Heuristics for Variants of Traveling Salesman Problem

A thesis submitted during 2019 to the University of Hyderabad in partial fulfillment of the award of a **Ph.D. degree** in School of Computer and Information Sciences

by

Pandiri Venkatesh

School of Computer and Information Sciences
University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500 046
Telangana, India

May 2019

CERTIFICATE

This is to certify that the thesis entitled **"Heuristics for Variants of Traveling Salesman Problem"** submitted by **Pandiri Venkatesh** bearing **Reg. No. 14MCPC02** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under my supervision and guidance.

This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

The student has the following publications before submission of the thesis for adjudication and has produced evidence for the same in the form of acceptance letter or the reprint in the relevant area of his research:

1. **Venkatesh Pandiri**, Alok Singh. "A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem". *Information Sciences,* ***463-464****: 261-281, 2018, Elsevier (ISSN: 0020-0255).* Work reported in this paper appears in **Chapter 8**.
2. **Venkatesh Pandiri**, Alok Singh. "A swarm intelligence approach for the colored traveling salesman problem". *Applied Intelligence,* ***48****: 4412-4428, 2018, Springer-Verlag (ISSN: 0924-669X).* Work reported in this paper appears in **Chapter 7**.
3. **Venkatesh Pandiri**, Alok Singh. "An artificial bee colony algorithm with variable degree of perturbation for the generalized covering traveling salesman problem". *Applied Soft Computing,* ***78****: 481-495, 2019, Elsevier (ISSN: 1568-4946).* Work reported in this paper appears in **Chapter 6**.
4. **Venkatesh Pandiri**, Gaurav Srivastava and Alok Singh. "A general variable neighborhood search algorithm for the k-traveling salesman problem". *Proceedings of the 2018*

*International Conference on Advances in Computing & Communications (ICACC-2018), Procedia Computer Science, **143**: 189-196, 2018, Elsevier (ISSN: 1877-0509).* Work reported in this paper appears in **Chapter 3**.

5. **Venkatesh Pandiri**, Alok Singh and Rammohan Mallipeddi. "A multi-start iterated local search algorithm for the maximum scatter traveling salesman problem". To appear in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC-2019), 2019, IEEE*. Work reported in this paper appears in **Chapter 2**.

and has made the presentations in the following conferences:

1. 2018 International Conference on Advances in Computing & Communications (ICACC-2018), September 13-15, 2018, Kochi, India.
2. 2018 International Conference on Harmony Search, Soft Computing and Applications (ICHSA-2018), February 7-9, 2018, Gurugram, India.

Further, the student has passed the following courses towards fulfillment of coursework requirement for Ph.D.:

Course Code	**Name**	**Credits**	**Pass/Fail**
CS 801	Data Structures and Algorithms	4	Pass
CS 802	Operating Systems and Programming	4	Pass
AI 810	Metaheuristic Techniques	4	Pass
AI 852	Learning & Reasoning	4	Pass

(Prof. Alok Singh)
Supervisor
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500 046, India

(Prof. Kavi Narayana Murthy)
Dean
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500 046, India

DECLARATION

I, **Pandiri Venkatesh**, hereby declare that this thesis entitled **"Heuristics for Variants of Traveling Salesman Problem"** submitted by me under the guidance and supervision of **Prof. Alok Singh** is a bonafide research work which is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Library is enclosed.

Date :

Name: Pandiri Venkatesh

Signature of the Student:

Reg. No.: 14MCPC02

Signature of the Supervisor:

Abstract

The traveling salesman problem (TSP) is about finding a minimum length Hamiltonian cycle over a given set of cities. Very few problems have as widespread applicability as TSP and are as well studied as TSP. Practically, any problem that consists of finding a sequence of actions, operations, tasks etc., can have a TSP aspect to it. Over the last five decades, numerous optimization problems have been studied by the researchers. Problems containing TSP aspect have a significant share in these problems. Despite the aspect of TSP in these problems, its not always possible to formulate them directly as TSP due to the involvement of additional constraints or the occurrence of TSP only as a sub-problem in them. To address such problems, many variants of TSP have been proposed in the literature by researchers from different fields. Apart from real world applications, these variants pose a serious challenge from theoretical perspective also. Any improvement that can be made in solving a TSP variant will have a direct impact on associated real world applications. Further, many TSP variants are not as widely studied as the TSP itself, and, there is a scope for significant improvement as far as heuristic solution approaches for such variants are concerned. Motivated by these facts, in this thesis, we have solved some of the $\mathcal{NP}$-hard variants of the TSP through various heuristic techniques.

Seven $\mathcal{NP}$-hard TSP variants covering a wide range of TSP variants have been addressed in this thesis. These seven TSP variants are as follows: maximum scatter traveling salesman problem, k-traveling salesman problem, family traveling salesman problem, covering salesman problem, generalized covering traveling salesman problem, colored traveling salesman problem and k-interconnected multi-depot multi-traveling salesman problem. These problems not only pose a serious challenge from theoretical point of view, but also have many real world applications in diverse domains such as transportation, logistics, manufacturing, networks, planning & scheduling, healthcare etc. For solving these problems, we have developed various heuristic approaches based on artificial bee colony algorithm, variable neighborhood search, large neighborhood search, iterated local search, hyper-heuristic and genetic algorithm. Among all these techniques, artificial bee colony algorithm and hyper-heuristics have been used to solve four and three problems respectively, whereas each of the remaining technique is used to solve a single problem only. Appropriate problem-specific knowledge has been incorporated in all our approaches to boost their performance. Computational results demonstrate the effectiveness of our approaches. Our approaches can be easily extended to solve other variants of TSP. Ideas presented in this thesis can be utilized to solve many other combinatorial optimization problems also.

To my parents, ***SRI Purushottam and SRIMATI Vijaya Kumari***
without their endless love, support and encouragement this would not have been possible.

Acknowledgements

It has been a wonderful journey for me towards the doctorate degree. There are many people who guided and helped me in many ways during this journey. I take this opportunity to express my gratitude to all of them.

I would like to offer my sincere gratitude to my supervisor **Prof. Alok Singh** who guided me from basics to advanced research in the field of combinatorial optimization using heuristic and metaheuristic techniques. I will always feel privileged for having had the pleasure of working with him. I am obliged to him for believing in me and giving me the freedom to pursue my ideas. His support, availability, patience and guidance have pushed me to offer my best, often more than I thought I was capable of doing. I can not thank him enough for all that he has done for me. I gratefully thank my Doctoral Review Committee (DRC) members, **Dr. Anupama Potluri** and **Prof. Rajeev Wankar** for their valuable insights and feedback about my research work, which improved the quality of work. I feel fortunate to be overseen by them.

I am thankful to each and every faculty member of the school, who have whole heartedly contributed to whatever I have learnt and have helped me to broaden my horizon. Other than my supervisor and DRC members, I am especially thankful to **Dr. S. Durga Bhavani, Prof. Kavi Narayana Murthy, Prof. Chakravarthy Bhagavati** and **Prof. S.K. Udgata** from whom I have learnt things beyond the subject and got the motivation to pursue my interest towards research. I am thankful to **Prof. André Rossi** and **Dr. Rammohan Mallipeddi** for collaborating with me to work on different problems. I am indebted to my senior lab mate **Dr. Sachchida Nand Chaurasia**, who had helped me in almost every phase of my Ph.D. I found him always available, whenever I needed him.

I am thankful to the Dean of the School **Prof. Kavi Narayana Murthy** for providing all the necessary facilities to pursue my research work.

I am thankful for the unstinting support that I received from the research infrastructure and the effervescent ambiance of the University. Due credit to the University for building a research oriented School of Computer & Information Sciences (SCIS), a library rich in a wide range of research books & articles, and most importantly a healthy campus atmosphere.

I am thankful to my fellow lab mates (**Shyam, Sachchida, Jaya, Abobakr, Gaurav, Edukondalu, Mallikarjuna, Kasi, and Sebanti**) for stimulating discussions, sleepless nights we were working together and for all the fun we had together in last five years.

I am grateful to a number of researchers in my field who furnished me with indispensable research data and clarified my doubts regarding the definitions and approaches they presented for the family traveling salesman problem (**Dr. L.F. Morán-Mirabal, Dr. R. Bernardino**), covering salesman problem (**Dr. M. Salari**), generalized covering traveling salesman problem (**Dr. M. Salari, Mr. M.H. Shaelaie**), colored traveling salesman problem (**Prof. J. Li, Prof. M.C. Zhou**) and k-interconnected multi-depot multi-traveling salesman problem (**Dr. C.E. Andrade**).

This journey would not have been wonderful without these three gangs, **PhD2014 Gang** (Abdul, Abhay, Vikas, Sandeep, Thirupathi, Anirudh, Raghu, Ayang, Abdul salam, Kagita and Melinda) - for many joyful parties and sweet moments, **Hostel Gang** (Rajesh, Venu, Anirudh, Rakesh, Renjith, Sharma, Vinod, Yogi, Sunil, Adimoolam and Srinu) - for many memorable times we had together in playing games, watching movies, cooking delicious food, trekking and many more adventures, **We11 Gang** (Subbu, Krishna, Kagita, Hanish, Murali, Tam, Siva, Nagarjuna, Charan and Anand) - for their constant support in my career. My special thanks to **Subbu, Krishna** and **Kagita** who are always available to help me whenever I am in need. I am thankful to all my Ph.D. colleagues for their support and encouragement. I could not start citing their names as the list would need pages, but I am thankful to many people I came to know during this journey.

A major part of my research work was supported by Council of Scientific & Industrial Research (CSIR), Government of India through Senior Research Fellowship. I am grateful to this organization for the financial support.

Finally, my heartfelt thanks to the people who mean a lot to me, my parents, **SRI Purushottam** and **SRIMATI Vijaya Kumari** for all their love, care, pain, sacrifice and constant encouragement in shaping my life. I am also thankful to my sister **Manasa** and brother-in-law **Ramesh** for their affection, patience and encouragement.

Pandiri Venkatesh

Contents

List of Figures

List of Tables

LIST OF TABLES

Chapter 1

Introduction

The traveling salesman problem (TSP) is considered as a classical problem in the domains of combinatorial optimization and operations research. It also enjoys the status of one among the most studied problems and one among those problems having wide range of real world applications. Given a set of cities along with distances between each pair of cities, the TSP seeks a minimum length Hamiltonian cycle over this set of cities. The distance matrix of a TSP instance specifying the distances between each pair of cities in that instance can be symmetric or asymmetric. Accordingly, a TSP instance is categorized as symmetric or asymmetric. Practically, any problem that consists of finding a sequence of actions, operations, tasks etc., can be casted as a TSP or a variant of it. Many real world problems like routing, networking, scheduling, ordering, timetabling etc., have a TSP aspect to them. TSP and most of its variants are $\mathcal{NP}$-hard. Over the last five decades, researchers across different fields have studied numerous optimization problems. Most of these problems belong to the class of $\mathcal{NP}$-hard problems. The TSP and its variants have a significant share in these problems. Because of its simplicity, widespread applicability and apparent intractability, the TSP became a fundamental problem to be discussed in universities and industry as a part of course curriculum/research.

In the TSP's rise to fame, the catchy name of it must have played a role in addition to its easily understandable yet intractable nature. The origin of the name "traveling salesman problem" is unclear. There does not seem to be any authentic source in the literature providing the evidence about who coined the name "traveling salesman problem". However, according to the paper of Morton and Land [6] published in 1955, it is believed that the name of traveling salesman problem is originated in the United States and this American term was used by Julia Robinson in her technical report [7] published in 1949. In her report also, it is clear that the name was not introduced by her. Although who coined the name is not known, the TSP research community believes that the TSP takes its name sometime in the 1930's or 1940's most probably at Princeton University according to the comments given by one of the early TSP researcher Merrill Flood [8] of the Princeton University and the RAND Corporation.

It is believed that the first published work on TSP [9] is by Karl Menger in 1932 [10, 11, 12] though he did not use the name "traveling salesman problem". In his report [9], Menger addressed a variation of the TSP where the objective was to find a shortest path connecting a given set of cities. He named this problem as "das botenproblem" in German which translates to the "messenger problem" in English. This problem was named so as each and every postal messenger faces this problem. The messenger problem and the TSP are equivalent as they can be reduced to one another via elementary transformations. Menger also pointed out that an optimal solution can be found by verifying all the permutations of the given cities and the algorithm that chooses the nearest unvisited city at each step can not guarantee an optimal solution. A few earlier papers describing applications of the TSP exists in the field of statistics where it is known by the name "mean minimum distances problem" [6, 13, 14, 15].

One of the earliest papers on the application of TSP [16] is due to the famous Indian statistician P. C. Mahalanobis in 1940 [10, 12]. In his paper, Mahalanobis discussed about forecasting of the jute crop in the Bengal region of British India during 1936-37. Mahalanobis was interested in finding a least cost tour for collecting data from a finite number of randomly sampled places, and those places are taken by considering different zones of which each zone consists of land of similar characteristics. Clearly, the TSP aspect can be seen in this application as there is a need of finding least cost tour for data collectors.

The first geometric instance of TSP was due to Dantzig *et al.* [17] in 1954. In their paper, they considered a geometric instance consisting of 49 cities from the United States. This motivated the TSP researchers to create new geometric instances by considering a set of actual cities and the distance between those cities. Over the years, with the increasing attention, the researchers considered numerous instances of varying sizes. In the early 1990's, Gerhard Reinelt [18] gathered all these instances ranging up to 85,900 cities into a traveling salesman problem library (TSPLIB)[1]. This library contains more than 100 instances and it is of great help to the researchers as it provides them a common test bed to test their approaches. In fact, all the chapters in this thesis make use of the instances which are derived from the instances available on TSPLIB.

[1]`http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html`

The first published work on solving a TSP instance was due to Dantzig *et al.* [17] in 1954. They solved an instance with 49 cities to optimality by representing the TSP as an integer linear program and using cutting plane method. Over the years, many instances of increasing sizes are solved by people from different fields by using branch-and-bound algorithms, branch-and-cut methods etc. [19, 20, 21, 22, 23, 24]. The Concorde is the fastest TSP solver so far and it was developed by Applegate *et al.* [10]. The largest solved TSP instance till date contains 85900 cities and it was solved by Concorde.

With the advancement of technology and human civilization, new problems including those with aspect of TSP continue to emerge. Despite the aspect of TSP in some of these problems, it is not always possible to formulate them directly as TSP due to either additional constraints involved or the occurrence of TSP only as a sub-problem in them. To address such problems, many variants of TSP have been proposed by researchers from different fields. Apart from real world applications, these variants pose a serious challenge from theoretical perspective. There exist several types of TSP variants. A TSP variant can have either a single salesman or multiple salesmen. In some TSP variants, a salesman can start his tour from any city, whereas in other variants particularly all those involving multiple salesmen, a salesman can start his tour only from a designated city called home city or depot. Further, in TSP variants having multiple salesmen, either all salesmen can start their tours from the same city (single depot variants) or each salesman can start his tour from his own city (multi-depot variants).

Any improvement that can be made in solving a variant of the TSP will have a direct impact on associated real world applications. Further, many TSP variants are not as widely studied as the TSP itself, and there is a scope for significant improvement as far as heuristic solution approaches for such variants are concerned. Motivated by these facts, in this thesis, we have addressed some of the $\mathcal{NP}$-hard variants of the TSP through various heuristic techniques.

Vehicle routing problems form an important subtype of TSP variants with multiple salesmen, where there are capacity restrictions on each salesman, a demand for each city and possibly several additional constraints depending on the real world situation to be modelled. There exists numerous vehicle routing problem variants, and innumerable solution approaches to solve them in the literature [25, 26]. In fact, vehicle routing problems are so widely studied, that there is a dedicated research journal for such problems[1]. Owing to their wide-spread popularity and availability of vast literature on them, vehicle routing problems are usually treated separately from other TSP variants, and most resources on TSP and its variants do not include vehicle

[1]Journal on Vehicle Routing Algorithms, Springer-Verlag

routing problems [11, 27]. Our thesis is no exception. We have neither provided any further discussion on vehicle routing problem & its variants nor solved any of them in this thesis.

TSP and its variants belong to the class of combinatorial optimization problems. Combinatorial optimization problems have been gaining a colossal attention from the researchers across different fields of study due to their practical importance. One will not be able to go through the life without encountering problems of this kind. A combinatorial optimization problem is discrete & finite by nature and requires an optimal solution from its large but finite set of feasible solutions. Such a problem is identified by its combinatorial explosion property which states that the number of feasible solutions grows exponentially with increase in the size of problem's instance. Problems evincing such a property pose a serious challenge to the researchers across different fields of study from both theoretical and practical perspectives. Such problems are inherently computational.

While exploring the history of combinatorial optimization, one can observe that several combinatorial optimization problems such as finding a shortest path from source to destination and assigning jobs to men were known since long. However, these problems were not studied in a systematic manner, and extemporary solutions were offered to such problems in a specific context as and when they arose. According to Schrijver [28], it is believed that the combinatorial optimization is a relatively young discipline. But the roots of this subject can be traced back to the year 1735 when the Swiss mathematician Leonhard Euler published his work on the *Seven Bridges of Königsberg*. This work laid the foundation of graph theory which in turn helped the combinatorial optimization to emerge as an individual discipline of mathematics. This discipline has been continuously evolving since then. The contributions of Kantorovich [29] and Dantzig [17] in developing the mathematical linear programming helped in strengthening this discipline. The advent and advancement of computer technology significantly curtailed the effort required on researchers' part in solving combinatorial optimization problems, thereby fuelling further growth of this discipline.

1.1 Combinatorial optimization

As mentioned already, a combinatorial optimization problem seeks an optimal solution from its finite set of feasible solutions. Optimality is determined with reference to the value of some cost function which is termed as the objective function. A combinatorial optimization problem can be either a minimization or a maximization problem depending on whether the value of the objective function needs to be minimized or maximized. Such a problem can be interpreted as an infinite set of instances where an instance of a problem refers to an input for that particular problem. For example, given an edge-weighted connected graph and two specified vertices of this graph, the shortest path problem seeks a path of minimum length between these two vertices. Clearly, an instance of this problem consists of an edge-weighted connected graph and two specified vertices of this graph.

On a given instance, a combinatorial optimization problem COP can be represented formally as a triplet (S, C, f), where

- S represents the search space or solution space containing all the feasible solutions.
- C represents the set of constraints that must be satisfied by each and every feasible solution.
- f represents the objective function whose value has to be either minimized or maximized.

The objective of COP is to find a globally optimal solution $s^* \in S$, i.e.,$f(s^*)$ is as good as or better than $f(s)\ \forall s \in S$. It is important to note that there may exist multiple optimal solutions for a given instance of a problem, i.e., optimal solution may not be unique. As mentioned earlier, depending on the objective function, one may have to find a solution with maximum or minimum objective function value. It can be observed that, by changing the sign of an objective function, a maximization problem can be transformed into minimization problem, and vice-versa. In COP, the variables, which take part in constructing the solutions are discrete quantities. The values of these variables are used in computing the value of the objective function. For example, in case of the shortest path problem mentioned above, the variables are edges of the graphs and the weights associated with edges constituting a path between the two specified vertices determine the cost of this path.

Depending on the associated constraints, a combinatorial optimization problem can have one or more of the following three aspects:

1. *Subset selection:* Solving the problem involves choosing a subset of objects from a given set of objects. Well known problems containing subset selection aspect are knapsack problem, minimum spanning tree problem and maximum clique problem.
2. *Permutation:* Solving the problem involves arranging a given set of objects into a particular order. Well known problems containing permutation aspect are traveling salesman problem and flow shop scheduling problem.

3. *Grouping:* Solving the problem involves partitioning a given set of objects into various groups. Well known problems containing grouping aspect are clustering, bin packing problem and graph coloring problem.

These three aspects are not mutually exclusive, and solving a combinatorial optimization problem may involve more than one aspect. For example, consider the multiple traveling salesman problem (MTSP) which is an extension of TSP, where more than one ($m > 1$) salesmen are present to visit $n > m$ cities though each city must be visited exactly once by only one salesman. To solve this problem, one has to partition the given set of cities into m groups, and also has to find an optimal ordering of cities within each group. Clearly, it can be observed that MTSP has the aspects of grouping and permutation both. Now, consider the example of generalized traveling salesman problem (GTSP) which is another extension of TSP. Given a set of n cities partitioned into $1 \leq m \leq n$ clusters, the GTSP is to find a minimum length tour comprising exactly one city from each of these clusters. To solve this problem, one has to select exactly one city from each cluster and also has to find an optimal ordering of the selected cities. Clearly, it can be observed that GTSP has the aspects of subset selection as well as permutation. Likewise, the multiple knapsack problem (MKP) has the aspects of subset selection and grouping both. The k-interconnected multi-depot multi-traveling salesman problem (k-IMDMTSP) considered in this thesis has all the three aspects of subset selection, grouping and permutation as explained subsequently.

In the literature, one can find that there exist polynomial time algorithms for solving some combinatorial optimization problems like shortest path problem, minimum spanning tree problem etc., to optimality. On the other hand, many other combinatorial optimization problems are known to be $\mathcal{NP}$-hard. For these problems, the execution time of any known exact algorithm grows exponentially with increase in the size of problem's instance, and it is not possible to even verify the optimality of a proposed solution in polynomial time. It is to be noted that a problem may be $\mathcal{NP}$-hard in general, but instances of a particular kind can still be solved to optimality in polynomial time. For example, maximum clique problem and vertex cover problems can be solved to optimality in polynomial time on planar graphs even though they are proven to be $\mathcal{NP}$-hard. Therefore, for $\mathcal{NP}$-hard combinatorial optimization problems, the exact methods can be applied to either instances of small size or instances comprising polynomially solvable cases. Often, the nature and the size of the instances ruled out the applicability of exact methods, and one has to look towards those methods which do not guarantee optimality, but can find high quality solutions within a reasonable amount of time. *Approximate methods* [30] is an umbrella term for such methods. Developing approximate methods for solving different combinatorial optimization problems is an active area of research since the last five decades.

1.2 Approximate methods

Approximate methods do not guarantee optimality but can provide promising solutions. Generally, these methods run in polynomial time. These methods may return solutions whose objective function values are close to that of optimal solution, but these solution can be quite separated from the optimal solution in the search space. This is due to the fact that the search space of an $\mathcal{NP}$-hard problem is generally of rugged nature, and contains numerous locally optimal solutions some of whom have objective function values very close to that of optimal solution. Approximate methods can be broadly divided into four categories. These categories are mentioned below:

- *Heuristic* means an intuitive method that finds a good solution quickly by exploiting the structure of the problem under consideration. However, no guarantee exists on the quality of the solution returned by a heuristic. Usually, a heuristic performs well if it incorporates appropriate knowledge about the structure of the problem at hand.
- *Metaheuristic* refers to a high-level heuristic which directs underlying problem-specific heuristics towards promising regions of the search space [30]. Metaheuristics are generic in nature and can be adapted to different problems without much modifications. Most of the metaheuristics are stochastic algorithms and a solution returned by a metaheuristic depends on the sequence of random variables generated. Metaheuristics usually require the problem to be represented in an appropriate form before they can be applied. Some of the popular metaheuristics are genetic algorithms [31, 32], ant colony optimization [30, 33], tabu search [34, 35], artificial bee colony algorithm [36], variable neighborhood search [37, 38] etc.
- *Approximation algorithm* is a heuristic that returns a solution with proven solution quality which is within a certain factor of the optimal solution.
- Methods which are obtained by prematurely terminating an exact method. For example, a mixed integer linear programming solver for a problem can be terminated after some

specified amount of time, and the best solution found in that time can be returned in case it is able to find any solution in that time.

From the relevant literature, one can observe that for many combinatorial optimization problems theoretical results exist that preclude the possibility of development of good polynomial time approximation algorithms for them. Prematurely terminating an exact method may also turn out to be futile in many cases as it may yield either a very poor solution or no solution at all. In these situations, one has to opt for heuristics and metaheuristics for want of any other viable option. In most of these cases, a hybrid approach, where a metaheuristic technique is used in combination with one or more problem-specific heuristics, is employed to get a good approximate solution. Work reported in this thesis makes use of several heuristic and metaheuristic techniques, viz. artificial bee colony algorithm, variable neighborhood search, large neighborhood search, iterated local search, hyper-heuristic and genetic algorithm. Next six sections provide overview of these six techniques.

1.3 Overview of artificial bee colony algorithm

The artificial bee colony (ABC) algorithm is a recent population based metaheuristic approach developed by Karaboga [39] in 2005 upon getting inspiration from the foraging behavior of honey bee swarm. From then on, a number of variants of the basic ABC algorithm have been proposed, e.g. [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53]. By observing the foraging behavior of real honey bees, entomologists discovered that there are three kinds of foraging bees based on the nature of jobs they do, viz. employed, onlooker and scout bees. The employed bees are those bees which are currently tapping food sources. The onlooker bees are those bees which are waiting in the hive to choose their food sources, whereas the scout bees are those bees which are currently searching for new food sources in the vicinity of the hive. In a colony, the foraging commences when scout bees are dispatched in search of new food sources in a purely random fashion. Whenever a scout bee finds a food source, it starts tapping the newly found food source and it is reclassified as employed bee. The employed bees are responsible for transporting the loads of nectar from the food sources to the hive and then performing a dance in a designated location of the hive. The nature and duration of a dance performed by an employed bee depends on the quality (nectar content) of the food source currently being tapped by it. Onlooker bees watch these dances and choose a food source stochastically after watching a number of dances. The probability with which an onlooker bee chooses a food source is directly proportional to its nectar content. Higher the nectar content of a food source, more onlookers tend to choose that food source. After performing the dance, the employed bee returns back to its associated food source along with all those onlooker bees which chose this food source. All these onlooker bees also start tapping this food source and are reclassified as employed bees. Whenever, a food source becomes empty, then all employed bees associated with it abandon it. These employed bees now become either scouts or onlookers.

Drawing parallel between this foraging behavior and stochastic search process carried out by a metaheuristic for finding an optimal solution, the scout bees can be perceived as performing the job of exploration, whereas employed and onlooker bees can be perceived as performing the job of exploitation. This served as the motivation for Karaboga [39] to develop the ABC algorithm. In the ABC algorithm, each food source represents a prospective solution to the problem currently under investigation and the nectar content of a food source represents the fitness of the solution that the food source currently represents. Like real bees, in this algorithm also, artificial bees are of three kinds, viz. scout, employed, and onlooker bees. Every employed bee is associated with a unique food source (solution), hence, the number of employed bees is same as the number of food sources. Commonly, the number of onlooker bees are also taken to be equal to the number of employed bees, but the number of onlooker bees can be different from the number of employed bees. The employed bee always becomes a scout and never an onlooker after abandoning its associated food source. For such a scout, a new food source (solution) is generated and this scout bee is associated with this food source to make it again employed. The new food source for the scout can be generated either in a totally random manner or according to some specific strategy. *We will use food source and solution interchangeably throughout this thesis in the context of ABC algorithm.*

The ABC algorithm is an iterative algorithm. It starts by associating all employed bees with randomly generated food sources (solution). Then during each iteration, every employed bee determines a food source in the neighborhood of its currently associated food source and evaluates its nectar content (fitness). If its nectar content is better than that of its currently associated food source then that employed bee moves to this new food source abandoning the old one, otherwise it retains its old food source. When all employed bees finished this process, they share the nectar information of their food sources with onlookers, each of whom chooses a food source by following a stochastic selection strategy which usually prefers good food sources over poor ones.

The roulette wheel selection method [32] and the binary tournament selection method [54] are the two most commonly used methods for selecting a food source for an onlooker bee. In roulette wheel selection method, a food source is selected randomly from among all the food sources (employed bee solutions) in such a way that the probability of selecting a food source is proportional to its relative fitness in the employed bee population, i.e., the probability P_i of selecting a food source i is as follows:

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

where f_i is the fitness of the solution represented by the food source i and n is the total number of food sources.

In binary tournament selection method, two different food sources (employed bee solutions) are selected uniformly at random from the employed bee population and their fitnesses are compared. The food source with better fitness is chosen with probability P_{bts}. Otherwise the worse one is chosen, i.e., the probability of selection of worse of the two solutions is $(1 - P_{bts})$.

After all onlookers have chosen their food sources, each of them determines a new food source in the neighborhood of their associated food sources and calculates its fitness. The best food source among all the neighboring food sources determined by the onlookers associated with a particular food source 'i' and food source i itself is selected as new food source i for next iteration. If a solution corresponding to a particular food source does not improve for a predetermined number of iterations, then that food source is deemed to be empty and its associated employed bee abandons it to become a scout. The manner in which this scout is handled is already discussed in a previous paragraph. After the new location of each food source is determined, another iteration of ABC algorithm begins. This entire process is repeated until termination condition is satisfied. The manner in which a new food source is determined in the neighborhood of a particular food source for employed and onlooker bees is problem dependent and even for the same problem can vary from one implementation of ABC algorithm to another. Further, the manner in which a neighboring food source is determined for an employed bee can be different from that of an onlooker bee.

Basically, the ABC algorithm effectively utilizes a simple fact that in the neighborhood of good solutions, chances of finding even better solutions are higher in comparison to the neighborhood of poor solutions, and hence, the neighborhood of good solutions should be explored more thoroughly. This is the reason behind deputing more onlookers for exploring the neighborhood of good solutions. However, if a solution is locally optimal with respect to the

Algorithm 1: Pseudo code of basic ABC

```
Input: Set of parameters for ABC
Output: Best solution found
Initialization Phase: Initialize the solutions and associate them with employed bees;
repeat
    Employed Bee Phase: Move the employed bees to their neighboring solutions according to
    the fitness values;
    Onlooker Bee Phase: Move the onlookers to their neighboring solutions according to the
    fitness values;
    Memorize Best Solution: Memorize the best solution achieved so far;
    Scout Bee Phase: Move the scouts for searching new solutions;
until Termination condition satisfied;
return $best\ solution$;
```

entire neighborhood then it can not be improved no matter how many attempts we make. The concept of scout bees aids in recovering from this situation. Instead of determining whether a solution is locally optimal or not, which can be computationally expensive as it involves exploring the entire neighborhood, the ABC algorithm assumes a solution to be locally optimal and discards it, if it has not improved over certain number of iterations say '$limit_{scout}$'. The employed bee associated with this discarded solution becomes a scout. A new solution is generated for this scout to convert it back to employed. Hence, the concept of scout bees is useful in replacing those solutions, which have not improved since long and which can be locally optimal, with new solutions. For a search process to be effective over its entire duration, a proper balance need to be maintained between the exploration and exploitation throughout the search process. The parameter $limit_{scout}$ governs this balance in the ABC algorithm, and hence, this parameter needs to be set properly. A lower value of this parameter prefers exploration over exploitation, whereas a higher value favors exploitation over exploration. The pseudo code of basic ABC algorithm is given in Algorithm 1.

For a comprehensive survey on ABC algorithm and its uses, the interested reader is directed to [55].

1.4 Overview of iterated local search algorithm

Iterated local search (ILS) is a single solution based metaheuristic which iteratively improves the solution quality. According to [56], ILS has many of the desirable features: simple, easy to implement, robust, and highly effective. The ILS mainly consists of four components, viz. initial solution generation, local search, acceptance criteria and perturbation procedure. Starting

Algorithm 2: Pseudo-code of basic ILS

```
Input: Set of parameters for ILS
Output: Best solution found
$S \leftarrow$ Initial_Solution();
while Termination condition not satisfied do
    $S_1 \leftarrow$ Local_Search($S$);
    $S \leftarrow$ Acceptance_Criteria($S, S_1, history$);
    $S \leftarrow$ Perturbation_Procedure($S$);
return $best$;
```

from an initial solution, an iterative process ensues. During each iteration, first a local search algorithm is applied on the current solution to find a locally optimal solution. Then depending on the acceptance criteria, this newly obtained locally optimal solution may replace the current solution. In order to escape from that locally optimal solution, a perturbation procedure is applied on the current solution to get a perturbed solution and next iteration begins with that perturbed solution as the current solution. The two commonly used acceptance criterias are replacing the current solution with the newly obtained solution if it is better than the current solution, and always replacing the current solution with the newly obtained solution. The former criteria leads to a first improvement type of strategy, whereas the latter yields a random-walk sort of strategy. ILS has been successfully applied to many optimization problems and has shown its effectiveness when compared with other approaches [57]. The pseudo-code of the basic ILS is given in Algorithm 2.

1.5 Overview of general variable neighborhood search algorithm

This section provides a brief introduction to variable neighborhood search, variable neighborhood descent and general variable neighborhood search.

Variable neighborhood search (VNS) is a metaheuristic proposed by Mladenović and Hansen[37] based on the systematic search of different neighborhood structures. The VNS is composed of two phases, viz. local search and shake phases. The local search phase performs the job of exploitation (finds locally optimal solution), whereas the shake phase performs the job of exploration (get rid of locally optimal solution).

The pseudo code of basic VNS is given in Algorithm 3. Let S be a solution and $F(S)$ be its fitness value, and let $\mathcal{N} = \{N_1, N_2, \ldots, N_{p_{max}}\}$ be a set of p_{max} different neighborhood structures. The VNS algorithm starts by generating an initial (random) solution, and setting the

Algorithm 3: Pseudo code of basic VNS

```
Input: Set of parameters for VNS
Output: Best solution found
$S \leftarrow$ Generate_Initial_Solution();
while termination condition not satisfied do
    $p \leftarrow 1$;
    while $p \leq p_{max}$ do
        $S' \leftarrow$ Shake($S$, $N_p$);
        $S'' \leftarrow$ Local_Search($S'$, $N_p$);
        if $F(S'') < F(S)$ then
            $S \leftarrow S''$;
            $p \leftarrow 1$;
        else
            $p \leftarrow p + 1$;
return $S$;
```

neighborhood indicator variable p to 1, then the algorithm generates a random solution S' in the neighborhood N_p of S by using shake phase. The S' is used as input to the local search phase which tries to get an improved solution S'' in the neighborhood N_p of S'. If S'' is better than the current solution S, then S will be replaced with S'' and the algorithm continues with the first neighborhood structure (i.e., p is reset to 1), otherwise the algorithm moves to the next neighborhood structure. The termination condition can be maximum time, maximum number of iterations, or some other quality measurements. The basic VNS can be perceived as a method which combines deterministic and stochastic changes of the neighborhood.

Algorithm 4: Pseudo code of VND

```
Input: Set of parameters for VND
Output: Best solution found
$S \leftarrow$ Generate_Initial_Solution();
repeat
    $p \leftarrow 1$;
    while $p \leq p_{max}$ do
        $S' \leftarrow$ Local_Search($S$, $N_p$);
        if $F(S') < F(S)$ then
            $S \leftarrow S'$;
            $p \leftarrow 1$;
        else
            $p \leftarrow p$+1;
until no improvement exists in any of $p_{max}$ neighborhoods;
return $S$;
```

The variable neighborhood descent (VND) [37] is a simple variation of the basic VNS, where the neighborhood is changed in a deterministic manner. The pseudo code of VND is given in Algorithm 4. The VND starts with an initial solution S and p=1. The search starts by exploring the neighborhood N_1 until there is no improvement possible. And then, the search continues in the neighborhood N_2. If an improved solution is found in the neighborhood N_2, then the VND goes back to the N_1 to explore the first neighborhood of this newly improved solution until there is no improvement possible. Otherwise, VND continues with N_3, and so on. The final solution obtained by the VND is locally optimal with respect to all p_{max} neighborhoods. The VND is very often used as a local search method, as the chance of getting a good solution is high by using it in comparison to a single neighborhood structure. When VND is used as a local search, it starts with a solution that is passed to it as input instead of an initial (random) solution.

The general variable neighborhood search [58] is a variant of VNS [37], which uses VND for the local search phase. The general variable neighborhood search has been applied successfully to solve numerous combinatorial optimization problems.

1.6 Overview of large neighborhood search algorithm

Shaw [59] developed the large neighborhood search (LNS) metaheuristic in 1998. The LNS based methods are known for their efficiency in exploring the complex neighborhoods. LNS improves the solution quality in an iterative manner by employing destroy and repair methods. The basic LNS has three components, viz. initial solution generation, destroy and repair method, acceptance criteria. A LNS approach begins by generating an initial input solution, and then an iterative process ensues. During each iteration, first a destroy method is applied on the input solution to get an infeasible solution and then a repair method is applied on it to get a feasible neighboring solution. Usually, a destroy method is of stochastic nature and destroys the same solution in a different way upon each call. Depending on the acceptance criteria used, the newly generated neighboring solution may replace the input solution and the next iteration commences. There are two commonly used acceptance criterias. In the first criteria, the input solution is replaced by the neighboring solution only in case it is better than the input solution. This criteria yields a first improvement sort of strategy. In the second criteria, the neighboring solution always replaces the input solution irrespective of their fitnesses. So this criteria results into a random-walk type of strategy. Large neighborhood search methods have been successfully used

Algorithm 5: Pseudo-code for the basic LNS

```
Input: Parameter values required for the LNS
Output: Best solution found by the LNS
$Current$ ← Create_Initial_Solution();
$Best$ ← $Current$;
while Termination condition is not met do
    $Neighbor$ ← Repair(Destroy($Current$));
    $Current$ ← Acceptance_Criteria($Current$, $Neighbor$);
    if $Current$ is better than $Best$ then
        $Best$ ← $Current$;
return $Best$;
```

to solve numerous combinatorial optimization problems, and have been shown to be effective in comparison to other approaches [60, 61, 62, 63, 64, 65, 66, 67, 68, 69]. The LNS derives its name from the fact that the neighborhood implicitly defined by a destroy and repair method is quite large in comparison to the neighborhoods explicitly defined for other metaheuristics. The main idea of the large neighborhood search method is to facilitate the exploration of search space in situations where it is difficult to explore the search space by using small neighborhoods, i.e., when the problem has tightly constrained small neighborhoods.

The pseudo-code for the basic LNS is provided in Algorithm 5, where the function *Create_Initial_Solution*() creates a new solution and returns it. The function *Destroy*(S) needs a solution S as input and returns an infeasible solution after deleting/adding some components from/to S. *Repair*(S) is another function that takes as input an infeasible solution S & converts it into a feasible solution, and then returns this feasible solution. *Acceptance_Criteria*(S, S') returns a solution from two input solutions S and S' as per the acceptance criteria used.

1.7 Overview of hyper-heuristics

The hyper-heuristics can be regarded as high level strategies that can adjust themselves according to the aspects of the problem instance at hand, [70]. Since the last ten years or so, hyper-heuristics are getting an increasing attention from the research community due to their level of generality in solving a problem. Denzinger *et al.* [71] coined the term *hyper-heuristic* as a strategy to combine some artificial intelligence based approaches for automated theorem proving, and does not provide any formal definition of this term. However, the basic idea of automating the design and/or selection of heuristics is proposed in early 1960s by Fisher *et al.* [72] and

Crowston *et al.* [73]. Cowling *et al.* [4] described the hyper-heuristics as the heuristics which can choose the suitable heuristics from a pool of available heuristics for a combinatorial optimization problem. According to Burke *et al.* [70], the hyper-heuristics can be described as high level strategies that handle a pool of low-level heuristics and work either by choosing a heuristic or producing a new heuristic from the components of available heuristics at each step in the search process and utilizing the heuristic chosen/produced. A hyper-heuristic and a metaheuristic differ fundamentally. A hyper-heuristic works in the search space of heuristics, whereas a metaheuristic directly works in the search space of solutions to the problem under consideration. A hyper-heuristic tries to find the most appropriate heuristic to solve the problem under consideration in the search space of available heuristics, whereas a metaheuristic tries to find the best solution in the search space of solutions to the problem under consideration [74]. The motivation of developing hyper-heuristics arises due to the fact that the different heuristics may perform differently for different instances of a problem, and even for the same instance, the performance of an individual heuristic can be different during different stages of the search process. Therefore, one may get better solutions if several heuristics are utilized in an appropriate manner.

Based on their purpose, hyper-heuristics can be of two types.

- Selective hyper-heuristics: methodologies for choosing/selecting from available low-level heuristics.
- Generative hyper-heuristics: methodologies for producing new heuristics using elements of available low-level heuristics.

Figure 1.1 illustrates the general framework of a selective hyper-heuristic. The domain barrier acts as insulator between high-level search strategy and the low-level heuristics. The high-level search strategy selects and applies the low-level heuristic by considering only the domain independent information. However, the design of domain-specific heuristics used as low level heuristics plays an important role in the performance of a hyper-heuristic. Further, a hyper-heuristic can be hybridized with a metaheuristic. Usually, this is done in one of the two ways, i.e., either a metaheuristic is utilized as a low-level heuristic within a hyper-heuristic framework or a hyper-heuristic is employed for local search search within a mataheuristic framework. Our approach for FTSP is based on the former strategy.

The copious literature on hyper-heuristics proves their effectiveness in solving combinatorial optimization problems, particularly when dealing with problems involving multiple aspects

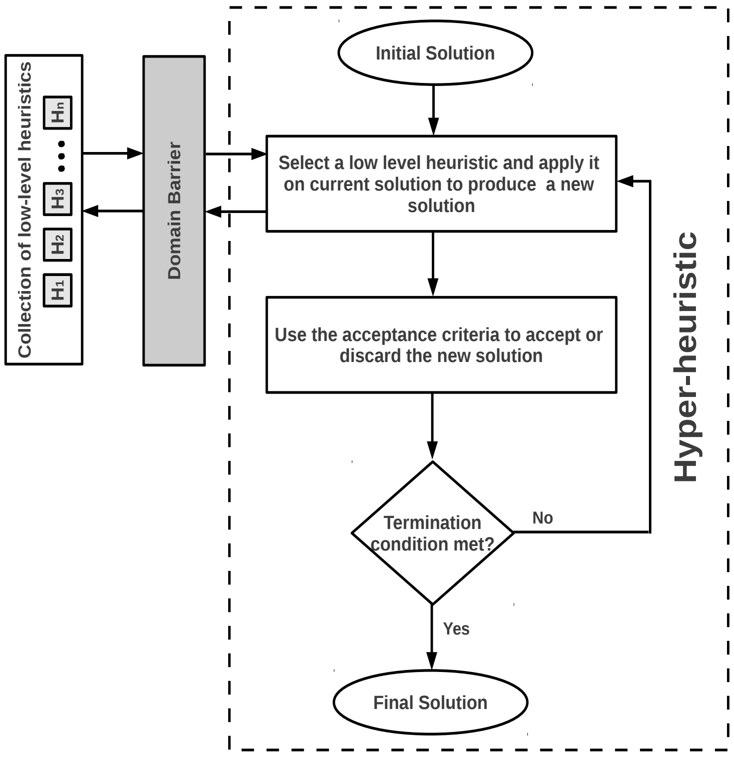

Collection of low-level heuristics
H1
H2
H3
Hn
Domain Barrier
Initial Solution
Select a low level heuristic and apply it on current solution to produce a new solution
Use the acceptance criteria to accept or discard the new solution
Termination condition met?
No
Yes
Final Solution
Hyper-heuristic

Figure 1.1: Framework of selective hyper-heuristic ([4], [5])

as they raise the level of generality. One may refer to [70] for a comprehensive survey on hyper-heuristics and their applications.

1.8 Overview of genetic algorithm

Genetic algorithm (GA) was introduced in 1960s by Holland [31] with the intention to mimic and study the evolutionary adaptation as it occurs in nature. GA has been widely used for generating high-quality solutions to various kinds of optimization and search problems. The success of GA is mainly due to its robustness and ability to explore various regions of the search space concurrently. GA provides both implicit parallelism and explicit parallelism. According to the schema theorem [75] proved by Holland, implicit parallelism is inherent in GA. Explicit parallelism may be achieved if different individuals of the population are manipulated and evaluated simultaneously.

GA starts by initializing a population of feasible solutions for the optimization problem under consideration. Generally, each individual in the population is generated randomly. However, for generating better initial solutions problem-specific information can be utilized. A fitness value is computed and assigned to each individual in the population by using a fitness function. Usually, the fitness function is same as the objective function, but it can be different also specially for those problems where there exist many solutions with same objective function value. These fitness values are used for comparing the solutions with one another. Once each and every solution in the population is assigned a fitness value, an iterative process ensues.

