in large category

#	Symmetric	opt	Proposed Algorithm	Nearest Neighbor
1	rl5915.txt	565530	8.186	25.10
2	rl5934.txt	556045	9.209	22.98
3	rl11849.txt	923288	10.3	22
4	usa13509.txt	19982859	10.3	23
5	brd14051.txt	469385	10.3	23
6	d18512.txt	645238	9.7	25
7	pla33810	66048945	9.24	17.08
8	pla85900	142382641	10.14	22.23

which were greater than that value were placed over the line. Which means that all the points that fell below the line were

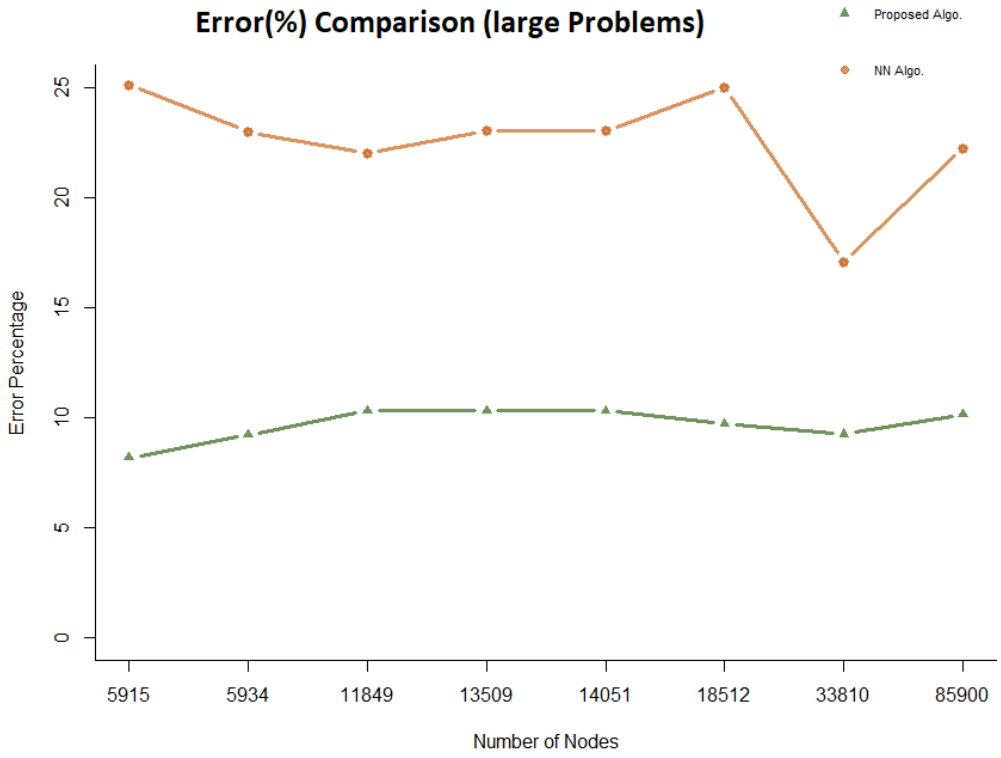

Figure 5.6: Error comparison of Proposed algorithm with NN in large category.

placed in the 3rd and 4th quarter, while all the points above the line were placed in 1st and 2nd quarter. Same technique was used with the y-axis points. So we calculated all the points which fall in 1st, 2nd, 3rd and 4th quarters. We considered every problem as an independent problem and solved it, in the end all problems were merged together. That produced an error margin of 10.14%, In table 5.7, results of each small problem, total best distance and computed error has been provided.