During each iteration or generation, according to a fitness-based selection mechanism, some members of the current population are selected to form a parent set through a selection mechanism. The selection mechanism ensures the principle of “survival of the fittest” by selecting solutions with better fitness into the parent set. This is due to the fact that the better solutions in the parent set are more likely to produce offspring with even higher fitness. Next, the solutions from the parent set are allowed to participate in the reproduction process in which the new solutions (offspring) are produced via genetic operators such as crossover and mutation. In crossover, generally two solutions (also called parents) are recombined to produce one or two new solutions (offspring). So, some components of each offspring will be inherited from one parent and the remaining will be inherited from the other parent. Whereas in mutation, an offspring will be generated from a single parent solution by making some random change in it according to the problem under consideration. These crossover and mutation operators can be applied in a sequential manner or in a mutual exclusive manner. These crossover and mutation operators are applied with their associated probabilities called crossover rate and mutation rate. The crossover rate decides what fraction of current generation will proceed to the next generation through crossover, whereas the mutation rate determines what fraction of solutions will be mutated. Since, crossover and mutation operators are applied with some probability, some individuals of current generation will proceed to the next generation without any change. This reproduction of solutions leads to a creation of new generation of the population. This process is repeated until the termination condition is satisfied. The termination condition may be a time constraint (amount of CPU time) or the total number of iterations or the number of consecutive iterations without improvement in quality of the best solution. The pseudo code of basic GA is given in Algorithm 6. Various components of GA are described in subsequent subsections.

Algorithm 6: Pseudo code of basic GA

```
Input: Set of parameters for GA
Output: Best solution found
Initialization Phase: Initialize the population with randomly generated solutions and evaluate
their fitness;
repeat
    Selection Phase: Perform parents selection according to a selection mechanism;
    Generation Phase: Apply mutation and/or crossover operators on selected parents to
    generate new offspring and evaluate the fitness;
    Population Replacement Phase: Select population for the next generation according to the
    population model used;
until Termination condition satisfied;
return $best\ solution$;
```

1.8.1 Crossover

Crossover is also known as recombination, and it is based on the idea that offspring generated by combining two different individuals have desirable features of both the parents. However, there is no denying about possibility of generating worse offspring, but some better offspring will be generated when crossover is applied many times. The nature of the problem under consideration and solution encoding play an important role in designing a suitable crossover operator. Numerous crossover operators have been proposed in the literature according to the type of problem and solution encoding strategies.

1.8.2 Mutation

The idea of mutation is to maintain sufficient diversity in the population in order to explore new regions of the search space. As part of mutation, a single parent is selected and a new offspring is produced by making some random change in it. Generally, there is no positional bias in mutation. For different types of problems and solution encoding schemes, different mutation operators have been proposed in the literature.

1.8.3 Population models

Population model decides the composition of next generation in terms of individuals' origin. Generational model and steady-state model are two main population models [76].

1.8.3.1 Generational model

In generational model, after each generation the entire population is replaced by the newly generated offspring.

1.8.3.2 Steady-state model

In steady-state model, only some members of the population are replaced by the same number of newly generated offspring in every generation. Generally, only a single offspring is generated during each generation and that replaces the worst fit member of the population. In steady-state model, best solutions are always kept in the population, and the newly generated offspring is immediately available for selection and reproduction. These two factors increase the possibility of finding better solutions faster. Moreover, in this model, duplicate solutions are prohibited in the population. This prevents premature convergence, which is a common phenomenon in generational model. Actually, in generational model, copies of the same solution may be present in the population. Generally, such solutions are high quality solutions, and because of that, within few generations they start dominating the entire population, thereby making crossover completely ineffective and mutation will be the only way to improve the solution quality. Therefore, even if there is any improvement in the solution quality, it is very slow. Such a phenomenon is termed as premature convergence.

1.8.4 Selection methods

Different selection methods basically vary in terms of the degree to which the better individuals in the population are favored and the degree of randomness employed to select the members of the parent set. There are many selection methods proposed in the literature [77]. Fitness proportionate selection [31], ranking selection [78], binary tournament selection [54] are commonly used selection methods

1.9 Scope of the thesis

This thesis is concerned with solutions of some $\mathcal{NP}$-hard TSP variants through various heuristic techniques, viz. artificial bee colony algorithm, variable neighborhood search, large neighborhood search, iterated local search, hyper-heuristic and genetic algorithm. We have considered

Table 1.1: List of the problems considered, their types and combinatorial optimization aspects

Problem name	**TSP variant type**	**Aspects of combinatorial optimization**
Maximum scatter traveling salesman problem	Single salesman, no depot	Permutation only
k-Traveling salesman problem	Single salesman, single depot	Permutation & subset selection
Family traveling salesman problem	Single salesman, single depot	
Covering salesman problem	Single salesman, no depot	
Generalized covering traveling salesman problem	Single salesman, single depot	
Colored traveling salesman problem	Multiple salesmen, single depot	Permutation & grouping
k-Interconnected multi-depot multi-traveling salesman problem	Multiple salesmen, multiple depots	Permutation, subset selection & grouping

seven problems in this thesis. These problems along with their TSP variant types and combinatorial optimization aspects are listed in Table 1.1, where by “no depot”, we mean a salesman can start his tour from any city. Please note that any TSP variant invariably has a permutation aspect to it. In addition, it can have subset selection and/or grouping aspect. Hence, these seven problems cover not only all possible combinations of combinatorial optimization aspects that can occur in a TSP variant, but also all types of TSP variants discussed earlier in this chapter. So these seven problems cover entire spectrum of TSP variants excluding those with roots in vehicle routing problem. These problems not only pose a serious challenge from theoretical point of view, but also have many real world applications in diverse domains such as transportation, logistics, manufacturing, networks, planning & scheduling, healthcare etc. *While describing each of these problems, we have followed the terminology available in the literature for that problem. In this thesis, we have used city, vertex, node and point interchangeably. Similarly, depot and home city are used interchangeably. Likewise, tour length and distance traveled by the salesman convey the same meaning throughout this thesis.*

This thesis is divided into nine chapters beginning with this introductory chapter. In the following, we summarize the content of the remaining eight chapters.

Chapter 2 deals with the maximum scatter traveling salesman problem (MSTSP). Similar to traveling salesman problem, MSTSP also seeks a Hamiltonian cycle. However, unlike TSP, in MSTSP, the objective of the MSTSP is to maximize the minimum length among all the constituent edges in the Hamiltonian cycle. The MSTSP finds important applications in manufacturing and medical imaging. In this chapter, we have proposed a multi-start iterated local search algorithm for the MSTSP. Two local search algorithms based on insertion and modified

2-opt moves have been developed as part of our approach. To investigate the performance of the proposed approach, benchmark instances from the standard TSPLIB are used. The computational results show the effectiveness of the proposed approach.

Chapter 3 presents two multi-start heuristic approaches for the k-traveling salesman problem (k-TSP). Given a set of n cities including a home city and a fixed value $1 < k \leq n$, the k-TSP is to find a minimum length tour which starts and ends at the home city and visits exactly k cities (including the home city) out of these n cities. Finding a feasible solution to k-TSP involves finding a subset of k cities including the home city first, and then a circular permutation representing a tour of these k cities. Among the two multi-start heuristic approaches for the k-TSP, the first approach is based on general variable neighborhood search algorithm (GVNS), whereas the latter approach is a hyper-heuristic (HH) approach. A variable neighborhood descent consisting of two neighborhood structures is used as local search in the GVNS. As part of the hyper-heuristic, two low level heuristics are considered. To the best of our knowledge, these are the first metaheuristic and hyper-heuristic approaches for the k-TSP. To evaluate the performance of our approaches, a set of benchmark instances is created utilizing instances from TSPLIB. Computational results on these benchmark instances show HH approach to be better than GVNS approach.

Chapter 4 concerned with a recently introduced variant of the generalized traveling salesman problem (GTSP) called the family traveling salesman problem (FTSP). Given a set of cities partitioned into multiple clusters termed as families, the FTSP is to find a tour visiting a pre-specified number of cities from each of these families in such a manner that the total distance traveled is minimized. To solve this problem efficiently, a hyper-heuristic approach is proposed that utilizes three large neighborhood search methods as low level heuristics. The solution obtained through the hyper-heuristic approach is improved further by using a local search phase. To investigate the performance of the proposed approach, computational experiments are performed on the 21 standard benchmark instances of the problem. The proposed approach obtained high-quality solutions in comparison to the state-of-the-art approaches on these instances. The proposed approach is able to find solutions as good as or better than current best known solutions for 17 out of 21 instances. Moreover, our approach is several times faster on most of the large instances. We have also reported the performance of our approach on a set of 60 new benchmark instances which are generated by utilizing the instances from the standard TSPLIB.

Chapter 5 addresses the covering salesman problem (CSP), which is an extension of the classical traveling salesman problem (TSP). Given a set of cities and a coverage radius associated with each one of them, the CSP seeks a Hamiltonian cycle over a subset of cities such that each city not in the subset is within the coverage radius of at least one city in the subset and that has minimum length among all Hamiltonian cycles over such subsets. The applications of CSP arises in emergency& disaster management and rural healthcare. In this chapter, we have proposed two hybrid metaheuristic approaches for the CSP. The first approach is based on the artificial bee colony algorithm, whereas the latter approach is based on the genetic algorithm. Both the approaches make use of a local search strategy with two neighborhood structures. Computational results on a wide range of benchmark instances demonstrate the superiority of our approaches over other state-of-the-art approaches in terms of solution quality.

Chapter 6 is devoted to the generalized covering traveling salesman problem (GCTSP). The GCTSP is a recently introduced variant of the covering salesman problem. Given a demand D and a set of cities that includes depot, facilities and customer cities. A coverage radius r_i is associated with each facility i, and a demand d_j is associated with each customer j. The objective of the GCTSP is to find a minimum length tour over a subset of facilities so that the sumtotal of demands of customers covered by this subset of facilities is at least D. A customer is said to be covered by a subset of facilities if it is within the coverage radius of one or more facilities. This problem finds important applications in humanitarian relief transportation and telecommunication networks. In this chapter, we have proposed an artificial bee colony algorithm with variable degree of perturbation for the GCTSP where the degree to which a solution is perturbed for generating its neighboring solution is reduced over iterations. Computational results on a wide range of benchmark instances and their analysis show the effectiveness of our proposed approach in comparison to other state-of-the-art approaches.

Chapter 7 deals with the recently introduced colored traveling salesman problem (CTSP), which is a variant of the multiple traveling salesman problem (MTSP). In the MTSP, given a set of cities, there are multiple salesmen to visit these cities though each city must be visited exactly once by one salesman only. On the other hand in case of the CTSP, every salesman has his exclusive cities to visit and a group of shared cities that are shared among different salesmen but should be visited exactly once by one salesman only. In this chapter, an artificial bee colony (ABC) algorithm based approach is proposed for the CTSP and its superiority over the state-of-the-art approaches is demonstrated experimentally in terms of both solution quality and computation time on the benchmark instances available in the literature. In addition, the encoding scheme that we have used to represent a CTSP solution within the ABC algorithm is theoretically analyzed and it is shown that our encoding scheme yields a solution space that is considerably smaller than the scheme used by the state-of-the-art approaches for the CTSP.

The second last chapter of the thesis, i.e., Chapter 8 is concerned with a problem which has all the three aspects of a combinatorial optimization problem, viz. the k-interconnected multi-depot multi-traveling salesman problem (k-IMDMTSP). Given a set of cities, this problem consists in finding a subset of cities of size k for locating the depots so that from each depot a salesman can begin and end the tour to visit the remaining cities. Each city should be visited exactly once by only one salesman. The k depots also need to be connected via a cycle which is termed as inner cycle, whereas each salesman's tour is termed as outer cycle. The objective of k-IMDMTSP is to minimize the sum total of the costs of inner and outer cycles. There are several practical applications of k-IMDMTSP such as designing cheaper ring networks with link-failure tolerance, hub location problems in telecommunication networks, transportation networks where multiple echelons are used, location-routing problems. This problem has the potential to address a variety of problems as it is a general problem that can change its aspects according to the combination of its constituent parameter values. In fact, k-IMDMTSP can become an altogether different problem depending on the choice of its parameter values. According to the No Free Lunch Theorem, it is not possible to have a general algorithm that can outperform all algorithms across all problems emanating from k-IMDMTSP due to various parameter values. However, an appropriate combination of different algorithms can successfully deal with all such problems emanating from k-IMDMTSP. Here, we have made an attempt in this direction with the help of hyper-heuristics. A hyper-heuristic based artificial bee colony algorithm is proposed for k-IMDMTSP. A new solution encoding scheme is proposed for representing a k-IMDMTSP solution within the proposed approach, and its associated search space is analyzed theoretically. It has been proved that our encoding scheme yields a search space that is considerably smaller in comparison to encoding schemes used previously. Experimental results on standard benchmark instances show that the proposed approach outperforms other state-of-the-art approaches available in literature in terms of both solution quality and running time.

Chapter 9 concludes the thesis by summarizing the contributions of the thesis. In addition, some guidelines and directions for future research are also provided.

Chapter 2

Maximum Scatter Traveling Salesman Problem

2.1 Introduction

This chapter is concerned with a TSP variant called maximum scatter traveling salesman problem (MSTSP). Like TSP, MSTSP also seeks a Hamiltonian cycle. However, unlike TSP, in MSTSP, the goal is to maximize the length of an edge which is having minimum length among all the edges in the Hamiltonian cycle. Figure 2.1 illustrates the difference between TSP and MSTSP on a TSPLIB graph instance **bayg29** which has 29 cities. It can be clearly seen that TSP solution tries to decrease the total distance traveled by the salesman, whereas MSTSP solution tries to maximize the minimum edge cost by making any two consecutive cities in the tour as much scattered as possible.

The MSTSP was introduced by Arkin *et al.* [79] by getting motivation from the application of it in manufacturing and medical imaging problems. The MSTSP is also referred to as the max-min 1-neighbor TSP. The general version of the MSTSP is the max-min m-neighbor TSP, in which the goal is to maximize the minimum cost of the edges between any city and all of its m-neighbors in the cycle. According to Arkin *et al.* [79], the manufacturing application of the MSTSP is encountered in their discussion with Boeing. In order to heat a workpiece, it is important to order the riveting operations as scattered as possible so as to allow the cooling time between the operations and the distance between neighboring rivets is maximum [80, 81]. Application of MSTSP in the medical imaging was studied by Arkin *et al.* [79] and Penavic [82]. While using a dynamic spatial reconstructor for imaging physiological functions, the

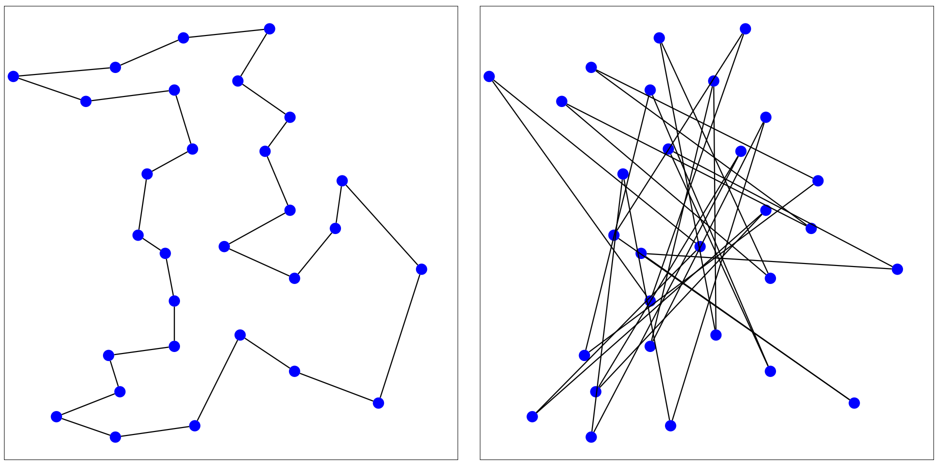

(a) TSP solution (b) MSTSP solution

Figure 2.1: Illustration of MSTSP by using instance **bayg29**

radiation sources are positioned along the top half of a circular ring and sensors are positioned directly opposite to radiation sources along the bottom half of the ring. The “firing sequence” governs the order in which the sources and their corresponding sensors are activated, usually in a periodic pattern. Two consecutive sources in a firing sequence have to be as much scattered as possible so as to avoid the interference between the radiations of them. This is clearly an instance of MSTSP.

There are many studies on the MSTSP proposing new approximation algorithms in different contexts of the MSTSP [83, 84, 85]. However, no metaheuristic approach has been proposed so far for this problem.

The multiple salesmen version of the MSTSP (MMSTSP) was proposed by Dong *et al.* [86]. While proposing the MMSTSP, they developed three variants of genetic algorithm (GA) to solve the MMSTSP. These three algorithms are based on greedy initialization (GAG), hill-climbing algorithm (HCGA), and simulated annealing algorithm (SAGA). The computational experiments demonstrated the effectiveness and distinct characteristics of these algorithms. Dong *et al.* [86] cited one potential application of the MSTSP in a situation where a person falsely accused of a murder that he did not commit and facing the capital punishment, escapes from the prison and starts a journey across the country to avoid getting recaptured. Obviously, such a person will look for a tour through his route network of safe places so that the smallest distance between consecutive locations is as large as possible [87].

A closely related problem to the MSTSP is the bottleneck traveling salesperson problem (BTSP), in which the goal is to minimize the maximum edge length in a Hamiltonian cycle [11, 88]. Maximum traveling salesman problem (MAX TSP) is another closely related problem to the MSTSP, where the goal is to find a maximum length Hamiltonian cycle [89].

There are many other applications of the MSTSP, which have been studied in the literature [90, 91, 92, 93].

In this chapter, we have proposed a multi-start iterated local search (MS-ILS) algorithm for the MSTSP and evaluated its performance on test instances from TSPLIB. Computational results show the effectiveness of the proposed approach.

The remainder of this chapter is organized as follows. Section 2.2 provides the formal definition of the MSTSP. The proposed MS-ILS approach for the MSTSP is described in Section 2.3. Section 2.4 presents the computational results on the instances from the TSPLIB. Finally, Section 2.5 contains some concluding remarks.

2.2 Problem definition

Given an undirected edge-weighted complete graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of cities, $E = \{(i, j)|i, j \in V\}$ is the set of edges, and a length or cost d_{ij} is associated with each edge $(i, j) \in E$. The MSTSP seeks a Hamiltonian cycle that maximizes the length of the edge which is having minimum length among all its constituent edges. By introducing a binary variable x_{ij} to indicate whether edge (i, j) is a part of Hamiltonian cycle ($x_{ij} = 1$) or not ($x_{ij} = 0$), the mathematical model for the MSTSP can be represented as follows:

$$\text{Maximize} \quad \min_{(i,j)\in H} d_{ij}, \quad H = \{(i, j)|x_{ij} = 1\} \tag{2.1}$$

subject to:

$$\sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk} = 1, \quad \forall j \in V, \tag{2.2}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V \tag{2.3}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E \tag{2.4}$$

Equation (2.1) is the objective function for the MSTSP and it maximizes the minimum edge cost. Equation (2.2) satisfies the constraints of indegree and outdegree of the cities as every city must have exactly one indegree and one outdegree in the tour. Equation (2.3) represents the sub tour elimination constraint. Equation (2.4) enforces the binary nature of the decision variable x_{ij}.

2.3 Multi-start iterated local search algorithm for the MSTSP

The proposed multi-start iterated local search (MS-ILS) for the MSTSP is an extension of the ILS and restarts the ILS multiple times, each time starting with a new solution generated by our initial solution generation procedure. The motivation for choosing the multi-start mechanism is due to the fact that the unproductive iterations may consume more time, and restarting the search from a newly generated initial solution yielded better solutions. The main components of our proposed approach are discussed in the following subsections.

2.3.1 Solution encoding and fitness

Within the MS-ILS approach, a solution is encoded by a linear permutation of the cities where the first city always occupies the first position. Actually, a solution to TSP is a circular permutation and a single circular permutation involving n cities can be represented using any of n linear permutations each beginning with a different city. By restricting the first city to first position, we have removed this redundancy in representation. Please note that none of the components of MS-ILS can modify the position of the first city.

The objective function presented in Equation (2.1) is used as the fitness function, i.e., the fitness of a solution is the minimum edge cost in the tour. As MSTSP is a maximization problem, a solution having a higher value of the fitness function is considered to be better than a solution having a lower value.

2.3.2 Initial solution generation

The initial solution generation procedure starts by selecting two cities uniformly at random and then an iterative process ensues. During each iteration, a city is selected uniformly at random and inserted in between the cities associated with the edge of minimum cost. This procedure is repeated until all the cities are inserted into the tour.

Algorithm 7: Pseudo-code for perturbing a solution

```
Input: A solution $S$
Output: A perturbed solution $S_1$

function Perturbation_Procedure($S$)
begin
    foreach city $c$ in tour of $S$ do
        Generate a random number $0 \leq \rho \leq 1$;
        if $\rho < P_{pert}$ then
            Add $c$ to a set of unassigned cities;
        else
            Copy $c$ to tour of $S_1$;
    foreach city $c$ in the set of unassigned cities in some random order do
        Follow the procedure described in Section 2.3.2 to insert $c$ into tour of $S_1$;
    return $S_1$;
```

2.3.3 Local search and perturbation procedures

The local search and perturbation procedures play a vital role in the ILS as they control the behavior of the search. In essence, an algorithm performs well if it maintains the delicate balance between the convergent and divergent behavior of the search. The local search tries to find a better solution by searching in the neighborhood of the current solution. The goal of the perturbation procedure is to escape from the present locally optimal solution by perturbing it, and providing a new starting solution to the local search to move the search to unexplored regions in the search space. In essence, the local search procedures do the job of exploitation, whereas the perturbation procedures do the job of exploration.

The local search procedure of the proposed MS-ILS consists of two heuristics h_1 and h_2, each of which has to handle two cases. In the first case (referred to as case 1 subsequently), there exists a single minimum cost edge, whereas in the second case (referred to as case 2 subsequently) there can be multiple edges with minimum cost. In the first case, these two heuristics will try to maximize the minimum edge cost, whereas in the latter case these heuristics will try to reduce the number of minimum cost edges by as much as possible. Accordingly, the best solutions are defined for these two cases. Please note that in latter case unless and until all minimum cost edges are replaced, minimum edge cost can not be increased, and that is why this latter case is processed differently from the former.

2.3.3.1 Insertion between the cities of minimum cost edge (h_1)

For case 1, this heuristic inserts a city between the cities of minimum cost edge in order to maximize the minimum edge cost. As part of this heuristic, each and every city has to be tried for insertion between the cities of minimum cost edge and the best among all the resulting solutions is accepted. For case 2, heuristic considers each minimum cost edge one-by-one in the order in which these edges occur in the solution. Again each city is inserted one-by-one between the cities of edge under consideration to reduce the number of minimum cost edges and the best among all the resulting solutions is accepted. If the number of minimum cost edges got reduced then this heuristic terminates and h_2 starts. Otherwise, next minimum cost edge is considered.

2.3.3.2 Minimum cost edge centric 2-opt move (h_2)

In a 2-opt move, basically two edges are removed from the tour and the resulting two paths are reconnected through two edges. In 2-opt move, the best tour is obtained after trying every pair of edges in the tour. Figure 2.2 provides an illustration of 2-opt move. In this figure, the two blue colored edges are removed from the tour and the two red colored edges are used to reconstruct the tour. The proposed heuristic is a modified version of 2-opt move, in which the minimum cost edge is always one of the two edges to be removed. In our heuristic, for case 1, to maximize the minimum edge cost, every other edge has to be tried with the minimum cost edge for removal and two new edges needs to be inserted at their place. The move that yields the maximum increase in minimum edge cost is accepted. For case 2, this heuristic proceeds in the same manner as in h_1 and control is passed to h_1 as soon as the number of minimum cost edges got reduced.

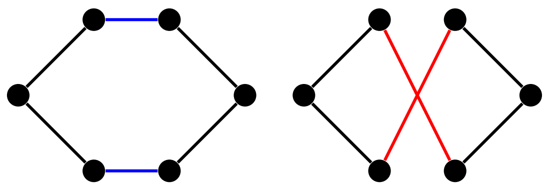

(a) Before 2-opt move (b) After 2-opt move

Figure 2.2: Illustration of 2-opt move

We have observed that instead of using h_1 and h_2 iteratively till no improvement, passing control to one another as soon as solution improves yields a better final solution. Hence, h_1 and h_2 are used in an interspersed manner.

2.3.3.3 Perturbation procedure

This perturbation procedure is helpful in escaping from locally optimal solution. If none of the heuristics h_1 and h_2 is able to improve the solution in terms of minimum edge cost or number of edges having minimum edge cost, then this perturbation procedure is used. As part of this procedure, destroy and repair mechanism is used. Each city from the tour is removed with a probability P_{pert}. All such removed cities are added back to the tour in an iterative manner similar to the initial solution generation procedure described in Section 2.3.2. Algorithm 7 provides the pseudo-code for the procedure to perturb a solution.

2.3.4 Acceptance criteria

Our acceptance criteria compare the quality (fitness) of the solution generated by the local search procedure with the solution before applying this procedure. An improved fitness solution is accepted all the times. The equal fitness solution is accepted in cases where there are multiple edges with the minimum edge cost and there is a decrease in the number of edges having that cost. Upon failing all the above mentioned cases, the perturbation procedure is applied and it may return a better or worse fitness solution. The search process starts from this newly returned solution.

The pseudo-code of the proposed MS-ILS approach is given in Algorithm 8.

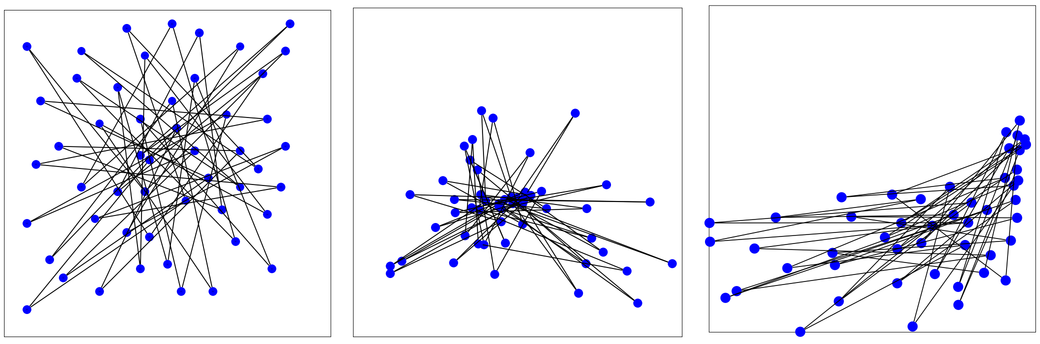

(a) MSTSP Solution for **eil51** (b) MSTSP Solution for **berlin52** (c) MSTSP Solution for **att48**

Figure 2.3: Plots of the solutions obtained by MS-ILS approach on different TSPLIB instances

Algorithm 8: Pseudo-code of the proposed approach MS-ILS for the MSTSP

```
Input: Set of parameters for the MS-ILS and a MSTSP instance
Output: Best solution found

$F(best) := -\infty$;
for $st := 1$ to $N_{rst}$ do
    $S \leftarrow$ Initial_Solution();
    while Termination condition not satisfied do

        /**********Applying heuristic $h_1$**********/
        $S_1 \leftarrow$ Apply_Heuristic_$h_1(S)$;
        if $F(S_1) > F(S)$ then
            $S \leftarrow S_1$;
        else if $F(S_1) := F(S)$ then
            if no.of edges with $F(S)$ decreased then
                $S \leftarrow S_1$;

        /**********Applying heuristic $h_2$**********/
        $S_1 \leftarrow$ Apply_Heuristic_$h_2(S)$;
        if $F(S_1) > F(S)$ then
            $S \leftarrow S_1$;
        else if $F(S_1) := F(S)$ then
            if no.of edges with $F(S)$ decreased then
                $S \leftarrow S_1$;

        /**********Dealing with best solution and local optimal solution**********/
        if $F(S) > F(best)$ then
            $best \leftarrow S$;
        else if $F(S_1) < F(S)$ then
            $S \leftarrow$ Perturbation_Procedure($S$);
return $best$;
```

2.4 Computational results

Our approach MS-ILS is tested on eighty two instances from the standard TSPLIB . The number of cities in these instances ranges from 14 to 1889 . Each of these instances represents an edge-weighted complete graph using an $n \times n$ distance matrix. MS-ILS is executed on each test instance ten times independently, each time with a different random seed. MS-ILS is implemented in C and executed on a Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM.

In all our experiments with the MS-ILS, we have used the following parameters – MS-ILS is restarted 100 times, i.e., N_{rst}=100, P_{pert}=0.2, and the MS-ILS terminates when there is no improvement in the solution continuously for 100 iterations. All these parameter values are chosen empirically after a large number of trails. For comparing the relative performance of heuristics, the MS-ILS is executed individually with heuristic h_1 (referred to as MS-ILS(h_1)), heuristic h_2 (referred to as MS-ILS(h_2)), and both the heuristics h_1 and h_2 (referred to as MS-ILS($h_1 + h_2$)).

Table 2.1 reports the performance of the three MS-ILS variants, viz. MS-ILS(h_1), MS-ILS(h_2), and MS-ILS($h_1 + h_2$). In this table, the first column represents the name of the instance. The second column (n) reports the number of cities. For each of the three approaches, the columns (Best, Worst & Average) report the best, worst and average costs over ten independent runs respectively. The best values are reported in bold to facilitate easy identification. The column (Time) reports the average execution times of ten different runs. The second last row named 'Overall' reports the overall average values of each column. The last row named 'NBV' reports the number of instances on which a best value is obtained by the corresponding MS-ILS variant. Out of the 82 instances, the MS-ILS($h_1 + h_2$) got the best values for 79 and 66 instances for the best and average objective function values respectively. The MS-ILS(h_2) got the best values for 71 instances for worst objective function value. The MS-ILS($h_1 + h_2$) got the best values 2561.23, 2552.30 and 2557.12 for average of Best, Worst and Average objective function values respectively over 82 instances.

This table shows that the performances of MS-ILS($h_1 + h_2$) and MS-ILS(h_2) are competitive with one another. However, MS-ILS($h_1 + h_2$) seems to perform slightly better in terms of overall solution quality as indicated by the values in second last row. When it comes to the comparison between two individual heuristics, MS-ILS(h_2) performed better than MS-ILS(h_1). The MS-ILS(h_1) performed worse than both MS-ILS(h_2) and MS-ILS($h_1 + h_2$). Figure 2.3 plots the solutions obtained by our MS-ILS($h_1 + h_2$) approach for three instances, viz. **bayg29**, **berlin52** and **att48**.

Table 2.1: Results of MS-ILS(h_1), MS-ILS(h_2) and MS-ILS($h_1 + h_2$)

Instance	n	MS-ILS(h_1)				MS-ILS(h_2)				MS-ILS($h_1 + h_2$)			
		Best	**Worst**	**Average**	**Time**	**Best**	**Worst**	**Average**	**Time**	**Best**	**Worst**	**Average**	**Time**
burma14	14	**498.00**	482.00	495.00	0.02	**498.00**	**498.00**	**498.00**	0.01	**498.00**	**498.00**	**498.00**	0.03
ulysses16	16	**677.00**	656.00	664.70	0.03	**677.00**	**677.00**	**677.00**	0.01	**677.00**	**677.00**	**677.00**	0.03
gr17	17	**239.00**	211.00	232.20	0.03	**239.00**	**239.00**	**239.00**	0.01	**239.00**	**239.00**	**239.00**	0.04
gr21	21	**370.00**	350.00	362.50	0.04	**370.00**	**370.00**	**370.00**	0.02	**370.00**	**370.00**	**370.00**	0.06
ulysses22	22	667.00	650.00	658.50	0.06	**687.00**	**687.00**	**687.00**	0.02	**687.00**	**687.00**	**687.00**	0.06
gr24	24	150.00	146.00	147.80	0.06	**164.00**	**164.00**	**164.00**	0.02	**164.00**	**164.00**	**164.00**	0.07
fri26	26	93.00	92.00	92.30	0.06	**102.00**	**102.00**	**102.00**	0.03	**102.00**	**102.00**	**102.00**	0.09
bayg29	29	169.00	162.00	165.10	0.09	**189.00**	**189.00**	**189.00**	0.03	**189.00**	**189.00**	**189.00**	0.11
bays29	29	216.00	200.00	208.60	0.09	**231.00**	**221.00**	229.00	0.03	**231.00**	**221.00**	**230.00**	0.11
dantzig42	42	65.00	63.00	64.00	0.14	**73.00**	**73.00**	**73.00**	0.08	**73.00**	71.00	72.60	0.21
swiss42	42	107.00	103.00	105.50	0.16	**129.00**	**129.00**	**129.00**	0.08	**129.00**	124.00	128.30	0.22
att48	48	961.00	898.00	919.90	0.22	**1103.00**	**1103.00**	**1103.00**	0.10	**1103.00**	**1103.00**	**1103.00**	0.27
gr48	48	466.00	453.00	458.50	0.19	**558.00**	**555.00**	**557.10**	0.10	**558.00**	545.00	555.40	0.29
hk48	48	919.00	902.00	910.60	0.19	**1098.00**	**1094.00**	**1097.20**	0.10	**1098.00**	1089.00	1095.80	0.28
eil51	51	33.00	32.00	32.60	0.19	**39.00**	**39.00**	**39.00**	0.11	**39.00**	**39.00**	**39.00**	0.31
berlin52	52	476.00	455.00	461.00	0.24	**541.00**	**541.00**	**541.00**	0.11	**541.00**	**541.00**	**541.00**	0.32
brazil58	58	1634.00	1560.00	1587.30	0.24	**1906.00**	**1906.00**	**1906.00**	0.14	**1906.00**	**1906.00**	**1906.00**	0.39
st70	70	57.00	55.00	56.00	0.41	**63.00**	**63.00**	**63.00**	0.21	**63.00**	**63.00**	**63.00**	0.54
eil76	76	35.00	33.00	33.80	0.39	**41.00**	**41.00**	**41.00**	0.26	**41.00**	**41.00**	**41.00**	0.66
pr76	76	7512.00	7046.00	7288.60	0.42	**9214.00**	9155.00	9208.10	0.27	**9214.00**	**9214.00**	**9214.00**	0.70
gr96	96	3954.00	3876.00	3919.60	0.64	**4778.00**	**4756.00**	**4763.90**	0.43	**4778.00**	**4756.00**	4763.50	1.09
rat99	99	86.00	84.00	84.80	0.63	**111.00**	**111.00**	**111.00**	0.47	**111.00**	**111.00**	**111.00**	1.19
kroA100	100	1778.00	1708.00	1732.80	0.72	**2101.00**	**2101.00**	**2101.00**	0.42	**2101.00**	**2101.00**	**2101.00**	1.08
kroB100	100	1665.00	1617.00	1645.90	0.65	**1935.00**	**1933.00**	1934.00	0.43	**1935.00**	**1933.00**	**1934.20**	1.11
kroC100	100	1831.00	1703.00	1744.30	0.65	**2253.00**	2226.00	2237.10	0.48	**2253.00**	**2230.00**	**2242.00**	1.25
kroD100	100	1673.00	1604.00	1628.20	0.68	**2067.00**	**2042.00**	**2052.60**	0.47	**2067.00**	2027.00	2047.10	1.22
kroE100	100	1742.00	1698.00	1717.90	0.68	**2002.00**	**1995.00**	**1999.90**	0.44	**2002.00**	1977.00	1995.60	1.11
rd100	100	582.00	577.00	578.90	0.65	**672.00**	**672.00**	**672.00**	0.42	**672.00**	**672.00**	**672.00**	1.09
eil101	101	36.00	34.00	34.80	0.69	**44.00**	**44.00**	**44.00**	0.48	**44.00**	**44.00**	**44.00**	1.23
lin105	105	1151.00	1108.00	1129.50	0.74	**1474.00**	**1460.00**	1465.90	0.49	**1474.00**	**1460.00**	**1467.50**	1.28
pr107	107	**6826.00**	**6826.00**	**6826.00**	0.71	6826.00	**6826.00**	**6826.00**	0.46	**6826.00**	**6826.00**	**6826.00**	1.18
gr120	120	443.00	430.00	436.30	1.01	**563.00**	561.00	562.60	0.66	**563.00**	**563.00**	**563.00**	1.69
pr124	124	5985.00	5852.00	5908.80	1.02	**7377.00**	**7377.00**	**7377.00**	0.67	**7377.00**	**7377.00**	**7377.00**	1.71
bier127	127	3856.00	3748.00	3817.90	1.11	**4828.00**	**4825.00**	4825.30	0.71	**4828.00**	**4825.00**	**4825.60**	1.81
ch130	130	372.00	363.00	367.40	1.08	**458.00**	**458.00**	**458.00**	0.78	**458.00**	**458.00**	**458.00**	1.99
pr136	136	6732.00	6498.00	6545.70	1.44	**8237.00**	**8237.00**	**8237.00**	0.81	**8237.00**	**8237.00**	**8237.00**	2.05
gr137	137	4399.00	4225.00	4317.40	1.16	**5398.00**	5367.00	5392.80	0.84	**5398.00**	**5377.00**	**5394.50**	2.11
pr144	144	6340.00	6047.00	6197.40	1.30	**7224.00**	**7166.00**	**7218.20**	0.94	**7224.00**	**7166.00**	**7218.20**	2.41
ch150	150	379.00	369.00	372.80	1.46	**454.00**	**454.00**	**454.00**	0.98	**454.00**	**454.00**	**454.00**	2.51
kroA150	150	1757.00	1709.00	1732.00	1.38	**2153.00**	**2153.00**	**2153.00**	0.93	**2153.00**	**2153.00**	**2153.00**	2.37
kroB150	150	1771.00	1708.00	1742.20	1.39	**2082.00**	**2082.00**	**2082.00**	0.94	**2082.00**	**2082.00**	**2082.00**	2.40
pr152	152	7247.00	7118.00	7147.50	1.56	8189.00	**8105.00**	8137.10	1.07	**8231.00**	**8105.00**	**8142.80**	2.71
u159	159	2884.00	2823.00	2854.30	1.53	**3406.00**	**3406.00**	**3406.00**	1.03	**3406.00**	**3406.00**	**3406.00**	2.61
si175	175	288.00	284.00	285.30	1.97	**304.00**	**304.00**	**304.00**	1.24	**304.00**	**304.00**	**304.00**	3.05
brg180	180	**9000.00**	**9000.00**	**9000.00**	1.80	**9000.00**	**9000.00**	**9000.00**	1.28	**9000.00**	**9000.00**	**9000.00**	3.11

continued ...

Instance	**n**	**MS-ILS(h_1)**				**MS-ILS(h_2)**				**MS-ILS($h_1 + h_2$)**			
		Best	**Worst**	**Average**	**Time**	**Best**	**Worst**	**Average**	**Time**	**Best**	**Worst**	**Average**	**Time**
rat195	195	120.00	117.00	117.60	2.20	**152.00**	**152.00**	**152.00**	1.67	**152.00**	**152.00**	**152.00**	4.09
d198	198	593.00	573.00	581.40	2.26	**738.00**	**738.00**	**738.00**	1.58	**738.00**	**738.00**	**738.00**	3.92
kroA200	200	1760.00	1707.00	1725.20	2.33	**2249.00**	**2236.00**	2242.60	1.86	**2249.00**	**2236.00**	**2244.90**	4.63
kroB200	200	1685.00	1626.00	1653.30	2.33	**2107.00**	2078.00	2100.90	1.74	**2107.00**	**2099.00**	**2104.60**	4.33
gr202	202	1112.00	1099.00	1105.90	2.60	**1462.00**	1449.00	**1453.10**	1.85	1452.00	**1451.00**	1451.90	4.55
ts225	225	7433.00	7159.00	7245.00	3.11	**8485.00**	**8485.00**	**8485.00**	2.01	**8485.00**	**8485.00**	**8485.00**	4.86
tsp225	225	182.00	179.00	180.80	3.20	**232.00**	**231.00**	**231.30**	2.40	**232.00**	**231.00**	231.10	5.71
pr226	226	8099.00	7884.00	7996.80	2.88	**9360.00**	**9360.00**	**9360.00**	2.09	**9360.00**	**9360.00**	**9360.00**	5.05
gr229	229	5297.00	5169.00	5223.80	3.04	**6871.00**	6820.00	6828.70	2.39	**6871.00**	**6821.00**	**6850.60**	5.97
gil262	262	106.00	104.00	105.00	4.22	**129.00**	**129.00**	**129.00**	3.01	**129.00**	128.00	128.90	7.27
pr264	264	5543.00	5398.00	5450.10	5.11	**6562.00**	6546.00	6555.30	3.86	**6562.00**	**6552.00**	**6556.70**	9.44
a280	280	127.00	124.00	125.00	4.33	**148.00**	**148.00**	**148.00**	3.09	**148.00**	**148.00**	**148.00**	7.36
pr299	299	2602.00	2535.00	2580.20	4.92	3454.00	**3431.00**	3438.00	4.68	**3471.00**	**3431.00**	**3451.60**	11.72
lin318	318	1888.00	1824.00	1846.80	6.31	**2396.00**	2374.00	**2386.70**	5.24	2387.00	**2381.00**	2385.70	12.45
rd400	400	553.00	544.00	548.50	10.17	**698.00**	**693.00**	**695.60**	7.80	**698.00**	692.00	695.40	18.43
fl417	417	1160.00	1135.00	1142.90	14.27	**1262.00**	**1262.00**	**1262.00**	6.64	**1262.00**	**1262.00**	**1262.00**	15.98
gr431	431	3004.00	2922.00	2959.00	11.03	**3932.00**	**3927.00**	**3931.00**	8.11	**3932.00**	3910.00	3929.30	19.43
pr439	439	3210.00	3116.00	3153.70	10.62	3899.00	**3886.00**	3891.00	8.36	**3909.00**	3883.00	**3893.30**	20.14
pcb442	442	1803.00	1769.00	1783.10	11.96	**2202.00**	**2202.00**	**2202.00**	9.34	**2202.00**	**2202.00**	**2202.00**	22.30
d493	493	622.00	606.00	612.10	19.60	**822.00**	**809.00**	**813.70**	11.49	815.00	803.00	812.50	27.61
att532	532	705.00	691.00	696.00	20.83	**893.00**	**891.00**	891.30	12.97	**893.00**	**891.00**	**891.70**	31.65
ali535	535	4545.00	4354.00	4449.10	17.06	**5741.00**	5693.00	5736.20	12.40	**5741.00**	**5741.00**	**5741.00**	29.79
si535	535	272.00	271.00	271.60	15.62	**282.00**	**280.00**	280.40	12.57	**282.00**	**280.00**	**281.60**	29.52
pa561	561	63.00	62.00	62.40	18.48	**83.00**	**82.00**	82.10	17.10	**83.00**	**82.00**	**82.50**	43.36
u574	574	1185.00	1169.00	1175.10	18.59	1533.00	**1526.00**	1528.70	17.97	**1534.00**	**1526.00**	**1529.00**	43.09
rat575	575	199.00	194.00	196.80	18.66	**268.00**	**268.00**	**268.00**	19.24	**268.00**	**268.00**	**268.00**	45.15
p654	654	3031.00	2931.00	2973.80	34.31	**3157.00**	**3157.00**	**3157.00**	16.92	**3157.00**	**3157.00**	**3157.00**	40.50
d657	657	1322.00	1303.00	1311.30	26.43	1743.00	**1737.00**	1739.30	28.85	**1747.00**	**1737.00**	**1739.60**	68.62
gr666	666	6351.00	6193.00	6277.00	25.14	**8099.00**	**7988.00**	**8062.00**	20.63	**8099.00**	7979.00	8032.40	49.30
u724	724	1217.00	1194.00	1204.60	29.97	**1561.00**	**1561.00**	**1561.00**	25.46	**1561.00**	**1561.00**	**1561.00**	60.33
rat783	783	229.00	222.00	224.80	34.90	**313.00**	**311.00**	**312.70**	41.66	**313.00**	**311.00**	312.20	96.79
nrw1379	1379	1010.00	995.00	1003.60	111.17	**1366.00**	**1362.00**	1363.30	137.72	**1366.00**	1361.00	**1363.60**	330.35
fl1577	1577	766.00	757.00	759.80	872.79	**1014.00**	**1005.00**	1008.30	672.42	**1014.00**	**1005.00**	**1008.60**	1649.23
d1655	1655	1353.00	1334.00	1338.70	963.57	**1741.00**	**1741.00**	**1741.00**	704.73	**1741.00**	**1741.00**	**1741.00**	1833.39
vm1748	1748	8548.00	8413.00	8490.30	1633.80	**10896.00**	**10896.00**	**10896.00**	660.43	**10896.00**	**10896.00**	**10896.00**	1700.45
u1817	1817	1157.00	1127.00	1136.20	1162.54	1552.00	1550.00	1551.70	1007.71	**1556.00**	**1552.00**	**1552.40**	2634.44
rl1889	1889	7737.00	7650.00	7684.00	1412.34	**10709.00**	**10709.00**	**10709.00**	1369.28	**10709.00**	**10709.00**	**10709.00**	3578.56
Overall		2130.61	2073.27	2098.82	80.15	2560.60	2551.46	2556.81	59.60	**2561.23**	**2552.30**	**2557.12**	152.80
NBV		6	2	2		76	**71**	58		**79**	69	**66**	

Table 2.2: Results of Wilcoxon signed rank test

	MS-ILS($h_1 + h_2$)							**MS-ILS**(h_2)					
	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant		$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant
MS-ILS(h_1)	80/82	3240	0	-7.770	-2.580	Yes	**MS-ILS**(h_1)	80/82	3240	0	-7.770	-2.580	Yes
MS-ILS(h_2)	40/82	443	377	-0.444	-2.580	No							

2.4.1 Wilcoxon signed rank test

To check whether there are significant differences among the performances of three MS-ILS variants, two tailed Wilcoxon signed rank test [94] has been conducted by setting the significance criteria to 5% (i.e. p-value $\leq$ 0.05). As part of this test, the difference between the normalized values of *'Average'* obtained by our approaches is ranked. Results of this statistical test are presented in Table 2.2. In this table, the '$NWT/Total$' means the number of instances without tie out of the total number of instances compared. The R^+ represents the sum of ranks for the instances where approaches on the top of table (MS-ILS($h_1 + h_2$)/MS-ILS(h_2)) performs better than its competitor mentioned on the left side of table, whereas the R^- represents the sum of ranks for the instances where approaches on the top of table (MS-ILS($h_1 + h_2$)/MS-ILS(h_2)) performs worse than its competitor mentioned on the left side of table. The test statistic Z is used because the number of instances are more than thirty ($NWT >$ 30). The Z value is used to compare with the critical value Z_{Cri} according to Wilcoxon signed rank test [94]. If $Z \leq Z_{Cri}$, then there is a significant difference between the performance of the two compared MS-ILS variants, otherwise the difference is not significant. Table 2.2 clearly shows that the results of both MS-ILS($h_1 + h_2$) and MS-ILS(h_2) are significant with respect to MS-ILS(h_1). On the other hand, comparison between MS-ILS($h_1 + h_2$) and MS-ILS(h_2) shows that there is no significant difference between them.