5.8 Summary

After carefully analyzing the results of proposed algorithm with 7 other algorithms in similar settings, we are able to conclude

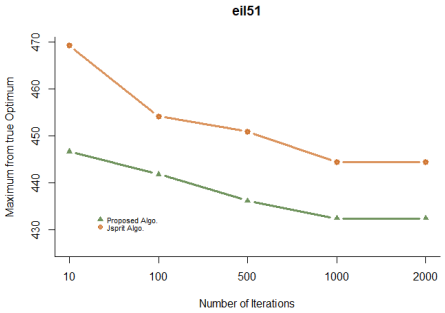

Figure 5.7: eil51 convergence graph comparison of proposed algorithm and *R&R*

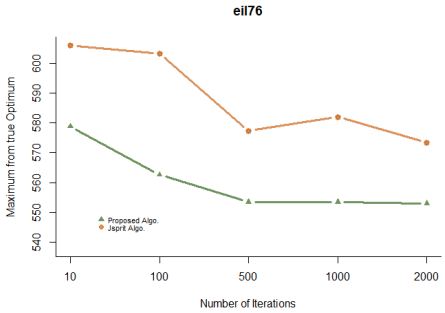

Figure 5.8: eil76 convergence graph comparison of proposed algorithm and *R&R*

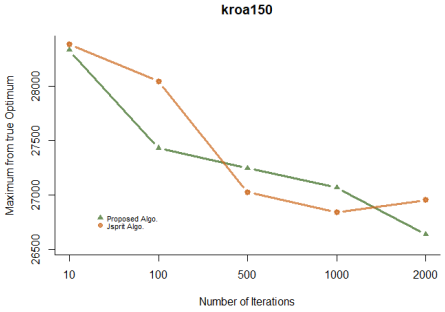

Figure 5.9: kroa150 convergence graph comparison of proposed algorithm and *R&R*

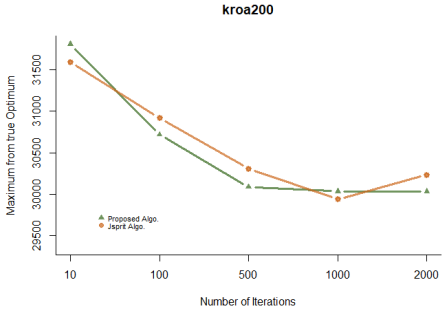

Figure 5.10: kroa200 convergence graph comparison of proposed algorithm and *R&R*

Table 5.7: Individual solution of the problem PLA85900

Name	Opt	Best	Error
Quad 1	-	42213702	-
Quad 2	-	41973309	-
Quad 3	-	36864495	-
Quad 4	-	35775820	-
PLA85900	142382641	156827326	10.14

that proposed algorithm outperforms all other well-known algorithms in terms of all important factors like error variance, convergence, and computation time. Here we will discuss average error and average time taken of proposed algorithm and R&R

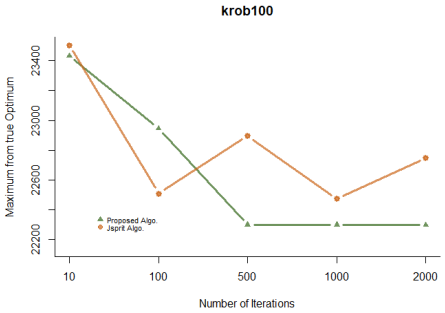

Figure 5.11: krob100 convergence graph comparison of proposed algorithm and *R&R*

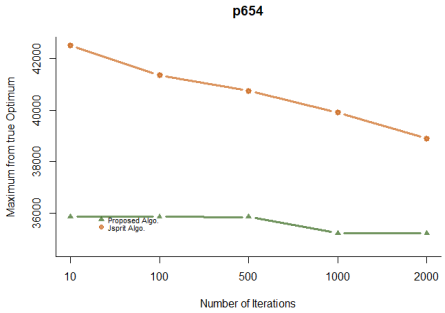

Figure 5.12: p654 convergence graph comparison of proposed algorithm and *R&R*

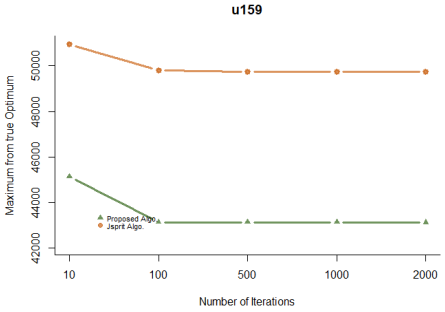

Figure 5.13: u159 convergence graph comparison of proposed algorithm and *R&R*

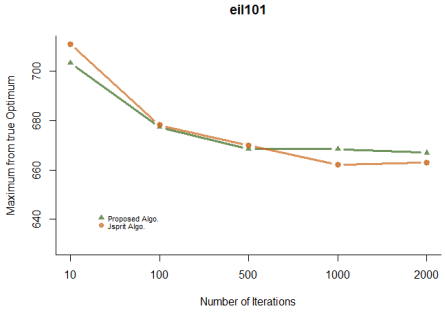

Figure 5.14: eil101 convergence graph comparison of proposed algorithm and *R&R*

algorithm. We have seen that proposed algorithm produced better results as compared to R&R for error margin, computation time, and convergence rate. The results have been provided in the table 5.1 and are further depicted with the help of diagrams.

For the 41 problems extracted from TSPLIB library for the small category, the average error variance for the *R&R* stood at 6.79% while for our proposed algorithm, this value was 2.33%. *R&R* is considered one of the best algorithms for TSP problem so the difference in this result is quite significant. Even for those cases where *R&R* produced a better result than our proposed

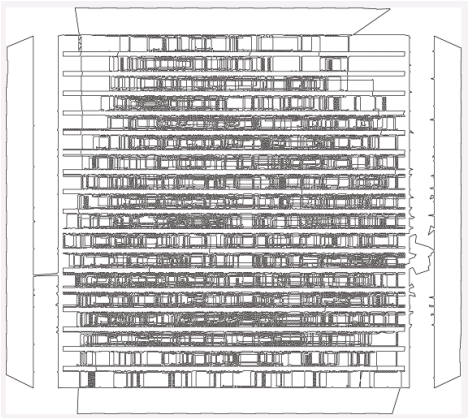

Figure 5.15: Solution graph of pla85900 problem

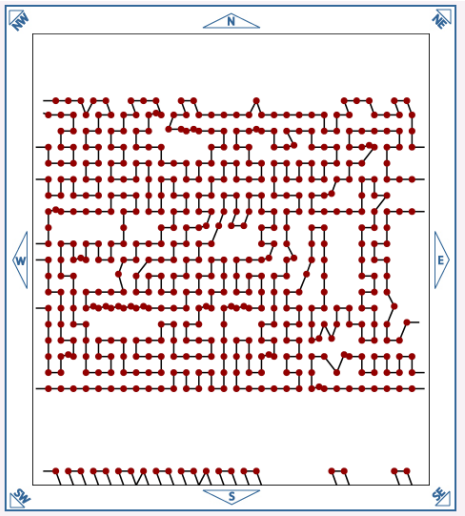

Figure 5.16: Zoomed view of solution graph of pla85900

algorithm, the difference was of very minor nature. The average time consumed by $R\&R$ and proposed algorithm is 64.63 seconds and 2.41 seconds respectively. This is again significant and highlights the efficacy of our proposed algorithm.

All the results of proposed algorithm and the nearest neighbor algorithm for all categories of symmetric graphs are mentioned in the tables 5.4, 5.5, 5.6. Graphs of error, time and convergence are also drawn to demonstrate the effectiveness of different algorithms. Overall, the results strongly advocate for the efficacy of our proposed algorithm for almost all kind of symmetric TSP problem instances.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

The proposed algorithm performed very well in the computational complexity of techniques that provide the practical approximation ratio with both symmetric and asymmetric problems. The algorithm finds a tour with minimum complexity and computational time of TSP using edge swap and graph compression techniques with impressive results in terms of approximation ratio, time and convergence. Graph compression technique has been applied to solve the relatively larger problems in reasonable time.

Problems from the TSPLIB library [20] has been divided into small, medium and big categories for benchmarking. In first comparison, we have compared and contrasted our results with the state-of-the-art algorithm known as ruin and recreate with respect to time, convergence and error variance from optimum results.