2.5 Conclusions

In this chapter, we have proposed a multi-start iterated local search algorithm to solve the maximum scatter traveling salesman problem (MSTSP) which contains only the permutation aspect. The key components of our approach are the proposed local searches based on insertion and modified 2-opt moves. The computational results on various test instances from TSPLIB show that the performance of MS-ILS with both the heuristics (viz. MS-ILS($h_1 + h_2$)) and MS-ILS with heuristic h_2 (viz. MS-ILS(h_2)) are quite competitive and there is no significant difference between the performances of the two. The MS-ILS with heuristic h_1 (viz. MS-ILS(h_1)) performed worse than the other two variants (MS-ILS(h_2) & MS-ILS($h_1 + h_2$)). Our MS-ILS approach being the first metaheuristic approach for the MSTSP will serve as the baseline approach for any future metaheuristic approaches for this problem. To facilitate this baselining, we have made use of publicly available TSPLIB instances to assess the performance of MS-ILS.

Chapter 3

k-Traveling Salesman Problem

3.1 Introduction

This chapter is devoted to k-traveling salesman problem (k-TSP). Given a set of n cities including a home city and a fixed value $1 < k \leq n$, the k-TSP consists in finding a subset of k cities including the home city and a tour visiting each city of this subset exactly once so that this tour has minimum length among all such tours over any subset of k cities that includes the home city. TSP can be considered as a special case of k-TSP with $k = n$. On the other hand, k-TSP in itself can be considered as a special case of prize collecting traveling salesman problem (PCTSP). The PCTSP was introduced by Balas *et al.* [95]. In this problem, a salesman gets a prize for visiting a city and incurs a penalty for not visiting a city. The objective of the PCTSP is to minimize the sumtotal of the distance traveled by the salesman and total penalties incurred while collecting a minimum amount of prize. If the associated penalties of all cities are zero, then the PCTSP becomes a quota traveling salesman problem (QTSP) [96, 97]. The k-TSP is a special case of the QTSP, where all the prizes are unitary and the quota is k [98]. k-TSP being a generalization of TSP is $\mathcal{NP}$-Hard. The applications of this problem arise in those situations where there are constraints on resources.

Several constant factor approximation schemes are known for the k-TSP [97, 99, 100, 101, 102]. The best known approximation ratio, i.e., a 2-approximation scheme for the k-TSP is achieved by the approximation scheme described in [101]. However, there are no metaheuristic approaches available for the k-TSP to the best of our knowledge. Compared to other variants of TSP, the k-TSP did not get that much attention from the researchers. Still it needs to be explored by the researchers belonging to various computational fields including those working in the field of metaheuristics.

Solving the k-TSP involves two aspects, viz. subset selection (selecting a subset containing k cities including the home city) and permutation (finding the best circular permutation of the k cities belonging to the selected subset). Any solution approach for k-TSP has to deal with both of these aspects in an appropriate manner in order to be effective over a wide range of instances. No matter how good the strategy is for subset selection in an approach for k-TSP, it will not yield a good solution for k-TSP if strategy to deal with permutation aspect is not designed properly. Likewise, an approach using a very good strategy to deal with permutation aspect, but a weak strategy for subset selection, will not succeed either. Further, relative importance of these two aspects may vary from one instance of the problem to another, and hence, an approach needs to adapt swiftly as per the aspects of the instance at hand in order to be efficient. Keeping all these facts in mind, we have developed two multi-start heuristic approaches for k-TSP. Our first approach is a simple and efficient general variable neighborhood search (GVNS) algorithm for the k-TSP that incorporates the variable neighborhood descent as local search. This approach utilizes two neighborhood structures one based on subset selection, whereas the other based on permutation. Our second approach is a hyper-heuristic approach which again incorporates two low level heuristics. The first low level heuristic caters to subset selection aspect, whereas the second low level heuristic caters to permutation aspect. Two versions of hyper-heuristic approach are presented in this chapter.

The remaining part of this chapter is organized as follows: Section 3.2 formally defines the k-TSP. Section 3.3 describes our multi-start general variable neighborhood search based approach for the k-TSP, whereas Section 3.4 describes our multi-start hyper-heuristic approach for the k-TSP. Computational results and their analysis are presented in Section 3.5. Section 3.6 outlines some concluding remarks.

3.2 Problem definition

Given a complete, edge-weighted, undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of n nodes in which the first node '1' represents the home city and the remaining $n - 1$ nodes represent other cities, $E = \{(i, j)|i, j \in V\}$ is the set of edges and a distance d_{ij} is associated with each edge $(i, j) \in E$. The objective of the k-TSP is to find a minimum length Hamiltonian cycle among all the Hamiltonian cycles over subgraphs induced by the subsets of V with exactly k nodes including the home city. We will denote such a subset of k cities by

V'. By introducing binary variable y_i to indicate whether a city i is part of this subset ($y_i = 1$) or not ($y_i = 0$), and another binary variable x_{ij} to indicate whether an edge (i, j) is part of Hamiltonian cycle over this subset ($x_{ij} = 1$) or not ($x_{ij} = 0$), the k-TSP can be formulated as follows:

$$\text{Minimize} \quad \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} \tag{3.1}$$

subject to:

$$\sum_{i \in V} y_i = k, \tag{3.2}$$

$$\sum_{i \in V} x_{1i} = 1 = \sum_{i \in V} x_{i1}, \tag{3.3}$$

$$\sum_{(k,i) \in E} x_{ki} + \sum_{(i,j) \in E} x_{ij} = 2y_i \quad \forall i \in V, \tag{3.4}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V' \subset V \tag{3.5}$$

$$x_{ij}, y_i \in \{0, 1\} \quad \forall (i, j) \in E, i \in V. \tag{3.6}$$

Equation (3.1) represents the objective function for the k-TSP and it minimizes the total distance. Equation (3.2) satisfies the constraint of visiting exactly k cities and Equation (3.3) ensures that the tour should start and end at the first city (i.e., home city). Equation (3.4) satisfies the constraints of indegree and outdegree of the visited cities. Equation (3.5) represents the sub tour elimination constraint. Equation (3.6) enforces the binary nature of the decision variables x_{ij} and y_i.

3.3 Multi-start general variable neighborhood search algorithm for the k-TSP

Inspired by the success of the general variable neighborhood search in solving several variants of the TSP [103, 104, 105, 106], we have proposed a multi-start general variable neighborhood search approach for the k-TSP which incorporates a VND strategy as local search. The proposed VND strategy explores the different neighborhood structures systematically by using the first improvement strategy [58]. The pseudo code of the proposed multi-start approach is given in Algorithm 9. Hereafter, this approach will be referred to as GVNS. GVNS being a multi-start approach, restarts N_{rst} number of times. The salient features of our GVNS approach for k-TSP are described in the following subsections.

3.3.1 Solution encoding and fitness

We have represented a solution by the linear permutation of k cities constituting the tour where home city always occupies the first position. Actually, a tour is a circular permutation and k linear permutations correspond to a single circular permutation comprising k cities. Hence, there is a redundancy when a linear permutation is used to represent a circular permutation. By fixing the fist position permanently for the home city, this redundancy got eliminated. It is to be noted that no component of GVNS can modify home city or its position.

The objective function (Equation (3.1)) itself is used as the fitness function, i.e., the fitness of a solution is the total distance traveled by the salesman. As k-TSP is a minimization problem, a solution having a lower value of the fitness function is regarded as more fit than a solution having a higher value.

3.3.2 Initial solution generation and shake phase

The initial solution generation procedure starts by inserting the home city at the first position in the tour and then an iterative process ensues. During each iteration a city which is not visited is selected uniformly at random and inserted into a random position in the salesman's tour. This procedure is repeated until the feasibility condition is satisfied, i.e., exactly k cities have been visited.

3.3.3 Variable neighborhood descent (VND)

VND is used as a local search very often due to its ability to systematically explore different neighborhood structures. The neighborhood structures can be designed as per the aspects of the problem at hand. We have proposed two neighborhood structures N_1 and N_2 for the k-TSP after giving due consideration to its aspects. The N_1 deals with the aspect of subset selection, whereas the N_2 deals with the aspect of permutation. These two neighborhoods are described below:

1. *Neighborhood N_1:* The first neighborhood N_1 is based on exchanging a visited city with an unvisited city. In this neighborhood, to create a new solution S' in the neighborhood of an existing solution S , a visited city is removed from S and an unvisited city is added

at the best possible position. By best possible position, we mean a position that yields a solution of least cost once the city is added to that position. To find the best possible position, we have to explore all the $k - 1$ positions.

2. *Neighborhood N_2:* The second neighborhood N_2 is based on swapping the positions of two visited cities in the tour. In this neighborhood, to create a new solution S' in the neighborhood of an existing solution S , two visited cities swap their positions, i.e., if i and j are two visited cities involved in swapping then i is moved to the place of j, and j is moved to the original place of i.

Owing to the fact that the initial neighborhood structures are explored more often than the latter ones, the performance of VND is influenced by the order in which various neighborhoods are explored. We have followed a deterministic order of exploring N_1 first and then N_2 (In fact, we have named them so based on this decision only). The proposed VND uses the first improvement strategy. In both the neighborhoods, viz. N_1 and N_2, if the VND finds a solution better than the current solution, then the current solution is immediately replaced with this new better solution, and the search starts afresh from the neighborhood N_1. This process continues until both the neighborhoods are explored completely without finding any improved solution. The current solution at this juncture is locally optimal with respect to both the neighborhoods and is returned as the solution found by VND.

3.4 Multi-start hyper-heuristic approach for the k-TSP

The hyper-heuristics individually and in hybridization with other metaheuristics have already been successfully applied to solve several variants of the TSP, e.g. [107, 108, 109, 110]. Motivated by the success of these approaches, we have developed a multi-start hyper-heuristic (HH) approach for k-TSP, where two low level heuristics are used. The number of restarts is governed by parameter N_{rst}. The motivation behind using a hyper-heuristic is that it can quickly adapt as per the aspects of the instance under consideration.

Following subsections describe the salient features of our HH approach for the k-TSP.

3.4.1 Solution encoding and fitness

Solution encoding and fitness function is same as used for GVNS (Section 3.3.1). Further, no component of HH can change home city or its position.

Algorithm 9: Pseudo code of GVNS Approach for k-TSP

```
Input: Set of parameters for GVNS and a $k$-TSP instance
Output: Best solution found
$best \leftarrow \infty$;
for $i \leftarrow 1$ to $N_{rst}$ do
    $S \leftarrow$ Generate_Initial_Solution();
    $p \leftarrow 1$;
    while termination condition not satisfied do
        $S' \leftarrow$ Shake($S$, $N_p$);
        $S'' \leftarrow$ VND($S'$);
        if $F(S'') < F(S)$ then
            $S \leftarrow S''$;
            $p \leftarrow 1$;
        else if $p < p_{max}$ then
            $p \leftarrow p + 1$;
        else
            $p \leftarrow 1$;
    if $S$ is better than $best$ then
        $best \leftarrow S$;
return $best$;
```

3.4.2 Generation of initial solutions

Each initial solution is generated in the same manner as described in Section 3.3.2 for GVNS.

3.4.3 Generation of neighboring solutions by using low level heuristics

An effective neighboring solution determination procedure should take into account all the aspects of the problem and should also maintain an appropriate balance among the considerations given to different aspects. The hyper-heuristic generates a solution S' in the neighborhood of current solution S using one of the low level heuristics. Our hyper-heuristic is provided with two low-level heuristics H_1 and H_2 to tackle the subset selection and permutation aspects respectively. These two heuristics are discussed below:

1. H_1: This heuristic handles the subset selection aspect of the k-TSP. This heuristic is a destroy and repair operator which partially destroys the tour and then repairs it. Each city in the tour is removed with probability ρ_r. All such removed cities are added to the set of unvisited cities which already contains all those cities which are not part of the tour. A city, which increases the cost of the tour by the least amount when inserted at

its best position, is chosen from this set of unvisited cities. By best position of a city, we mean a position that yields the least increase in the cost of the tour after the city is inserted at that position in comparison to all other positions. For determining this city, all possible combinations of unvisited cities and insertion positions in the tour need to be checked. The city so determined is inserted at its best position in the tour. This procedure is repeated until the feasibility condition is satisfied, i.e., exactly k cities have been visited. Notice that, in this heuristic, there may be a change in the constituent cities of the tour and their respective positions too.

2. *H_2:* This heuristic tackles the permutation aspect of the k-TSP. This heuristic is also a destroy and repair operator which partially destroys the tour and then repairs it. Each city in the tour is removed with a probability ρ_r. All such removed cities are added to a set which is initially empty, and then, one-by-one, a city is chosen randomly from this set and inserted at best possible position in the tour. This process continues until all the removed cities are inserted back into the tour. Notice that, in this heuristic, there is no change in the constituent cities, but there may be a change in their respective positions in the tour.

Algorithm 10 provides the pseudo code for generating a neighboring solution where the values 1 and 2 respectively corresponds to H_1 and H_2. Basically, *Create_Neighbor(S, j)* takes as input a solution S & a parameter j indicating the choice of the heuristic and returns a neighboring solution S' by applying the chosen heuristic on S.

3.4.4 Other features: selection mechanism and acceptance criteria

The selection mechanism plays a vital role in hyper-heuristic yielding a good solutions. There are many selection mechanisms available in the literature. Out of these, we have examined the hyper-heuristic with random and greedy selection mechanisms. In random selection mechanism, a low level heuristic is selected at random and used to return a solution at each step, whereas in greedy selection mechanism, all the low level heuristics are used at each step and the best solution among all the solutions obtained through these low level heuristics is returned. The heuristic whose solution is returned is deemed to be selected by greedy selection mechanism at that step. Motivation of choosing only these two mechanisms is due to the fact that the number of low-level heuristics are less (i.e., only 2) in our hyper-heuristic, and other mechanisms can be beneficial only when there are large number of low-level heuristics [70]. The two versions

Algorithm 10: Pseudo-code for generating a neighboring solution

```
Input: A solution $S$
Output: A neighboring solution $S'$
function Create_Neighbor($S$, $j$)
begin
if $j == 1$ then
    foreach city $c$ in the tour as per their order do
        Generate a random number $r$ such that $0 \leq r \leq 1$;
        if $r < \rho_r$ then
            Add $c$ to the set of unvisited cities;
        else
            Copy $c$ to the tour in $S'$;
    while $S'$ contains less than $k$ cities do
        Insert an unvisited city into the tour of $S'$ as per heuristic $H_1$;
    return $S'$;                                                    } Procedure of $H_1$.

else if $j == 2$ then
    foreach city $c$ in the tour as per their order do
        Generate a random number $r$ such that $0 \leq r \leq 1$;
        if $r < \rho_r$ then
            Add $c$ to a set of unassigned cities;
        else
            Copy $c$ to the tour in $S'$;
    foreach city $c$ in the set of unassigned cities in some random order do
        Insert $c$ into the tour of $S'$ as per heuristic $H_2$ ;
    return $S'$;                                                    } Procedure of $H_2$.
end function
```

of HH with random and greedy selection mechanisms will be referred to as HH_RAND and HH_GREEDY respectively.

There are many acceptance criterias available in the literature [70]. Out of these, we examined our hyper-heuristic with AA (all acceptance), OI (only improvement) acceptance criterias. Out of these two, only improvement (OI) criteria obtained the better results, and hence, the results with this criteria only are reported in this chapter.

The pseudo-code of our HH approach is given in Algorithm 11, where N_{rst} is the number of times the hyper-heuristic is restarted. *Selection_Mechanism(S, $\mathbb{S}_{LH}$)* is a function that takes as input a solution S and a set $\mathbb{S}_{LH}$ of low level heuristics and returns a solution as per selection mechanism by making use of *Create_Neighbor()* function one or more times as the case may be.

Algorithm 11: Pseudo code of HH approach for k-TSP

```
Input: Set of parameters for HH Algorithm and a $k$-TSP instance
Output: Best solution found
$best \leftarrow \infty$;
for $i \leftarrow 1$ to $N_{rst}$ do
    $S \leftarrow$ Generate_Initial_Solution();
    if $S$ is better than $best$ then
        $best \leftarrow S$;
    while termination condition not satisfied do
        $S' \leftarrow$ Selection_Mechanism($S$, $\mathbb{S}_{LH}$)
        if $S'$ is better than $S$ then
            $S \leftarrow S'$;
        if $S$ is better than $best$ then
            $best \leftarrow S$;
return $best$;
```

3.5 Computational results

Since our approaches, viz. GVNS, HH_RAND and HH_GREEDY are the first heuristic approaches for k-TSP, no benchmark instances are available in the literature for the k-TSP. Hence, it is inevitable for us to utilize the fresh test instances for evaluating the performance of our approaches. Our test instances for k-TSP are derived from the instances publicly available in TSPLIB . These instances have cities ranging from 14 to 783, and have a $n \times n$ distance matrix format. We have considered three different scenarios, each having a specific value for k. These three scenarios are as follows:

1. **Small** – Small k value: $k = \frac{1}{4}n$
2. **Medium** – Medium k value: $k = \frac{1}{2}n$
3. **Large** – Large k value: $k = \frac{3}{4}n$

Please note that the n is the total number of cities in an instance and the salesman has to visit k cities including the home city which is taken to be first city in all these instances.

All our approaches are implemented in C and executed on a Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM. All the approaches, viz. GVNS, HH_RAND and HH_GREEDY are executed on each test instance ten times independently, each time with a different random seed. In the GVNS approach, we have used the following values for the two parameters: Number of restarts N_{rst} is set to 100, and number of neighborhoods p_{max} is set to 2. In both HH_RAND and HH_GREEDY, we have used the following parameter values: Number of restarts N_{rst} is again set to 100 like GVNS, and the probability ρ_r of a city to be removed from the tour is set to 0.05. For our approaches, we choose two termination criterias with short and long time to compare our approaches from the perspective of convergence behavior. The chosen termination criterias are

1. **Short Run(SR)** – Termination after a time of $0.05 \times n$ seconds

2. **Long Run(LR)** – Termination after a time of $0.2 \times n$ seconds

As the termination criteria is according to the time and our approaches are multi-start approaches, each execution of an approach after a fresh start is allowed for time $\frac{T}{N_{rst}}$, where T is total time allowed for the approach.

This leads to six tables based on three scenarios and two termination criterias. Tables 3.1, 3.2 and 3.3 report the performance of GVNS, HH_RAND and HH_GREEDY under short run.

Table 3.1: Results of various approaches under small scenario with SR termination criteria

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	70	691	749	720.60	**686**	**718**	**703.00**	693	734	716.30
ali535	133	22648	34030	26606.60	13147	**14540**	13771.50	**12406**	15259	**13614.00**
att48	12	**1925**	**1925**	**1925.00**	**1925**	**1925**	**1925.00**	**1925**	**1925**	**1925.00**
att532	133	5373	6113	5661.60	**4014**	**4388**	**4234.60**	4024	4523	4271.90
bayg29	7	**332**	**332**	**332.00**	**332**	**332**	**332.00**	**332**	**332**	**332.00**
bays29	7	**400**	**400**	**400.00**	**400**	**400**	**400.00**	**400**	**400**	**400.00**
berlin52	13	**679**	**679**	**679.00**	**679**	**679**	**679.00**	**679**	**679**	**679.00**
bier127	31	**10619**	11029	10804.60	10687	**11014**	**10792.00**	10692	11028	10801.90
brazil58	14	**4965**	5030	4983.70	**4965**	**4965**	**4965.00**	**4965**	**4965**	**4965.00**
brg180	45	650	710	680.00	540	**560**	**551.00**	**520**	580	554.00
burma14	3	**359**	**359**	**359.00**	**359**	**359**	**359.00**	**359**	**359**	**359.00**
ch130	32	1149	1351	1261.10	**1130**	1291	**1226.30**	1167	**1290**	1238.60
ch150	37	1318	1350	1330.20	**1276**	**1336**	1316.90	**1276**	1359	**1312.40**
d198	49	5085	5257	5189.10	**5027**	**5102**	**5058.40**	5069	5147	5102.40
d493	123	10665	11769	11226.70	**9399**	**9713**	**9509.50**	9451	9814	9644.50
d657	164	18371	24327	20230.40	**12808**	**13623**	**13250.00**	13142	14110	13469.10
dantzig42	10	**145**	**145**	**145.00**	**145**	**145**	**145.00**	**145**	**145**	**145.00**
eil101	25	**107**	**108**	**107.30**	**107**	109	107.60	**107**	109	**107.30**
eil51	12	**82**	**82**	**82.00**	**82**	**82**	**82.00**	**82**	**82**	**82.00**
eil76	19	**102**	**102**	**102.00**	**102**	**102**	**102.00**	**102**	**102**	**102.00**
fl417	104	2304	4656	2795.10	**2258**	**2267**	**2260.70**	**2258**	2283	2264.50
fri26	6	**243**	**243**	**243.00**	**243**	**243**	**243.00**	**243**	**243**	**243.00**
gil262	65	593	633	612.60	**540**	**580**	**565.20**	555	590	575.20
gr120	30	**1308**	1362	1314.20	**1308**	**1318**	**1312.60**	**1308**	1385	1318.50
gr137	34	17802	18065	17996.50	**17399**	**17999**	**17780.20**	17510	18155	17874.80
gr17	4	**234**	**234**	**234.00**	**234**	**234**	**234.00**	**234**	**234**	**234.00**
gr202	50	8175	8573	8421.70	8175	8426	8332.30	**8142**	**8364**	**8263.40**
gr21	5	**324**	**324**	**324.00**	**324**	**324**	**324.00**	**324**	**324**	**324.00**
gr229	57	20604	24168	21836.10	**18589**	**19489**	**19062.20**	19129	19946	19438.80
gr24	6	**264**	**264**	**264.00**	**264**	**264**	**264.00**	**264**	**264**	**264.00**
gr431	107	16084	17874	17042.20	**14959**	**15966**	**15542.80**	15303	16233	15816.80
gr48	12	**874**	**874**	**874.00**	**874**	**874**	**874.00**	**874**	**874**	**874.00**
gr666	166	54939	98321	71424.80	**28464**	**30988**	29394.80	28630	31235	**29340.20**

continued . . .

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
gr96	24	10465	**10465**	**10465.00**	**10460**	10561	10474.00	**10460**	10786	10529.20
hk48	12	**2827**	**2827**	**2827.00**	**2827**	**2827**	**2827.00**	**2827**	**2827**	**2827.00**
kroA100	25	4998	5104	5023.10	**4970**	**5061**	**5011.80**	4998	5111	5030.20
kroA150	37	5725	**6182**	5936.10	**5690**	6454	5938.60	**5690**	6231	**5907.20**
kroA200	50	6438	7137	6775.60	**6220**	**6576**	**6393.50**	6272	6747	6493.90
kroB100	25	4353	4788	4588.80	**4305**	4684	**4473.70**	**4305**	**4605**	4536.60
kroB150	37	6410	**6854**	6588.00	**6071**	6938	**6468.20**	6482	6855	6666.60
kroB200	50	**6388**	7625	7076.80	6630	**7060**	6884.30	6414	7280	**6827.00**
kroC100	25	**4964**	5346	5048.60	**4964**	5103	4988.70	**4964**	**4982**	**4966.50**
kroD100	25	**4762**	5062	**4855.60**	4787	**5014**	4905.60	**4762**	5029	4874.00
kroE100	25	**3905**	3977	3918.70	**3905**	3984	3916.80	**3905**	**3935**	**3914.50**
lin105	26	**2606**	2751	2632.90	**2606**	**2680**	**2613.40**	**2606**	**2680**	**2613.40**
lin318	79	9133	10402	9654.90	8936	**9433**	**9175.60**	**8901**	10136	9463.10
p654	163	13671	26915	18722.80	**7348**	**8162**	**7834.00**	**7348**	8173	8030.50
pa561	140	609	710	639.10	**532**	570	**547.70**	536	**569**	552.20
pcb442	110	11994	12664	12206.70	11254	**11861**	**11558.30**	**11247**	12136	11734.50
pr107	26	**8443**	9311	8590.80	**8443**	**8449**	**8444.50**	**8443**	8729	8472.20
pr124	31	14952	**14952**	14952.00	**14640**	**14952**	**14920.80**	14952	**14952**	14952.00
pr136	34	21174	**23108**	**22186.60**	**21116**	26162	22862.80	21289	24118	22821.80
pr144	36	**14538**	**16119**	**14710.50**	**14538**	17196	15312.90	**14538**	16815	15327.30
pr152	38	23373	**24412**	**23895.70**	23373	24456	23988.40	**23195**	25027	24069.60
pr226	56	**20033**	27165	24288.50	**20033**	**25416**	**22968.40**	21814	26450	24669.20
pr264	66	10188	15034	11532.90	**9472**	**10512**	**9924.00**	9672	11073	10290.50
pr299	74	11877	13258	12554.00	11513	**11835**	**11653.60**	**11449**	12792	12164.50
pr439	109	22438	24316	23516.90	**20736**	**21704**	**21175.60**	20984	22266	21468.20
pr76	19	**23450**	**23450**	**23450.00**	**23450**	**23450**	**23450.00**	**23450**	**23450**	**23450.00**
rat195	48	592	618	603.30	**557**	594	**579.20**	568	**586**	580.40
rat575	143	1885	2156	1994.10	1612	**1696**	**1661.80**	**1599**	1710	1665.90
rat783	195	3939	6776	4466.50	**2343**	**2572**	**2478.90**	2533	2977	2690.00
rat99	24	**284**	291	285.70	**284**	**287**	285.20	**284**	**287**	**284.50**
rd100	25	**1438**	1613	1517.80	**1438**	**1521**	1472.30	**1438**	1545	**1462.70**
rd400	100	3750	4084	3931.00	**3383**	**3681**	**3526.30**	3458	3733	3597.00
si175	43	4968	5246	5107.40	4878	**4932**	**4901.40**	**4875**	5034	4942.00
si535	133	11777	13804	12116.70	11271	**11820**	**11460.40**	**11231**	12280	11543.60
st70	17	**120**	**125**	123.50	**120**	**125**	123.50	**120**	**125**	**123.40**
swiss42	10	**192**	**192**	**192.00**	**192**	**192**	**192.00**	**192**	**192**	**192.00**
ts225	56	**28828**	**28828**	**28828.00**	**28828**	**28828**	**28828.00**	**28828**	**28828**	**28828.00**
tsp225	56	957	1049	997.40	**923**	**970**	**952.50**	939	1009	970.10
u159	39	9176	9629	9354.30	**8983**	9623	9262.30	9085	**9332**	**9198.70**
u574	143	11188	12850	11794.80	**8384**	**9183**	**8747.50**	8711	9303	9016.40
u724	181	19865	34745	22750.10	**11293**	**12282**	**11802.20**	12423	14590	13555.80
ulysses16	4	**935**	**935**	**935.00**	**935**	**935**	**935.00**	**935**	**935**	**935.00**
ulysses22	5	**747**	**747**	**747.00**	**747**	**747**	**747.00**	**747**	**747**	**747.00**

Table 3.2: Results of various approaches under medium scenario with SR termination criteria

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	140	1446	1575	1508.80	**1358**	**1451**	**1397.70**	1373	1462	1409.90
ali535	267	107094	256356	132364.09	**52272**	75825	56897.30	53331	**65386**	**56496.10**
att48	24	**3603**	**3603**	**3603.00**	**3603**	**3603**	**3603.00**	**3603**	**3603**	**3603.00**
att532	266	17045	36396	20907.50	10766	12388	**11172.60**	**10654**	**12112**	11272.30
bayg29	14	**626**	**626**	**626.00**	**626**	**626**	**626.00**	**626**	**626**	**626.00**
bays29	14	**733**	**733**	**733.00**	**733**	**733**	**733.00**	**733**	**733**	**733.00**
berlin52	26	**1874**	**1928**	1883.70	**1874**	**1928**	**1879.40**	**1874**	1992	1895.80
bier127	63	27519	28758	28119.00	26377	**27617**	**27030.10**	**26145**	28091	27127.20
brazil58	29	8001	8170	8060.60	7993	**8077**	**8028.10**	**7978**	8132	**8028.10**
brg180	90	1310	1500	1432.00	**1110**	**1170**	**1135.00**	**1110**	1190	1158.00
burma14	7	**1272**	**1272**	**1272.00**	**1272**	**1272**	**1272.00**	**1272**	**1272**	**1272.00**
ch130	65	2594	2915	2772.30	**2408**	**2570**	**2506.60**	2463	2646	2548.50
ch150	75	2927	3158	3067.30	**2793**	**2929**	**2883.30**	2821	3039	2917.30
d198	99	7297	7582	7400.50	7080	**7155**	**7133.00**	**7058**	7270	7181.40
d493	246	21354	35272	24460.20	15041	**15615**	15268.40	**14651**	15640	**15221.00**

continued . . .

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
d657	328	42682	137294	55356.60	**34359**	**53018**	**38065.90**	37322	70533	42231.90
dantzig42	21	**260**	**260**	**260.00**	**260**	**260**	**260.00**	**260**	**260**	**260.00**
eil101	50	234	246	241.40	228	**237**	**232.30**	**227**	243	234.60
eil51	25	**175**	185	180.00	**175**	187	180.40	**175**	**184**	**178.80**
eil76	38	218	227	223.50	219	229	222.80	**217**	**226**	**221.60**
fl417	208	9821	21810	11752.50	6908	**7551**	7231.90	**6662**	7719	**7202.70**
fri26	13	**414**	**414**	**414.00**	**414**	**414**	**414.00**	**414**	**414**	**414.00**
gil262	131	1163	1260	1222.80	1044	**1134**	**1106.00**	**1042**	1176	1125.40
gr120	60	2736	3048	2892.10	**2690**	2889	**2767.10**	2694	**2884**	2782.30
gr137	68	29920	32460	31354.10	**29491**	**30557**	**29779.10**	29541	32365	30564.10
gr17	8	**517**	**517**	**517.00**	**517**	**517**	**517.00**	**517**	**517**	**517.00**
gr202	101	14509	15476	14988.80	14313	**14944**	**14596.30**	**14221**	15056	14656.70
gr21	10	**918**	**918**	**918.00**	**918**	**918**	**918.00**	**918**	**918**	**918.00**
gr229	114	45181	48567	47082.30	**41518**	**43585**	**42233.50**	42403	45229	43489.80
gr24	12	**504**	**504**	**504.00**	**504**	**504**	**504.00**	**504**	**504**	**504.00**
gr431	215	66497	92715	72582.30	**38134**	**41630**	**39686.70**	38469	43318	40311.60
gr48	24	**1819**	1836	1820.70	**1819**	1836	1822.40	**1819**	**1819**	**1819.00**
gr666	333	221489	810842	298109.91	**128728**	**190163**	**148593.80**	149752	302646	184097.20
gr96	48	20765	22069	21437.60	**20688**	**20881**	**20766.90**	20733	21617	21083.80
hk48	24	**4701**	4759	4712.30	**4701**	4759	4710.20	**4701**	**4735**	**4707.40**
kroA100	50	9572	10280	9885.00	**9184**	**9736**	**9369.90**	**9184**	10176	9666.80
kroA150	75	12866	13704	13374.90	**11783**	**12515**	**12150.20**	11812	13350	12662.80
kroA200	100	14631	15852	15139.60	12945	**13824**	**13475.90**	**12850**	14484	13585.50
kroB100	50	10026	10974	10452.60	**9096**	**9797**	**9485.60**	9150	10286	9912.80
kroB150	75	12276	14361	13278.90	11703	**12066**	**11861.10**	**11535**	12531	12038.90
kroB200	100	14297	15955	15271.80	**13080**	**14369**	**13775.50**	13434	14808	14284.50
kroC100	50	9668	10288	9986.00	**9457**	**10027**	**9702.40**	9709	10216	9884.10
kroD100	50	8962	9582	9223.20	**8719**	9134	**8870.20**	**8719**	**9103**	8884.60
kroE100	50	9283	10093	9804.20	9130	**9452**	**9259.30**	**9102**	9724	9452.40
lin105	52	5880	5954	5899.80	**5848**	**5883**	**5863.10**	**5848**	5954	5885.20
lin318	159	20453	25916	22657.50	**18600**	20346	**19462.40**	19114	**20322**	19786.80
p654	327	33482	256641	59420.50	**22241**	**73518**	**30505.20**	25556	94437	34445.80
pa561	280	1743	3178	1963.30	1285	**1407**	**1326.50**	**1232**	1501	1348.80
pcb442	221	31481	44188	34710.10	25214	**26075**	**25609.40**	**25149**	26379	25663.90
pr107	53	29652	31418	30722.90	18052	**29261**	**21452.50**	**18028**	29927	27145.80
pr124	62	**22998**	25088	23584.70	**22998**	**22998**	**22998.00**	**22998**	23321	23037.20
pr136	68	48496	52908	50164.50	47147	**47981**	**47663.60**	**47016**	48757	47875.20
pr144	72	29464	32664	31845.20	**28402**	**32321**	**30364.40**	29297	32674	30776.80
pr152	76	37881	47105	**41807.80**	37928	**46495**	41887.00	**36637**	47630	44106.90
pr226	113	39638	42326	40438.60	**38941**	**40461**	**39597.80**	39496	41838	40768.80
pr264	132	32000	36703	34025.30	**27898**	**29623**	**29189.20**	28699	32174	30294.50
pr299	149	26696	29615	27820.20	**23694**	**24100**	**23964.80**	23855	25679	24668.90
pr439	219	56023	91406	63483.30	**40440**	45601	**42148.30**	41011	**44208**	42267.10
pr76	38	41254	42786	41816.10	41258	**41976**	**41516.30**	**41248**	42472	41849.90
rat195	97	1185	1288	1240.80	**1159**	**1184**	**1171.80**	1161	1213	1185.50
rat575	287	5232	10894	5963.60	**3672**	**4242**	**3886.20**	3943	4586	4148.70
rat783	391	8682	41474	12405.30	**7511**	**18160**	**8928.20**	8091	24225	10010.50
rat99	49	577	599	588.10	**574**	**589**	**580.80**	575	590	582.70
rd100	50	3192	3371	3268.60	**3168**	**3236**	**3200.40**	3192	3322	3234.20
rd400	200	8839	12104	9610.70	7556	**7819**	**7683.20**	**7487**	7943	7731.70
si175	87	10500	11042	10750.40	**10188**	**10487**	**10320.20**	10244	10522	10404.10
si535	267	23554	32045	24735.10	**23039**	**26630**	**23525.70**	23101	27112	23609.40
st70	35	**260**	**278**	267.20	**260**	279	265.90	**260**	280	**263.00**
swiss42	21	**458**	**458**	**458.00**	**458**	**458**	**458.00**	**458**	**458**	**458.00**
ts225	112	**56828**	58257	57589.00	**56828**	**57656**	**57229.90**	**56828**	**57656**	57312.70
tsp225	112	1867	2044	1943.70	**1766**	**1849**	**1820.00**	1783	1877	1835.40
u159	79	18608	19524	19023.80	**18401**	**18762**	**18550.10**	**18401**	18939	18703.70
u574	287	28540	75322	35589.70	**20544**	**25252**	**22068.90**	22698	28472	24273.60
u724	362	40096	174467	57592.40	**36221**	**85596**	**42480.70**	40110	104609	47298.50
ulysses16	8	**1685**	**1685**	**1685.00**	**1685**	**1685**	**1685.00**	**1685**	**1685**	**1685.00**
ulysses22	11	**1902**	1903	1902.20	**1902**	**1902**	**1902.00**	**1902**	**1902**	**1902.00**

Table 3.3: Results of various approaches under large scenario with SR termination criteria

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	210	2268	2749	2386.60	2102	**2201**	**2139.90**	**2094**	2254	2167.70
ali535	401	196679	717736	270305.09	**118366**	**217688**	**144713.59**	136714	239308	151024.00
att48	36	**6563**	6656	**6583.10**	**6563**	6693	6586.30	**6563**	**6636**	6584.80
att532	399	31837	94931	39125.70	**22442**	**29921**	**24166.70**	22556	36159	25062.20
bayg29	21	**999**	**999**	**999.00**	**999**	**999**	**999.00**	**999**	**999**	**999.00**
bays29	21	**1194**	**1204**	**1196.40**	**1194**	**1204**	1197.80	**1194**	**1204**	1196.80
berlin52	39	**4174**	4436	4350.10	4251	4458	4342.80	**4174**	**4409**	**4289.90**
bier127	95	**51113**	57192	53949.80	51565	**53103**	**52269.80**	51593	54638	52822.80
brazil58	43	**11614**	11964	11785.40	**11614**	**11619**	**11616.10**	**11614**	11699	11623.30
brg180	135	2070	2310	2186.00	1650	**1750**	**1695.00**	**1620**	1820	1721.00
burma14	10	**1642**	**1642**	**1642.00**	**1642**	**1642**	**1642.00**	**1642**	**1642**	**1642.00**
ch130	97	4186	4635	4418.50	4012	**4212**	**4112.10**	**4003**	4421	4210.70
ch150	112	4814	5288	5001.40	4565	4843	4677.90	**4503**	**4842**	**4676.20**
d198	148	**9622**	11450	10300.10	9627	**10115**	**9861.70**	9758	10358	9971.00
d493	369	31965	78152	38732.00	**24533**	**31294**	**26597.20**	25635	31763	26990.10
d657	492	61969	297626	87099.60	55571	**131286**	**64648.20**	**54983**	184501	71055.90
dantzig42	31	**427**	**449**	435.10	**427**	459	432.40	**427**	455	**430.70**
eil101	75	403	430	418.50	**396**	**409**	**403.70**	**396**	418	406.90
eil51	38	290	300	294.80	290	**299**	295.20	**289**	300	**292.90**
eil76	57	348	366	358.70	341	**356**	**347.90**	**339**	363	350.80
fl417	312	12024	41153	16353.30	8816	13523	**9714.70**	**8810**	**13431**	10007.60
fri26	19	**601**	**601**	**601.00**	**601**	**601**	**601.00**	**601**	**601**	**601.00**
gil262	196	1835	2065	1960.60	**1695**	**1800**	**1748.00**	1717	1829	1773.00
gr120	90	4688	5027	4828.60	**4424**	**4622**	**4530.40**	4525	4770	4675.70
gr137	102	44356	52121	48566.10	**44220**	**45256**	**44751.00**	44382	47195	45816.80
gr17	12	**951**	**951**	**951.00**	**951**	**951**	**951.00**	**951**	**951**	**951.00**
gr202	151	22651	24576	23681.50	**22119**	**23100**	**22712.60**	22530	23388	22979.00
gr21	15	**1501**	**1501**	**1501.00**	**1501**	**1501**	**1501.00**	**1501**	**1501**	**1501.00**
gr229	171	73882	83324	77747.30	**69201**	**73782**	**70951.70**	70598	74436	72796.40
gr24	18	**844**	**844**	**844.00**	**844**	**844**	**844.00**	**844**	**844**	**844.00**
gr431	323	123736	243300	142645.80	**87161**	**102599**	**90941.30**	87748	103074	90947.30
gr48	36	3113	3261	3178.60	3113	**3231**	3159.40	**3104**	3380	**3144.90**
gr666	499	344171	1734123	498885.31	**264813**	**722902**	**335465.19**	275516	954405	365561.41
gr96	72	32425	35322	33450.30	**31437**	**32452**	**31798.20**	31706	32746	32017.80
hk48	36	7311	7677	7422.60	7326	7559	7442.30	**7278**	**7386**	**7318.70**
kroA100	75	15418	17074	16558.20	**14492**	**15586**	**14917.90**	14775	16542	15444.60
kroA150	112	20019	21807	20799.70	**18629**	**19592**	**19017.80**	19176	20717	19679.40
kroA200	150	22354	24847	23937.70	21056	**22196**	**21563.50**	**20723**	22617	21834.10
kroB100	75	15857	17040	16456.70	**14831**	**15416**	**15029.20**	14880	16045	15470.00
kroB150	112	19312	21896	20549.40	18103	**19018**	**18427.50**	**17729**	19749	18709.50
kroB200	150	22895	24442	23885.60	21241	**22346**	**21765.70**	**21043**	22965	21833.00
kroC100	75	15144	17257	15990.40	**14412**	**15052**	**14711.40**	14581	16423	15468.80
kroD100	75	15053	16798	15795.30	**14382**	**15088**	**14803.50**	14454	15968	15240.70
kroE100	75	15250	17118	16202.60	**14776**	**15605**	**15123.00**	15060	16272	15609.70
lin105	78	9405	9796	9650.70	**9058**	**9499**	**9206.30**	9161	9986	9406.10
lin318	238	34430	43832	36058.30	31370	**34198**	**32202.50**	**30682**	34447	32321.50
p654	490	41141	578099	98659.90	**30845**	**180175**	**50757.90**	34879	302242	64742.90
pa561	420	2944	8298	3535.90	**2205**	**3141**	**2405.70**	2288	3231	2478.10
pcb442	331	50388	92044	55843.70	40037	45301	42130.40	**40012**	**43452**	**41496.20**
pr107	80	39574	40944	40270.60	36843	**38060**	**37386.20**	**36468**	39045	37798.40
pr124	93	40230	43145	41802.20	39381	**39599**	**39533.50**	**39237**	41298	39974.90
pr136	102	73061	80892	77248.70	**69939**	**72538**	**71350.90**	71737	74652	73310.70
pr144	108	45204	54047	48876.10	**42721**	**46123**	**44607.20**	44266	49345	46872.50
pr152	114	60924	69307	64608.10	**58097**	**62759**	**60589.30**	60829	64174	62321.60
pr226	169	51519	64822	57824.90	**49198**	61479	**53353.00**	49732	**61193**	54024.20
pr264	198	44120	68691	49413.40	**39130**	**45863**	**41945.90**	41061	47907	44298.30
pr299	224	42458	51252	44269.00	**36657**	**39024**	**37847.50**	36977	41521	38599.80
pr439	329	97027	194132	114449.20	**69371**	**86472**	**75268.70**	72548	90071	76568.90
pr76	57	66833	70073	68378.40	64990	**66566**	**65593.50**	**64694**	66988	65642.70
rat195	146	1852	1981	1917.90	**1782**	**1856**	**1810.90**	1783	1888	1827.00
rat575	431	7984	29090	10308.30	**6896**	**11120**	**7451.90**	7176	14586	8085.90
rat783	587	11660	77251	18571.30	**10561**	**39014**	**13791.10**	11371	52236	15729.80
rat99	74	912	952	940.80	**876**	**913**	**894.70**	888	948	918.90
rd100	75	5197	5836	5552.50	**5096**	**5577**	**5287.30**	5226	5618	5488.10
rd400	300	14507	22552	15634.30	**11796**	**13400**	**12277.90**	11973	14055	12521.10
si175	131	15998	17139	16352.30	**15708**	**15978**	**15871.30**	15720	16068	15938.10

continued …

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
si535	401	36651	56842	39029.00	**35417**	**45057**	**36630.60**	35538	45353	36684.00
st70	52	444	468	456.00	**428**	**451**	**440.20**	**428**	465	446.60
swiss42	31	**760**	782	770.70	**760**	**774**	**762.50**	**760**	788	764.10
ts225	168	86484	93538	89286.20	**85656**	**88741**	**87418.20**	86535	89828	88558.40
tsp225	168	2987	3236	3070.30	**2780**	2936	**2875.80**	2813	**2932**	2877.80
u159	119	27890	31140	29311.50	**27621**	**29816**	**28556.40**	27790	29891	29101.20
u574	430	44488	180915	60303.20	**36238**	**62734**	**40817.20**	39009	85203	45108.60
u724	543	55269	340356	86381.70	**52148**	**166699**	**65839.60**	52708	212251	71306.40
ulysses16	12	**3183**	**3184**	3183.70	**3183**	**3184**	3183.40	**3183**	**3184**	**3183.30**
ulysses22	16	**2941**	**2942**	2941.70	**2941**	**2942**	2941.90	**2941**	**2942**	**2941.20**

On the other hand, Tables 3.4, 3.5 and 3.6 report the performance of our three approaches under long run. In all these tables, the first column represents the name of the instance with total number of cities in the end. The second column (k) shows the number of cities (including the home city) that needs to be visited by the salesman. The columns (Best, Worst & Average) report the best, worst and average solution quality over ten independent runs respectively. The best values are marked in bold to facilitate easy identification. Table 3.7 presents a summary regarding the relative performance of our approaches in terms of the number of instances on which the algorithm on the left side (HH_RAND/HH_GREEDY) obtained better (‘$<$’), same (‘=’) or worse solution (‘$>$’) than the algorithm on the upper side (GVNS/HH_GREEDY). This performance comparison has been done according to the best and the average solution quality. The summary of results in Tables 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 are reported in the rows of SR_Small, SR_Medium, SR_Large, LR_Small, LR_Medium and LR_Large respectively. Finally, the overall performance of our approaches by considering all three scenarios and two termination criterias is reported in the last two rows named as ‘Total’. These tables clearly show that both HH_RAND and HH_GREEDY performed better than GVNS in terms of best and average solution quality both in all three scenarios under both the termination conditions. Further, the relative performance of HH_RAND and HH_GREEDY improve as we move from small to large scenario. However, there is not much difference in relative performance under two different termination conditions. When it comes to the comparison between HH_RAND and HH_GREEDY, the former performed better than the latter.

The poor performance of the GVNS can be attributed to higher computational cost of exploring the neighborhood N_1 which is used more often than N_2. Further, these two neighborhoods are explored multiple times in an iteration of GVNS. On the other hand, HH_RAND uses either H_1 or H_2 and HH_GREEDY uses both H_1 and H_2 in an iteration. Moreover, H_1 and H_2 can perform much more exploration than a single move in N_1/N_2. So when GVNS is executed for the same amount of time as HH_RAND and HH_GREEDY, it is not able to explore as much search space as explored by the latter two approaches leading to its poor performance. In a likewise manner, the superior performance of HH_RAND over HH_GREEDY can be explained. HH_GREEDY applies both H_1 and H_2 on the current solution, whereas HH_RAND applies only one of these two heuristics randomly. As a result, single iteration of HH_GREEDY requires more time than HH_RAND. Consequently, HH_GREEDY gets lesser number of iterations in comparison to HH_RAND, when both HH_RAND over HH_GREEDY are executed for the same amount of time. In this situation, HH_GREEDY can outperform HH_RAND only when the gain of using both the heuristics in an iteration surpasses the loss HH_GREEDY incurs due to the lesser number of iterations. This is clearly not the case with k-TSP.

Figure 3.1 plots the solutions obtained by our approaches for the instance eil51 with different k values (12, 25, 38). In all the plots of this figure, the cities are shown as blue circles, and the first city (i.e., home city) where the salesman has to start and end is shown as red colored square. This figure depicts the variation in the tour and its length according to the value of k.

Table 3.4: Results of various approaches under small scenario with LR termination criteria

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	70	687	742	717.50	**670**	**713**	**698.40**	683	736	711.10
ali535	133	18832	28075	22382.20	**12602**	**13612**	**13220.30**	12980	15030	13486.80
att48	12	**1925**	**1925**	**1925.00**	**1925**	**1925**	**1925.00**	**1925**	**1925**	**1925.00**
att532	133	4596	4873	4699.50	**3980**	**4258**	**4132.90**	4093	4392	4236.90
bayg29	7	**332**	**332**	**332.00**	**332**	**332**	**332.00**	**332**	**332**	**332.00**
bays29	7	**400**	**400**	**400.00**	**400**	**400**	**400.00**	**400**	**400**	**400.00**
berlin52	13	**679**	**679**	**679.00**	**679**	**679**	**679.00**	**679**	**679**	**679.00**
bier127	31	**10619**	11029	10804.60	10687	11014	10792.00	10692	**10813**	**10745.80**
brazil58	14	**4965**	5030	4983.70	**4965**	**4965**	**4965.00**	**4965**	**4965**	**4965.00**
brg180	45	650	710	686.00	540	580	554.00	**530**	**560**	**546.00**
burma14	3	**359**	**359**	**359.00**	**359**	**359**	**359.00**	**359**	**359**	**359.00**
ch130	32	1149	1351	1261.10	**1130**	**1291**	1226.30	**1130**	1296	**1219.70**
ch150	37	1318	1350	1330.20	**1276**	**1336**	1316.90	**1276**	1386	**1310.50**
d198	49	5080	5244	5169.10	**5028**	**5120**	**5066.40**	5037	5130	5075.40
d493	123	10032	10868	10404.90	9411	**9576**	**9489.60**	**9394**	9767	9567.40
d657	164	14029	14919	14424.60	**12299**	**12751**	**12554.10**	12562	13607	12942.80
dantzig42	10	**145**	**145**	**145.00**	**145**	**145**	**145.00**	**145**	**145**	**145.00**
eil101	25	**107**	**108**	**107.30**	**107**	109	107.60	**107**	109	**107.30**
eil51	12	**82**	**82**	**82.00**	**82**	**82**	**82.00**	**82**	**82**	**82.00**
eil76	19	**102**	**102**	**102.00**	**102**	**102**	**102.00**	**102**	**102**	**102.00**
fl417	104	2285	2493	2324.90	**2257**	**2269**	**2259.40**	2258	2283	2262.20
fri26	6	**243**	**243**	**243.00**	**243**	**243**	**243.00**	**243**	**243**	**243.00**
gil262	65	574	641	606.70	**545**	**570**	**561.00**	553	579	564.70
gr120	30	**1308**	1362	1314.20	**1308**	**1318**	**1312.60**	**1308**	1386	1323.40
gr137	34	17802	18065	17996.50	**17399**	**17999**	**17780.20**	17445	18065	17813.20
gr17	4	**234**	**234**	**234.00**	**234**	**234**	**234.00**	**234**	**234**	**234.00**
gr202	50	8191	8564	8422.40	**8142**	**8404**	8301.40	**8142**	8428	**8283.50**
gr21	5	**324**	**324**	**324.00**	**324**	**324**	**324.00**	**324**	**324**	**324.00**
gr229	57	20252	24811	21926.40	**18555**	19805	**19102.80**	18970	**19600**	19195.30
gr24	6	**264**	**264**	**264.00**	**264**	**264**	**264.00**	**264**	**264**	**264.00**
gr431	107	15556	16227	15960.10	**14857**	**15612**	**15374.30**	15350	16221	15819.40
gr48	12	**874**	**874**	**874.00**	**874**	**874**	**874.00**	**874**	**874**	**874.00**
gr666	166	29083	32399	31037.20	**27358**	**28430**	**28045.50**	27820	29347	28660.90
gr96	24	10465	**10465**	**10465.00**	**10460**	10561	10474.00	**10460**	10786	10529.20

continued . . .

Instance	k	GVNS			HH_RAND			HH_GREEDY		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
hk48	12	**2827**	**2827**	**2827.00**	**2827**	**2827**	**2827.00**	**2827**	**2827**	**2827.00**
kroA100	25	4998	5104	5023.10	**4970**	**5061**	**5011.80**	4998	5104	5017.20
kroA150	37	5725	6182	5936.10	**5690**	6454	5938.60	**5690**	**6154**	**5826.70**
kroA200	50	6438	7155	6780.70	**6202**	**6496**	**6358.00**	6421	6663	6503.10
kroB100	25	4353	4788	4588.80	**4305**	**4684**	**4473.70**	4501	4788	4574.50
kroB150	37	6410	6854	6588.00	6071	6938	6468.20	**5812**	**6781**	**6426.20**
kroB200	50	6414	7543	7051.00	**6370**	**7034**	**6711.20**	6440	7219	6811.80
kroC100	25	**4964**	5346	5048.60	**4964**	**5103**	**4988.70**	**4964**	5227	4991.50
kroD100	25	**4762**	5062	4855.60	4787	5014	4905.60	**4762**	**4980**	**4843.70**
kroE100	25	**3905**	3977	3918.70	**3905**	3984	3916.80	**3905**	**3935**	**3911.00**
lin105	26	**2606**	2751	2632.90	**2606**	2680	2613.40	**2606**	**2606**	**2606.00**
lin318	79	9108	10190	9624.10	**8912**	**9626**	**9162.50**	9219	9813	9429.30
p654	163	8152	8328	8255.70	7348	**8128**	**7439.20**	**7341**	8173	7633.80
pa561	140	575	658	607.70	**523**	**563**	**540.20**	526	568	547.40
pcb442	110	11628	12267	12000.80	**11099**	**11716**	**11457.00**	11539	12054	11782.70
pr107	26	**8443**	9311	8590.80	**8443**	**8449**	**8444.50**	**8443**	8729	8472.20
pr124	31	14952	**14952**	14952.00	**14640**	**14952**	**14920.80**	14952	**14952**	14952.00
pr136	34	21174	**23108**	**22186.60**	**21116**	26162	22862.80	21174	24564	22398.40
pr144	36	14538	**16119**	**14710.50**	14538	17196	15410.30	**14327**	16363	15483.40
pr152	38	**23373**	**24412**	**23895.70**	**23373**	24544	24001.70	23700	25078	24298.30
pr226	56	21202	27255	24446.10	**20033**	26595	**23665.00**	20125	**25902**	24008.30
pr264	66	10864	15034	11990.60	9432	**10432**	9931.20	**9232**	11032	**9819.20**
pr299	74	11773	13272	12205.60	**11392**	**11916**	**11611.50**	11675	12622	12055.20
pr439	109	22243	23926	22911.30	**20874**	**21267**	**21064.90**	20987	21728	21355.50
pr76	19	**23450**	**23450**	**23450.00**	**23450**	**23450**	**23450.00**	**23450**	**23450**	**23450.00**
rat195	48	593	618	603.50	**557**	**586**	580.80	559	590	**573.90**
rat575	143	1694	1817	1758.80	1591	**1663**	**1622.30**	**1589**	1691	1640.80
rat783	195	2453	2746	2577.60	**2192**	**2274**	**2222.90**	**2192**	2283	2234.60
rat99	24	**284**	291	285.70	**284**	**287**	285.20	**284**	**287**	**284.50**
rd100	25	**1438**	1613	1517.80	**1438**	**1521**	**1472.30**	**1438**	1545	1488.30
rd400	100	3762	4013	3884.90	**3362**	**3506**	**3439.90**	3486	3769	3596.80
si175	43	4968	5246	5107.40	**4881**	**4947**	**4906.00**	4906	5010	4945.30
si535	133	11688	13780	12051.00	**11208**	**11815**	**11374.40**	11347	12280	11575.00
st70	17	**120**	**125**	123.50	**120**	**125**	123.50	**120**	**125**	**123.40**
swiss42	10	**192**	**192**	**192.00**	**192**	**192**	**192.00**	**192**	**192**	**192.00**
ts225	56	**28828**	**28828**	**28828.00**	**28828**	**28828**	**28828.00**	**28828**	**28828**	**28828.00**
tsp225	56	977	1028	999.10	**915**	**964**	942.10	**915**	973	**939.70**
u159	39	9176	9629	9354.30	**8983**	9623	9262.30	**8983**	**9412**	**9188.60**
u574	143	9146	9602	9364.20	**8310**	**8725**	**8520.20**	8453	9194	8870.50
u724	181	11737	13354	12395.80	10176	**10733**	**10362.50**	**10071**	11162	10578.20
ulysses16	4	**935**	**935**	**935.00**	**935**	**935**	**935.00**	**935**	**935**	**935.00**
ulysses22	5	**747**	**747**	**747.00**	**747**	**747**	**747.00**	**747**	**747**	**747.00**

Table 3.5: Results of various approaches under medium scenario with LR termination criteria

Instance	k	GVNS			HH_RAND			HH_GREEDY		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	140	1376	1510	1443.60	**1314**	**1417**	**1363.90**	1362	1436	1400.10
ali535	267	59255	107878	72798.10	46457	53987	**49143.40**	**46195**	**52936**	49680.30
att48	24	**3603**	**3603**	**3603.00**	**3603**	**3603**	**3603.00**	**3603**	**3603**	**3603.00**
att532	266	11285	13411	12027.20	**9991**	**10800**	**10396.80**	10285	10924	10466.30
bayg29	14	**626**	**626**	**626.00**	**626**	**626**	**626.00**	**626**	**626**	**626.00**
bays29	14	**733**	**733**	**733.00**	**733**	**733**	**733.00**	**733**	**733**	**733.00**
berlin52	26	**1874**	1928	1883.70	**1874**	1928	1879.40	**1874**	**1917**	**1878.30**
bier127	63	27404	28361	27942.20	**26062**	27335	26841.20	26102	**27329**	**26745.50**
brazil58	29	8001	8170	8060.60	7993	8077	8028.10	**7978**	**8065**	**8005.80**
brg180	90	1370	1470	1412.00	**1080**	**1120**	**1103.00**	1110	1160	1130.00
burma14	7	**1272**	**1272**	**1272.00**	**1272**	**1272**	**1272.00**	**1272**	**1272**	**1272.00**
ch130	65	2483	2920	2742.70	**2423**	**2564**	**2512.60**	2544	2657	2588.20
ch150	75	2977	3158	3077.00	**2761**	**2948**	**2851.20**	2856	2968	2914.30
d198	99	7270	7517	7396.70	**7073**	**7198**	**7124.30**	7126	7318	7209.60
d493	246	15997	17188	16514.90	**14223**	**14803**	**14568.80**	14356	15119	14674.80
d657	328	32740	54078	37002.30	24462	27495	**25991.30**	**24424**	**26937**	26093.40

continued . . .

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
dantzig42	21	**260**	**260**	**260.00**	**260**	**260**	**260.00**	**260**	**260**	**260.00**
eil101	50	240	245	242.20	228	**238**	234.00	**227**	239	**232.80**
eil51	25	**175**	185	180.00	**175**	187	180.40	**175**	**184**	**178.80**
eil76	38	218	227	223.50	219	229	222.80	**216**	**226**	**218.70**
fl417	208	7404	8593	7848.30	**6404**	**6955**	**6597.00**	6483	7157	6853.40
fri26	13	**414**	**414**	**414.00**	**414**	**414**	**414.00**	**414**	**414**	**414.00**
gil262	131	1141	1290	1224.20	1037	**1101**	**1078.20**	**1034**	1136	1095.60
gr120	60	2826	3001	2891.00	2713	2871	**2767.70**	**2710**	**2843**	2772.40
gr137	68	30456	32460	31654.50	**29363**	**30127**	**29700.00**	29673	31016	29936.60
gr17	8	**517**	**517**	**517.00**	**517**	**517**	**517.00**	**517**	**517**	**517.00**
gr202	101	14333	15415	14983.00	**14182**	**14670**	**14399.40**	14199	14958	14646.80
gr21	10	**918**	**918**	**918.00**	**918**	**918**	**918.00**	**918**	**918**	**918.00**
gr229	114	45078	49105	47002.90	**41005**	**42574**	**41642.70**	41242	44401	42704.70
gr24	12	**504**	**504**	**504.00**	**504**	**504**	**504.00**	**504**	**504**	**504.00**
gr431	215	41906	45594	43969.60	**37071**	**39334**	**37973.30**	38207	40976	38820.40
gr48	24	**1819**	1836	1820.70	**1819**	1836	1822.40	**1819**	**1819**	**1819.00**
gr666	333	140432	253851	160908.09	93994	107511	98839.90	**89448**	**107124**	**95744.70**
gr96	48	20807	22097	21431.10	**20688**	**20937**	**20756.80**	**20688**	21930	21095.20
hk48	24	**4701**	4759	4712.30	**4701**	4759	4710.20	**4701**	**4735**	**4707.40**
kroA100	50	9335	10417	10035.50	**9184**	**9525**	**9218.10**	**9184**	9949	9565.90
kroA150	75	12166	14020	13228.10	**11625**	**12379**	**12015.30**	11936	13270	12621.80
kroA200	100	14213	15717	15060.50	**12753**	**13668**	**13235.30**	13011	14069	13595.80
kroB100	50	9244	11025	10291.70	9312	**9787**	9577.40	**9096**	9980	**9537.00**
kroB150	75	12751	14361	13368.80	**11642**	**12132**	**11880.40**	11910	12412	12102.50
kroB200	100	14004	15955	14971.90	**13178**	**14114**	**13704.90**	13222	14637	14030.90
kroC100	50	9668	10288	9986.00	9493	10049	9726.20	**9457**	**9923**	**9654.40**
kroD100	50	8962	9582	9223.20	**8719**	9150	8853.50	**8719**	**8969**	**8805.50**
kroE100	50	9345	10141	9781.70	9132	**9631**	**9303.70**	**9102**	9792	9386.40
lin105	52	5857	5921	5880.50	**5848**	**5919**	**5873.20**	5857	5995	5909.80
lin318	159	19174	22569	21183.30	**17766**	**19464**	**18660.00**	18696	20284	19536.10
p654	327	27188	73557	34053.80	19380	**21433**	**20473.00**	**19332**	22443	20649.30
pa561	280	1429	1768	1497.50	**1211**	**1300**	**1237.30**	1224	1327	1262.40
pcb442	221	25899	27504	27015.00	24536	**25147**	**24921.30**	**24451**	26191	25208.20
pr107	53	29652	31206	30625.80	**18028**	**29798**	**23769.60**	18165	30212	28257.10
pr124	62	**22998**	24923	23890.80	**22998**	**22998**	**22998.00**	**22998**	23321	23037.20
pr136	68	49652	52064	50590.10	**46890**	**47931**	**47316.90**	47016	51128	48389.10
pr144	72	30810	33628	32062.30	**28402**	**31557**	**30063.60**	29380	32167	31107.10
pr152	76	38369	46768	42781.70	**37336**	**45171**	**40178.30**	38355	46485	41500.30
pr226	113	**38718**	40662	39691.60	38914	**39830**	**39292.40**	39231	43349	40238.70
pr264	132	31251	36187	33866.70	**27711**	**29598**	**28317.30**	28247	29903	28856.50
pr299	149	24843	27832	25756.10	**23475**	**24544**	**23849.00**	23636	25119	24498.30
pr439	219	45318	49432	46788.20	38874	**41581**	**39778.20**	**38313**	43040	40781.30
pr76	38	41254	42786	41816.10	41258	**41976**	**41516.30**	**41248**	42465	41699.30
rat195	97	1185	1288	1238.80	**1140**	**1165**	**1152.80**	1144	1208	1171.60
rat575	287	3853	4512	4052.60	**3349**	**3526**	**3444.20**	3431	3604	3489.00
rat783	391	7258	17344	8431.00	**5007**	**5580**	**5174.90**	5132	5858	5304.60
rat99	49	577	599	588.80	576	591	581.00	**574**	**589**	**579.40**
rd100	50	3192	3371	3268.60	**3168**	**3236**	**3196.80**	3190	3260	3210.30
rd400	200	7951	8542	8192.20	**7013**	**7555**	**7287.70**	7428	7799	7577.10
si175	87	10467	11042	10700.80	10265	**10436**	**10319.10**	**10208**	10556	10382.70
si535	267	23460	30386	24433.90	22940	**25599**	**23316.70**	**22896**	25926	23360.90
st70	35	**260**	278	267.20	**260**	279	265.90	**260**	**273**	**262.60**
swiss42	21	**458**	**458**	**458.00**	**458**	**458**	**458.00**	**458**	**458**	**458.00**
ts225	112	**56828**	**57656**	57436.90	**56828**	**57656**	**57229.90**	**56828**	57949	57271.30
tsp225	112	1904	2035	1964.00	**1729**	**1852**	**1788.10**	1818	1926	1868.10
u159	79	18491	19617	19031.20	18401	**18762**	**18516.70**	**18399**	18955	18580.10
u574	287	20940	26302	22489.60	**17355**	**19033**	**18214.10**	17553	19438	18249.20
u724	362	33596	71937	39033.60	**21802**	**25214**	23468.30	22138	25262	**23227.40**
ulysses16	8	**1685**	**1685**	**1685.00**	**1685**	**1685**	**1685.00**	**1685**	**1685**	**1685.00**
ulysses22	11	**1902**	1903	1902.20	**1902**	**1902**	**1902.00**	**1902**	**1902**	**1902.00**

Table 3.6: Results of various approaches under large scenario with LR termination criteria

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
a280	210	2149	2452	2269.60	2066	**2185**	**2116.90**	**2043**	2199	2135.60
ali535	401	144825	291826	171311.59	113209	147140	120854.20	**108369**	**140219**	**118111.20**
att48	36	**6563**	6656	6583.10	**6563**	6693	6586.30	**6563**	**6605**	**6575.90**
att532	399	23148	34207	25235.50	19245	**21196**	**19729.40**	**18858**	21441	19747.60
bayg29	21	**999**	**999**	**999.00**	**999**	**999**	**999.00**	**999**	**999**	**999.00**
bays29	21	**1194**	1204	1196.40	**1194**	1204	1197.80	**1194**	**1194**	**1194.00**
berlin52	39	**4174**	4436	4350.10	4251	4458	4342.80	**4174**	**4385**	**4323.80**
bier127	95	52853	57192	54586.10	**50324**	**53100**	**51530.10**	51427	53888	52671.70
brazil58	43	**11614**	11964	11785.40	**11614**	**11619**	11615.60	**11614**	**11619**	**11614.70**
brg180	135	2030	2310	2163.00	**1600**	**1710**	**1663.00**	1650	1780	1693.00
burma14	10	**1642**	**1642**	**1642.00**	**1642**	**1642**	**1642.00**	**1642**	**1642**	**1642.00**
ch130	97	4156	4628	4412.20	**3907**	4176	**4068.50**	3956	**4166**	4089.40
ch150	112	4860	5288	4994.90	**4499**	**4705**	**4599.60**	4625	4812	4716.00
d198	148	9819	11440	10493.80	9428	**9713**	**9586.10**	**9386**	10358	9796.30
d493	369	26223	31451	27158.30	23483	**25003**	**23868.10**	**23380**	26149	24260.50
d657	492	52098	130389	61837.70	**41410**	**48611**	**42767.90**	41976	49857	43707.80
dantzig42	31	**427**	449	435.10	**427**	459	432.40	**427**	**427**	**427.00**
eil101	75	409	432	419.20	397	**408**	**402.70**	**389**	419	403.30
eil51	38	**290**	300	294.80	**290**	299	295.20	**290**	**296**	**293.30**
eil76	57	351	367	359.00	**336**	**356**	**346.10**	339	359	347.80
fl417	312	9690	16049	11174.40	8337	**8998**	**8603.10**	**8242**	11504	9027.80
fri26	19	**601**	**601**	**601.00**	**601**	**601**	**601.00**	**601**	**601**	**601.00**
gil262	196	1817	1999	1896.00	**1672**	**1759**	**1717.80**	1703	1829	1751.10
gr120	90	4585	5027	4763.90	4420	**4598**	**4528.20**	**4380**	4770	4589.00
gr137	102	46737	50307	48199.10	44182	**45781**	**44665.50**	**43912**	47350	45554.80
gr17	12	**951**	**951**	**951.00**	**951**	**951**	**951.00**	**951**	**951**	**951.00**
gr202	151	22701	24537	23731.80	**21563**	**22761**	**22444.20**	21954	23633	22854.10
gr21	15	**1501**	**1501**	**1501.00**	**1501**	**1501**	**1501.00**	**1501**	**1501**	**1501.00**
gr229	171	72938	81767	75845.00	**67848**	**72217**	**70154.60**	68553	75220	71644.00
gr24	18	**844**	**844**	**844.00**	**844**	**844**	**844.00**	**844**	**844**	**844.00**
gr431	323	93079	107716	97093.70	81904	**87997**	**84147.10**	**81144**	89321	84736.20
gr48	36	3113	3261	3178.60	3113	**3231**	3159.40	**3104**	**3231**	**3135.00**
gr666	499	273618	763260	341435.41	**194897**	**253287**	**208981.80**	204846	259577	215325.91
gr96	72	32498	35322	33822.20	**31437**	**32452**	**31707.80**	31526	34413	32144.10
hk48	36	7311	7677	7422.60	7326	**7559**	7442.30	**7278**	7561	**7317.70**
kroA100	75	15461	16943	16280.90	14592	**14969**	**14747.30**	**14500**	16437	15267.10
kroA150	112	19192	21807	20779.40	**18210**	**19202**	**18724.50**	18917	20031	19344.40
kroA200	150	22926	25030	23673.70	20794	**21736**	**21323.40**	**20740**	22159	21542.40
kroB100	75	15846	17040	16344.20	**14744**	**15412**	**15045.60**	14799	15775	15312.50
kroB150	112	19506	21896	20560.30	17695	**18717**	**18140.40**	**17501**	19749	18705.70
kroB200	150	22964	24418	23548.40	20673	**21541**	**21191.40**	**20508**	23134	21695.30
kroC100	75	15536	17257	16162.50	14201	**15093**	**14615.30**	**14067**	15770	14922.90
kroD100	75	15211	16798	15943.00	**14171**	**15145**	**14691.00**	14288	15710	15139.00
kroE100	75	15452	17118	16102.00	**14640**	**15605**	**15154.10**	14843	15956	15217.90
lin105	78	9178	9982	9523.10	**9034**	**9310**	**9206.20**	**9034**	9408	9267.10
lin318	238	32100	37616	33558.10	**29829**	**31398**	**30479.10**	30625	32702	31419.10
p654	490	34821	198432	53572.60	**24514**	44335	**29129.30**	25671	**42799**	29650.40
pa561	420	2361	3538	2548.40	**2031**	**2419**	**2147.40**	2067	2458	2152.20
pcb442	331	41889	46635	42991.40	**37941**	**40196**	**38976.70**	38419	40523	39116.90
pr107	80	39770	41113	40531.90	**36627**	**37745**	**37072.50**	36661	39331	38251.40
pr124	93	40471	43934	41953.00	**39174**	**39772**	**39562.40**	39528	42758	40375.60
pr136	102	75922	80389	77741.50	**69690**	**72784**	**71071.10**	69905	75594	72198.40
pr144	108	46889	51287	48861.80	**41452**	**48138**	**43983.30**	42164	49900	46620.10
pr152	114	61758	69307	64513.60	**57431**	**60424**	**59372.00**	58114	65974	62302.90
pr226	169	49552	60248	56582.70	**47516**	**54238**	**50928.10**	50315	61193	54532.20
pr264	198	43189	60377	48234.90	**39503**	**41739**	**40366.70**	40896	44796	42967.90
pr299	224	38188	43058	40204.00	**35942**	**38155**	**37030.40**	36957	39966	37949.40
pr439	329	74268	97896	78410.10	**64497**	**77142**	**68425.10**	66365	77489	70468.70
pr76	57	66186	69717	67805.60	64251	66734	65505.40	**64142**	**66117**	**64878.40**
rat195	146	1891	1983	1929.10	**1753**	**1828**	**1780.40**	1783	1844	1817.20
rat575	431	6595	11320	7231.10	**5452**	**5942**	**5591.30**	5498	6018	5610.30
rat783	587	10561	36820	13406.80	**8487**	**12529**	**9152.30**	8820	15063	9685.50
rat99	74	928	960	942.00	**861**	**904**	**884.50**	866	927	894.50
rd100	75	5221	5819	5523.20	**5094**	**5494**	**5264.50**	5192	5503	5325.20
rd400	300	12224	14484	12864.20	**11326**	12423	**11671.80**	11398	**12231**	11765.10
si175	131	16074	17136	16426.20	**15625**	**15915**	**15821.20**	15756	16016	15889.00

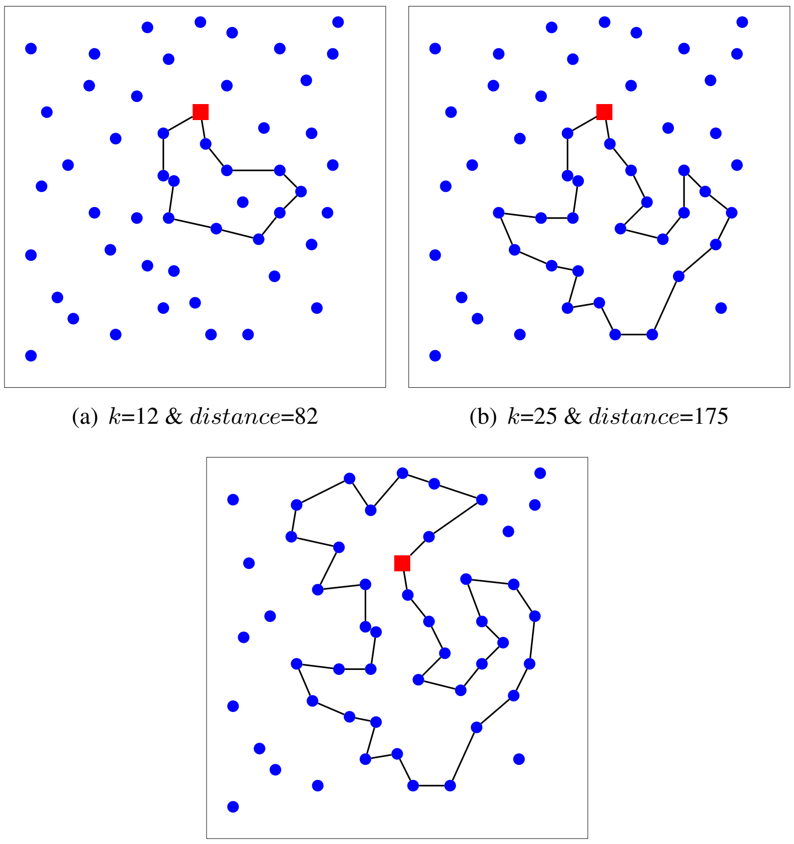
continued ...

Instance	k	**GVNS**			**HH_RAND**			**HH_GREEDY**		
		Best	Worst	Average	Best	Worst	Average	Best	Worst	Average
si535	401	35926	49376	37418.00	35141	42336	**35974.00**	**35114**	**42018**	35993.00
st70	52	444	468	456.00	**428**	**448**	**436.60**	**428**	451	437.70
swiss42	31	**760**	782	770.70	**760**	**774**	**762.50**	**760**	784	763.30
ts225	168	86898	90490	88150.60	**85656**	**87783**	**86888.40**	86070	89191	87537.20
tsp225	168	2934	3236	3067.70	**2665**	**2888**	**2773.60**	2800	2934	2855.50
u159	119	29181	30761	29968.30	**27413**	**28672**	**27927.30**	27817	30354	28991.80
u574	430	35032	64203	39064.40	28950	**31892**	30120.80	**28376**	32163	**30045.00**
u724	543	49460	155156	62157.80	**37730**	**48702**	**40120.60**	39467	54504	42298.80
ulysses16	12	**3183**	**3184**	3183.70	**3183**	**3184**	3183.40	**3183**	**3184**	**3183.30**
ulysses22	16	**2941**	**2942**	2941.70	**2941**	**2942**	2941.90	**2941**	**2942**	**2941.20**

Table 3.7: Performance comparison summary

Scenario		Best Solution Quality						Average Solution Quality					
		GVNS			**HH_GREEDY**			**GVNS**			**HH_GREEDY**		
		$<$	=	$>$	$<$	=	$>$	$<$	=	$>$	$<$	=	$>$
SR_Small	**HH_RAND**	40	33	3	28	36	12	49	20	7	40	21	5
	HH_GREEDY	36	33	7	-	-	-	46	21	9	-	-	-
SR_Medium	**HH_RAND**	54	19	3	32	24	20	62	11	3	55	13	8
	HH_GREEDY	54	20	2	-	-	-	61	11	4	-	-	-
SR_Large	**HH_RAND**	57	15	4	42	15	19	65	6	5	59	6	11
	HH_GREEDY	57	14	5	-	-	-	68	6	2	-	-	-
LR_Small	**HH_RAND**	43	31	2	30	37	9	49	20	7	38	20	18
	HH_GREEDY	38	33	5	-	-	-	50	21	5	-	-	-
LR_Medium	**HH_RAND**	53	19	4	35	22	19	63	11	2	49	12	15
	HH_GREEDY	54	20	2	-	-	-	63	11	2	-	-	-
LR_Large	**HH_RAND**	59	15	2	39	16	21	65	6	5	57	6	13
	HH_GREEDY	60	15	1	-	-	-	70	6	0	-	-	-
Total	**HH_RAND**	306	132	18	206	150	100	353	74	29	298	78	80
	HH_GREEDY	299	135	22	-	-	-	358	76	22	-	-	-

For understanding the convergence behavior of our approaches, three instances of different sizes, namely $gr229$, $ali535$ and $p654$, have been considered. Figures 3.2, 3.3 and 3.4 plot the convergence behavior of our three approaches, viz. GVNS, HH_RAND and HH_GREEDY respectively for the instances $gr229$, $ali535$ and $p654$ under small, medium and large scenarios. The convergence graphs depict that both HH_RAND and HH_GREEDY converges faster than the GVNS. When it comes to the comparison between HH_RAND and HH_GREEDY, there is only a minute difference in their convergence behavior except for the large scenario of $p654$, where HH_RAND clearly converges faster than HH_GREEDY.


(a) k=12 & $distance$=82

(b) k=25 & $distance$=175

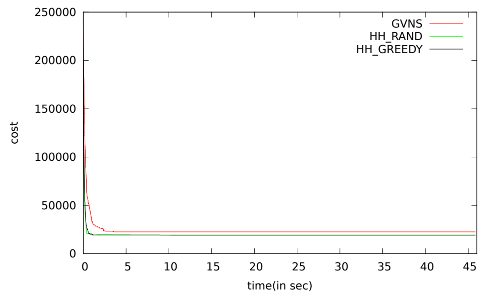
(c) k=38 & $distance$=287

Figure 3.1: Solutions of instance eil51 obtained by GVNS with different k values

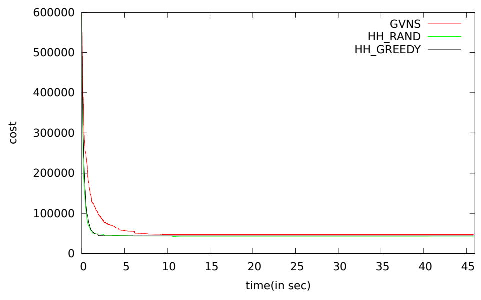
3.5.1 Wilcoxon signed rank test

To check the significance of the results obtained by our approaches, Wilcoxon signed rank test[94] has been performed. The calculator available online[1] is used to conduct the two-tailed Wilcoxon signed rank test with significance criteria set to 5% (i.e. p-value $\leq$ 0.05). Table 3.8 identifies the significant one among our approaches. In Table 3.8, the difference between the normalized values of *Average* obtained by HH_GREEDY / HH_RAND and the compared approach is ranked. The column $NWT/Total$ provides the number of instances without tie out of the total number of instances compared. The R^+ indicates the sum of the ranks for the instances where HH_GREEDY / HH_RAND performed better than its competitor, whereas the R^- indicates the sum of the ranks for the instances where HH_GREEDY / HH_RAND

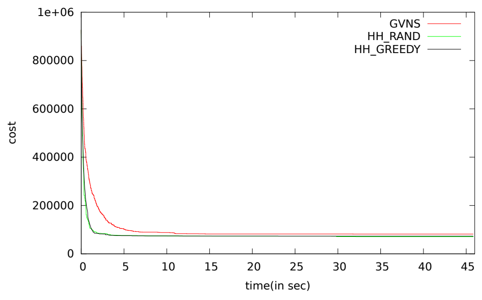
[1]`https://mathcracker.com/wilcoxon-signed-ranks.php`



(a) Small scenario



(b) Medium scenario



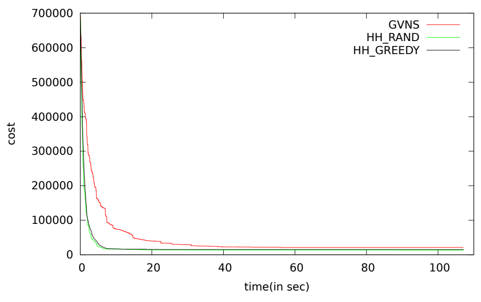
(c) Large scenario

Figure 3.2: Convergence behavior on instance gr229 under different scenarios

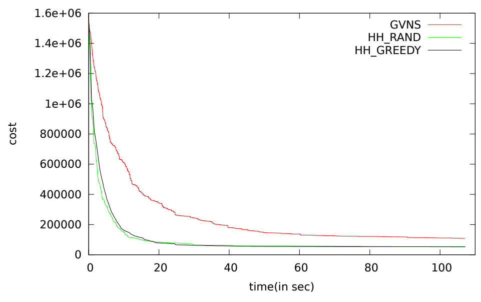
performed worse than its competitor. As the number of instances without tie are more than thirty ($NWT > 30$), the test statistic Z is used to compare with the critical value Z_{Cri} according to Wilcoxon signed rank test [94]. If $Z \leq Z_{Cri}$, then there is a significant difference between the performance of the two compared approaches, otherwise the difference is not significant. Table 3.8 clearly shows that HH_RAND is significant with respect to both GVNS and HH_GREEDY. Moreover, HH_GREEDY is significant with respect to GVNS.

Table 3.8: Results of Wilcoxon signed rank test between our approaches

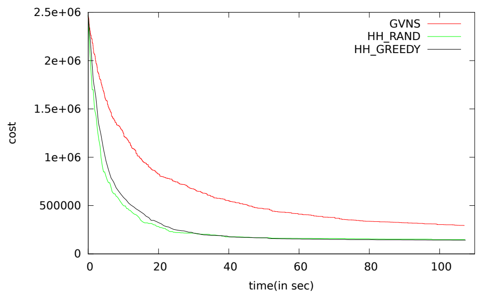
	HH_RAND vs...						**HH_GREEDY vs...**					
	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant
GVNS	382/456	71909	1244	-16.361	-1.960	yes	380/456	70838	1552	-16.169	-1.960	yes
HH_GREEDY	378/456	63175	8456	-12.871	-1.960	yes						



(a) Small scenario

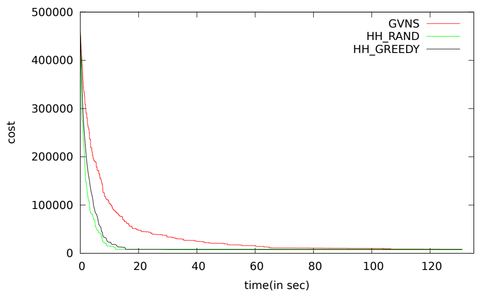


(b) Medium scenario

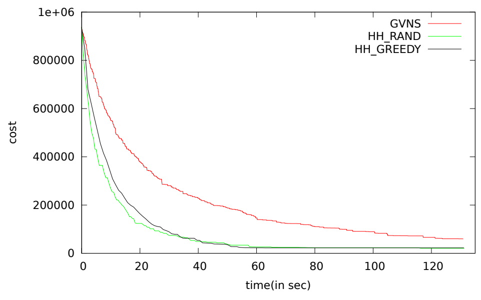


(c) Large scenario

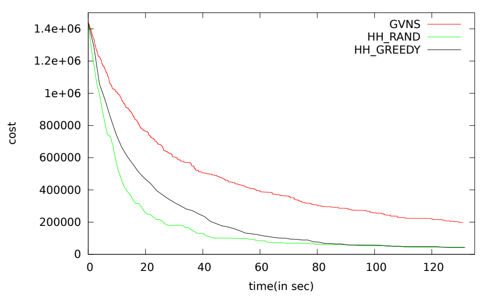
Figure 3.3: Convergence behavior on instance ali535 under different scenarios



(a) Small scenario



(b) Medium scenario



(c) Large scenario

Figure 3.4: Convergence behavior on instance p654 under different scenarios

3.6 Conclusions

In this chapter, we have proposed two approaches for the k-TSP based on general variable neighborhood search and hyper-heuristic. The main component of the proposed GVNS approach is the variable neighborhood descent which uses two neighborhood structures based on exchange (N_1) and swap (N_2) operations. These two neighborhood structures effectively handle both the aspects of the k-TSP, viz. subset selection and permutation. Likewise, hyper-heuristic also incorporates two heuristics (H_1 and H_2) as low level heuristics to handle subset selection and permutation aspects. Two versions of hyper-heuristic, viz. HH_RAND and HH_GREEDY are proposed based on two different selection mechanisms. To evaluate the performance of the various approaches proposed (viz. GVNS, HH_RAND and HH_GREEDY), various k-TSP test instances are derived from the publicly available instances in TSPLIB. All the details regarding how these instances are derived from TSPLIB instances have been provided in this chapter to facilitate reuse of these instances. Computational results on these test instances show that both HH_RAND and HH_GREEDY performed better than GVNS. As far as comparison between HH_RAND and HH_GREEDY is concerned, the former performed better than the latter.

As our approaches are the first heuristic approaches for k-TSP, these approaches will be used as the baseline approaches for evaluating the performance of future heuristic approaches for this problem.

Chapter 4

Family Traveling Salesman Problem

4.1 Introduction

The family traveling salesman problem (FTSP) is an extension of the generalized traveling salesman problem (GTSP) which is a well known TSP variant [111]. Given a set of n nodes partitioned into $1 \leq m \leq n$ clusters, the GTSP is to find a minimum length tour comprising exactly one node from each of these clusters. The family traveling salesman problem (FTSP) is a generalization of the GTSP, where a cluster is termed as a family and the goal is to find a minimum length tour comprising a pre-specified number of nodes (≥ 1) from each family. If exactly one node needs to be chosen from each family then the FTSP is same as the GTSP, and if in addition, each family contains a single node then the FTSP is same as the traveling salesman problem (TSP). Hence, the FTSP is $\mathcal{NP}$-hard as it can be considered as a generalization of the TSP . Clearly, solving this problem involves aspects of both subset selection (selecting pre-specified number of cities from each family) and permutation (arranging the selected cities into a tour of minimum length).

Morán-Mirabal *et al.* [112] introduced the FTSP upon getting motivation from an application of order picking in modern warehouses, where similar items can be stored at different places as the latest technologies like radio frequency identification (RFID) facilitate item localization. For solving the FTSP, the authors have presented a binary integer programming model and two randomized heuristics. The first heuristic is a biased random-key genetic algorithm (BRKGA), whereas the other is a greedy randomized adaptive search procedure with evolutionary path relinking (GRASP + evPR) .

Bernardino *et al.* [113] proposed several mixed integer linear programming formulations for the FTSP based on compact (multi-commodity flow) and non-compact (branch-and-cut) models. Empirical analysis found the non-compact models to be more effective than the compact ones. For solving the large size instances, the authors have proposed an iterated local search (ILS) metaheuristic. The ILS is able to get better solutions within a reasonable time for seven out of eight instances which could not be solved by the exact methods.

To the best of our knowledge, only these two papers are available in the literature on the FTSP.

To solve the FTSP, a hyper-heuristic approach is proposed in this chapter that uses three large neighborhood search methods as low level heuristics. Computational results on a set of 21 standard benchmark instances show that the proposed approach is able to get much better solutions in comparison to the state-of-the-art approaches on most of the instances. Besides, it is several times faster in comparison to these approaches on most of the large instances. Hence, the proposed approach demonstrates the benefit of using large neighborhood search methods as low level heuristics in a hyper-heuristic framework. We have also reported the performance of our approach on a set of 60 new benchmark instances which are generated by using the instances available in the standard TSPLIB.