In second comparison, the results of the proposed algorithm are compared with 6 other well known algorithms published in research paper [7], algorithms selected for comparisons are NN,

GA, SA, TS, ACO and TPO. The performance of our proposed algorithm has proved to be extremely impressive. NN algorithm has been the fastest followed by the proposed algorithm, TPO and GA. Based on the problem size, different algorithms shows different patterns in time consumption. Tabu search and ACO depicts exponential time increase as the problem size increases, while others show a slight increase in time with problem size. In average error comparison, proposed algorithm performed outstanding followed by Tabu search, TPO and GA. Tabu search better performance in average error could be due to its diversification mechanism like swap, reversion and insertion techniques and the tabu list which ensures the best selection of the solution candidate. TS still shows up to 64% of error margin in a problem of 442 nodes. This problem can be addressed by using some other algorithms in tabu list mechanism. In accuracy GA shows consistent results with error margins less than 15% in all selected problems. It uses mutation process which derives it toward better solution. If we compare the individual results, proposed algorithm performs best followed by TS, TPO and GA.

Results of asymmetric type problems have been presented in table 5.3, with some more refinements in the proposed algorithm, results for asymmetric TSP problems can become consistent. Some of those points have also been mentioned in future work.

6.2 Future work and Modification

- **Choice of base architecture**: The base architecture used in this thesis is nearest neighbor. Other algorithms can also be used to feed a base Hamiltonian cycle to the algorithm. However, the main reasons for their selection of the greedy approach is it's simplicity and time efficiency. As the selection of the Hamiltonian cycle at the initial stage

is significant for the overall performance of the algorithm, considerable amount of research is needed to select an effective sub-optimal Hamiltonian cycle algorithm which can give good results as compared to greedy without effecting the time efficiency of proposed algorithm.

- **Different Mutations**: In the proposed algorithm, a single mutation is used for simplicity sake i.e. two edges are removed from the graph and added up with two more edges to complete the graph. However, this single mutation does not guarantee better results. Ruin and recreate algorithm [11] uses multiple mutations like swap, reversion and insertion to ensure a closer solution to the optimum value. Other mutations could also be used in this algorithm to keep it simple and efficient.

- **Greedy approach**: This algorithm uses a greedy approach for the selection of best tour after mutation. Greedy approach tends to eliminate or ignore other possibly better solutions from the table, so in the near future, we will aim to adopt a better selection criteria such as threshold acceptance or simulated annealing etc.

- **Data set and architecture**: In this work, we have only used Euclidean 2D and Ceil 2D type problems from TSPLIB library. This benchmark library has some other important problems as well where data is available in other formats like MAN-3D (Manhattan distance in 3d), geographical distances or matrix form [20]. The proposed algorithm can be improved to accept any input type. The architecture used in this algorithm can also be improved to cater for the bigger problems in memory. Right now we have divided the last problem in 4 small problems to solve it in memory. PLA85900 is an example with 85900 nodes. It consumes a

lots of resources to solve a matrix of 85900 * 85900 in memory. Architecture can be modified to solve any matrix using arrays. For example, instead of placing the whole matrix in memory, there should be a way to only place required information in memory, which will speed up the algorithm as well.

- **Better graph compression technique**: Graph compression technique is perhaps the most important task that demands attention as finding TSP solutions for larger problems tends to become problemtic due to demand of resources. We have used a simple graph compression technique in this thesis but the technique is too simple for our liking. The proposed technique activates an edge if and only if it is near to the required node. A new and better technique can be devised for compression which can take into account all the nearest points in 360 degrees. This will be of great help in graphs where many points are assembled in a small area. Moreover, a separate module can also be developed which can be used with other algorithms for graph compression purpose.

In this thesis, we have proposed a very simple and efficient approximation technique to solve both symmetric and asymmetric TSP problems using 2-opt local search with some enhancements. Graph compression technique has been used for finding the solution of larger sized problems. For benchmarking, TSPLIB library problems have been used. With the help of these new and simple techniques, we have empirically justified the performance of our proposed technique taking into account the factors of time, accuracy and complexity.