The remaining part of this chapter is organized as follows: Section 4.2 defines the FTSP formally and illustrates it with the help of an example. A large neighborhood search based hyper-heuristic approach to solve the FTSP is described in Section 4.3. Section 4.4 reports the computational results obtained by our approach and compares them with the results of the state-of-the-art approaches. Finally, some concluding remarks are provided in Section 4.5

4.2 Problem definition

Given an undirected, edge-weighted, complete graph $G = (V, E)$, where $V = \{0, 1, 2, \dots, N\}$ is the set of $N+1$ nodes in which the node 0 represents the depot and the remaining N nodes are partitioned into nf families. $E = \{(i, j) | i, j \in V\}$ is the set of edges, and each edge $(i, j) \in E$ has an associated distance d_{ij}. Let us assume that F_i, n_i and m_i represent the set of nodes, total number of nodes and specified number of nodes to visit in each family $i \in \{1, 2, \dots, nf\}$ respectively. So $V = \{0\} \cup F_1 \cup F_2 \cup \dots F_{nf}$, the number of nodes excluding depot, i.e., $N = n_1 + n_2 + \dots + n_{nf}$ or $(\sum\limits_{i=1}^{nf} n_i = N)$, and the predefined number of nodes to visit excluding depot, i.e., $M = m_1 + m_2 + \dots + m_{nf}$ or $(\sum\limits_{i=1}^{nf} m_i = M)$. The FTSP seeks a minimum length tour starting and ending at the depot 0 and comprising specified number of nodes from each family. We will denote a subset of $M+1$ nodes containing depot 0 and specified number of nodes from each family by V'. We will use binary variable y_i to specify whether a node i belongs to V' ($y_i = 1$) or not ($y_i = 0$), and another binary variable x_{ij} to specify whether an edge (i,j) belongs to the tour ($x_{ij} = 1$) or not ($x_{ij} = 0$). With the help of these notational conventions, the FTSP can be formulated as follows:

$$\text{Minimize} \quad \sum_{i\in V}\sum_{j\in V} d_{ij}x_{ij} \tag{4.1}$$

subject to:

$$\sum_{j\in F_i} y_j = m_i, \quad \forall i \in \{1,2,\ldots,nf\}, \tag{4.2}$$

$$\sum_{i\in V} x_{0i} = 1 = \sum_{i\in V} x_{i0}, \tag{4.3}$$

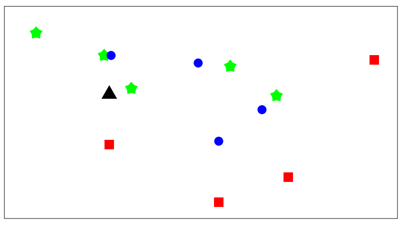
$$\sum_{(i,j)\in E} x_{ij} + \sum_{(j,k)\in E} x_{jk} = 2y_j, \quad \forall j \in V, \tag{4.4}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1, \quad \forall S \subset V' \subset V, \tag{4.5}$$

$$x_{ij}, y_i \in \{0,1\}, \quad \forall (i,j) \in E, i \in V. \tag{4.6}$$

The objective function of the FTSP is represented by Equation (4.1) which minimizes the total tour length. Equation (4.2) enforces the constraint of visiting exactly n_i nodes from each family $i \in \{1,2,\ldots,nf\}$, and Equation (4.3) ensures that a tour must start and end at the depot (i.e., node 0). Equation (4.4) enforces the constraints on the indegree and the outdegree in the tour of the visited nodes. Equation (4.5) is the sub tour elimination constraint. Equation (4.6) restricts the values of decision variables x_{ij} and y_i to either 0 or 1.

Figure 4.1 illustrates the FTSP by using an instance **burma14_1**. As given in Figure 4.1 (a), this instance has 14 cities, out of which the first city is the depot which is represented as black triangle, and other 13 cities are partitioned into 3 families having 4, 5, and 4 cities respectively. The cities belonging to first, second and third families are represented as red squares, green stars and blue circles respectively. The required number of cities to be visited from these 3 families are 2, 2, and 2 respectively. Figure 4.1 (b) depicts a feasible FTSP solution for this instance.


(a) **burma14_1** instance

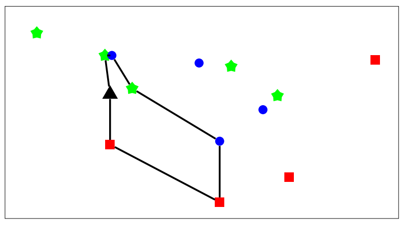

(b) A feasible FTSP solution

Figure 4.1: Illustration of the FTSP

4.3 A large neighborhood seach based hyper-heuristic approach for the FTSP

Our hyper-heuristic(HH) approach for the FTSP makes use of several large neighborhood search methods as low level heuristics. In addition, the best solution obtained through hyper-heuristic is improved further through a local search phase. The salient features of the proposed hyper-heuristic approach have been described in the following subsections.

4.3.1 Solution encoding

Within the proposed HH approach, a solution of the FTSP is represented as a linear permutation of the $M + 1$ visited cities where the depot always occupies the first position. Actually, a tour of the FTSP is a circular permutation and $M + 1$ linear permutations corresponds to a single circular permutation comprising $M + 1$ cities. Hence, there is a redundancy when a linear permutation is used to represent a circular permutation. By reserving the first position for the depot, this redundancy got eliminated. It is worth mentioning that, none of the components of HH can alter this position of depot.

Theorem 1: By recalling M as the number of cities required to be visited out of N cities excluding the depot, nf as the number of families, n_i as the number of cities in each family i ($\sum_{i=1}^{nf} n_i = N$), and m_i as the specified number of cities to visit from family i ($\sum_{i=1}^{nf} m_i = M$). The search space size using our encoding is $\prod_{i=1}^{nf} \binom{n_i}{m_i} \times M!$

Proof. From each family i, m_i cities need to be visited out of n_i cities. These m_i cities can be chosen from n_i cities in $\binom{n_i}{m_i}$ ways. The total chosen M cities can be ordered in $M!$ ways.

Hence, $(\prod_{i=1}^{nf} \binom{n_i}{m_i}) \times M!$ is the size of the search space using our encoding. $\square$

4.3.2 Fitness

The objective function (Equation (4.1)) itself is used as the fitness function, i.e., fitness of a solution is the total length of the tour being represented by that solution. As FTSP is a minimization problem, a solution having a smaller value of the fitness function is regarded as better in comparison to a solution having a larger value.

4.3.3 Initial solution generation

An initial solution is generated iteratively by using a greedy strategy. Starting with the partial tour containing only the depot, an iterative process ensues where during each iteration an unvisited city which yields least increase in the cost of the tour is chosen. To identify this city, all the unvisited cities belonging to those families for which specified number of cities are yet to be included in the tour have to be examined in all the possible insertion positions in the tour at that moment. The identified city will be inserted at its best possible insertion position in the tour. This procedure repeats until a feasible solution is obtained, i.e., specified number of cities are included from each family in the tour.

4.3.4 Neighboring solution generation via large neighborhood search methods

A neighboring solution generation procedure has predominant role in directing the search process towards optimal solution by maintaining a proper balance among the efforts spent on different aspects of a problem. The hyper-heuristic produces a neighboring solution by applying one of the low level heuristics on the current solution. Our hyper-heuristic approach utilizes three large neighborhood search methods as low-level heuristics, viz. LNS_1, LNS_2 and LNS_3 to deal with permutation and subset selection aspects of the FTSP. These three large neighborhood search methods are described below:

1. LNS_1*:* This method tackles the permutation aspect of the FTSP. As part of this method, a tour is destroyed by removing some cities randomly from the tour and a repair mechanism is followed. All such removed cities are added to a set and then the repair mechanism chooses cities from this set randomly, one-by-one, and inserts them at best possible positions in the tour. This repair mechanism continues until it gets a feasible solution by

inserting all the removed cities back into the tour. The number of cities to be removed is restricted by a lower and upper limits and it can be any number between these limits. The lower limit is 1 and upper limit is represented by a parameter NR. It is important to note that, in this method, no change is permitted in the constituent cities of the tour, but there can be a change in their respective positions in the tour.

2. *LNS_2:* This method simultaneously deals with subset selection and permutation aspects of the FTSP. As part of the destroy mechanism, a tour is destroyed by removing some cities from the tour randomly and an infeasible solution is returned. All these removed cities are added to the set of unvisited cities which already contains the unvisited cities that are not part of the tour. All unvisited cities belonging to families still having specified number of cities in the tour are removed from further consideration by deleting them from the set of unvisited cities. The repair mechanism iteratively chooses a city from this set of unvisited cities that yields the cheapest insertion cost. To identify this city, all the unvisited cities need to be examined in all the possible insertion positions in the tour at that moment. This repair mechanism is repeated until a feasible solution is obtained that satisfies the constraint of each family. Further, as soon as the specified number of cities are reached for a family, all unvisited cities belonging to that family are removed from the set of unvisited cities. Like the LNS_1, in this method also, the number of cities to be removed is restricted by the same parameter NR. It is pertinent to mention that, in this method, the changes are permitted in both the constituent cities and their respective positions in the tour.

3. *LNS_3:* Like LNS_2, this method also handles both subset selection and permutation aspects of the FTSP. As part of the destroy mechanism, some unvisited cities are selected randomly from the set of all unvisited cities and added to the tour one-by-one at their best possible position thereby making the solution infeasible as number of cities in the tour for some families exceed their specified number. This infeasible solution is handled by repair mechanism which removes the cities from the tour to get a feasible solution. The repair mechanism iteratively chooses a city from the tour whose removal reduces the cost of the tour by maximum amount. To identify this city, all the cities in the tour belonging to families having excess representation in the tour need to be examined. This repair mechanism is repeated until it gets a feasible solution that satisfies the constraint of each family. The number of cities to be added is restricted by a lower and upper limits

and it can be any number between these limits. The lower limit is 1 and the upper limit is represented by a parameter NA. It is important to note that, in this method, changes are permitted in both the constituent cities and their respective positions in the tour.

The pseudo-code for the procedure of generating a neighboring solution is given in Algorithm 12 in which the values 1, 2 and 3 respectively correspond to LNS_1, LNS_2 and LNS_3. Basically, *Create_Neighbor(S, j)* requires as input a solution S & a parameter j indicating the choice of the large neighborhood search method, and returns a neighboring solution S' by applying the chosen large neighborhood search method on S.

4.3.5 Other features: selection mechanism and acceptance criteria

The choice of the selection mechanism influences the quality of the solution returned by a hyper-heuristic. We have examined the hyper-heuristic with three selection mechanisms. These mechanisms are described below.

- **Random:** In this selection mechanism, at each step, a low level heuristic is selected at random and then a neighboring solution obtained by applying that selected heuristic is returned. The hyper-heuristic with random selection mechanism will be referred to as HH(Random).

- **Greedy:** In this selection mechanism, at each step, all the low level heuristics are used to get neighboring solutions and the best solution among them is returned. The heuristic whose solution is returned is deemed to be selected by the greedy selection mechanism at that step. The hyper-heuristic with greedy selection mechanism will be referred to as HH(Greedy).

- **Rand2:** In this selection mechanism, at each step, two low level heuristics are selected at random from the available three low level heuristics, and these two are used to get two neighboring solutions, and best solution among these two is returned. The heuristic whose solution is returned is deemed to be selected by rand2 selection mechanism at that step. The hyper-heuristic with rand2 selection mechanism will be referred to as HH(Rand2).

Even though, there exist various selection mechanisms in the literature, the reason for choosing only these three mechanisms lies in the fact that the number of low-level heuristics are

Algorithm 12: Pseudo-code for the procedure of generating a neighboring solution

```
Input: $S$ (An input solution)
Output: $S'$ (A neighboring solution of $S$)
function Create_Neighbor($S$, $j$)
begin
Set $S'$ to $S$;
if $j == 1$ then
    /**********Applying $LNS\_1$**********/
    if $NR > M$ then
        Set $r$ to a random value between 1 and $M$;
        Randomly remove $r$ number of cities from $S'$ and add them to a set $UA$;
    else
        Set $r$ to a random value between 1 and $NR$;
        Randomly remove $r$ number of cities from $S'$ and add them to a set $UA$;
    foreach city c chosen randomly from $UA$ do
        Follow the procedure described in $LNS\_1$ (Section 4.3.4-Item 1) to insert $c$ into $S'$;
else if $j == 2$ then
    /**********Applying $LNS\_2$**********/
    Set $U_c$ to all unvisited cities;
    if $NR > M$ then
        Set $r$ to a random value between 1 and $M$;
        Randomly remove $r$ number of cities from $S'$ and add them to the set of unvisited cities
        $U_c$;
    else
        Set $r$ to a random value between 1 and $NR$;
        Randomly remove $r$ number of cities from $S'$ and add them to the set of unvisited cities
        $U_c$;
    while $S'$ is infeasible do
        Follow the procedure described in $LNS\_2$ (Section 4.3.4-Item 2) to insert a city $c$ from
        $U_c$ into $S'$;
else if $j == 3$ then
    /**********Applying $LNS\_3$**********/
    if $NA > (N - M)$ then
        Set $r$ to a random value between 1 and $(N - M)$;
        Add $r$ number of randomly chosen cities in their best positions in $S'$;
    else
        Set $r$ to a random value between 1 and $NA$;
        Add $r$ number of randomly chosen cities in their best positions in $S'$;
    while $S'$ is infeasible do
        Follow the procedure described in $LNS\_3$ (Section 4.3.4-Item 3) to remove a city $c$ from
        $S'$;
return $S'$;
end function
```

less (i.e., only 3) in our hyper-heuristic approach, and other mechanisms can be beneficial only when there are large number of low-level heuristics [70].

There are many acceptance criterias available in the literature [70]. Out of these, we have examined our hyper-heuristic with AM (all moves), OI (only improvement) acceptance criterias and the following proposed acceptance criteria: Compare the fitness of the solution obtained after applying the low-level heuristic with the fitness of the current solution. If the solution obtained after applying the low-level heuristic is better, it will always be accepted. However, if this solution is worse, then it will be accepted only when the current solution has not improved for a $Limit_{noimp}$ number of consecutive iterations. The main reason behind accepting a worse solution is to save computation time by abandoning an unworthy locally optimal solution. Our acceptance criteria is compared with AM and OI criterias. In comparison to these criterias, our acceptance criteria provided the better results and only those results are reported here.

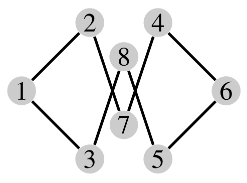
4.3.6 The local search phase

The best solution obtained by HH approach is improved further through four local search methods. All these local search methods work on the first improvement strategy, i.e., in the search process once an improved solution is encountered then this improved solution immediately replaces the current solution and the next local search method is applied. These local search methods are applied one after the other in an iterative manner until no improvement is possible through any of them. In essence, a solution returned by the local search phase is locally optimal with respect to all these four methods. These four local search methods are explained below in the order in which they are applied.

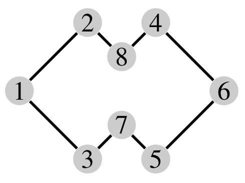
1. *Swap:* This method tries to swap the positions of two cities within the tour in order to get an improved solution. It returns either an improved solution or the same solution if there is no swap possible between any pair of cities in the tour that improves the solution.

2. *Relocate:* This method tries to relocate a city by removing a city first, and then, reinserting it at the best possible position in the tour. This method returns either an improved solution or the same solution if no relocation that improves the solution is possible for any city in the tour.

3. *Exchange:* This method tries to exchange a visited city from a family with an unvisited city from the same family. For evaluating the benefit of an exchange, the visited city is

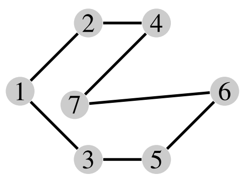


(a) Prior to swap move

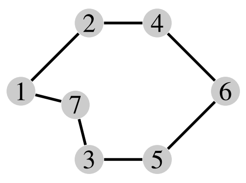


(b) After swap move

Figure 4.2: Illustration of swap move

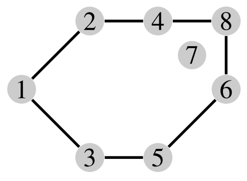


(a) Prior to relocate move

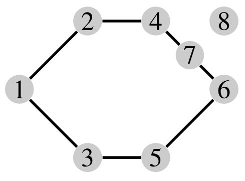


(b) After relocate move

Figure 4.3: Illustration of relocate move

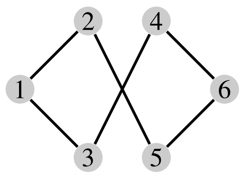


(a) Prior to exchange move

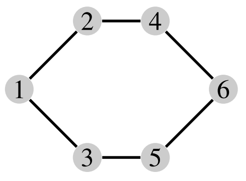


(b) After exchange move

Figure 4.4: Illustration of exchange move



(a) Prior to 2-opt move



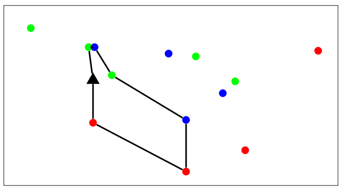
(b) After 2-opt move

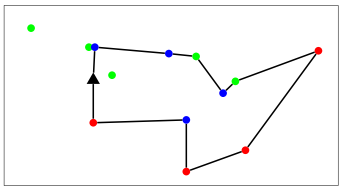
Figure 4.5: Illustration of 2-opt move

removed from the tour and unvisited city is inserted at its best possible position. This local search returns either an improved solution or the same solution if no visiting city can be exchanged with an unvisited city such that the tour length is reduced.

4. $2 - opt\ move$: This method tries to improve a solution by removing two edges from the tour and then adding two new edges in place of them in order to get a valid tour. It returns either an improved solution or the same solution if there is no improvement possible for any pair of edges in the tour.

These four methods are illustrated in Figures 4.2–4.5. The pseudo-code for the HH approach is given in Algorithm 13 where the function *Selection_Mechanism(S, $\mathbb{S}_{LH}$)* needs as input a solution S and a set $\mathbb{S}_{LH}$ of low level heuristics, and returns a new solution as per selection mechanism by making use of *Create_Neighbor()* function one or more times.


(a) **burma14_1**


(b) **burma14_2**

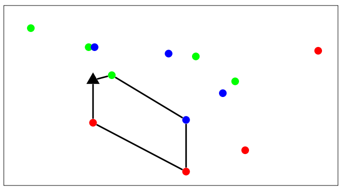

(c) **burma14_3**

Figure 4.6: FTSP solutions of **burma14** instance by our approach

Algorithm 13: HH approach for the FTSP

```
Input: Parameters required for the HH algorithm and an instance of the FTSP
Output: Best solution achieved by HH algorithm
$S \leftarrow$ Generate_Initial_Solution();
$best \leftarrow S$;
$S.counter \leftarrow 0$;
while termination condition is not met do
    $S' \leftarrow$ Selection_Mechanism($S$, $\mathbb{S}_{LH}$)
    if $S'$ is better than $S$ then
        $S.counter \leftarrow 0$;
        $S \leftarrow S'$;
        if $S$ is better than $best$ then
            $best \leftarrow S$;
    else
        $S.counter++$;
        if $S.counter$ is greater than $Limit_{noimp}$ then
            replace $S$ with neighboring solution $S'$;
$best \leftarrow Local\_Search\_Phase(best)$;
return $best$;
```

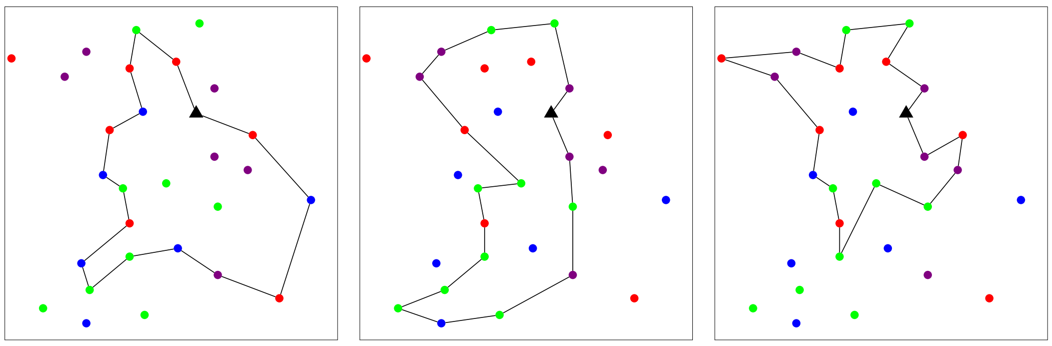

(a) **bayg29_1** (b) **bayg29_2** (c) **bayg29_3**

Figure 4.7: FTSP solutions of **bayg29** instance by our approach

4.4 Computational results

For testing the performance of our HH approach, benchmark instances of [112] are used. Morán-Mirabal *et al.* [112] tested the performance of their approaches on 21 instances. These instances are having number of nodes ranging from 14 to 1002, number of families ranging from 3 to 40 and number of nodes (excluding the depot) to be visited by the salesman ranging from 4 to 486. Other details about these 21 instances can be found in Table 8 of [113].

We have implemented all the three versions of HH approach, viz. HH(Random), HH(Greedy)

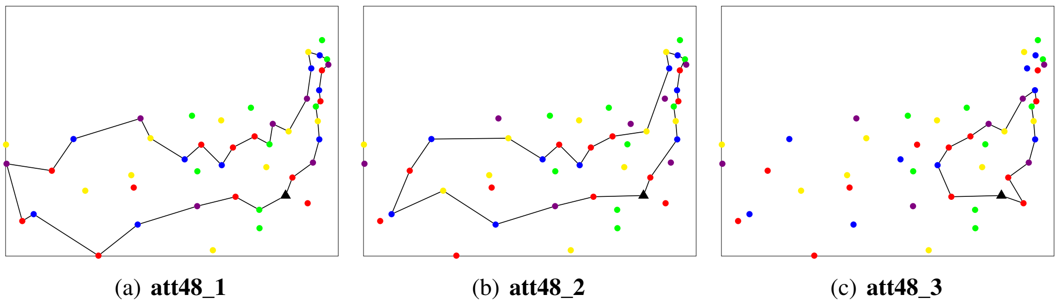

(a) **att48_1** (b) **att48_2** (c) **att48_3**

Figure 4.8: FTSP solutions of **att48** instance by our approach

and HH(Rand2) in C. These approaches have been executed on a Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM. For benchmarking, our approaches are compared with biased random key genetic algorithm (BRKGA-FTSP) [112], greedy randomized adaptive search procedure (GRASP+evPR-FTSP) [112] and iterated local search (ILS-FTSP) [113]. Like the approaches of [112] and [113], our approaches are also executed five independent times on each instance. Our approaches use maximum time limit as the termination condition (see while loop in Algorithm 13), i.e., our approaches terminate when the total execution time exceeds $0.01 \times (N + 1)$ seconds.

Figures 4.6, 4.7 and 4.8 plot the solutions obtained by our approaches for different instances of **burma14**, **bayg29** and **att48** respectively. Please note that all our approaches found the same solution in all the five runs on these 3 instances. In all these plots, the depot is shown as a black triangle. As shown in Figure 4.6, the **burma14** instances have three families of cities and they are represented in three colors (red, green, blue). Figure 4.7 plots the solutions of **bayg29** instances which have four families of cities and they are represented in four colors (red, green, blue, violet). Figure 4.8 plots the solutions of **att48** instances which have five families of cities and they are represented in five colors (red, green, blue, violet, yellow).

4.4.1 Parameter settings

Being the generic approaches, our HH approaches do not require many parameters. Three parameter values are used in all our HH approaches. The first parameter is NR which specifies the maximum number of cities that can be removed in both LNS_1 and LNS_2 methods. To determine this value, we have examined the set $\{25, 50, 75\}$ only to find that with value 50 good objective function values are obtained (i.e. $NR = 50$). The second parameter is NA which specifies the maximum number of cities that can be added in LNS_3 method. Here also, we examined the set $\{25, 50, 75\}$ and found that with value 50 good objective function values are obtained (i.e. $NA = 50$). The third and last parameter is $Limit_{noimp}$ which is useful in escaping from local optima. If a solution has not improved over $Limit_{noimp}$ number of iterations, then the neighboring solution will be accepted even if it has a fitness worse than the original solution. To determine this value, we have examined the set $\{50, 75, 100\}$ and found that value 50 provides the good objective function values (i.e. $Limit_{noimp} = 50$).

4.4.2 Comparison of our approaches with previously proposed approaches

Table 4.1 compares the performance of our approaches (HH(Random), HH(Greedy), HH(Rand2)) with the approaches proposed in [112] and [113] on 21 benchmark instances. In this table, the first column (*Name*) provides the name of the instance. The second column (*Nodes*) shows the number of nodes including the depot. The third column (*Families*) lists the number of families in the respective instance. The fourth column (*Visits*) provides the total number of cities to be visited by the salesman. For each approach, the columns *Average*, *Best* and *Worst* report the average, best and worst objective function values obtained over five runs by that approach. The column *time* reports the average execution time in seconds over five runs by that approach. The detailed results of BRKGA-FTSP and GRASP+evPR-FTSP are taken from [112] where they executed both the approaches on a system having 2.50 GHz Core-i5-2450M processor and 6 GB of RAM. They reported the average and best objective function values in addition to the average execution time over five runs. The detailed results of ILS-FTSP are taken from [113] where they executed ILS approach on a system having 3.60 GHz Core-i7 processor and 8 GB of RAM. They reported average, best and worst objective function values in terms of percentage deviation (%) from the best values of those instances in [112]. From the best result and percentage deviation (%) information for each instance, we have calculated the average, best and worst objective values for ILS-FTSP approach. Like ILS-FTSP, for our approaches (HH(Random), HH(Greedy), HH(Rand2)) also we reported average, best and worst objective function values in addition to the average execution time over five runs. The last row *No. of Best* counts the number of instances on which a best solution is obtained by the corresponding approach. The best values among all the approaches are represented in bold font for easy identification. Table 4.1 clearly shows the superiority of our approaches over previously proposed approaches. Our approaches returned solutions of much better quality on most of the instances. According to the average objective function values, comparison among three versions of HH shows that HH(Greedy) obtained better values on higher number of instances (i.e., 14 out of 21), whereas HH(Random)

Table 4.1: Results of various approaches on 21 instances available in the literature

Instance				**BRKGA - FTSP**			**GRASP + evPR - FTSP**			**ILS - FTSP**				**HH(Random) - FTSP**				**HH(Greedy) - FTSP**				**HH(Rand2) - FTSP**			
Name	Nodes	Families	Visits	Average	Best	Time	Average	Best	Time	Average	Best	Worst	Time	Average	Best	Worst	Time	Average	Best	Worst	Time	Average	Best	Worst	Time
burma_1	14	3	6	**13.93**	**13.93**	0.01	**13.93**	**13.93**	0.01	**13.93**	**13.93**	**13.93**	0.00	**13.93**	**13.93**	**13.93**	0.15	**13.93**	**13.93**	**13.93**	0.15	**13.93**	**13.93**	**13.93**	0.15
burma_2	14	3	10	**25.66**	**25.66**	0.01	**25.66**	**25.66**	0.03	**25.66**	**25.66**	**25.66**	0.00	**25.66**	**25.66**	**25.66**	0.15	**25.66**	**25.66**	**25.66**	0.15	**25.66**	**25.66**	**25.66**	0.15
burma_3	14	3	4	**11.89**	**11.89**	0.01	**11.89**	**11.89**	0.01	**11.89**	**11.89**	**11.89**	0.00	**11.89**	**11.89**	**11.89**	0.15	**11.89**	**11.89**	**11.89**	0.15	**11.89**	**11.89**	**11.89**	0.15
bayg29_1	29	4	16	**5345.86**	**5345.86**	3.40	**5345.86**	**5345.86**	8.05	**5345.86**	**5345.86**	**5345.86**	0.00	**5345.86**	**5345.86**	**5345.86**	0.30	**5345.86**	**5345.86**	**5345.86**	0.30	**5345.86**	**5345.86**	**5345.86**	0.30
bayg29_2	29	4	17	**5791.01**	**5791.01**	1.40	**5791.01**	**5791.01**	75.74	5791.59	5791.09	5792.17	0.00	**5791.01**	**5791.01**	**5791.01**	0.30	**5791.01**	**5791.01**	**5791.01**	0.30	**5791.01**	**5791.01**	**5791.01**	0.30
bayg29_3	29	4	18	**5544.33**	**5544.33**	1.60	**5544.33**	**5544.33**	0.03	5602.55	**5544.33**	5664.64	0.00	**5544.33**	**5544.33**	**5544.33**	0.30	**5544.33**	**5544.33**	**5544.33**	0.30	**5544.33**	**5544.33**	**5544.33**	0.30
att48_1	48	5	34	**23686.02**	**23686.02**	143.40	23709.62	**23686.02**	2938.77	24590.80	24555.28	24614.49	0.00	**23686.02**	**23686.02**	**23686.02**	0.49	**23686.02**	**23686.02**	**23686.02**	0.49	**23686.02**	**23686.02**	**23686.02**	0.49
att48_2	48	5	25	20609.09	20609.09	62.80	20635.57	20635.57	7199.22	20751.30	20609.10	20951.21	0.00	**20609.08**	**20609.08**	**20609.08**	0.49	**20609.08**	**20609.08**	**20609.08**	0.49	**20609.08**	**20609.08**	**20609.08**	0.49
att48_3	48	5	15	**9024.58**	**9024.58**	1.80	**9024.58**	**9024.58**	0.05	**9024.58**	**9024.58**	**9024.58**	0.00	**9024.58**	**9024.58**	**9024.58**	0.49	**9024.58**	**9024.58**	**9024.58**	0.49	**9024.58**	**9024.58**	**9024.58**	0.49
bier127_1	127	10	62	36950.75	36913.74	498.80	36856.17	36800.39	3.65	35833.41	35037.86	36278.38	4.00	33709.87	33709.74	33710.08	1.28	**33709.73**	**33709.73**	**33709.74**	1.28	33709.74	**33709.73**	**33709.74**	1.28
bier127_2	127	10	85	98333.46	98216.10	1413.80	98370.63	97615.41	2966.56	94397.78	94051.71	94566.38	4.00	**89714.58**	**89519.01**	90272.30	1.28	89860.99	**89519.01**	**90130.12**	1.28	89791.96	89631.29	90261.67	1.28
bier127_3	127	10	60	50891.36	50513.10	1048.60	50920.77	50715.49	11.26	48933.78	48699.92	49501.72	3.00	48052.30	**47827.02**	48191.14	1.28	47922.96	47861.52	**48139.01**	1.28	**47921.17**	47835.25	**48139.01**	1.28
a280_1	280	20	179	2164.40	2126.34	14760.40	1898.17	1891.16	218.28	1972.29	1946.57	1994.23	37.00	1737.52	1723.47	1744.54	2.82	**1723.92**	**1714.91**	**1730.80**	2.82	1735.66	1728.59	1741.44	2.82
a280_2	280	20	156	1980.84	1925.28	14759.20	1702.33	1697.48	2700.88	1697.48	1688.13	1717.34	36.00	1550.17	1547.85	1552.23	2.82	**1549.16**	**1546.97**	1552.84	2.81	1550.19	1547.49	**1551.90**	2.81
a280_3	280	20	141	17740.78	1720.23	14758.40	1598.49	1597.25	6.91	-	1584.75	-	-	1436.37	1430.36	1449.04	2.81	**1427.25**	**1421.90**	**1432.21**	2.81	1436.68	1428.53	1453.36	2.81
gr666_1	666	30	357	2733.62	2625.69	22271.60	1848.50	1817.06	6610.22	1685.87	1642.35	1714.58	532.00	1497.02	1484.58	1506.26	6.77	**1494.66**	**1478.83**	1508.50	6.78	1496.41	1491.64	**1501.20**	6.76
gr666_2	666	30	328	2390.46	2275.80	22269.00	1451.10	1443.05	4005.06	1305.38	1293.96	1337.56	535.00	**1221.36**	**1212.01**	1229.90	6.76	1224.45	1216.03	1233.08	6.75	1222.71	1213.84	**1228.50**	6.75
gr666_3	666	30	328	3062.17	2426.59	22269.75	1403.00	1384.18	7199.99	1337.39	1320.12	1356.22	562.00	1162.22	1159.60	1164.65	6.74	1159.37	**1154.87**	**1164.50**	6.72	**1159.35**	1155.76	1165.31	6.75
pr1002_1	1002	40	486	432665.72	421061.62	31568.00	164721.66	163461.80	20.72	148129.08	145303.34	150630.05	2156.00	**129699.92**	**129086.97**	**130162.18**	10.92	130680.57	129093.55	131774.52	10.73	130811.09	129189.30	132356.59	10.97
pr1002_2	1002	40	538	445954.00	421761.00	31566.75	184440.19	182144.12	8.86	161598.27	159933.64	163638.28	2008.00	142594.02	141714.00	143649.66	12.01	142664.55	**141586.38**	143889.34	12.24	**142244.41**	141763.23	**142903.69**	11.54
pr1002_3	1002	40	463	326561.62	284856.22	31565.50	149838.12	149456.62	227.56	141759.61	137385.12	144225.64	1666.00	**117196.44**	116625.08	**118491.34**	11.04	117522.10	116207.23	118623.29	10.76	117506.23	**115744.56**	120641.81	10.68
No. of Best				8	8		7	8		5	6	5		13	13	11		**14**	**17**	**15**		12	11	**15**	

Table 4.2: Average distribution (in terms of percentage) of three LNS methods in our approaches on 21 instances available in the literature

Name of approach	LNS_1	LNS_2	LNS_3
HH(Random)	33.40	33.29	33.31
HH(Greedy)	42.75	28.75	28.50
HH(Rand2)	34.04	31.66	34.30

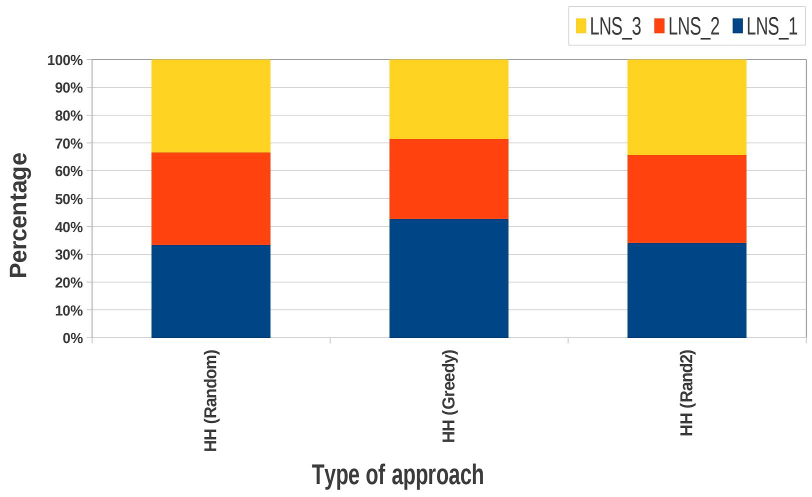


Figure 4.9: Graphical representation of average distribution values presented in Table 4.2

and HH(Rand2) obtained better values for 13 and 12 number of instances respectively. As the previous approaches were executed on systems which are different from the system used to execute our approaches, the running times of the previous approaches can not be compared precisely with our approaches. However, a rough comparison of execution times can always be made. Based on publicly available information about processor speeds and capabilities, the system used to execute the approaches of [112] is inferior to the system used to execute our approaches, whereas the system used in [113] is superior. Even after considering this fact, we can safely say that our approaches are several times faster than previous approaches on most of the instances with more than 100 nodes.

To analyze the relative contribution of three LNS methods in the performance of our approaches, the average distribution (in terms of percentage) of each of these three methods (LNS_1, LNS_2 and LNS_3) selected is recorded for 21 benchmark instances available in the literature and presented in Table 4.2. Figure 4.9 shows these average distributions graphically.

The usage of LNS_1 shows the importance of ordering the cities in the tour. The usage of both LNS_2 and LNS_3 shows the importance of selecting proper subset of cities and at the same time ordering these selected cities in the tour. From the Table 4.2 and the Figure 4.9, it can be observed that the usage of three low level heuristics is almost same in both HH(Random) and HH(Rand2), thus giving almost equal importance to all three heuristics. However, in HH(Greedy), more importance has been given to LNS_1 than LNS_2 and LNS_3 (LNS_2 and LNS_3 got almost equal importance).

4.4.3 Comparison of our approaches on the newly generated instances

In addition to the 21 benchmark instances available in the literature, we have generated 60 new instances for FTSP based on instances available in the TSPLIB . These newly generated instances have number of nodes ranging from 51 to 1379, number of families ranging from 2 to 70 and number of nodes (excluding the depot) to be visited by the salesman ranging from 30 to 773. Actually, we thought that 21 instances are not sufficient to evaluate the performance of an approach. Hence, to facilitate future researchers to test the performance of their approaches on a larger data set, we generated these new instances and reported the results of our approaches for these new instances. These instances can be obtained from the authors through e-mail. Table 4.3 compares the performance of our approaches (HH(Random), HH(Greedy), HH(Rand2)) on these 60 newly generated instances. We have reported the results of all our approaches in same format as in Table 4.1. The best values among all our approaches are represented in bold font for easy identification. According to the average objective function values, comparison among three versions of HH shows that HH(Greedy) obtained better values on higher number of instances (i.e., 36 out of 60), whereas HH(Random)and HH(Rand2) obtained better values for 16 and 24 number of instances respectively.

For these newly generated instances also, the average distribution (in terms of percentage) of each of these three heuristics (LNS_1, LNS_2 and LNS_3) selected by our approaches is presented in the Table 4.4 and the Figure 4.10. From the Table 4.4 and the Figure 4.10, it can be observed that the usage pattern of three low level heuristics is almost same in both HH(Greedy) and HH(Rand2) where less importance has been given to LNS_1 than LNS_2 and LNS_3 (there is only a minute difference in the usage of LNS_2 and LNS_3).

Table 4.3: Results of our approaches on 60 newly generated instances

Instance				**HH(Random) - FTSP**				**HH(Greedy) - FTSP**				**HH(Rand2) - FTSP**			
Name	Nodes	Families	Visits	Average	Best	Worst	Time	Average	Best	Worst	Time	Average	Best	Worst	Time
eil51_1	51	2	35	**270.00**	**270.00**	**270.00**	0.52	270.40	**270.00**	271.00	0.52	**270.00**	**270.00**	**270.00**	0.52
eil51_2	51	3	30	**274.00**	**274.00**	**274.00**	0.52	**274.00**	**274.00**	**274.00**	0.52	**274.00**	**274.00**	**274.00**	0.52
eil51_3	51	3	37	**303.00**	**303.00**	**303.00**	0.52	**303.00**	**303.00**	**303.00**	0.52	**303.00**	**303.00**	**303.00**	0.52
kroC100_1	100	5	44	**10358.00**	**10358.00**	**10358.00**	1.01	**10358.00**	**10358.00**	**10358.00**	1.01	**10358.00**	**10358.00**	**10358.00**	1.01
kroC100_2	100	5	53	**11404.60**	**11395.00**	**11443.00**	1.01	11462.60	**11395.00**	11564.00	1.01	11436.00	**11395.00**	11508.00	1.01
kroC100_3	100	5	69	**15088.00**	**15088.00**	**15088.00**	1.01	**15088.00**	**15088.00**	**15088.00**	1.01	**15088.00**	**15088.00**	**15088.00**	1.01
ch130_1	130	6	81	3573.80	**3566.00**	3581.00	1.31	**3566.20**	**3566.00**	**3567.00**	1.31	3566.80	**3566.00**	3570.00	1.31
ch130_2	130	6	87	3876.00	3868.00	3893.00	1.31	**3867.20**	**3857.00**	**3875.00**	1.31	3871.20	**3857.00**	3882.00	1.31
ch130_3	130	7	76	3518.40	**3518.00**	3520.00	1.31	**3518.00**	**3518.00**	**3518.00**	1.31	**3518.00**	**3518.00**	**3518.00**	1.31
kroA150_1	150	8	114	21216.60	21142.00	21313.00	1.51	21197.00	**21063.00**	21266.00	1.51	**21179.80**	21157.00	**21218.00**	1.51
kroA150_2	150	8	94	16023.20	**15978.00**	16124.00	1.51	**15978.20**	**15978.00**	**15979.00**	1.51	15980.80	**15978.00**	15985.00	1.51
kroA150_3	150	8	71	12824.20	12793.00	12883.00	1.51	**12741.00**	**12741.00**	**12741.00**	1.51	12783.60	**12741.00**	12844.00	1.51
si175_1	175	10	135	16479.80	16460.00	16492.00	1.76	16473.00	**16459.00**	16514.00	1.76	**16472.00**	16463.00	**16486.00**	1.76
si175_2	175	9	100	12453.00	**12429.00**	12469.00	1.76	**12438.60**	12433.00	**12443.00**	1.76	12444.20	12439.00	12450.00	1.76
si175_3	175	9	127	15427.80	15417.00	15436.00	1.76	**15412.60**	**15397.00**	**15421.00**	1.76	15418.00	15408.00	15432.00	1.76
brg180_1	180	9	108	**1384.00**	**1330.00**	**1420.00**	1.81	1388.00	1340.00	**1420.00**	1.81	**1384.00**	1340.00	**1420.00**	1.81
brg180_2	180	9	79	**1026.00**	980.00	1080.00	1.81	1038.00	**970.00**	1080.00	1.81	**1026.00**	1010.00	**1050.00**	1.81
brg180_3	180	10	97	1310.00	**1280.00**	1340.00	1.81	**1300.00**	**1280.00**	**1330.00**	1.81	1326.00	**1280.00**	1410.00	1.81
kroB200_1	200	10	126	17833.00	**17703.00**	17885.00	2.01	17800.60	17753.00	17868.00	2.01	**17796.00**	17738.00	**17865.00**	2.01
kroB200_2	200	10	103	17953.20	17903.00	17994.00	2.01	**17769.80**	**17659.00**	**17828.00**	2.01	17790.00	17681.00	17916.00	2.01
kroB200_3	200	10	127	19626.00	**19562.00**	19710.00	2.01	**19589.40**	19574.00	19620.00	2.01	19604.60	19569.00	**19619.00**	2.01
ts225_1	225	12	142	78454.60	**78206.00**	79271.00	2.26	**78337.40**	**78206.00**	**78863.00**	2.26	78530.40	**78206.00**	79082.00	2.26
ts225_2	225	12	119	70803.60	**70792.00**	70844.00	2.26	**70792.00**	**70792.00**	**70792.00**	2.26	**70792.00**	**70792.00**	**70792.00**	2.26
ts225_3	225	12	121	71893.00	**71829.00**	71989.00	2.26	**71829.00**	**71829.00**	**71829.00**	2.26	**71829.00**	**71829.00**	**71829.00**	2.26
pr226_1	226	12	130	54161.60	**53998.00**	54267.00	2.27	**54034.00**	**53998.00**	54080.00	2.27	54039.20	54035.00	**54056.00**	2.27
pr226_2	226	12	139	49059.20	**48978.00**	49129.00	2.27	**48978.00**	**48978.00**	**48978.00**	2.27	48985.40	**48978.00**	49015.00	2.27
pr226_3	226	12	135	**48744.00**	**48744.00**	**48744.00**	2.27	**48744.00**	**48744.00**	**48744.00**	2.27	**48744.00**	**48744.00**	**48744.00**	2.27
gr229_1	229	11	131	87373.40	**87174.00**	87743.00	2.30	87343.80	87220.00	87446.00	2.30	**87307.20**	87253.00	**87334.00**	2.30
gr229_2	229	11	146	88265.40	**87599.00**	88658.00	2.30	88191.20	88090.00	88355.00	2.30	**88116.20**	87847.00	**88311.00**	2.30
gr229_3	229	12	123	60478.00	**60333.00**	60575.00	2.30	60435.20	60366.00	60542.00	2.30	**60408.60**	60366.00	**60490.00**	2.30
gil262_1	262	13	113	1062.40	1060.00	1065.00	2.63	1058.00	**1057.00**	**1059.00**	2.63	**1057.40**	**1057.00**	**1059.00**	2.63
gil262_2	262	13	148	**1432.20**	1429.00	**1435.00**	2.63	1433.00	**1427.00**	1439.00	2.63	1435.00	1429.00	1449.00	2.63
gil262_3	262	13	145	**1320.40**	**1314.00**	1328.00	2.64	1322.20	1316.00	**1327.00**	2.63	1323.20	1318.00	**1327.00**	2.64
lin318_1	318	15	184	23407.80	23292.00	23514.00	3.19	**22990.80**	**22945.00**	**23032.00**	3.19	23044.20	22961.00	23126.00	3.19
lin318_2	318	17	189	24455.60	24360.00	24601.00	3.19	**24269.40**	**24175.00**	**24310.00**	3.19	24338.80	24286.00	24417.00	3.19
lin318_3	318	16	152	21213.00	21097.00	21316.00	3.18	**21109.60**	**21063.00**	**21161.00**	3.18	21150.60	21105.00	21193.00	3.18
rd400_1	400	20	227	9049.20	9016.00	9107.00	4.04	9073.60	9011.00	9203.00	4.05	**9039.00**	**8984.00**	**9090.00**	4.04
rd400_2	400	19	183	7347.20	7288.00	7415.00	4.02	**7287.20**	**7199.00**	7344.00	4.03	7297.00	7252.00	**7331.00**	4.02
rd400_3	400	20	186	7751.40	7711.00	7832.00	4.02	7791.20	7763.00	7815.00	4.02	**7746.60**	**7686.00**	**7808.00**	4.03
gr431_1	431	21	231	95523.40	95290.00	95723.00	4.35	**95261.40**	94916.00	**95651.00**	4.35	95333.00	**94480.00**	95899.00	4.34
gr431_2	431	22	257	**95116.60**	**94512.00**	**95498.00**	4.36	95163.40	94625.00	95620.00	4.37	95326.00	95001.00	95715.00	4.36
gr431_3	431	24	222	89002.60	88846.00	89268.00	4.34	**88548.40**	88361.00	**88675.00**	4.35	88557.80	**88333.00**	89007.00	4.35
pr439_1	439	22	261	67166.00	66794.00	67657.00	4.46	**66132.60**	66054.00	**66243.00**	4.44	66422.60	**66031.00**	66733.00	4.45
pr439_2	439	21	262	65717.80	65243.00	66301.00	4.46	**65305.20**	**65039.00**	**65538.00**	4.45	65661.60	65142.00	66060.00	4.45
pr439_3	439	22	215	59550.20	59162.00	59853.00	4.42	**57884.20**	**57306.00**	**58406.00**	4.43	58369.80	57690.00	58640.00	4.44

continued . . .

Instance				**HH(Random) - FTSP**				**HH(Greedy) - FTSP**				**HH(Rand2) - FTSP**			
Name	Nodes	Families	Visits	Average	Best	Worst	Time	Average	Best	Worst	Time	Average	Best	Worst	Time
att532_1	532	27	301	14273.80	**14196.00**	14310.00	5.41	14258.80	14206.00	**14301.00**	5.41	**14254.60**	14212.00	14305.00	5.40
att532_2	532	26	244	13972.00	13904.00	13995.00	5.36	**13838.80**	**13761.00**	13949.00	5.36	13901.20	13833.00	**13943.00**	5.36
att532_3	532	27	293	14394.60	14325.00	14500.00	5.37	14331.80	**14198.00**	14485.00	5.41	**14324.80**	14294.00	**14371.00**	5.42
ali535_1	535	25	272	92031.80	91707.00	92572.00	5.44	**91557.00**	**91255.00**	**92057.00**	5.44	91800.60	91259.00	92688.00	5.41
ali535_2	535	28	291	117342.00	115499.00	119599.00	5.49	**114063.40**	**113583.00**	**114581.00**	5.45	114520.20	113997.00	115122.00	5.49
ali535_3	535	26	260	94711.20	93252.00	96554.00	5.44	**90926.80**	**90494.00**	**91207.00**	5.42	91126.80	90994.00	91329.00	5.41
pa561_1	561	28	298	**1438.40**	**1420.00**	1450.00	5.71	1445.40	1432.00	1458.00	5.69	1439.00	1429.00	**1449.00**	5.69
pa561_2	561	29	254	1253.00	1243.00	1261.00	5.67	1244.80	1237.00	1254.00	5.67	**1241.40**	**1232.00**	**1252.00**	5.66
pa561_3	561	29	283	1413.20	**1397.00**	1431.00	5.69	**1402.60**	1399.00	**1406.00**	5.67	1406.80	1401.00	1412.00	5.71
d657_1	657	34	338	25647.00	25444.00	25801.00	6.73	25528.40	25245.00	25821.00	6.72	**25444.40**	**25230.00**	**25762.00**	6.70
d657_2	657	33	359	28607.60	28517.00	28701.00	6.73	**28450.00**	**28363.00**	**28635.00**	6.81	28547.60	28462.00	28642.00	6.80
d657_3	657	31	345	26918.60	26778.00	27075.00	6.72	**26881.20**	26725.00	**26995.00**	6.72	26935.00	**26641.00**	27111.00	6.73
nrw1379_1	1379	70	773	**32798.00**	32564.00	**32941.00**	26.02	32872.80	32644.00	33001.00	22.48	32830.00	**32563.00**	33046.00	21.92
nrw1379_2	1379	69	737	**31202.20**	31111.00	**31286.00**	23.86	31505.00	31131.00	31637.00	22.41	31242.40	**31077.00**	31413.00	22.94
nrw1379_3	1379	69	747	**31746.20**	31651.00	**31961.00**	25.77	31804.80	**31353.00**	32079.00	22.95	31750.60	31496.00	32004.00	22.59
No. of Best				16	28	13		**36**	**38**	**36**		24	28	29	

Table 4.4: Average distribution (in terms of percentage) of three LNS methods in our approaches on 60 newly generated instances

Name of approach	LNS_1	LNS_2	LNS_3
HH(Random)	33.30	33.37	33.33
HH(Greedy)	19.51	42.11	38.39
HH(Rand2)	18.13	42.03	39.83

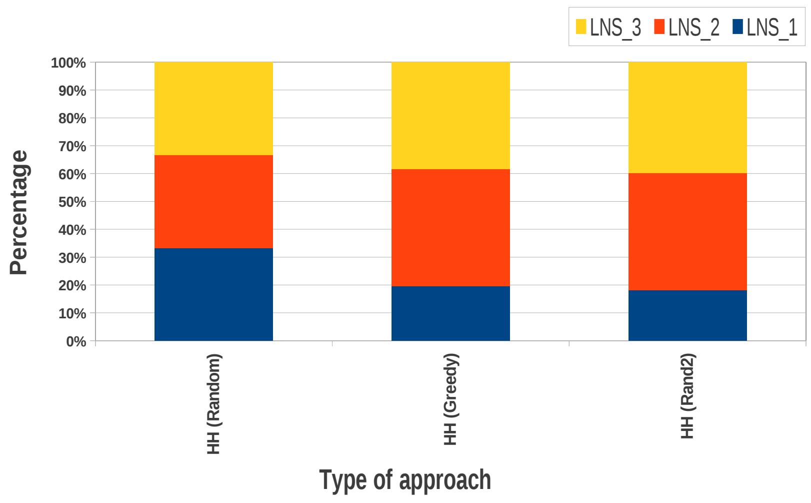


Figure 4.10: Graphical representation of average distribution values presented in Table 4.4

4.4.4 Statistical significance of our approaches

To check the significance of the results provided by our approaches in comparison to previous approaches, Wilcoxon signed rank test[94] has been performed. The calculator available online[1] is used to conduct the two-tailed Wilcoxon signed rank test with significance criteria set to 5% (i.e. p-value $\leq$ 0.05). Table 4.5 verifies the significance of our approaches in comparison to previous approaches on the 21 benchmark instances. In Table 4.5, the difference between the normalized values of *Average* obtained by HH(Random) / HH(Greedy) / HH(Rand2) and the compared approach is ranked. The column $NWT/Toal$ provides the number of instances without tie out of the total number of instances compared. The R^+ indicates the sum of the ranks for the instances where HH(Random) / HH(Greedy) / HH(Rand2) performed better than its competitor, whereas the R^- indicates the sum of the ranks for the instances where HH(Random)

[1]`https://mathcracker.com/wilcoxon-signed-ranks.php`

Table 4.5: Results of Wilcoxon signed rank test between our approaches and previous approaches

	HH(Random) vs...							**HH(Greedy) vs...**							**HH(Rand2) vs...**					
Previous approaches	$NWT/Total$	R^+	R^-	W	$W_{Cri,N}$	Significant		$NWT/Total$	R^+	R^-	W	$W_{Cri,N}$	Significant		$NWT/Total$	R^+	R^-	W	$W_{Cri,N}$	Significant
BRKGA - FTSP	13/21	91	0	0	12	yes		13/21	91	0	0	12	yes		13/21	91	0	0	12	yes
GRASP + evPR - FTSP	14/21	105	0	0	15	yes		14/21	105	0	0	15	yes		14/21	105	0	0	15	yes
ILS - FTSP	16/21	136	0	0	23	yes		16/21	136	0	0	23	yes		16/21	136	0	0	23	yes

/ HH(Greedy) / HH(Rand2) performed worse than its competitor. As the number of instances without tie are less than thirty ($NWT < 30$), the test statistic W is used to compare with the critical value $W_{Cri,N}$ as per the Wilcoxon signed rank test [94]. If $W \leq W_{Cri,N}$, then there is a significant difference between the performance of the two compared approaches, otherwise the difference is not significant. Table 4.5 clearly shows that the results of our approaches are significant with respect to three previous approaches, viz. BRKGA-FTSP, GRASP-FTSP and ILS-FTSP with $W \leq W_{Cri,N}$ and $R^+ > R^-$.

Table 4.6: Results of Wilcoxon signed rank test between our approaches

	HH(Greedy) vs...							**HH(Rand2) vs...**					
	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant		$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant
HH(Random)	67/81	1699	579	-3.498	-1.960	yes		64/81	1642	438	-4.026	-1.960	yes
HH(Rand2)	64/81	1257	823	-1.451	-1.960	no							

Table 4.6 identifies the significant one among our approaches on both the benchmark instances and newly generated instances. In this case, the number of instances without tie are greater than thirty ($NWT > 30$). Therefore, the test statistic Z is used to compare with the critical value Z_{Cri} as per the Wilcoxon signed rank test [94]. If $Z \leq Z_{Cri}$, then there is a significant difference between the performance of the two compared approaches, otherwise the difference is not significant. Table 4.6 clearly shows that HH(Greedy) is significant with respect to HH(Random) and there is no significant difference between the performances of HH(Greedy) and HH(Rand2). On the other hand, comparison between HH(Random) and HH(Rand2) shows that HH(Rand2) is significant with respect to HH(Random).

4.5 Conclusions

In this chapter, we have addressed the family traveling salesman problem (FTSP) with a hyper-heuristic approach. Three versions (HH(Random), HH(Greedy) and HH(Rand2)) of hyper-heuristic based on different selection mechanisms have been presented. These three versions have been evaluated on 21 benchmark instances of [112], and the obtained results have been compared with the state-of-the-art approaches ([112] and [113]). This comparison clearly shows the superiority of our approaches over all the other approaches. On most of the instances, HH(Random), HH(Greedy) and HH(Rand2) are able to get much better results. Our approaches are also several times faster than all the other approaches on most of the instances with more than 100 nodes. Further, Wilcoxon signed rank test shows the significance of the results obtained by our approaches with respect to all the other approaches.

We have generated 60 new benchmark instances for the FTSP based on the instances available in standard TSPLIB and reported the results of our approaches on them to facilitate future researchers to evaluate the performance of their approaches on a larger benchmark set comprising 81 instances in total. As far as the relative performance among our approaches is concerned, HH(Greedy) obtained better values on higher number of instances for both sets of benchmark instances viz. set of 21 instances from literature (i.e., 14 out of 21) and the set of 60 newly generated instances (i.e., 36 out of 60). Wilcoxon signed rank test on our approaches on both the benchmark instances and newly generated instances shows the significance of the results obtained by HH(Greedy) and HH(Rand2) with respect to HH(Random). However, no significant difference is found between the performances of HH(Greedy) and HH(Rand2).

Chapter 5

Covering Salesman Problem

5.1 Introduction

In the TSP, every city needs to be visited exactly once. However, it may not be possible in some practical scenarios particularly when there are limitations on resources like fuel, manpower etc. Keeping in mind these scenarios, Current and Schilling [114] formulated the covering salesman problem (CSP) that can be considered as a variant of the TSP. Unlike TSP, all the cities need not be visited in CSP. However, every city has to be either visited or has to be within a pre-specified distance from at least one visited city in the tour. If a city is within a pre-specified distance from a visited city in the tour then it is is considered to be covered. Hence, in CSP, every city has to be either visited or covered. Given a set of n cities with associated coverage radius c_i for each city $i \in \{1, 2, \ldots, n\}$, the CSP seeks a Hamiltonian cycle over a subset of cities such that remaining cities which are not part of this subset are covered by at least one city in this subset and that has minimum length among all Hamiltonian cycles over such subsets. TSP can be regarded as a special case of CSP where every city has the coverage radius of 0. Hence, CSP is also $\mathcal{NP}$-hard. As referred by Current and Schilling [114], the application of CSP arises in rural healthcare delivery, where all the zones can not be visited. In this situation, the goal is to visit a smaller number of zones, and the people residing in the unvisited zones need to travel to the nearest visited zone to avail the healthcare service. Solving this problem involves aspects of subset selection (selecting a subset of cities that covers the remaining cities) and permutation (finding the Hamiltonian cycle of minimum length over the selected subset of cities).

While introducing the CSP, Current and Schilling [114] proposed a heuristic method to solve it. They divided the procedure of solving CSP into two parts. In the first part, the associated set covering problem (SCP) is solved to determine a minimum number of cities that can cover all the cities. Then in the latter part, an optimal tour is determined by applying the TSP solver on the subset of cities obtained as a result of solving the SCP. There may exist more than one optimal solutions to the SCP. In such a situation, the TSP solver is applied on all the optimal solutions of the SCP and the minimum length tour among all the resulting tours is returned as a solution of the CSP.

Two local search algorithms, viz. $LS1$ and $LS2$ were proposed by Golden *et al.* [115] to solve the CSP by making use of different operations like removal-reinsertion, swap, and perturbation to guide the search process towards optimal solutions. The first local search algorithm $LS1$ uses removal-reinsertion operation in which some cities are removed from the tour, and later a subset of unvisited cities are reinserted into the tour to get a feasible solution with improved quality. The second local search algorithm $LS2$ uses removal-reassignment and perturbation procedures to enhance a solution. It uses the removal-reassignment heuristic to find a subset of cities, and then it applies the *Lin-Kernighan heuristic* [116] on this subset to find a tour on its constituent cities.

Salari *et al.* [117] developed a heuristic based on integer linear programming (ILP) for solving the CSP. The proposed heuristic is a combination of both ILP and heuristic search, and improves the quality of a given initial feasible solution. It uses a destroy and repair operator which makes the given solution infeasible by removing some cities from the tour and then inserts some cities back to the tour to again make it a feasible solution. This feasible solution is improved further by optimally solving an ILP formulation.

Salari *et al.* [118] developed an ant colony optimization (ACO) algorithm by hybridizing it with a dynamic programming (DP) technique and 3-opt move. Comparison of this hybrid ACO algorithm with approaches available in the literature showed this approach to be superior. In addition, this paper also provides an integer linear formulation for CSP, and solves it through CPLEX for small sized instances.

A variation of the CSP based on geometry was proposed by Arkin and Hassin [119], in which each neighborhood is a set of cities and if a city from that set is intersected, then that neighborhood is considered as covered. The goal is to find a tour of minimum length which traverses all the neighborhoods by intersecting with them. Several approximation algorithms exist in the literature to solve this version [119, 120, 121].

There exist several variants of CSP in the literature. The covering tour problem (CTP) was formulated by Gendreau *et al.* [122], in which the set of cities can be partitioned into three different subsets G1, G2 & G3. G1 contains cities that must be visited, G2 contains cities that have to be either visited or covered and G3 contains mandatory cities to be covered. The goal of the CTP is to find a tour of minimum length that visits all the cities in G1 and a subset of cities in G2 to cover all the cities in G3 and all the unvisited cities in G2. A heuristic approach and a branch-and-cut based exact approach were presented in [122] for solving the CTP.

Hachicha *et al.* [123] introduced a variant of CTP with multiple vehicles called multi-vehicle covering tour problem (MVCTP). In addition to the constraints of CTP, multiple vehicles are there in MVCTP. The objective of MVCTP is to find the tours for multiple vehicles while minimizing the total cost of these tours. In order to solve the MVCTP, the authors proposed three heuristics which are based on savings, sweep and route-first/cluster-second algorithms respectively. The savings and sweep algorithms are actually developed for the vehicle routing problem (VRP) and they are suitably modified according to the MVCTP. Another two variants of CTP are generalized covering tour problem [124] and maximum covering tour problem [125].

Another closely related variant of the CSP is the ring star problem (RSP), in which a city has to be either visited or assigned to a visited city in the ring. Visiting a city incurs a routing cost and assigning a city incurs an assignment cost. The objective of RSP is to minimize the sum of both routing and assignment costs [126]. If there are multiple rings and each ring has a restriction on the number of visited or assigned cities, then that variant of RSP is called as capacitated m-ring star problem (CMRSP) [127, 128]. A problem quite similar to RSP is known as the median cycle problem (MCP) [129]. In addition to the constraints of RSP, MCP imposes an upper bound on the maximum assignment cost.

Generalized traveling salesman problem (GTSP) also can be considered as a special case of the CSP in which a set of cities is given as the multiple disjoint clusters and the goal of GTSP is to find a tour of minimum length that visits exactly one city from each cluster. The GTSP can be seen as a CSP with a constraint that every city covers every other city in the cluster to which it belongs [111].

In this chapter, we have proposed two hybrid metaheuristic approaches for solving the CSP. Our first approach is based on the artificial bee colony algorithm (ABC), whereas the latter one is based on the genetic algorithm (GA). These two metaheuristics have already been applied to solve innumerable discrete optimization problems. Success of these two metaheuristics in solving discrete optimization problems and absence of any approach based on these metaheuristics for CSP have motivated us to develop two hybrid approaches for CSP utilizing these metaheuristics. Both these approaches make use of problem specific knowledge in an appropriate manner in their operators. These two approaches utilize several first improvement or best improvement based local search strategies defined over various neighborhood structures. In ABC approach, these strategies are used in a sequential manner to generate a neighboring solution. On the other hand, in GA, some of these strategies are used to enhance the solution produced by crossover/mutation. Computational results on benchmark instances available in the literature show the effectiveness of our approaches.

The remaining part of this chapter is organized as follows. Section 5.2 provides the formal definition of the CSP. The first approach based on ABC for the CSP is described in Section 5.3, whereas Section 5.4 presents the second approach based on GA. Section 5.5 is devoted to presentation of computational results and their analysis. Finally, some concluding remarks are provided in Section 5.6.

5.2 Problem definition

The covering salesman problem can be defined formally in the following manner: Given an edge-weighted undirected complete graph $G = (V, E)$ which has the set of cities $V = \{1, 2, \ldots, n\}$, and the set of edges $E = \{(i, j)|i, j \in V\}$. Each edge $(i, j) \in E$ has an associated distance $d_{ij} \in \mathbb{R}^+$. A city is considered as covered by city i, if it is within the coverage radius c_i of the city i. It is to be noted that a city covers itself also as per this definition. The set of cities covered by city i is denoted by A_i, and the set of cities which can cover the city i is denoted by B_i. Please note that A_i and B_i both contain city i. The CSP seeks a Hamiltonian cycle over a subset of cities such that the cities which are not part of this subset are covered by at least one city in this subset and that has minimum length among all Hamiltonian cycles over such subsets. By introducing binary variables x_{ij} to specify whether the edge (i, j) belongs to the Hamiltonian cycle ($x_{ij} = 1$) or not ($x_{ij} = 0$), a mathematical model for the CSP can be formulated as follows:

$$\text{Minimize} \quad \sum_{i\in V}\sum_{j\in V} d_{ij}x_{ij} \tag{5.1}$$

subject to:

$$\sum_{j\in B_i} \sum_{\substack{k\in V \\ (k,j)\in E}} x_{kj} \geq 1, \quad \forall i \in V, \tag{5.2}$$

$$\sum_{(h,i)\in E} x_{hi} = \sum_{(i,j)\in E} x_{ij}, \quad \forall i \in V, \tag{5.3}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V, \tag{5.4}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E. \tag{5.5}$$

The objective function of the CSP is represented by Equation (5.1) which minimizes the total tour length. Equation (5.2) ensures that each city either belongs to the tour or is covered by at least one city belonging to the tour. Equation (5.3) enforces the indegree and outdegree constraints on every city. Equation (5.4) is a sub tour elimination constraint. Equation (5.5) restricts the values of decision variables x_{ij} to either 0 or 1.

Figure 5.1 provides an illustration of the CSP. The instance of 51 cities with their coverage radii can be seen in Figure 5.1 (a), where the cities are shown as blue points and their associated radii are shown by red circles. Figure 5.1 (b) shows a feasible solution for this instance, where 9 cities are visited by the salesman and the remaining 42 cities are covered by at least one of these 9 visited cities.

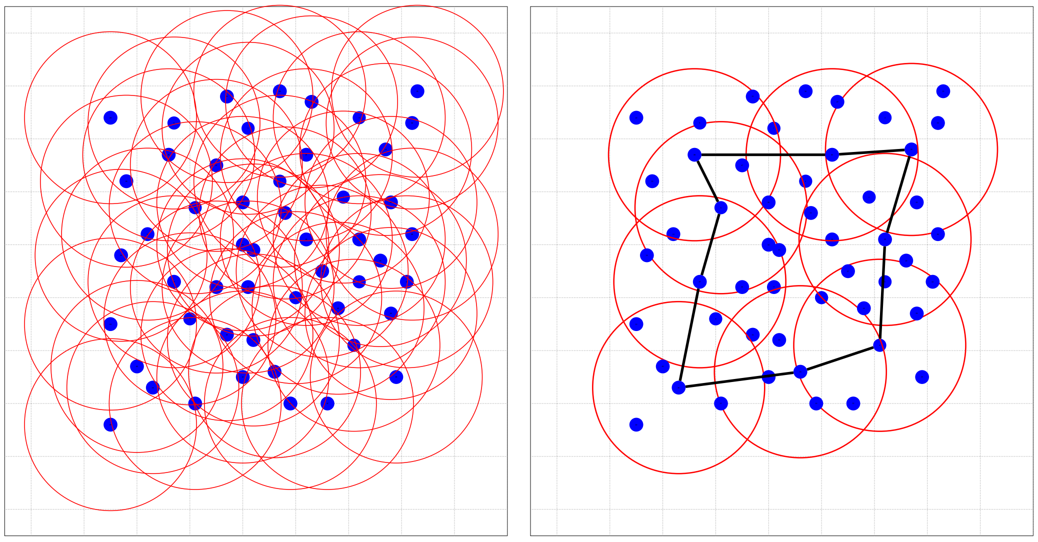

(a) An instance of 51 cities with their coverage radii (b) A feasible solution for the CSP

Figure 5.1: Illustration of CSP with an example

5.3 Artificial bee colony algorithm for the CSP

This section describes the proposed artificial bee colony algorithm based approach for the CSP. Subsequent subsections provide the details about different components of this approach.

5.3.1 Solution encoding of CSP

Within the ABC algorithm, a feasible solution of CSP is encoded as a linear permutation of visited cities. This encoding is also used in [118]. Suppose the given number of cities, $|V| = n$, and number of visited cities, $|V'| = m$, the search space size of our encoding to CSP is $\sum\limits_{m=1}^{n} \binom{n}{m} \times m!$. A feasible solution of a CSP can have m cities ranging from 1 to n. $\binom{n}{m}$ represents the number of ways a subset of m cities can be chosen from given set of n cities. $m!$ represents the number of ways the chosen m cities can be arranged.

5.3.2 Fitness

The fitness function is same as the objective function given (Equation (5.1)), i.e., the total distance traveled by the salesman for covering all the cities. As CSP is a minimization problem, a solution having a smaller value of the fitness function is regarded as better in comparison to a solution having a larger value

5.3.3 Generation of initial solutions

Each initial solution is generated via an iterative procedure which begins by adding all the cities to the list of uncovered cities and then an iterative process follows. During each iteration, a city is chosen randomly from the list of uncovered cities, and inserted at a best possible position in the partially constructed tour. A best possible position is the one where inserting the city yields the smallest increase in the cost. All those cities which are now covered by this newly added city are removed from the list of uncovered cities and another iteration begins. This process is repeated as long as the list of uncovered cities is not empty.

5.3.4 Mechanism for choosing a food source

Binary tournament selection method is used to choose an employed bee solution for an onlooker bee. In this method, two employed bee solutions are selected uniformly at random from the population of employed bee solutions and a comparison is made between their fitnesses. The better one of the two solutions is chosen with probability P_{bts}. Otherwise, the worse one is chosen, i.e., the probability of selection of worse of the two solutions is $(1 - P_{bts})$. The motivation behind choosing binary tournament selection method instead of the commonly used roulette wheel selection method is that the fitness values are large and nearer to each other for some instances. Actually, when a population contains only individuals with large, nearly equal fitness, the selection probability of all individuals becomes nearly identical and roulette wheel selection method fails to induce any selection pressure, thereby impairing the performance of the ABC algorithm. Algorithm 14 provides the pseudo-code for the binary tournament selection.

Algorithm 14: Pseudo-code of the binary tournament selection

```
Input: Population of solutions
Output: Selected solution

Select two solutions $S_1$ and $S_2$ randomly from the population;
Generate a random number $r$ such that $0 \le r \le 1$;
if $r < P_{bts}$ then
    return the best solution between $S_1$ and $S_2$;
else
    return the worst solution between $S_1$ and $S_2$;
```

5.3.5 Neighboring solution generation

Since the problem of CSP involves the aspects of both subset selection and permutation, the neighboring solution generation process must give due considerations to both these aspects so that the search process can be directed towards an optimal solution. Keeping this in mind, we have developed four methods for generating a neighboring solution. These four methods are used one after the other in the order in which they are described below.

1. ***Subset neighbor (SSNB):*** This method tries to find out a proper subset of cities for the CSP. Initially, it will make the given solution infeasible by removing some cities from the tour. Later, it will add some cities to the tour to make the solution feasible. Each city from the tour is removed with a probability Deg_{pert}, which makes the solution infeasible. To make this solution feasible again, an iterative procedure is followed to add the cities to the tour. In each iteration, a city is chosen uniformly at random from the unvisited cities and added to the tour in a position that yields the least insertion cost.

   The feasible solution resulting from this procedure may have redundant cities. A visited city is redundant if all the cities covered by it are also covered by other visited cities.

Removal of a redundant city may decrease the tour length without affecting the feasibility of the CSP solution. These redundant cities must be removed in order to get an improved solution. In case there exists more than one redundant cities, then a redundant city that yields the maximum reduction in the cost of the solution is removed and the redundant cities are recomputed. This redundant city removal procedure is continued until no redundant city remains in the solution. Both the input and output solutions of this method are compared and the best of these two solutions is passed to the next method.

2. ***Local search with two neighborhood structures (LS2N):*** The proposed LS2N uses two exchange based neighborhood structures N_1 and N_2. These neighborhood structures are described below:

   (a) The first neighborhood N_1 considers a city in the tour and tries to exchange it with a pair of cities so that there is an improvement in the cost of the solution without violating feasibility. Actually, every city i may have its own set of pairs of cities which can cover all the cities covered by city i. Recalling that A_i is the set of cities covered by city i (Section 5.2), the city $i \in V'$ can be exchanged with a pair of cities $j, k \in V$ satisfying the constraints $A_i \subseteq A_j \cup A_k$, $A_i \nsubseteq A_j$ and $A_i \nsubseteq A_k$. For efficiency considerations, this set of pairs is pre-computed for each city. The neighborhood N_1 considers the cities one-by-one from the tour in the order in which they appear in the tour, and tries to exchange them with one of the pairs in their respective set of pairs. The neighborhood N_1 follows the first improvement strategy.

   In order to check the possibility of exchange of city i with the pair of cities j and k, first the city i is removed from the tour then j and k are inserted to the tour in their respective best possible positions. If one of the cities (j/k) is already visited, then only the other unvisited city is inserted. If both cities of the pair are already visited then there is no need to insert any city, and this is surely an improving move. If there is no improvement in the solution, j and k are removed and i is reinserted at its original position and another pair is considered for checking the possibility of exchange. The pairs are tried for exchange in some random order. If the city i encounters a pair of cities that can improve the solution, then the solution is updated immediately with that pair of cities and the search process moves to the next city in the tour (if we are at the last city then move to the first city). This process continues until no beneficial exchange is possible for any city in the tour.

At this juncture redundant city removal procedure as described for SSNB is applied to remove redundant cities. Then the second neighborhood is explored in case either exploring first neighborhood yields an improved solution or is explored for the very first time. Otherwise, LS2N stops. It is to be noted that we are computing the pairs for each city without any regard to the current composition of the tour. This allows us to pre-compute all such pairs once and for all.

(b) The second neighborhood N_2 considers a city in the tour and tries to exchange it with an unvisited city that it covers. Like N_1, here also the cities are considered one-by-one according to their order in the tour. Unlike N_1, the neighborhood N_2 follows a best improvement strategy as there are limited number of cities that a city can cover. A city $i \in V'$ can be exchanged with an unvisited city $j \in A_i$ if such an exchange leads to a feasible solution with reduced cost. Please note that feasibility depends not only on choice of city j, but also on current composition of the tour. Like N_1, here also city i is removed first and city j is inserted to the tour in its best position to check the possibility of exchange. A city i is examined for exchange with all unvisited cities $j \in A_i$, and a best city is selected for exchange. Here, a best city is the one that decreases the cost of the solution by maximum amount. Once the solution is updated, the search process moves to the next city in the tour (if we are at the last city then move to the first city). This process continues until there is no exchange possible for every city in the tour. Now, the redundant city removal procedure as described for SSNB is applied to remove redundant cities. Now, again the first neighborhood is explored in case exploring the second neighborhood yields an improved solution. Otherwise, LS2N stops.

Hence, LS2N consists of exploring these two neighborhoods one after the other in an alternating manner until there is no improved solution in either of the neighborhoods. In essence, a solution obtained by LS2N is a locally optimal solution with respect to both the neighborhoods N_1 and N_2.

3. ***General exchange (GE):*** This method is based on general one-one exchange and tries to exchange a visited city with an unvisited city. If the exchange of city $i \in V'$ with the city $j \in V - V'$ yields an improved feasible solution, then i is exchanged with j. Again the feasibility of the resulting solution depends not only on choice of j, but also on the current composition of the tour. A city i is examined for exchange with all unvisited cities

j, and the best city is selected for exchange. Cities in the tour are considered for exchange one-by-one in the order in which they occur in the tour. After considering the last city in the tour, again the first city in the tour is considered for exchange. This process continues until no city in the tour can be exchanged with a city not in the tour to reduce the objective function value. At this juncture, the redundant city removal procedure is applied.

4. ***Permutation neighbor (PTNB):*** This method tries to find out a proper permutation of cities for the CSP. In this method, each city from the tour is removed with a probability Deg_{pert}, and later these removed cities are only added to the tour in their best positions. This method is similar to *SSNB* method described previously except for the fact that only cities that are removed from the tour are considered for insertion. Compared to *SSNB* method, in *PTNB* method, there is no need to remove the redundant cities as there is no change in the constituent cities of the tour.

Another crucial factor for the success of an ABC approach lies in correctly estimating, how strong the perturbations need to be while generating a neighboring solution. In the initial iterations, higher degree of perturbation helps in improving the randomly generated initial solutions quality quickly. However, during final iterations, higher degree of perturbation may fail to improve a solution as during final iterations, a solution is very near to an optimal/locally optimal solution and perturbing it too much may move this solution away from the optimal/locally optimal solution. Keeping this in mind, a variable degree of perturbation strategy is used within our ABC approach. The main idea of this strategy is that the degree of perturbation needs to be high in the initial iterations, and it has to be reduced as the algorithm progresses in order to get good solutions. The parameter Deg_{pert} controls the degree of perturbation, and it varies from max_{deg} to min_{deg} over max_{it} initial iterations. The value of Deg_{pert} in such an iteration it is calculated as follows:

$$Deg_{pert} := \left(\frac{max_{deg} - min_{deg}}{max_{it}}\right)(max_{it} - it) + min_{deg} \tag{5.6}$$

Beyond max_{it} iterations, the parameter Deg_{pert} takes the fixed value of min_{deg} as too small value for Deg_{pert} fails to produce a neighboring solution different from the original solution. The pseudo-code for neighboring solution generation utilizing the four methods described above is given in Algorithm 15, where N_q represents the q^{th} neigborhood.

Algorithm 15: Neighboring solution generation utilizing four proposed methods

```
Input: A solution $S$
Output: A neighboring solution $S'$

begin
    $S' \leftarrow \emptyset$;                                                   }
    foreach visited city c in $S$ do                                            }
        Generate a random number $0 \leq r \leq 1$;                             }
        if $r < Deg_{pert}$ then                                                }
            do not copy $c$ to $S'$;                                            }
        else                                                                    } SSNB method.
            copy $c$ to $S'$;                                                   }
    while $S'$ is infeasible do                                                 }
        Add an unvisited city according to $SSNB$ method;                       }
    $S \leftarrow S'$;
    $first \leftarrow 1$;                                                       }
    $flag \leftarrow 1$;                                                        }
    $q \leftarrow 1$;                                                           }
    repeat                                                                      }
        $flag \leftarrow 0$;                                                    }
        $S' \leftarrow$ Neighborhood_Search($S$, $N_q$);                        }
        if $F(S') < F(S)$ or $first = 1$ then                                   } LS2N method.
            if $F(S') < F(S)$ then                                              }
                $S \leftarrow S'$;                                              }
            $q \leftarrow 3 - q$;                                               }
            $flag \leftarrow 1$;                                                }
            $first \leftarrow 0$;                                               }
    until $flag = 0$;                                                           }
    $S' \leftarrow$ GE($S$);                                                    }
    if $F(S') < F(S)$ then                                                      } GE method.
        $S \leftarrow S'$;                                                      }
    $S' \leftarrow \emptyset$;                                                  }
    $U \leftarrow \emptyset$;                                                   }
    foreach visited city c in $S$ as per their order do                         }
        Generate a random number $0 \leq r \leq 1$;                             }
        if $r < Deg_{pert}$ then                                                }
            Add $c$ to $U$;                                                     } PTNB method.
        else                                                                    }
            Copy $c$ into $S'$;                                                 }
    foreach city c in $U$ in some random order do                               }
        Insert $c$ into $S'$ using the $PTNB$ method;                           }
    return $S'$;                                                                }
```

5.3.6 Other features

We have used different number of employed bees and onlooker bees which is different from the practice of using the same number of employed and onlooker bees as suggested with original ABC algorithm. Actually, the aim of onlooker bee phase is to provide more chances to the better solutions to improve themselves. It is based on the simple fact that chances of finding even better solutions in the vicinity of good solutions is more in comparison to poor solutions. Now, how many more chances good solutions need in comparison to poor ones depend on the fitness landscape. Usually, for a permutation problem like CSP, there is a huge difference in fitness between the good solutions and bad solutions and neighboring solution generation procedure more often than not can not bridge this gap. Hence, for such problems good solutions need to be given much more chances than what they usually get by taking equal number of employed and onlooker bees. This reason has motivated us to use more number of onlooker bees than employed bees.

If an employed bee solution does not improve over a fixed number of trials $limit_{scout}$, then the corresponding employed bee abandons that solution and becomes a scout. By number of trials for a solution, we mean the number of times that solution is used for neighboring solution generation. A solution can get more than one trial in an iteration depending on the number of onlookers selecting that solution. Therefore, the number of scout bees in an iteration is equal to the number of employed bee solutions that have not improved over last $limit_{scout}$ trials. This number can be zero or one or more than one. Since we are reducing the degree of perturbation over iterations, assigning a random solution to a scout bee will not be a good move as degree of perturbation at the time of replacement will not be suitable for quick improvement of this randomly generated solution. So, we assign a neighboring solution of just abandoned solution, even if it is worse than the just abandoned solution, to the scout bee to make it employed again.

The pseudo-code of our ABC approach for the CSP is provided in Algorithm 16, where NEB and NOB represent the number of employed bees and the number of onlooker bees respectively. *Initialization_Of_Solutions (*NEB*)* is a function which produces NEB number of initial solutions according to the process described in Section 5.3.3. *New_Solution(*S*)* is a function which generates a new solution in the neighborhood of the solution S as per the process described in Section 5.3.5. *Select_Solution (*$S_1, S_2, \ldots, S_{NEB}$*)* is a function that selects a solution to be associated with an onlooker bee from employed bee solutions $S_1, S_2, \ldots, S_{NEB}$ as per the process described in Section 5.3.4.

5.4 Genetic algorithm for the CSP

We have developed a steady-state genetic algorithm [130] for the CSP where part of the population is reinitialized whenever the population suffers from the lack of diversity. Hereafter,

Algorithm 16: ABC approach for the CSP

```
Input: Values for different parameters of the ABC Algorithm and a CSP instance
Output: Best solution found

for $i \leftarrow 1$ to $NEB$ do                                                  } Initialization phase
    $S_i \leftarrow$ Initialization_Of_Solutions($NEB$);
$S_{best} \leftarrow$ best solution among $S_1, S_2, \ldots, S_{NEB}$;
while termination criterion is not met do
    for $i \leftarrow 1$ to $NEB$ do                                              } Employed bee phase
        $S' \leftarrow$ New_Solution($S_i$);
        if $S'$ is better than $S_i$ then
            $S_i \leftarrow S'$;
    for $i \leftarrow 1$ to $NOB$ do                                              } Onlooker bee phase
        $S_p \leftarrow$ Select_Solution($S_1, S_2, \ldots, S_{NEB}$);
        $S' \leftarrow$ New_Solution($S_p$);
        if $S'$ is better than $S_p$ then
            $S_p \leftarrow S'$;
    for $i \leftarrow 1$ to $NEB$ do                                              } Memorizing best solution & Scout bee phase
        if $S_i$ is better than $S_{best}$ then
            $S_{best} \leftarrow S_i$;
        else if $S_i$ has not improved over last $limit_{scout}$ trials then
            $S_i \leftarrow$ New_Solution($S_i$);
return $S_{best}$;
```

this approach will be referred to as GA. Our GA utilizes problem specific knowledge in crossover and mutation. Following subsections describe the salient features of our GA.

5.4.1 Solution encoding

The solution encoding used by GA is similar to one used by ABC (Section 5.3.1) with one small difference, i.e., a permutation always begins with smallest visited city (note that cities are numbered from 1 to n). This allows us to represent a tour, which is a circular permutation, uniquely using a linear permutation. Otherwise, a tour comprising m visited cities can be represented by m different linear permutations each beginning with a different city among m visited cities leading to redundancy in solution representation. This is specially significant for our GA as population members interact with one another through crossover and our GA being steady-state enforces the constraint of uniqueness of each solution in the population. This also yields the reduced search space. Using the same notational conventions as used in Section 5.3.1, the search space size of this encoding is $\sum_{m=1}^{n} \binom{n}{m} \times (m-1)!$. A feasible solution of CSP can have m cities ranging from 1 to n. $\binom{n}{m}$ represents the number of ways a subset of m cities can be chosen from the given set of n cities. $(m-1)!$ represents the number of ways the $m-1$ cities excluding the smallest among these m cities, which has the fixed position, can be arranged.

If a solution generated through crossover/mutation/initial solution generation procedure does not comply with this encoding, then it will be transformed into an equivalent compliant solution before checking its uniqueness and inserting into the population.

5.4.2 Fitness

Like the ABC approach (Section 5.3.2), here also the objective function itself is used as the fitness function.

5.4.3 Initial solution generation

Each initial solution is generated in the same manner as described in Section 5.3.3. A newly generated solution is checked for uniqueness with respect to population members generated so far, and, in case it is found to be different from all existing members, it is included in the initial population, or else it is discarded. We have not applied redundant city removal procedure during initial population generation as doing so can reduce the diversity of the initial population.

5.4.4 Selection

Each parent for crossover/mutation is selected via binary tournament selection method (Algorithm 14), where the solution with better fitness is selected with probability $Prob_{bt}$.

5.4.5 Crossover

We have developed a crossover operator for CSP taking into consideration its special aspects. The crossover operator begins by copying the first parent to the child. Then the cities belonging to the second parent are inserted at their best possible positions in the child with probability $Prob_{cp}$ if they are not already present in the child. Half of the time these cities are inserted in the order in which they occur in the second parent and half of the time in random order. These insertions make some cities redundant in child solution. So redundant city removal procedure is applied on the child to remove redundant cities.

Our GA uses crossover and mutation in a mutually exclusive manner where crossover is used with probability $Prob_{cross}$, or else mutation is used.

5.4.6 Mutation

Our GA utilizes two mutation operators, and only one of them is used under the circumstance of not using crossover. The first mutation operator is used with probability $Prob_{fm}$, otherwise the second mutation operator is used. Both mutation operators try to copy each city in the tour to the child with probability $Prob_{cp}$. This may yield an infeasible child. To make this child again feasible, the first mutation operator makes use of only those cities which are present in the parent but could not be copied to the child. On the other hand, second mutation operator makes use of the cities other than those used by first mutation operator to the maximum extent possible. However, we may need to consider a city used by first mutation operator in case such a city is not covered by any city currently present in the child. The cities to be considered are added to a list L and then an iterative process ensues. During each iteration, a city i is chosen uniformly at random from this list and added to the child at the best possible position after deleting i from L. All those cities j in L which are covered by i are also deleted from L in case each city in A_j is either covered by a city already in child or belongs to L itself. After this another iteration begins. This process is repeated till L becomes empty. Now, child is subjected to redundant city removal procedure to improve its objective function value.

5.4.7 Local search

Each new solution produced by crossover/mutation is improved using LS2N (Section 5.3.5). If after this step, solution becomes better than the best solution found so far, then GE exchange (Section 5.3.5) is also applied in a bid to improve it further.

5.4.8 Population replacement and re-initialization

Our genetic algorithm being steady-state produces a single solution in each generation that replaces the worst member of the population if it is unique with respect to current population members. In case, this solution is found to be identical to any of the current member of the population then it is discarded.

Another unique feature of our GA is partial population re-initialization whenever there is a lack of diversity in the population. If we fail to find a solution different from current population members in Gen_R consecutive generations, then this indicates lack of diversity in the population. To alleviate this problem, we re-initialize part of the population. Let $bval$ be the objective function value of the best solution found so far. All solutions with objective function value equal to $bval$ are always retained. Remaining solutions are retained with probability $Prob_R$ only, otherwise they are replaced with solutions generated in the same manner as initial population. While doing re-initialization, uniqueness of each member of the population is always maintained. Once the population is re-initialized, it is forbidden to reinitialize it again for next 1000 generations.

Algorithm 17 provides the pseudo-code for GA where *Crossover(P_1, P_2)*, *Mutation_One(P)*, *Mutation_Two(P)*, *LS2N(P)* and *GE(P)* are five functions implementing our crossover operator (Section 5.4.5), first mutation operator (Section 5.4.6), second mutation operator (Section 5.4.6), LS2N local search (Section 5.4.7) and GE exhange local search (Section 5.4.7) respectively. Binary tournament selection is performed by the function *Binary_Tournament(S_1,S_2,..., S_{ps})*. This function selects a solution from solutions S_1,S_2,...,S_{ps} and returns the solution selected.

Algorithm 17: GA for the CSP

```
Input: Values for different parameters of the GA and a CSP instance
Output: Best solution found

Randomly generate $ps$ initial solutions $S_1, S_2, \ldots, S_{ps}$;
$S_{best} \leftarrow$ best solution among $S_1, S_2, \ldots, S_{ps}$;
while termination criterion is not met do
    if ($u01 < Prob_{cross}$) then
        $P_1 \leftarrow Binary\_Tournament(S_1, S_2, \ldots, S_{ps})$;
        repeat
            $P_2 \leftarrow Binary\_Tournament(S_1, S_2, \ldots, S_{ps})$;
        until ($P_1 \neq P_2$);
        $C \leftarrow Crossover(P_1, P_2)$;
    else
        $P \leftarrow Binary\_Tournament(S_1, S_2, \ldots, S_{ps})$;
        if ($u01 < Prob_{fm}$) then
            $C \leftarrow Mutation\_One(P)$;
        else
            $C \leftarrow Mutation\_Two(P)$;
    $C \leftarrow LS2N(C)$;
    if ($C$ is better than $S_{best}$) then
        $C \leftarrow GE(C)$;
        $S_{best} \leftarrow C$;
    Apply population replacement and population re-initialization policies;
return $S_{best}$;
```

5.5 Computational results

Both the approaches, viz. ABC and GA are tested on the benchmark instances of CSP [118]. These instances are in Euclidean distance matrix format (i.e., $n \times n$ matrix), and are derived from the TSPLIB . These instances have cities ranging from 51 to 783. These instances are divided into three categories based on the number of cities they contain, viz. small ($51 \leq n \leq 100$), medium ($150 \leq n \leq 200$) and large ($532 \leq n \leq 783$).