Bibliography

[1] H. Chen, G. Tan, G. Qian, and R. Chen, "Ant colony optimization with tabu table to solve tsp problem," in *2018 37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 2523–2527.

[2] G. Weidong, F. Jinqiao, W. Yazhou, Z. Hongjun, and H. Jidong, "Parallel performance of an ant colony optimization algorithm for tsp," in *2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*. IEEE, 2015, pp. 625–629.

[3] E. Aarts, J. Korst, and W. Michiels, "Simulated annealing," in *Search methodologies*. Springer, 2005, pp. 187–210.

[4] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," *Handbooks in operations research and management science*, vol. 7, pp. 225–330, 1995.

[5] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[6] M. Khalil, J.-P. Li, Y. Wang, and A. Khan, "Algorithm to solve travel salesman problem efficently," in *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. IEEE, 2016, pp. 123–126.

[7] A. H. Halim and I. Ismail, "Combinatorial optimization: comparison of heuristic algorithms in travelling salesman problem," *Archives of Computational Methods in Engineering*, vol. 26, no. 2, pp. 367–380, 2019.

[8] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[9] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling salesman problem, theory and applications*, vol. 1, 2010.

[10] C. Chauhan, R. Gupta, and K. Pathak, "Survey of methods of solving tsp along with its implementation using dynamic programming approach," *International journal of computer applications*, vol. 52, no. 4, 2012.

[11] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[12] D. Lin, X. Wu, and D. Wang, "Exact heuristic algorithm for traveling salesman problem," in *2008 The 9th International Conference for Young Computer Scientists*. IEEE, 2008, pp. 9–13.

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

[14] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun, "Certification of an optimal tsp tour through 85,900 cities," *Operations Research Letters*, vol. 37, no. 1, pp. 11–15, 2009.

[15] M. Bläser, K. Panagiotou, and B. R. Rao, "A probabilistic analysis of christofides algorithm," in *Scandinavian Workshop on Algorithm Theory*. Springer, 2012, pp. 225–236.

[16] Z. Xu, L. Xu, and B. Rodrigues, "An analysis of the extended christofides heuristic for the k-depot tsp," *Operations research letters*, vol. 39, no. 3, pp. 218–223, 2011.

[17] R. Li, J. Qiao, and W. Li, "A modified hopfield neural network for solving tsp problem," in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. IEEE, 2016, pp. 1775–1780.

[18] V. Chvátal, W. Cook, G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a large-scale traveling-salesman problem," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 7–28.

[19] W. Cook, "History of the tsp," *The Traveling Salesman Problem*, 2009.

[20] G. Reinelt, "Tsplib a traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[21] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[22] E. Aarts, E. H. Aarts, and J. K. Lenstra, *Local search in combinatorial optimization*. Princeton University Press, 2003.

[23] W. Zhang, "Depth-first branch-and-bound versus local search: A case study," in *AAAI/IAAI*, 2000, pp. 930–935.

[24] G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *Journal of computational physics*, vol. 90, no. 1, pp. 161–175, 1990.

[25] I. K. Gupta, A. Choubey, and S. Choubey, "Randomized bias genetic algorithm to solve traveling salesman problem," in *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2017, pp. 1–6.

[26] B. L. Lin, X. Sun, and S. Salous, "Solving travelling salesman problem with an improved hybrid genetic algorithm." *Journal of computer and communications.*, vol. 4, no. 15, pp. 98–106, 2016.

[27] A. Hussain, Y. S. Muhammad, and M. N. Sajid, "A simulated study of genetic algorithm with a new crossover operator using traveling salesman problem," *Journal of Mathematics (ISSN 1016-2526)*, vol. 51, no. 2, pp. 00–00, 2019.

[28] F. Glover, "Tabu searchpart i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[29] L. Eskandari, A. Jafarian, P. Rahimloo, and D. Baleanu, "A modified and enhanced ant colony optimization algorithm for traveling salesman problem," in *Mathematical Methods in Engineering*. Springer, 2019, pp. 257–265.