Both the approaches are implemented in C and the executions are carried out on a Linux based system which has 3.10 GHz Intel Core-i5-2400 processor and 4 GB RAM. It is to be noted that no multi-core programming feature is used while implementing our approaches. In all our experiments with the ABC, we have used a population of 100 employed bees ($NEB = 100$), 150 onlooker bees ($NOB = 150$), $limit_{scout}$ = 50, P_{bts} = 0.8 and the degree of perturbation Deg_{pert} is varied from 1 ($max_{deg} = 1$) to 0.1 ($min_{deg} = 0.1$) for 100 iterations ($max_{it} = 100$). After 100 iterations the degree of perturbation is constant at the minimum value of the degree of perturbation (i.e., min_{deg}). Our ABC approach terminates if there is no improvement in the solution quality consecutively for 100 iterations. In all our experiments with GA, we have used a population of 400 chromosomes, $Prob_{bt} = 0.8$, $Prob_{cross} = 0.66$, $Prob_{cp} = 0.8$, $Prob_{fm} = 0.25$, $Gen_R = 4$ and $Prob_R = 0.33$. Our genetic algorithm terminates when best solution has not improved over Max_Iter iterations and it has executed for at least I iterations. Max_Iter is set to 2000 for $n \leq 100$, otherwise it is set to 10000. I is set to Max_Iter for $n \leq 500$, otherwise it is set to 50000. All the parameter values for both ABC & GA are chosen empirically after large number of trials. For benchmarking, our approaches are compared with four state-of-the-art approaches available in the literature, viz. LS2 [115], SN [117], CPLEX and Hybrid ACO [118]. Basically, CPLEX refers to the approach of solving integer linear programming formulation provided in [118] through CPLEX. Results of CPLEX with maximum allowed time of one hour are reported for small instances only in [118]. These four approaches were executed on 1.66 GHz PC with Intel Core Duo processor. Like LS2, SN and Hybrid ACO, our approaches are also executed for five independent times on each of the instances.

Tables 5.1, 5.2 and 5.3 report the performance of our approaches, viz. ABC and GA along with aforementioned approaches on the three categories of instances, viz. small, medium and large respectively. In all these tables, data for the approaches CPLEX, LS2, SN and Hybrid ACO is taken from [118], where CPLEX is used to solve only small instances as mentioned

Table 5.1: Results of various approaches on small sized instances

Instance	**NC**	**CPLEX**				**LS2**				**SN**				**Hybrid ACO**				**ABC**				**GA**			
		Best	Rel-gap	Time	Gap	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT
eil51	7	164	0.00	149	0.00	**164**	**164.0**	**0.00**	1	**164**	**164.0**	**0.00**	3	**164**	**164.0**	**0.00**	2	**164**	**164.0**	**0.00**	1	**164**	**164.0**	**0.00**	1
eil51	9	159	0.00	220	0.00	**159**	**159.0**	**0.00**	1	**159**	**159.0**	**0.00**	2	**159**	**159.0**	**0.00**	2	**159**	**159.0**	**0.00**	1	**159**	**159.0**	**0.00**	1
eil51	11	147	0.00	681	0.00	**147**	**147.0**	**0.00**	1	**147**	**147.0**	**0.00**	2	**147**	**147.0**	**0.00**	3	**147**	**147.0**	**0.00**	1	**147**	**147.0**	**0.00**	1
berlin52	7	3887	0.00	140	0.00	**3887**	**3887.0**	**0.00**	1	**3887**	**3887.0**	**0.00**	2	**3887**	3887.4	0.01	2	**3887**	**3887.0**	**0.00**	1	**3887**	**3887.0**	**0.00**	1
berlin52	9	3430	0.00	212	0.00	**3430**	**3430.0**	**0.00**	1	**3430**	**3430.0**	**0.00**	2	**3430**	**3430.0**	**0.00**	2	**3430**	**3430.0**	**0.00**	1	**3430**	**3430.0**	**0.00**	1
berlin52	11	3262	0.00	255	0.00	**3262**	**3262.0**	**0.00**	1	**3262**	**3262.0**	**0.00**	2	**3262**	**3262.0**	**0.00**	2	**3262**	**3262.0**	**0.00**	1	**3262**	**3262.0**	**0.00**	1
st70	7	288	0.00	490	0.00	**288**	**288.0**	**0.00**	1	**288**	**288.0**	**0.00**	4	**288**	**288.0**	**0.00**	2	**288**	**288.0**	**0.00**	1	**288**	**288.0**	**0.00**	1
st70	9	259	0.00	1391	0.00	**259**	**259.0**	**0.00**	2	**259**	**259.0**	**0.00**	4	**259**	**259.0**	**0.00**	6	**259**	**259.0**	**0.00**	1	**259**	**259.0**	**0.00**	1
st70	11	250	14.51	3600	1.21	**247**	**247.0**	**0.00**	2	**247**	**247.0**	**0.00**	4	**247**	**247.0**	**0.00**	4	**247**	**247.0**	**0.00**	1	**247**	**247.0**	**0.00**	1
eil76	7	219	13.68	3600	5.80	**207**	**207.0**	**0.00**	1	**207**	**207.0**	**0.00**	4	**207**	**207.0**	**0.00**	2	**207**	**207.0**	**0.00**	1	**207**	**207.0**	**0.00**	1
eil76	9	198	22.71	3600	7.03	186	186.0	0.54	1	**185**	**185.0**	**0.00**	4	186	186.0	0.54	3	186	186.0	0.54	1	**185**	**185.0**	**0.00**	1
eil76	11	177	21.90	3600	4.12	**170**	**170.0**	**0.00**	1	**170**	**170.0**	**0.00**	4	**170**	**170.0**	**0.00**	2	**170**	**170.0**	**0.00**	1	**170**	**170.0**	**0.00**	1
pr76	7	50275	0.00	2488	0.00	**50275**	**50275.0**	**0.00**	2	**50275**	**50275.0**	**0.00**	4	**50275**	**50275.0**	**0.00**	10	**50275**	**50275.0**	**0.00**	1	**50275**	**50275.0**	**0.00**	1
pr76	9	45387	5.71	3600	0.09	**45348**	45462.2	0.25	2	**45348**	**45348.0**	**0.00**	4	**45348**	**45348.0**	**0.00**	6	**45348**	**45348.0**	**0.00**	1	**45348**	**45348.0**	**0.00**	1
pr76	11	44060	12.91	3600	2.40	**43028**	**43028.0**	**0.00**	2	**43028**	**43028.0**	**0.00**	4	**43028**	**43028.0**	**0.00**	12	**43028**	**43028.0**	**0.00**	1	**43028**	**43028.0**	**0.00**	1
rat99	7	508	17.29	3600	4.53	**486**	**486.0**	**0.00**	2	**486**	**486.0**	**0.00**	7	**486**	**486.0**	**0.00**	6	**486**	**486.0**	**0.00**	1	**486**	**486.0**	**0.00**	1
rat99	9	530	40.63	3600	16.48	**455**	**455.0**	**0.00**	2	**455**	**455.0**	**0.00**	7	**455**	**455.0**	**0.00**	6	**455**	**455.0**	**0.00**	1	**455**	**455.0**	**0.00**	1
rat99	11	473	35.09	3600	6.53	**444**	**444.0**	**0.00**	2	**444**	**444.0**	**0.00**	7	**444**	**444.0**	**0.00**	6	**444**	**444.0**	**0.00**	1	**444**	**444.0**	**0.00**	1
kroA100	7	12762	39.07	3600	31.92	**9674**	**9674.0**	**0.00**	3	**9674**	**9674.0**	**0.00**	7	**9674**	**9674.0**	**0.00**	7	**9674**	**9674.0**	**0.00**	1	**9674**	**9674.0**	**0.00**	1
kroA100	9	10130	27.61	3600	10.60	**9159**	**9159.0**	**0.00**	2	**9159**	**9159.0**	**0.00**	7	**9159**	**9159.0**	**0.00**	9	**9159**	**9159.0**	**0.00**	1	**9159**	**9159.0**	**0.00**	1
kroA100	11	12843	49.46	3600	44.29	**8901**	**8901.0**	**0.00**	2	**8901**	**8901.0**	**0.00**	7	**8901**	**8901.0**	**0.00**	10	**8901**	**8901.0**	**0.00**	1	**8901**	**8901.0**	**0.00**	1
kroB100	7	-	-	3600	-	**9537**	**9537.0**	**0.00**	3	**9537**	**9537.0**	**0.00**	7	**9537**	**9537.0**	**0.00**	4	**9537**	**9537.0**	**0.00**	1	**9537**	**9537.0**	**0.00**	1
kroB100	9	10517	36.97	3600	13.82	**9240**	**9240.0**	**0.00**	3	**9240**	**9240.0**	**0.00**	7	**9240**	**9240.0**	**0.00**	9	**9240**	**9240.0**	**0.00**	1	**9240**	**9240.0**	**0.00**	1
kroB100	11	-	-	3600	-	**8842**	**8842.0**	**0.00**	3	**8842**	**8842.0**	**0.00**	7	**8842**	**8842.0**	**0.00**	8	**8842**	**8842.0**	**0.00**	1	**8842**	**8842.0**	**0.00**	1
kroC100	7	10477	22.34	3600	3.22	**9723**	**9723.0**	**0.00**	3	**9723**	**9723.0**	**0.00**	7	**9723**	9724.0	0.01	5	**9723**	**9723.0**	**0.00**	1	**9723**	**9723.0**	**0.00**	1
kroC100	9	10020	30.76	3600	11.44	**9171**	**9171.0**	**0.00**	2	**9171**	**9171.0**	**0.00**	7	**9171**	**9171.0**	**0.00**	10	**9171**	**9171.0**	**0.00**	1	**9171**	9196.0	0.27	1
kroC100	11	11226	47.91	3600	30.05	**8632**	**8632.0**	**0.00**	2	**8632**	**8632.0**	**0.00**	7	**8632**	**8632.0**	**0.00**	5	**8632**	**8632.0**	**0.00**	1	**8632**	**8632.0**	**0.00**	1
kroD100	7	10316	18.25	3600	7.17	**9626**	**9626.0**	**0.00**	2	**9626**	**9626.0**	**0.00**	6	**9626**	**9626.0**	**0.00**	4	**9626**	**9626.0**	**0.00**	1	**9626**	**9626.0**	**0.00**	1
kroD100	9	-	-	3600	-	**8885**	**8885.0**	**0.00**	3	**8885**	**8885.0**	**0.00**	7	**8885**	**8885.0**	**0.00**	5	**8885**	**8885.0**	**0.00**	1	**8885**	**8885.0**	**0.00**	1
kroD100	11	-	-	3600	-	**8725**	**8725.0**	**0.00**	3	**8725**	**8725.0**	**0.00**	7	**8725**	**8725.0**	**0.00**	11	**8725**	**8725.0**	**0.00**	1	**8725**	**8725.0**	**0.00**	1
kroE100	7	10609	14.39	3600	9.11	**10150**	**10150.0**	**0.00**	2	**10150**	**10150.0**	**0.00**	6	**10150**	**10150.0**	**0.00**	7	**10150**	**10150.0**	**0.00**	1	**10150**	**10150.0**	**0.00**	1
kroE100	9	10719	26.10	3600	16.88	**8991**	**8991.0**	**0.00**	2	**8991**	**8991.0**	**0.00**	7	8992	8992.0	0.01	5	8992	8992.0	0.01	1	**8991**	**8991.0**	**0.00**	1
kroE100	11	11879	53.50	3600	40.58	**8450**	**8450.0**	**0.00**	2	**8450**	**8450.0**	**0.00**	7	**8450**	8451.2	0.01	10	**8450**	**8450.0**	**0.00**	1	**8450**	**8450.0**	**0.00**	1
rd100	7	3836	24.00	3600	10.84	**3461**	3485.6	0.71	2	**3461**	3493.8	0.95	6	**3461**	**3461.0**	**0.00**	4	**3461**	**3461.0**	**0.00**	1	**3461**	**3461.0**	**0.00**	1
rd100	9	4049	52.00	3600	26.77	**3194**	**3194.0**	**0.00**	2	**3194**	**3194.0**	**0.00**	6	**3194**	**3194.0**	**0.00**	6	**3194**	**3194.0**	**0.00**	1	**3194**	**3194.0**	**0.00**	1
rd100	11	3946	49.00	3600	35.04	**2922**	**2922.0**	**0.00**	2	**2922**	**2922.0**	**0.00**	6	**2922**	**2922.0**	**0.00**	6	**2922**	**2922.0**	**0.00**	1	**2922**	**2922.0**	**0.00**	1
Avg		-	21.12	2867.40	10.06	8325.69	8329.55	0.04	1.92	**8325.67**	8326.58	0.03	5.31	8325.72	8325.96	0.02	5.71	8325.72	**8325.72**	0.02	**1.00**	**8325.67**	8326.36	**0.01**	**1.00**
NB		9			9	35	33			**36**	**35**			34	31			34	34			**36**	**35**		

Table 5.2: Results of various approaches on medium sized instances

Instance	**NC**	**LS2**				**SN**				**Hybrid ACO**				**ABC**				**GA**			
		Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT	Best cost	Avg.cost	Deviation	TT
kroA150	7	**11423**	11800.0	3.30	4	**11423**	**11423.0**	**0.00**	11	**11423**	**11423.0**	**0.00**	6	**11423**	**11423.0**	**0.00**	1	**11423**	**11423.0**	**0.00**	2
kroA150	9	**10056**	10062.4	0.06	3	**10056**	10057.6	0.02	10	**10056**	**10056.0**	**0.00**	10	**10056**	10057.8	0.02	5	**10056**	10060.8	0.05	1
kroA150	11	**9439**	**9439.0**	**0.00**	3	**9439**	**9439.0**	**0.00**	9	**9439**	**9439.0**	**0.00**	8	**9439**	**9439.0**	**0.00**	1	**9439**	**9439.0**	**0.00**	1
kroB150	7	**11457**	11491.2	0.30	4	**11457**	**11457.0**	**0.00**	10	**11457**	**11457.0**	**0.00**	9	**11457**	**11457.0**	**0.00**	1	**11457**	**11457.0**	**0.00**	3
kroB150	9	**10121**	**10121.0**	**0.00**	4	**10121**	**10121.0**	**0.00**	10	**10121**	**10121.0**	**0.00**	6	**10121**	**10121.0**	**0.00**	1	**10121**	**10121.0**	**0.00**	1
kroB150	11	**9611**	**9611.0**	**0.00**	4	**9611**	**9611.0**	**0.00**	10	**9611**	**9611.0**	**0.00**	11	**9611**	**9611.0**	**0.00**	1	**9611**	**9611.0**	**0.00**	1
kroA200	7	**13285**	13666.4	2.87	6	**13285**	13327.0	0.32	15	13286	13286.0	0.01	12	13286	13286.0	0.01	3	**13285**	**13285.0**	**0.00**	5
kroA200	9	**11708**	11716.8	0.08	5	**11708**	11731.6	0.20	14	11710	11710.0	0.02	12	**11708**	**11708.0**	**0.00**	6	**11708**	**11708.0**	**0.00**	5
kroA200	11	**10748**	10848.6	0.94	5	**10748**	10865.6	1.09	13	10760	10764.2	0.15	15	**10748**	**10748.0**	**0.00**	12	**10748**	10752.8	0.04	6
kroB200	7	13100	13511.6	3.53	5	**13051**	13181.2	1.00	15	**13051**	13061.0	0.08	8	**13051**	**13051.0**	**0.00**	2	**13051**	**13051.0**	**0.00**	4
kroB200	9	11900	11964.8	0.85	5	**11864**	11878.4	0.12	14	**11864**	11871.2	0.06	12	**11864**	**11864.0**	**0.00**	7	**11864**	**11864.0**	**0.00**	3
kroB200	11	10676	10809.6	1.56	5	**10644**	10656.8	0.12	13	**10644**	**10644.0**	**0.00**	9	**10644**	**10644.0**	**0.00**	7	**10644**	**10644.0**	**0.00**	4
Avg		11127.00	11253.53	1.12	4.42	**11117.25**	11145.77	0.24	12.00	11118.50	11120.73	0.03	10.29	11117.33	**11117.48**	**0.00**	3.92	**11117.25**	11118.05	0.01	**3.00**
NB		9	3			**12**	5			9	7			11	**10**			**12**	**10**		

Table 5.3: Results of various approaches on large sized instances

Instance	NC	LS2				SN				Hybrid ACO				ABC					GA				
		Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	TT	Best cost	Avg.cost	Deviation	TB	TT
att532	3	52399	52760.4	2.33	44	52412	52794.4	2.40	35	51616	51841.2	0.55	37	51792	51955.0	0.77	248	408	**51557**	**51823.4**	**0.52**	70	88
att532	5	42634	43280.0	2.69	34	42387	43047.2	2.14	39	42212	42687.4	1.29	41	42389	42503.6	0.85	161	269	**42145**	**42268.0**	**0.29**	73	115
att532	7	38186	38551.8	2.76	38	38016	38244.0	1.94	31	37741	38088.4	1.52	44	37680	37736.0	0.58	147	218	**37517**	**37560.4**	**0.12**	74	125
ali535	3	1370	1387.0	1.46	25	1368	1381.4	1.05	11	**1367**	**1370.0**	**0.22**	28	1372	1375.8	0.64	242	417	1369	1384.0	1.24	46	69
ali535	5	1184	1201.2	1.62	35	1206	1210.2	2.39	21	1185	**1188.4**	**0.54**	30	1189	1190.6	0.73	164	277	**1182**	1189.8	0.66	55	99
ali535	7	1094	1103.6	2.09	30	1086	1093.4	1.15	40	1083	1084.2	0.30	33	1087	1088.4	0.68	147	231	**1081**	**1082.8**	**0.17**	63	112
u574	3	23330	23454.6	2.02	33	23286	23457.0	2.03	37	**22990**	**23101.4**	**0.48**	35	23080	23212.4	0.97	328	552	23096	23237.0	1.07	63	97
u574	5	19417	19578.6	3.02	42	19213	19392.8	2.05	28	19014	19113.6	0.58	46	19076	19155.2	0.80	334	455	**19004**	**19097.8**	**0.49**	86	121
u574	7	16940	17090.8	2.98	34	16880	17056.0	2.77	44	**16597**	16789.0	1.16	33	16722	16765.8	1.02	174	262	16637	**16688.4**	**0.55**	78	132
rat575	3	3755	3783.6	2.07	34	3754	3771.6	1.74	40	**3707**	**3726.2**	**0.52**	44	3724	3737.6	0.83	589	789	3720	3738.4	0.85	83	109
rat575	5	3062	3088.0	2.66	44	3104	3116.2	3.60	30	3034	3053.2	1.50	46	3017	3036.0	0.93	244	359	**3008**	**3030.0**	**0.73**	60	97
rat575	7	2667	2698.6	3.24	35	2645	2678.8	2.48	24	2635	2651.0	1.42	45	2638	2645.4	1.20	201	280	**2614**	**2626.8**	**0.49**	48	75
p654	3	25158	25206.6	0.32	34	25155	25206.0	0.31	36	25166	25182.8	0.22	53	25205	25224.2	0.39	1238	1519	**25127**	**25133.4**	**0.03**	36	88
p654	5	23226	23258.4	0.23	25	23211	23224.8	0.09	35	23242	23289.4	0.36	49	23285	23291.0	0.37	393	616	**23205**	**23215.6**	**0.05**	48	142
p654	7	22121	22233.4	0.52	26	22126	22138.6	0.09	18	22125	22130.8	0.06	47	22123	22125.2	0.03	84	272	**22118**	**22119.6**	**0.01**	87	181
d657	3	30715	30987.2	2.34	39	30653	30878.0	1.98	36	**30278**	**30498.4**	**0.73**	56	30626	30697.8	1.39	647	952	30404	30708.8	1.42	120	146
d657	5	26589	26738.2	3.28	28	26385	26663.8	2.99	50	**25890**	26132.6	0.94	50	25917	26094.4	0.79	435	610	25922	**26055.8**	**0.64**	168	212
d657	7	23429	23545.0	1.97	43	23320	23551.6	2.00	41	23198	23426.8	1.46	54	23204	23252.8	0.71	280	396	**23090**	**23186.0**	**0.42**	150	200
gr666	3	2005	2009.0	1.41	24	2006	2011.4	1.53	23	**1981**	**1993.6**	**0.64**	44	2013	2016.4	1.79	762	1088	2007	2015.4	1.74	77	121
gr666	5	1690	1695.8	2.47	29	1672	1685.4	1.84	19	**1655**	1665.2	0.62	46	1659	1667.4	0.75	454	668	1657	**1663.8**	**0.53**	76	144
gr666	7	1485	1490.4	1.39	25	1474	1481.2	0.76	34	**1470**	**1474.8**	**0.33**	40	1479	1481.0	0.75	374	504	1471	1476.0	0.41	90	159
u724	3	25045	25266.8	2.94	52	25048	25262.4	2.92	51	**24545**	**24746.0**	**0.82**	52	24679	24795.2	1.02	960	1374	24607	24765.6	0.90	161	198
u724	5	20563	20799.0	3.12	45	20575	20844.6	3.34	42	20305	20543.2	1.85	51	20245	20339.8	0.84	356	594	**20170**	**20336.4**	**0.82**	136	196
u724	7	18015	18043.0	2.06	51	17916	18230.4	3.12	44	17737	17943.4	1.50	49	17798	17845.8	0.94	353	503	**17679**	**17740.4**	**0.35**	107	215
rat783	3	5093	5141.0	3.15	32	5076	5119.4	2.72	44	5021	5045.0	1.22	55	5036	5049.8	1.32	1466	1938	**4984**	**5030.8**	**0.94**	214	250
rat783	5	4216	4237.2	3.25	46	4164	4221.6	2.87	45	4117	4172.0	1.66	52	4122	4140.2	0.88	638	902	**4104**	**4119.0**	**0.37**	218	264
rat783	7	3651	3678.8	2.30	50	3692	3700.8	2.91	36	3615	3638.2	1.17	52	3609	3635.4	1.10	569	750	**3596**	**3602.2**	**0.17**	181	224
Avg		16260.70	16381.78	2.17	36.19	16215.93	16350.48	2.00	**34.59**	16056.52	16169.49	0.84	44.48	16102.44	16150.30	0.82	444.00	637.15	**16039.67**	**16107.24**	**0.59**	98.81	147.37
NB		0	0			0	0			10	8			0	0				**17**	**19**			

already. In Table 5.1, there are 4 columns under CPLEX. The description for these 4 columns is provided next. The column named *'best'* reports the cost of the best solution obtained through CPLEX. Some rows have '-' under various columns, which indicates the inability of the CPLEX to obtain even one feasible solution within the allowed time of one hour. The column named *'Rel-gap'* reports the relative MIP gap (the gap between the upper bound and smallest among all lower bounds corresponding to open nodes in the search tree) found by CPLEX. The column named *'Time'* reports the execution time of CPLEX. The column named *'gap'* reports the gap between the upper bound obtained through CPLEX, and the cost of the best solution among all the heuristics. In all these three tables, viz. Table 5.1, Table 5.2 and Table 5.3, the first column named *'Instance'* reports the name of the instance with number of cities at the end. The second column named *'NC'* reports the number of cities that each city covers. The columns named *'Best cost'* & *'Avg. cost'* report the best and average costs of five runs, respectively. The column named *'Deviation'* reports the percentage deviation of $Avg.cost$ from the best cost among all the heuristics. This deviation is computed as $100\times\frac{Avg.cost-best}{best}$, where $best$ is the best cost among all the heuristics. The columns *'TT'* in these three tables report the average execution times over the five runs. In Table 5.3, columns *'TB'* report average time till best over five runs. Actually for large instances, [118] reported only average time till best (and not average execution time) for various approaches. However, we have reported average time till best and average execution time for both of our approaches. The second last row named *'Avg'* in all these three tables, viz. Table 5.1, Table 5.2 and Table 5.3, reports the average values for each column, whereas the last row named *'NB'* reports the number of instances on which a best solution is obtained by the corresponding approach. The best values are marked in bold to facilitate easy identification.

Table 5.4: Summary of performance comparison on average solution quality (Avg. cost)

Categories		**LS2**			**SN**			**Hybrid ACO**			**GA**		
		$<$	=	$>$	$<$	=	$>$	$<$	=	$>$	$<$	=	$>$
Small	**ABC**	2	33	1	1	33	2	3	33	0	1	33	2
	GA	3	32	1	1	34	1	5	30	1	-	-	-
Medium	**ABC**	9	3	0	6	5	1	4	7	1	2	8	2
	GA	9	3	0	6	5	1	5	6	1	-	-	-
Large	**ABC**	24	0	3	24	0	3	12	0	15	4	0	23
	GA	26	0	1	25	0	2	19	0	8	-	-	-
Total	**ABC**	35	36	4	31	38	6	19	40	16	7	41	27
	GA	38	35	2	32	39	4	29	36	10	-	-	-

Table 5.1 shows the results of different approaches for the category of small instances. Out

of 36 instances, both the GA and the SN performed same in getting the best values for best cost and Avg. cost on 36 and 35 instances, respectively. Both the GA and the SN got the same value 8325.67 for the average of best solutions of 36 instances. On average, the ABC obtained the best value 8325.72 for the average performance (Avg. Cost) of the approaches over five different runs. The GA obtained best percentage of 0.01% for average deviation.

Table 5.2 reports the results of different approaches for the category of medium instances. Out of 12 instances, the GA is able to get best values for 12 and 10 instances for the best cost and Avg. cost, respectively. Both the GA and the SN got the same value 11117.25 for the average of best solutions of 12 instances. On average, the ABC obtained the best value 11117.48 for the average performance (Avg. Cost) of the approaches over five different runs. The ABC obtained best percentage of 0.00% for average deviation.

Table 5.3 presents the results of different approaches for the category of large instances. Out of 27 instances, the GA is able to get best values for 17 and 19 number of instances for the best cost and Avg. cost respectively. The GA also got the overall best values 16039.67, 16107.24 for the average of best solutions (best cost), average performance (Avg. cost) respectively for 27 instances. For the average deviation also GA only obtained the best percentage of 0.59%. Our approaches are able to get best overall average solution quality for all the three categories of instances as seen in the second last row of the Tables 5.1, 5.2 and 5.3.

Table 5.4 gives a overall summary of the relative performance of our approaches based on the detailed results presented in previous three tables. In this table, our two approaches (ABC and GA) are compared with other approaches on the average solution quality (Avg. cost). The columns of this table show the number of instances on which the algorithm on the left side (ABC or GA) obtained better ('$<$'), same ('$=$') or worse ('$>$') solution in comparison to the algorithm on the upper side (LS2 or SN or Hybrid ACO or GA). The summary of results in Table 5.1, Table 5.2 and Table 5.3 are reported in the rows corresponding to 'Small', 'Medium' and 'Large' respectively. Finally, the overall performance of our approaches on all the three categories of instances is reported in the last two rows named as 'Total'. Hybrid ACO is the best known approach so far for solving the CSP. Therefore, it is important to compare the performance of our approaches with this Hybrid ACO. Considering all the 75 instances (small+medium+large), the comparison among ABC and Hybrid ACO shows that ABC is able to get better or equal results for 59 instances and worse results for 16 instances. Whereas for GA, these figures are 65 and 10 compared to Hybrid ACO. These 4 tables clearly demonstrate the superiority of ABC and GA over other 4 approaches in terms of solution quality. As mentioned in second paragraph of this

Table 5.5: Results of Wilcoxon signed rank test

	ABC							**GA**					
	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant		$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant
LS2	39/75	763	17	-5.205	-1.960	yes		40/75	814	6	-5.430	-1.960	yes
SN	37/75	662	41	-4.684	-1.960	yes		36/75	657	9	-5.090	-1.960	yes
Hybrid ACO	35/75	322	308	-0.115	-1.960	no		39/75	728	52	-4.717	-1.960	yes
							ABC	33/75	547	14	-4.762	-1.960	yes

section, ABC and GA have been executed on a system different from the one used to execute LS2, SN, Hybrid ACO and CPLEX. Hence, execution times can not be compared precisely. However, our approaches are definitely slower on large instances in comparison to LS2, SN and Hybrid ACO. As far as comparison among our two proposed approaches is concerned, GA performed better than ABC by obtaining better or equal results for 68 instances and worse results for 7 instances. GA is also faster in comparison to ABC.

To check whether there are significant differences among the performances of various approaches for CSP, two tailed Wilcoxon signed rank test [94] has been conducted by setting the significance criteria to 5%, i.e., p-value $\leq$ 0.05. As part of this test, the difference between the normalized values of *'Avg.cost'* obtained by different approaches is ranked. Results of this statistical test are presented in Table 5.5. In this table, the column named $NWT/Total$ reports the number of instances without tied values out of the total number of instances used in comparison. The column named R^+ reports the sum of ranks for the instances where the approach on the top of the table (ABC/GA) performs better than its contender mentioned on the left side of the table, whereas the column R^- reports the sum of ranks for the instances where the approach on the top of the table (ABC/GA) performs worse than its contender mentioned on the left side of the table. As the number of instances without tie exceeds thirty ($NWT >$ 30) in all the cases, we have used the test statistic Z. The resulting Z value is compared with the critical value Z_{Cri} as per the Wilcoxon signed rank test [94]. A Z value not exceeding Z_{Cri} ($Z \leq Z_{Cri}$) indicates a significant difference between the performance of the two approaches being compared, or else the difference is insignificant. Table 5.5 shows that the results of ABC are significant with respect to LS2 & SN. However, there is no significant difference between the performances of ABC and Hybrid ACO. On the other hand, comparison between GA and other approaches shows that the results of GA are significant with respect to all the other approaches (LS2, SN, Hybrid ACO and ABC).

Table 5.6: Comparison of results of GA and ABC after executing them for thirty runs with other approaches

Instance	**NC**	**LS2**				**SN**				**Hybrid ACO**				**ABC**					**GA**				
		Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	Best cost	Avg.cost	Deviation	TB	TT	Best cost	Avg.cost	Deviation	TB	TT
att532	3	52399	52760.4	2.45	44	52412	52794.4	2.52	35	51616	**51841.2**	**0.67**	37	51754	51958.7	0.89	331	493	**51498**	51888.1	0.76	68	97
att532	5	42634	43280.0	2.91	34	42387	43047.2	2.35	39	42212	42687.4	1.50	41	42281	42488.7	1.02	155	254	**42058**	**42280.0**	**0.53**	85	120
att532	7	38186	38551.8	2.79	38	38016	38244.0	1.97	31	37741	38088.4	1.55	44	37608	37783.9	0.74	146	219	**37506**	**37603.7**	**0.26**	51	124
ali535	3	1370	1387.0	1.46	25	1368	1381.4	1.05	11	**1367**	**1370.0**	**0.22**	28	1370	1375.0	0.59	308	478	1369	1387.1	1.47	38	70
ali535	5	1184	1201.2	1.62	35	1206	1210.2	2.39	21	1185	**1188.4**	**0.54**	30	1187	1190.7	0.74	162	272	**1182**	1195.4	1.13	44	101
ali535	7	1094	1103.6	2.28	30	1086	1093.4	1.33	40	1083	1084.2	0.48	33	1084	1088.5	0.88	147	231	**1079**	**1082.8**	**0.36**	54	111
u574	3	23330	23454.6	2.02	33	23286	23457.0	2.03	37	**22990**	**23101.4**	**0.48**	35	23080	23214.3	0.98	446	654	23016	23210.1	0.96	64	99
u574	5	19417	19578.6	3.14	42	19213	19392.8	2.16	28	19014	19113.6	0.69	46	19076	19172.6	1.00	262	386	**18983**	**19108.6**	**0.66**	84	120
u574	7	16940	17090.8	3.39	34	16880	17056.0	3.18	44	16597	16789.0	1.57	33	16619	16762.8	1.41	183	271	**16530**	**16688.4**	**0.96**	100	139
rat575	3	3755	3783.6	2.20	34	3754	3771.6	1.88	40	3707	**3726.2**	**0.65**	44	3718	3741.3	1.06	577	774	**3702**	3735.1	0.89	78	108
rat575	5	3062	3088.0	3.11	44	3104	3116.2	4.05	30	3034	3053.2	1.94	46	3004	3034.3	1.31	249	364	**2995**	**3024.2**	**0.98**	62	101
rat575	7	2667	2698.6	3.28	35	2645	2678.8	2.52	24	2635	2651.0	1.45	45	2628	2649.5	1.40	193	273	**2613**	**2633.8**	**0.79**	37	76
p654	3	25158	25206.6	0.35	34	25155	25206.0	0.35	36	25166	25182.8	0.25	53	25186	25237.2	0.47	1087	1375	**25119**	**25135.0**	**0.06**	37	89
p654	5	23226	23258.4	0.23	25	23211	23224.8	0.09	35	23242	23289.4	0.36	49	23285	23295.6	0.39	344	575	**23205**	**23217.2**	**0.05**	58	144
p654	7	22121	22233.4	0.52	26	22126	22138.6	0.09	18	22125	22130.8	0.06	47	22121	22123.9	0.03	106	284	**22118**	**22120.0**	**0.01**	45	181
d657	3	30715	30987.2	2.34	39	30653	30878.0	1.98	36	**30278**	**30498.4**	**0.73**	56	30502	30689.9	1.36	774	1072	30404	30750.6	1.56	111	142
d657	5	26589	26738.2	3.73	28	26385	26663.8	3.44	50	25890	26132.6	1.38	50	25860	26104.7	1.27	370	553	**25777**	**26014.7**	**0.92**	188	232
d657	7	23429	23545.0	2.36	43	23320	23551.6	2.39	41	23198	23426.8	1.85	54	23115	23258.2	1.11	265	391	**23002**	**23134.8**	**0.58**	135	200
gr666	3	2005	2009.0	1.41	24	2006	2011.4	1.53	23	**1981**	**1993.6**	**0.64**	44	1998	2015.6	1.75	847	1174	1996	2012.8	1.61	93	129
gr666	5	1690	1695.8	2.59	29	1672	1685.4	1.96	19	1655	1665.2	0.74	46	1659	1667.3	0.87	393	585	**1653**	**1663.8**	**0.66**	69	138
gr666	7	1485	1490.4	1.73	25	1474	1481.2	1.11	34	1470	**1474.8**	**0.67**	40	1473	1480.5	1.06	326	463	**1465**	1475.3	0.71	83	161
u724	3	25045	25266.8	3.30	52	25048	25262.4	3.28	51	24545	24746.0	1.17	52	24556	24783.5	1.32	1058	1466	**24460**	**24714.4**	**1.04**	168	208
u724	5	20563	20799.0	3.12	45	20575	20844.6	3.34	42	20305	20543.2	1.85	51	20245	20351.0	0.90	463	703	**20170**	**20346.5**	**0.87**	142	200
u724	7	18015	18043.0	2.30	51	17916	18230.4	3.36	44	17737	17943.4	1.74	49	17793	17878.2	1.37	344	495	**17637**	**17745.2**	**0.61**	148	229
rat783	3	5093	5141.0	3.15	32	5076	5119.4	2.72	44	5021	5045.0	1.22	55	5030	5051.0	1.34	1304	1792	**4984**	**5032.5**	**0.97**	196	236
rat783	5	4216	4237.2	3.40	46	4164	4221.6	3.02	45	4117	4172.0	1.81	52	4109	4145.4	1.16	569	843	**4098**	**4126.8**	**0.70**	201	254
rat783	7	3651	3678.8	2.90	50	3692	3700.8	3.52	36	3615	3638.2	1.77	52	3609	3638.7	1.78	501	686	**3575**	**3607.1**	**0.90**	160	222
Avg		16260.70	16381.78	2.17	36.19	16215.93	16350.48	2.00	**34.59**	16056.52	16169.49	0.84	44.48	16072.22	16154.85	1.04	441.11	636.30	**16007.19**	**16108.67**	**0.78**	96.26	149.30
NB		0	0			0	0			4	8			0	0				**23**	**19**			

Since our approaches are stochastic in nature, we think that five runs are not sufficient to properly ascertain their behavior specially for large instances. Hence, we have also executed our approaches thirty independent times on large instances. Table 5.6 presents the results of different approaches for the category of large instances after running our approaches for thirty times, while retaining the results of other approaches for five runs only. Out of 27 instances, the GA is able to get best values for 23 and 19 number of instances for the best cost and Avg. cost, respectively. Here, the comparison with earlier approaches can not be considered fair, but it will help future researchers to ascertain the performance of their approaches over 30 runs.

5.6 Conclusions

In this chapter, we have presented two hybrid metaheuristic approaches to solve the covering salesman problem. The first one is based on artificial bee colony algorithm, whereas the second one is based on genetic algorithm. The proposed approaches make use of first improvement or best improvement based local searches defined over various neighborhood structures. The proposed approaches are evaluated and compared with the state-of-the-art approaches on the benchmark instances available in the literature [118]. Computational results and their analysis show that our approaches have performed as good as or better than the state-of-the-art approaches in terms of solution quality. As far as comparison between our two proposed approaches is concerned, the GA found solutions of same or better quality than the ABC in shorter times on most of the instances.

Chapter 6

Generalized Covering Traveling Salesman Problem

6.1 Introduction

The generalized covering traveling salesman problem (GCTSP) is a variant of the CSP discussed in the previous chapter. Shaelaie *et al.* [1] formulated the GCTSP by introducing the potential applications of it in humanitarian relief transportation and telecommunication networks. Given a set of n cities which includes depot, facilities and customer cities. A coverage radius r_i is associated with each facility i, and a demand d_j is associated with each customer j. The objective is to construct a minimum length tour over a subset of facilities so that the sumtotal of demands of customers covered by this subset of facilities is at least D. A customer is said to be covered by a subset of facilities if it is within the coverage radius of one or more facilities belonging to this subset. To solve this problem, one has to find a subset of facilities that can satisfy the demand D, and also has to arrange them in an order that minimize the total distance traveled by the salesman. This problem can be considered as a mix of CSP and QTSP (Section 3.1).

While introducing GCTSP, Shaelaie *et al.* [1] proposed two mathematical models (Flow-based and Node-based formulations) and two metaheuristic approaches (memetic algorithm (MA) and variable neighborhood search (VNS)) to solve this problem. Both the metaheuristic approaches (MA and VNS) make use of two local searches viz. the standard 2-opt local search for TSP and drop-and-add local search. The latter local search removes a facility from the tour and in its place tries to add one or two facilities from among its nearest N facilities so that the objective function can be improved while retaining the feasibility. The 2-opt local search is applied first on a solution followed by drop-and-add local search. Each local search is applied in a random order till there is no improvement in tour length. Both MA and VNS outperformed the two mathematical formulations on the benchmark instances considered. As far as comparison between MA and VNS is concerned, for small and medium size instances, both approaches have obtained same number of best solutions, whereas for the large size instances, MA outperformed VNS by providing more number of best solutions than VNS. These are the only approaches available in the literature for this problem.

In this chapter, we have proposed an artificial bee colony (ABC) algorithm based approach for the GCTSP. Unlike the MA and the VNS approaches of [1] which utilize two local searches explicitly to improve the solutions obtained through them, our ABC approach does not make use of any local search explicitly. However, it is equipped with a procedure to remove redundant facilities from a newly generated solution. A redundant facility is one whose removal does not effect the feasibility of a GCTSP solution. Computational results on the same set of benchmark instances as used in [1] show the effectiveness of our ABC approach.

The remainder of the chapter is organized as follows. Section 6.2 formally defines the GCTSP. Our proposed ABC approach for GCTSP is described in Section 6.3. Section 6.4 presents the computational results on benchmark instances and analyses them. Finally, Section 6.5 outlines some concluding remarks.

6.2 Problem definition

Given a graph $G = (V, E)$, where V consists of three sets of vertices, viz. the depot 0, the set of facilities F, and the set of customers C. Hence, $V = \{0\} \cup F \cup C$. Further, $E = \{(i, j) \mid i, j \in F \cup \{0\}\}$. A cost or distance t_{ij} is associated with each edge $(i, j) \in E$, and every customer i has a demand d_i, that will be satisfied by a tour when it is within the coverage distance of at least one facility on that tour. The facilities that are part of the tour are called visited facilities. The GCTSP seeks a tour of minimum length over $\{0\} \cup F$ that begins and ends at the depot 0 so that the sumtotal of the satisfied demands of the customers is at least D. Note that, only a subset F' of the set of facilities F ($F' \subseteq F$) that can satisfy the total demand of at least D needs to be part of a tour. Assume $|V| = n$, and $|F'| = m$. By introducing binary variables, x_{ij} to indicate whether edge (i, j) is part of the tour (i.e., $x_{ij} = 1$) or not (i.e., $x_{ij} = 0$), y_{ij} to indicate whether the demand of customer $i \in C$ is satisfied by the facility $j \in F$ (i.e., $y_{ij} = 1$) or not (i.e., $y_{ij} = 0$), and another binary variable z_i to indicate whether a facility $i \in F$ is visited (i.e., $z_i = 1$) or not (i.e., $z_i = 0$), the mathematical model of the GCTSP can be formulated as follows:

$$\text{Minimize} \quad \sum_{(i,j)\in E} t_{ij}x_{ij} \tag{6.1}$$

subject to:

$$\sum_{i\in C}\sum_{j\in F'} d_i y_{ij} \geq D, \tag{6.2}$$

$$\sum_{j\in F} y_{ij} \leq 1 \quad \forall i \in C, \tag{6.3}$$

$$\sum_{j\in F} x_{0j} = 1 = \sum_{j\in F} x_{j0} \tag{6.4}$$

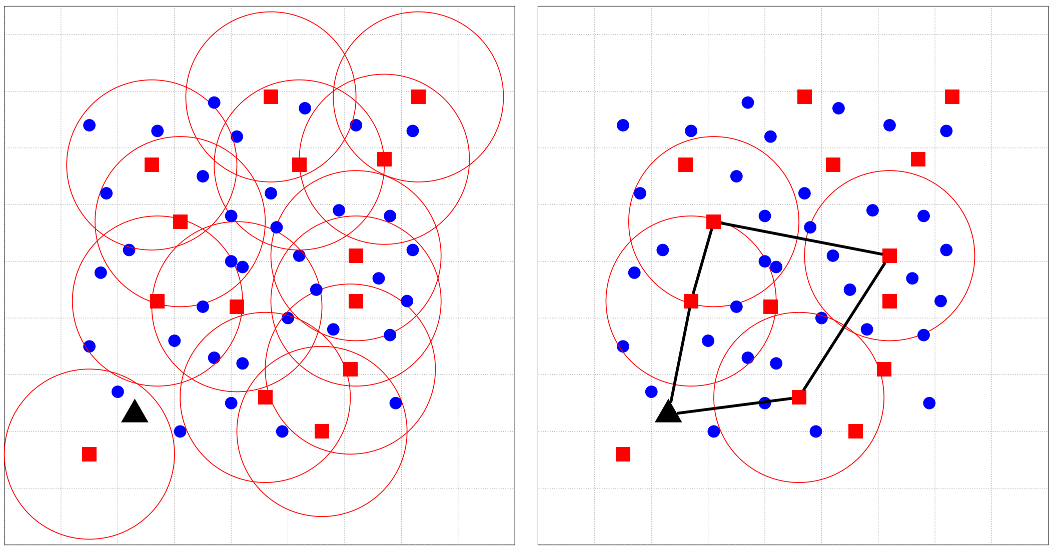
$$\sum_{(i,j)\in E} x_{ij} + \sum_{(j,i)\in E} x_{ji} = 2z_i \quad i \in F \tag{6.5}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \quad \forall S \cup \{0\} \subset F' \cup \{0\} \subset F \cup \{0\}, \quad 2 \leq |S| \leq m-2 \tag{6.6}$$

$$x_{ij}, y_{ij}, z_i \in \{0,1\} \quad \Big[\forall i, j \in F \cup \{0\}\Big], \Big[\forall i \in C, \quad j \in F\Big], \Big[\forall i \in F \cup \{0\}\Big] \tag{6.7}$$

Equation (6.1) is the objective function that minimizes the total distance traveled by the salesman. Equation (6.2) enforces the constraint that the subset of facilities F' that are part of the tour satisfies the given demand D. Equation (6.3) represents the constraint that each customer must be allocated to at most one facility for satisfying that customer's demand. Equation (6.4) assures that the tour must begin and end at the depot 0. Equation (6.5) satisfies the constraints of indegree and outdegree of the visited facilities. Equation (6.6) represents the sub tour elimination constraints. Finally, the Equation (6.7) implies the binary nature of decision variables x_{ij}, y_{ij} and z_i.

Figure 6.1 illustrates GCTSP with the help of an example. Figure 6.1 (a) depicts a GCTSP instance where the depot, facilities and customer vertices are shown as black triangle, red squares and blue circles respectively. There are 36 customers, 14 facilities and 1 depot leading to a total of 51 vertices. The total demand D that needs to be satisfied by a tour is 24 and each customer has a unitary demand. A feasible solution to this GCTSP instance is shown in Figure 6.1 (b) where 4 facilities are visited by the salesman out of 14 facilities and the demands of 24 customers which are within the coverage radius of one or more of these 4 visited facilities are met.



(a) A GCTSP instance with 51 cities (b) A feasible solution

Figure 6.1: Illustration of a feasible solution for the GCTSP

6.3 Artificial bee colony algorithm for the GCTSP

We have developed an artificial bee colony algorithm based approach for the GCTSP whose main components are discussed in the following subsections.

6.3.1 Solution encoding

To represent the solution of GCTSP, we have used the encoding which represents a solution as a linear permutation of the visited facilities. This encoding is also used in [1].

Theorem 1: By recalling the given graph $G = (V, E)$, where $V = \{0\} \cup F \cup C$ is the set of vertices including the depot 0, the set of facilities F, and the set of customers C. Here, the number of cities $|V| = n$, and the number of facilities $|F| = m$. The search space size of the GCTSP according to our encoding is $\sum\limits_{k=1}^{m} \binom{m}{k} \times k!$

Proof. The GCTSP can possibly have $1 \leq k \leq m$ facilities. These k facilities can be chosen from given m facilities in $\binom{m}{k}$ ways. And the chosen k facilities can be ordered in $k!$ ways.

Therefore, the search space size of the GCTSP is $\sum_{k=1}^{m} \binom{m}{k} \times k!$ $\square$

6.3.2 Fitness

The fitness function is same as the objective function given in Equation (6.1) , i.e., a solution's fitness is the total distance traveled by the salesman. Therefore, a solution with a lower value of the fitness function is considered to be better than a solution with a higher value.

6.3.3 Initialization of solutions

Our initial solution generation procedure follows an iterative process where a facility is selected in each iteration. During each iteration, one of the following three heuristics is chosen uniformly at random to select a facility which is inserted into the best possible position in the salesman's tour. To find this, one has to check all the possible insertion positions and then the insertion is carried out at the position which yields the least increase in the cost. This procedure is repeated until a feasible solution is obtained, i.e., one satisfying the given demand.

- *Least insertion cost*: This heuristic selects a facility randomly from the top three facilities which yield least increase in the cost while inserting them in to the salesman tour.
- *Covering the maximum uncovered*: This heuristic selects a facility randomly from the top three facilities which cover the largest number of uncovered customers.
- *Completely random*: This heuristic selects a facility uniformly at random from the facilities which are able to cover at least one uncovered customer.

The strategy of considering top three facilities instead of only the top is taken from [1] and is motivated by the intention to generate diverse solutions.

Three heuristics were also used in [1] to generate the initial solutions. The first heuristic of [1] is a modified form of nearest neighbor heuristic for TSP, where instead of selecting the nearest unvisited facility at each step, a facility among top three nearest unvisited facilities is selected at random. The second and third heuristics are similar to our second and third heuristics. However, while generating an initial solution, unlike our approach, a facility is always added at the last position in [1] instead of the best possible position. Further, in our approach, none of the three heuristics generates an initial solution in-toto. In our approach, each initial solution is generated by combination of these three heuristics where during each iteration one of the three heuristics is chosen at random to select an unvisited facility. On the other hand, in [1], the first heuristic alone is used to generate 50% of initial solutions, whereas the second heuristic alone is used to generate 30% of initial solutions. 15% solutions were generated using a combination of the first and the second heuristic where during each iteration a facility is selected either by the first heuristic or by the second heuristic. Remaining 5% solutions were generated using the third heuristic alone.

6.3.4 Mechanism for choosing a food source

Binary tournament selection method (Section 5.3.4) is used for choosing an employed bee solution for an onlooker bee, where the probability of selection of the better solution is P_{bts}.

6.3.5 Generating a neighboring food source

An effective neighboring solution generation process should exploit various problem aspects in such a manner so as to guide the search process towards an optimal solution. Solving GCTSP involves two crucial aspects. First, choosing a subset of facilities to be part of the tour, and then determining an ordering among the selected facilities. Our neighboring solution generation process is designed keeping in mind these two aspects, viz. subset selection and permutation. Hence, it makes use of two methods which are used one after the other. These two methods are described below:

1. ***Subset-DRO:*** This method handles the subset selection aspects of facilities in GCTSP. This method is a destroy and repair operator which partially destroys the tour and then repairs it. Each facility in the tour is removed with a probability D_p. This destroy procedure makes the solution infeasible. To make the solution feasible again, it is repaired by adding some facilities to the solution under consideration. As part of the repair, following score, which is introduced in [1], is computed for each facility i that is not part of the tour:

$$Score(i) = \frac{AdditionCover(i)}{(AdditionCost(i) + 1)} \tag{6.8}$$

   Here, $AdditionCover(i)$ is the total number of uncovered customers that will be covered by adding facility 'i' into the tour. $AdditionCost(i)$ is the increase in cost of the tour by adding facility 'i' into its best position in the tour. The facility which has the best score is added at its best possible position in the tour. The procedure of adding facilities into the tour repeats until the solution under consideration becomes feasible. Note that a facility to be added must have at least one uncovered customer. This repair operator is similar to addition procedure used by VNS in [1] with one small difference. In our case, always the facility with the maximum score is added, whereas the addition procedure of [1] choose one facility uniformly at random from among top δ facilities with maximum scores. By following the policy of adding the facility with maximum score has avoided a sort of the facilities available for addition in each iteration, thereby yielding some savings in computational time. The feasible solution obtained after adding the facilities may contain redundant facilities. A redundant facility is the one whose removal does not effect the feasibility of a GCTSP solution, i.e., satisfying the given demand. If there are multiple redundant facilities, then a facility is removed which yields maximum decrease in the cost of the solution. The removal procedure continues till there is no redundant facility left in the solution. If the resulting solution after this method is better than the original input solution then this solution replaces the original input solution, otherwise the original input solution is passed to the next method. It is to be noted that there is no provision in MA and VNS of [1] to deal with redundant facilities and such facilities remain in the solutions thereby adversely affecting their qualities.

2. ***Permut-DRO:*** This method handles the permutation aspect of facilities in GCTSP. This method is also a destroy and repair operator like *Subset-DRO*. In this method, some visited facilities are removed from the solution under consideration and then these deleted facilities are reinserted at their best possible positions only to restore the feasibility. In this method, first, each visited facility is removed with probability D_p. All these removed facilities are added to a set, and then, one-by-one, a facility is chosen at random from this set and inserted at best possible position in the tour.

   The main difference between these two methods is that, in the first method, the facilities constituting the tour and their order in the tour may change, whereas in the second method there is no change in the constituent facilities. However, their order in the tour can change.

Like the approach of previous chapter, here also we have utilized a variable degree of perturbation in our ABC approach. The degree of perturbation is directly controlled by parameter D_p. The parameter D_p varies over $iter_{max}$ initial iterations from max_{dp} to min_{dp}. The value of D_p in an iteration $iter$ is calculated as follows:

$$D_p := \left(\frac{max_{dp} - min_{dp}}{iter_{max}}\right)(iter_{max} - iter) + min_{dp} \tag{6.9}$$

Beyond $iter_{max}$ iterations, the parameter D_p takes the fixed value of min_{dp}.

We have not utilized any explicit local search like 2-opt and drop-and-add local searches of [1] to improve the solution obtained through neighboring solution generation process. In fact, if neighboring solution generation process is designed properly keeping in mind various crucial aspects of a problem, explicit local searches will be redundant in most cases. Our computational results prove this point. The pseudo-code for generating a neighboring solution can be seen in Algorithm 18.

Algorithm 18: Neighboring solution generation

```
Input: A solution $S$
Output: A neighboring solution $S'$
begin
    $S' \leftarrow S$;                                                        } Subset-DRO method.
    foreach visited facility $f$ in $S'$ do
        Generate a random number $r$ such that $0 \leq r \leq 1$;
        if $r < D_p$ then
            Remove $f$ from $S'$;
        else
            Keep $f$ in $S'$;
    while $S'$ is infeasible do
        Add an unvisited facility by following procedure described in $Subset - DRO$ method;
    $S \leftarrow S'$;
    $S' \leftarrow \emptyset$;                                                } Permut-DRO method.
    $U \leftarrow \emptyset$;
    foreach visited facility $f$ in $S$ as per their order in $S$ do
        Generate a random number $r$ such that $0 \leq r \leq 1$;
        if $r < D_p$ then
            Add $f$ to $U$;
        else
            Copy $f$ into $S'$;
    foreach facility $f$ in $U$ in some random order do
        Insert $f$ into $S'$ using the $Permut - DRO$ method;
    return $S'$;
```

6.3.6 Other features

Other features are same as the approach of the previous chapter (Section 5.3.6). Like the approach of the previous chapter, the proposed ABC algorithm uses different number of employed bees and onlooker bees. If an employed bee solution does not improve over a fixed number of trials $limit_{scout}$, then the corresponding employed bee abandons that solution and becomes a scout. To make this scout again employed, this scout is assigned a neighboring solution of just abandoned solution irrespective of the fitness of this neighboring solution.

The pseudo-code of our ABC approach is given in Algorithm 19, where n_{eb} and n_{ob} are, respectively, the number of employed and the number of onlooker bees. *Initialize()* is a function that generates and returns an initial solution as per the procedure described in Section 6.3.3. *New_Solution(S)* function returns a new solution in the neighborhood of the solution S as per the procedure described in Section 6.3.5. *Select_Solution($S_1, S_2, \ldots, S_{n_{eb}}$)* function selects a solution for an onlooker bee from employed bee solutions $S_1, S_2, \ldots, S_{n_{eb}}$ using binary tournament selection method (Section 6.3.4) and returns the solution selected.

Algorithm 19: Pseudo code of our ABC approach

```
Input: Set of parameters for the ABC Algorithm and a GCTSP instance
Output: Best solution found
for $i \leftarrow 1$ to $n_{eb}$ do                                              } Initialization phase
    $S_i \leftarrow$ Initialize();
$best \leftarrow$ best solution among $S_1, S_2, \ldots, S_{n_{eb}}$;
while Termination condition not satisfied do
    for $i \leftarrow 1$ to $n_{eb}$ do                                         } Employed bee phase
        $S' \leftarrow$ New_Solution($S_i$);

        if $S'$ is better than $S_i$ then
            $S_i \leftarrow S'$;
    for $i \leftarrow 1$ to $n_{ob}$ do                                         } Onlooker bee phase
        $S_p \leftarrow$ Select_Solution($S_1, S_2, \ldots, S_{n_{eb}}$);
        $S' \leftarrow$ New_Solution($S_p$);

        if $S'$ is better than $S_p$ then
            $S_p \leftarrow S'$;
    for $i \leftarrow 1$ to $n_{eb}$ do                                         } Memorizing best solution
        if $S_i$ is better than $best$ then                                       &
            $best \leftarrow S_i$;                                                dealing with scouts

        else if $S_i$ has not improved over last $limit_{scout}$ trials then
            $S_i \leftarrow$ New_Solution($S_i$);
return $best$;
```

Table 6.1: Parameter settings of ABC

Parameter	**Description**	**Test set**	**Best value**
n_{eb}	Number of employed bees	$\{75, 100, 125\}$	100
n_{ob}	Number of onlooker bees	$\{125, 150, 175\}$	150
$limit_{scout}$	Number of iterations without improvement after which a solution can be discarded	$\{25, 50, 75\}$	50
P_{bts}	Probability of selecting better of the two solutions in binary tournament selection	$\{0.7, 0.8, 0.9\}$	0.8
min_{dp}	Minimum probability to destroy the salesman tour in both methods of neighboring solution generation	$\{0, 0.1, 0.2\}$	0.1
max_{dp}	Maximum probability to destroy the salesman tour in both methods of neighboring solution generation	$\{0.8, 0.9, 1.0\}$	1.0

6.4 Computational results

For testing the performance of ABC, we have used the same test instances as used in [1]. Shaelaie *et al.* [1] used 115 instances to evaluate the performance of their approaches. The number of vertices in these instances range from 51 to 1000. Instances having number of vertices up to 76, between 100 & 200 and between 535 & 1000 are categorized as small, medium, and large size instances respectively. All the 115 instances are Euclidean and contain the co-ordinates of their constituent vertices. The first vertex in these instances is depot, whereas the remaining vertices are partitioned into facilities and customers. Each customer is supposed to have unit demand, and the value of D is chosen to be largest integer ℓ less than or equal to either 50% or 75% or 100% of number of vertices to get at least that ℓ customers covered. In these instances, the set of customers covered by a facility is explicitly specified instead of specifying a coverage radius. There are 32 small instances, 69 medium instances and 14 large instances.

Since ABC algorithm is a stochastic algorithm, therefore, choosing proper parameter values is crucial for the success of ABC. We have used the Taguchi's method [131, 132] to determine various parameter values. There are six parameters in our approaches. We have identified three candidate parameter values for each of these six parameters. These six parameters along with their description and candidate values are listed in Table 6.1. These candidate parameter values are arrived at by considering literature on ABC algorithm, our own previous experience with ABC algorithm, and some preliminary experimentations. In order to determine the value of six parameters each having three candidate values as per Taguchi's method, 18 experiments involving various combinations of candidate parameter values were performed. The combination of parameter values for each of these experiments were determined as per the composition of the rows of $L18$ orthogonal array [131]. Based on these experiments, we have used a population of 100 employed bees ($n_{eb} = 100$), 150 onlooker bees ($n_{ob} = 150$), $P_{bts} = 0.8$, $limit_{scout} = 50$ and the value of D_p ranges from 1.0 to 0.1, i.e. $max_{dp} = 1.0$ and $min_{dp} = 0.10$ in subsequent experiments. These values are also reported in the last column of Table 6.1.

We have implemented ABC in C and executed it on a PC with a 3.10 GHz Intel Core-i5-2400 processor with 4 GB RAM. For benchmarking, we have compared our approach with previously proposed approaches in [1], viz. Flow-based formulation, Node-Based formulation, memetic algorithm and variable neighborhood search, which are represented as Flow-based formulation, Node-based formulation, MA and VNS in the following tables. Our approach is executed on each test instance five times independently like MA and VNS approaches of Shaelaie *et al.* [1]. On small and medium instances, our approach terminates when best solution fails to improve over 500 iterations. The approaches of Shaelaie *et al.* [1] were executed on a PC with a 2.27GHz Intel Core-i5 processor with 4 GB RAM. As these approaches were executed on a system which is different from the system used to execute ABC approach, the running times of these approaches can not be compared precisely with ABC. However, a rough comparison is possible. As both systems use Core-i5 processors, execution times of ABC can be adjusted by multiplying by a factor of $\frac{3.10}{2.27}$ for a rough comparison with approaches of Shaelaie *et al.* [1].

6.4.1 Comparison of our approaches with approaches of [1]

Tables 6.2–6.4 compare the performance of our approach ABC with different approaches proposed in [1]. Table 6.2 , 6.3 and 6.4 reports the results for small, medium, and large size instances respectively. In these tables, the results of Flow-based formulation, Node-based formulation, MA and VNS have been taken from Shaelaie *et al.* [1]. For each of these three tables, the best results over all the approaches are shown in bold font for easy identification. In 6.2 and 6.3 , the first column represents the name of the instance, whereas the next four columns represents the number of vertices ($|V|$), number of facilities ($|F|$), number of customers ($|C|$) and the given demand (D). For both the formulations, viz. Flow-based and Node-based formulations, the columns 'Obj.', 'Gap(%)' and 'Time' reports the objective function value, estimated gap from the optimality (if any) and execution times required by CPLEX to solve the respective formulations. In the columns of MA, VNS, and ABC approaches, the 'Average Obj.' and 'Time' report the average objective function value and average execution time over

Table 6.2: Comparison of various approaches on small size instances

Name	$\lvert V\rvert$	$\lvert F\rvert$	$\lvert C\rvert$	D	*Flow-based formulation*			*Node-based formulation*			*MA*		*VNS*		*ABC*	
					Obj.	Gap(%)	Time	Obj.	Gap(%)	Time	Average Obj.	Time	Average Obj.	Time	Average Obj.	Time
S1	51	15	35	17	**71.33**	0.00	3.00	**71.33**	0.00	0.75	**71.33**	0.13	**71.33**	0.05	**71.33**	0.15(0.20)
S2	51	15	35	26	**105.98**	0.00	3.25	**105.98**	0.00	1.08	**105.98**	0.12	**105.98**	0.06	**105.98**	0.20(0.27)
S3	51	15	35	35	**172.36**	0.00	3.25	**172.36**	0.00	0.08	**172.36**	0.09	**172.36**	0.07	**172.36**	0.23(0.31)
S4	51	20	30	15	**75.29**	0.00	11.78	**75.29**	0.00	3.55	**75.29**	0.14	**75.29**	0.06	**75.29**	0.17(0.23)
S5	51	20	30	22	**102.59**	0.00	37.83	**102.59**	0.00	9.47	**102.59**	0.17	**102.59**	0.07	**102.59**	0.20(0.27)
S6	51	25	25	12	**38.49**	0.00	1.25	**38.49**	0.00	0.24	**38.49**	0.12	**38.49**	0.05	**38.49**	0.15(0.20)
S7	51	25	25	18	**82.38**	0.00	131.23	**82.38**	0.00	25.28	**82.38**	0.17	**82.38**	0.07	**82.38**	0.20(0.27)
S8	51	25	25	25	**140.62**	0.00	108.05	**140.62**	0.00	4.05	**140.62**	0.16	**140.62**	0.10	**140.62**	0.21(0.29)
S9	52	16	35	17	**1378.45**	0.00	1.83	**1378.45**	0.00	0.67	**1378.45**	0.12	**1378.45**	0.05	**1378.45**	0.16(0.22)
S10	52	16	35	26	**2198.91**	0.00	4.05	**2198.91**	0.00	0.59	**2198.91**	0.11	**2198.91**	0.05	**2198.91**	0.20(0.27)
S11	52	21	30	15	**669.76**	0.00	0.56	**669.76**	0.00	0.19	**669.76**	0.13	**669.76**	0.06	**669.76**	0.15(0.20)
S12	52	21	30	22	**1554.50**	0.00	60.91	**1554.50**	0.00	4.08	**1554.50**	0.15	**1554.50**	0.06	**1554.50**	0.17(0.23)
S13	52	21	30	30	**3910.04**	0.00	173.47	**3910.04**	0.00	13.94	**3910.04**	0.14	**3910.04**	0.10	**3910.04**	0.18(0.25)
S14	52	26	25	12	**572.13**	0.00	0.47	**572.13**	0.00	0.97	**572.13**	0.15	**572.13**	0.06	**572.13**	0.15(0.20)
S15	52	26	25	18	**1240.67**	0.00	249.61	**1240.67**	0.00	53.17	**1240.67**	0.14	**1240.67**	0.07	**1240.67**	0.18(0.25)
S16	52	26	25	25	**2958.78**	0.00	1179.13	**2958.78**	0.00	746.22	**2958.78**	0.15	**2958.78**	0.12	**2958.78**	0.21(0.29)
S17	52	16	35	17	**1340.29**	0.00	2.19	**1340.29**	0.00	0.81	**1340.29**	0.15	**1340.29**	0.06	**1340.29**	0.14(0.19)
S18	52	16	35	26	**2291.58**	0.00	10.77	**2291.58**	0.00	2.11	**2291.58**	0.13	**2291.58**	0.06	**2291.58**	0.20(0.27)
S19	52	21	30	15	**878.30**	0.00	3.36	**878.30**	0.00	0.38	**878.30**	0.15	**878.30**	0.05	**878.30**	0.14(0.19)
S20	52	21	30	22	**1521.64**	0.00	45.94	**1521.64**	0.00	6.88	**1521.64**	0.14	**1521.64**	0.06	**1521.64**	0.20(0.27)
S21	52	21	30	30	**3590.08**	0.00	38.06	**3590.08**	0.00	5.17	**3590.08**	0.13	**3590.08**	0.10	**3590.08**	0.18(0.25)
S22	52	26	25	12	**479.76**	0.00	0.42	**479.76**	0.00	0.81	**479.76**	0.13	**479.76**	0.05	**479.76**	0.15(0.20)
S23	52	26	25	18	**1334.81**	0.00	432.09	**1334.81**	0.00	44.74	**1334.81**	0.15	**1334.81**	0.07	**1334.81**	0.18(0.25)
S24	52	26	25	25	**2894.96**	0.00	3484.84	**2894.96**	0.00	578.73	**2894.96**	0.19	**2894.96**	0.09	**2894.96**	0.19(0.26)
S25	76	23	52	26	**98.94**	0.00	130.92	**98.94**	0.00	30.03	**98.94**	0.16	**98.94**	0.07	**98.94**	0.21(0.29)
S26	76	23	52	39	**147.89**	0.00	358.36	**147.89**	0.00	45.19	**147.89**	0.15	**147.89**	0.09	**147.89**	0.29(0.40)
S27	76	30	45	22	**86.16**	0.00	78.69	**86.16**	0.00	50.33	**86.16**	0.17	**86.16**	0.08	**86.16**	0.21(0.29)
S28	76	30	45	33	**119.54**	0.00	141.55	**119.54**	0.00	38.92	**119.54**	0.19	**119.54**	0.09	**119.54**	0.26(0.36)
S29	76	38	37	18	**57.79**	0.00	57.94	**57.79**	0.00	34.25	**57.79**	0.14	**57.79**	0.08	**57.79**	0.22(0.30)
S30	76	38	37	27	**106.39**	15.31	3600.00	**106.39**	16.07	3600.00	**106.39**	0.24	**106.39**	0.10	**106.39**	0.25(0.34)
S31	76	38	37	37	**173.59**	16.03	3600.00	**173.59**	0.00	1846.03	**173.59**	0.27	**173.59**	0.21	**173.59**	0.28(0.38)
S32	76	38	37	18	**20279.16**	0.00	2653.58	21638.57	33.99	3600.00	**20279.16**	0.15	**20279.16**	0.09	**20279.16**	0.17(0.23)
No. Best					**32**			31			**32**		**32**		**32**	
Average					**1583.54**	0.98	519.01	1626.02	1.56	335.90	**1583.54**	0.15	**1583.54**	0.08	**1583.54**	0.19(0.26)

Table 6.3: Comparison of various approaches on medium size instances

Name	$\|V\|$	$\|F\|$	$\|C\|$	D	**Flow-based formulation**			**Node-based formulation**			**MA**		**VNS**		**ABC**	
					Obj.	Gap(%)	Time	Obj.	Gap(%)	Time	Average Obj.	Time	Average Obj.	Time	Average Obj.	Time
M1	100	30	69	34	**6043.13**	15.58	3600.00	6061.44	12.34	3600.00	**6043.13**	0.23	**6043.13**	0.11	**6043.13**	0.30(0.41)
M2	100	30	69	51	**8471.25**	0.00	2763.55	**8471.25**	0.00	1358.34	**8471.25**	0.27	**8471.25**	0.12	**8471.25**	0.39(0.53)
M3	100	40	59	29	**3836.03**	0.00	3111.47	**3836.03**	0.00	2589.89	**3836.03**	0.28	**3836.03**	0.10	**3836.03**	0.22(0.30)
M4	100	40	59	44	6424.29	33.82	3600.00	6424.29	30.47	3600.00	**6288.73**	0.33	**6288.73**	0.13	**6288.73**	0.28(0.38)
M5	100	40	59	59	**8789.24**	13.04	3600.00	**8789.24**	0.00	1310.11	**8789.24**	0.36	**8789.24**	0.28	**8789.24**	0.34(0.46)
M6	100	50	49	24	**3148.77**	0.00	3285.84	**3148.77**	10.95	3600.00	**3148.77**	0.21	**3148.77**	0.12	**3148.77**	0.22(0.30)
M7	100	50	49	36	5783.34	45.20	3600.08	6056.26	46.18	3600.00	**5777.62**	0.38	**5777.62**	0.16	5779.58	0.26(0.36)
M8	100	50	49	49	9114.16	35.46	3600.00	**9094.20**	26.20	3600.00	**9094.20**	0.40	**9094.20**	0.43	**9094.20**	0.34(0.46)
M9	100	30	69	34	**4362.92**	0.00	2479.83	**4362.92**	0.00	251.84	**4362.92**	0.18	**4362.92**	0.10	**4362.92**	0.31(0.42)
M10	100	30	69	51	**8105.04**	13.70	3600.00	**8105.04**	14.85	3600.00	**8105.04**	0.25	**8105.04**	0.14	**8105.04**	0.35(0.48)
M11	100	40	59	29	**3592.77**	10.99	3600.00	**3592.77**	0.00	736.28	**3592.77**	0.23	**3592.77**	0.11	**3592.77**	0.22(0.30)
M12	100	40	59	44	7007.12	38.64	3600.00	7326.59	33.70	3600.00	**6948.96**	0.33	**6948.96**	0.15	**6948.96**	0.32(0.44)
M13	100	50	49	24	**2929.98**	0.00	2763.84	**2929.98**	0.00	2554.08	**2929.98**	0.30	**2929.98**	0.10	**2929.98**	0.19(0.26)
M14	100	50	49	36	**5446.84**	38.22	3600.00	**5446.84**	33.41	3600.00	**5446.84**	0.32	**5446.84**	0.14	**5446.84**	0.26(0.36)
M15	100	30	69	34	5274.42	23.29	3600.00	**5164.91**	0.00	1298.50	**5164.91**	0.29	**5164.91**	0.11	**5164.91**	0.26(0.36)
M16	100	30	69	51	**7298.66**	12.33	3600.00	**7298.66**	3.59	3600.00	**7298.66**	0.30	**7298.66**	0.14	**7298.66**	0.33(0.45)
M17	100	40	59	29	4662.59	32.54	3600.00	**4580.82**	19.13	3600.00	**4580.82**	0.25	**4580.82**	0.13	**4580.82**	0.24(0.33)
M18	100	40	59	44	**6880.86**	35.94	3600.00	6917.87	34.47	3600.00	**6880.86**	0.35	**6880.86**	0.15	**6880.86**	0.31(0.42)
M19	100	50	49	24	**3749.47**	34.96	3600.00	3753.16	21.93	3600.00	**3749.47**	0.24	**3749.47**	0.11	**3749.47**	0.25(0.34)
M20	100	50	49	36	6549.37	50.35	3600.00	6170.11	43.45	3600.00	**5936.53**	0.40	**5936.53**	0.17	**5936.53**	0.29(0.40)
M21	100	50	49	49	**8506.35**	17.83	3600.68	8643.74	16.03	3600.00	**8506.35**	0.46	**8506.35**	0.44	**8506.35**	0.34(0.46)
M22	100	30	69	34	5842.87	8.83	3600.00	5842.87	21.07	3600.00	**5704.71**	0.27	**5704.71**	0.10	**5704.71**	0.29(0.40)
M23	100	30	69	51	**7513.86**	0.00	334.38	**7513.86**	0.00	53.09	**7513.86**	0.26	**7513.86**	0.13	**7513.86**	0.36(0.49)
M24	100	40	59	29	**5029.98**	33.38	3600.00	**5029.98**	36.85	3600.00	5072.39	0.34	**5029.98**	0.11	**5029.98**	0.26(0.36)
M25	100	40	59	44	**6456.74**	20.15	3600.00	**6456.74**	4.61	3600.00	**6456.74**	0.34	**6456.74**	0.13	**6456.74**	0.33(0.45)
M26	100	50	49	24	4420.68	42.92	3600.00	4490.01	56.73	3600.00	**4280.76**	0.25	**4280.76**	0.11	**4280.76**	0.26(0.36)
M27	100	50	49	36	**5833.35**	41.01	3600.00	5976.48	42.91	3600.00	**5833.35**	0.35	**5833.35**	0.14	**5833.35**	0.24(0.33)
M28	100	50	49	49	**8143.74**	25.06	3600.00	**8143.74**	13.07	3600.00	**8143.74**	0.46	**8143.74**	0.42	**8143.74**	0.31(0.42)
M29	100	30	69	34	**4750.53**	0.00	1815.30	**4750.53**	0.00	82.46	**4750.53**	0.21	**4750.53**	0.09	**4750.53**	0.26(0.36)
M30	100	30	69	51	**7784.01**	14.81	3600.00	**7784.01**	0.00	835.74	**7784.01**	0.26	**7784.01**	0.12	**7784.01**	0.33(0.45)
M31	100	40	59	29	**3396.99**	6.23	3600.00	**3396.99**	0.00	171.93	**3396.99**	0.29	**3396.99**	0.12	**3396.99**	0.22(0.30)
M32	100	40	59	44	**6584.91**	33.59	3600.00	**6584.91**	24.36	3600.00	**6584.91**	0.37	**6584.91**	0.17	**6584.91**	0.31(0.42)
M33	100	40	59	59	**9949.88**	15.22	3600.00	**9949.88**	0.00	899.30	**9949.88**	0.32	**9949.88**	0.34	**9949.88**	0.38(0.52)
M34	100	50	49	24	**3066.63**	20.66	3600.00	**3066.63**	14.26	3600.00	**3066.63**	0.29	**3066.63**	0.10	**3066.63**	0.20(0.27)
M35	100	50	49	36	5750.13	44.72	3600.00	5750.13	46.67	3600.00	**5657.02**	0.41	**5657.02**	0.13	**5657.02**	0.24(0.33)
M36	100	50	49	49	**8009.42**	11.93	3600.00	**8009.42**	11.73	3600.00	**8009.42**	0.47	**8009.42**	0.51	**8009.42**	0.33(0.45)
M37	100	30	69	34	**1726.09**	0.00	1251.83	**1726.09**	0.00	493.80	**1726.09**	0.22	**1726.09**	0.09	**1726.09**	0.25(0.34)
M38	100	30	69	51	**2501.29**	0.00	104.81	**2501.29**	0.00	44.00	**2501.29**	0.22	**2501.29**	0.13	**2501.29**	0.32(0.44)
M39	100	40	59	29	**1518.85**	31.55	3600.00	1540.25	35.57	3600.00	**1518.85**	0.28	**1518.85**	0.11	**1518.85**	0.22(0.30)
M40	100	40	59	44	**2107.69**	20.41	3600.00	**2107.69**	26.42	3600.00	**2107.69**	0.34	**2107.69**	0.15	**2107.69**	0.32(0.44)
M41	100	50	49	24	1570.85	41.36	3600.00	1632.52	54.75	3600.00	**1499.04**	0.23	**1499.04**	0.11	**1499.04**	0.21(0.29)
M42	100	50	49	36	2238.72	41.59	3600.00	2558.96	50.54	3600.00	**2208.52**	0.31	**2208.52**	0.16	**2208.52**	0.26(0.36)
M43	100	50	49	49	**3228.85**	7.83	3600.00	**3228.85**	3.61	3600.00	**3228.85**	0.30	**3228.85**	0.61	**3228.85**	0.37(0.51)
M44	150	45	104	52	6800.55	36.43	3600.00	6611.13	30.72	3600.00	**6557.02**	0.33	**6557.02**	0.15	**6557.02**	0.35(0.48)

continued ...

Name	$\|V\|$	$\|F\|$	$\|C\|$	D	**Flow-based formulation**			**Node-based formulation**			**MA**		**VNS**		**ABC**	
					Obj.	Gap(%)	Time	Obj.	Gap(%)	Time	Average Obj.	Time	Average Obj.	Time	Average Obj.	Time
M45	150	45	104	78	9558.62	29.37	3600.00	9494.90	23.62	3600.00	**9489.22**	0.43	9503.10	0.24	**9489.22**	0.47(0.64)
M46	150	60	89	44	5941.15	54.81	3600.00	5983.58	55.35	3600.00	**5393.39**	0.36	**5393.39**	0.17	**5393.39**	0.30(0.41)
M47	150	60	89	66	8557.71	52.88	3600.00	8171.50	48.29	3600.00	**8149.13**	0.64	**8149.13**	0.26	8171.50	0.41(0.56)
M48	150	75	74	37	4950.34	57.27	3600.00	**4583.03**	56.53	3600.00	**4583.03**	0.35	**4583.03**	0.19	**4583.03**	0.30(0.41)
M49	150	75	74	55	7388.02	59.08	3600.00	8329.72	66.61	3600.00	**6841.86**	0.57	**6841.86**	0.23	**6841.86**	0.37(0.51)
M50	150	75	74	74	11127.67	50.08	3600.00	10067.26	35.43	3600.00	**10061.50**	0.70	10170.60	1.41	**10061.50**	0.52(0.71)
M51	150	45	104	52	6152.91	35.80	3600.00	6377.95	30.71	3600.00	**6133.27**	0.37	**6133.27**	0.19	**6133.27**	0.33(0.45)
M52	150	45	104	78	**8629.85**	25.70	3600.00	8730.54	16.41	3600.00	**8629.85**	0.45	**8629.85**	0.22	**8629.85**	0.48(0.66)
M53	150	60	89	44	5200.64	56.50	3600.00	5305.46	58.71	3600.00	**5199.33**	0.39	**5199.33**	0.16	**5199.33**	0.28(0.38)
M54	150	60	89	66	8712.97	55.90	3600.00	7624.55	48.94	3600.00	7471.30	0.64	**7471.14**	0.24	**7471.14**	0.38(0.52)
M55	150	75	74	37	5218.44	62.19	3600.00	7246.78	83.33	3600.00	**4898.98**	0.38	**4898.98**	0.18	**4898.98**	0.27(0.37)
M56	150	75	74	55	7573.78	60.09	3600.00	7959.61	73.71	3600.00	**7169.79**	0.57	**7169.79**	0.27	**7169.79**	0.35(0.48)
M57	200	60	139	69	6286.19	47.90	3600.00	6356.29	44.03	3600.00	6281.59	0.61	**6281.11**	0.27	**6281.11**	0.49(0.67)
M58	200	60	139	104	**9476.31**	30.88	3600.00	**9476.31**	20.52	3600.00	**9476.31**	0.64	9499.65	0.37	**9476.31**	0.75(1.02)
M59	200	80	119	59	6162.23	60.83	3600.00	6199.21	71.84	3600.00	**5620.09**	0.66	**5620.09**	0.26	**5620.09**	0.39(0.53)
M60	200	80	119	89	9241.57	63.05	3600.00	8827.56	57.22	3600.00	**8464.17**	0.91	**8464.17**	0.41	**8464.17**	0.56(0.76)
M61	200	100	99	49	5439.22	64.73	3600.00	5549.83	73.90	3600.00	**4895.78**	0.57	**4895.78**	0.28	4907.93	0.35(0.48)
M62	200	100	99	74	8388.30	66.49	3600.00	8346.32	71.59	3600.00	**7366.94**	1.01	7383.64	0.44	7379.09	0.43(0.59)
M63	200	60	139	69	6562.64	45.87	3600.00	6814.37	36.87	3600.00	**6257.61**	0.49	6471.74	0.25	**6257.61**	0.43(0.59)
M64	200	60	139	104	9626.72	34.50	3600.00	9642.84	19.73	3600.00	9588.34	0.80	9579.08	0.39	**9550.47**	0.63(0.86)
M65	200	80	119	59	6307.84	59.72	3600.00	6005.98	56.01	3600.00	**5442.88**	0.63	**5442.88**	0.24	**5442.88**	0.36(0.49)
M66	200	80	119	89	8854.88	54.51	3600.00	9175.32	51.43	3600.00	8368.61	0.84	**8359.44**	0.40	**8359.44**	0.58(0.79)
M67	200	100	99	49	5226.24	61.70	3600.00	5342.40	67.10	3600.00	4887.60	0.57	**4872.50**	0.23	**4872.50**	0.35(0.48)
M68	200	100	99	74	9338.61	64.42	3600.00	8593.37	63.76	3600.00	**7217.00**	0.79	**7217.00**	0.35	**7217.00**	0.48(0.66)
M69	200	100	99	99	11966.08	44.77	3600.00	11088.01	38.97	3600.00	**10754.88**	1.03	10841.55	4.32	**10754.88**	0.62(0.85)
No. Best					33			30			63		62		**65**	
Average					6262.20	32.59	3390.05	6262.62	30.31	3053.34	6045.62	0.42	6051.23	0.28	**6044.80**	0.34(0.46)

five different runs. The value in parenthesis under column 'Time' for ABC reports the adjusted execution times of our approach as explained in the beginning of this section. The last two rows in both the Tables 6.2 and 6.3 report the number of best solutions found by each algorithm (row named 'No. Best') and the average values of each column (row named 'Average'). On small instances, all the approaches except for node based formulation performed the same in terms of solution quality on all the instances. Node based formulation performed slightly worse in terms of solution quality. Our ABC approach got best known results on all 32 instances like MA and VNS. In comparison to MA and VNS approaches, our approach is clearly slower on small instances. However, it is faster than the two exact approaches. For medium instances, exact approaches are too slow and they fail to find the proven optimal solution on most instances in maximum allowed execution time of 3600 seconds. Their solution quality is also quite inferior in comparison to three metaheuristic approaches, viz. MA, VNS, and ABC. As far as comparison among three metaheuristic approaches is concerned, ABC performed the best, followed by MA and VNS. ABC obtained the best known solution values for 65 instances. In comparison, MA and VNS obtained best known solution values for 63 and 62 instances respectively. ABC obtained better overall average solution quality than other approaches. Our ABC is slower than MA and VNS on majority of instances. However, on some instances it is clearly faster.

On 14 large instances, Shaelaie *et al.* [1] reported the results of MA and VNS after executing them for a fixed amount of time. They have reported three sets of results with time limit of 30, 60 and 120 seconds. For each instance, best and average solution obtained by MA and VNS over five different runs were reported under each of the three time limits. Results of exact approaches were not reported on these large instances presumably because they were too slow to be of any use on these large instances. To compare our approaches with the previous approaches (MA and VNS) on these large instances, we have also used the CPU time as termination criteria like Shaelaie *et al.* [1] . However, we have adjusted our time limits to $30 \times \frac{2.27}{3.10}$, $60 \times \frac{2.27}{3.10}$ and $120 \times \frac{2.27}{3.10}$ to compensate for difference in processing speeds of the system used to execute our approach ABC and the system used to execute MA and VNS approaches. The comparison of the results according to three different termination criterias based on time limits have been reported in Table 6.4 . The columns 'Best Obj.' and 'Average Obj.' represents the best and average objective function values obtained by respective approaches over five different runs. The last two rows in this Table 6.4, i.e., row named 'No. Best' reports the number of best solutions achieved by each algorithm, and row named 'Average' reports the average value of the corresponding column. The results presented in Table 6.4 clearly demonstrates the superiority

Table 6.4: Comparison of various approaches on large size instances

Name	**30 sec**†						**60 sec**†						**120 sec**†					
	Best Obj.			Average Obj.			Best Obj.			Average Obj.			Best Obj.			Average Obj.		
	MA	**VNS**	**ABC**	**MA**	**VNS**	**ABC**	**MA**	**VNS**	**ABC**	**MA**	**VNS**	**ABC**	**MA**	**VNS**	**ABC**	**MA**	**VNS**	**ABC**
L1	**260.92**	261.32	**260.92**	265.66	262.58	**261.00**	**260.92**	261.32	**260.92**	265.66	262.58	**261.03**	**260.92**	261.32	**260.92**	265.03	262.58	**261.03**
L2	568.82	569.46	**568.01**	571.59	592.01	**568.29**	568.82	569.32	**567.95**	571.59	591.25	**568.27**	568.82	569.32	**567.95**	570.56	585.43	**568.06**
L3	221.78	**219.01**	**219.01**	224.43	222.36	**220.14**	221.78	**219.01**	**219.01**	224.43	221.86	**220.11**	221.78	**219.01**	**219.01**	224.43	221.86	**220.13**
L4	537.71	521.77	**519.32**	540.90	534.94	**521.65**	521.77	521.62	**519.36**	536.31	534.36	**520.62**	521.77	521.62	**519.33**	536.01	531.29	**520.73**
L5	174.33	175.90	**173.47**	177.27	178.07	**173.53**	174.33	175.90	**173.47**	177.27	178.07	**173.52**	174.33	175.90	**173.47**	177.27	178.07	**173.54**
L6	**409.69**	421.19	410.41	413.59	432.73	**411.23**	409.69	416.06	**409.60**	413.59	426.54	**409.60**	409.69	416.06	**409.60**	413.59	425.91	**410.15**
L7	**9659.42**	9860.09	9702.00	9783.33	10262.22	**9719.99**	**9659.42**	9860.09	9679.05	9775.89	10236.86	**9698.64**	**9659.42**	9860.09	**9659.42**	9746.22	10222.04	**9678.74**
L8	7445.34	7428.56	**7425.21**	7475.27	7444.14	**7432.41**	7445.34	7428.56	**7423.48**	7475.27	7434.01	**7426.71**	7445.34	**7410.18**	7423.06	7474.78	7430.33	**7424.93**
L9	3406315.00	3466420.00	**3310439.50**	3451099.00	3690228.25	**3322650.50**	3363319.00	3462163.00	**3304538.50**	3433883.25	3688099.50	**3316275.25**	3359436.25	3420977.50	**3293890.50**	3428711.25	3669406.25	**3301858.50**
L10	5809963.00	5956696.00	**5763888.50**	5882573.50	6204090.50	**5771842.50**	5801249.00	5944205.50	**5722106.50**	5861252.00	6195724.00	**5742636.50**	5797975.00	5940558.50	**5687455.00**	5851822.50	6169100.50	**5696322.50**
L11	2715609.50	**2698974.25**	2706040.00	2848941.50	2830881.25	**2717308.25**	2715609.50	**2698974.25**	**2698974.25**	2841862.00	2825327.00	**2704331.25**	2715609.50	2698974.25	**2698278.00**	2840534.25	2776202.50	**2699507.00**
L12	**4852149.00**	4969874.00	4870647.50	4966469.00	5354164.50	**4908488.00**	**4788332.00**	4954128.00	4818295.50	4925313.50	5348881.50	**4837634.00**	**4766605.50**	4954128.00	4802720.00	4910558.50	5330731.00	**4810786.50**
L13	2466742.00	**2402721.75**	2467296.00	2540296.75	2623601.25	**2491357.00**	2466742.00	**2402721.75**	2405866.50	2531566.75	2592871.00	**2437261.00**	2459972.75	**2399977.75**	2423088.00	2525418.25	2592322.25	**2434717.25**
L14	**4391044.00**	4485000.50	4397055.00	4425225.00	4729981.50	**4420655.50**	**4361460.00**	4481123.50	4391208.00	**4404383.50**	4652509.00	4408860.50	**4338690.50**	4479302.50	4339843.00	4391780.00	4643592.50	**4366767.00**
No. Best	5	3	**8**	0	0	**14**	4	3	**10**	1	0	**13**	4	3	**10**	0	0	**14**
Average	1690078.50	1714224.62	**1681046.00**	1723861.25	1818062.62	**1689400.62**	1679712.38	1711626.25	**1668588.75**	1715550.12	1808807.00	**1676162.62**	1675539.38	1708096.50	**1661750.50**	1712016.62	1800086.62	**1666372.62**

†-Time limit for MA/VNS. Time limit for ABC was $\frac{2.27}{3.10}$ times the time limit for MA/VNS.

of our approach ABC on large size instances. Our approach has obtained more number of best known values and better overall average solution quality than MA and VNS on both quality parameters (Best Obj. and Average Obj.).

Table 6.5 presents a summary of comparison of ABC with Flow-based formulation, Node-based formulation, MA and VNS approaches in terms of number of instances on which ABC obtained better ('$<$'), same ('=') or worse solutions ('$>$'). On small and medium instances this comparison is done in terms of average solution quality, and on large instances in terms of both best and average solution quality under each of the three time limits. Actually, Shaelaie *et al.* [1] reported only average solution quality for small and medium instances, whereas for large instances both best and average solution qualities were reported. Also they have not reported the results of Flow-based formulation and Node-based formulation for the large instances. This table shows that relative performance of ABC improves with instance size and with increase in time limit in comparison to MA and VNS approaches. Further, on large instances ABC performed consistently better in terms of average solution quality, which shows its robustness.

Table 6.5: Summary of results comparing ABC with other approaches

ABC vs...	**Flow-based formulation**			**Node-based formulation**			**MA**			**VNS**		
Types of instances	$<$	=	$>$	$<$	=	$>$	$<$	=	$>$	$<$	=	$>$
Small	0	32	0	1	31	0	0	32	0	0	32	0
Medium	36	33	0	38	31	0	6	59	4	7	59	3
Large(30sec)-Best	-	-	-	-	-	-	8	1	5	11	1	2
Large(30sec)-Avg	-	-	-	-	-	-	14	0	0	14	0	0
Large(60sec)-Best	-	-	-	-	-	-	10	1	3	11	2	1
Large(60sec)-Avg	-	-	-	-	-	-	13	0	1	14	0	0
Large(120sec)-Best	-	-	-	-	-	-	10	2	2	11	1	2
Large(120sec)-Avg	-	-	-	-	-	-	14	0	0	14	0	0
TOTAL	36	65	0	39	62	0	75	95	15	82	95	8

Figure 6.2 plots the solutions obtained by our approach on four different instances, viz S2, S3, S7 and S8 where the depot, facilities and customer vertices are shown as black triangle, red squares and blue circles respectively. All these four instances are optimally solved by our approaches. This can be inferred from Table 6.2 as ABC obtained same solutions as obtained by exact approaches with optimality gap of zero on these four instances. These plots show how the facilities and the customers are distributed and how the customers are covered by the facilities in these instances. These plots also depict the composition of an optimal solution for each of these instances. Instances S2 and S3 have 15 facilities and 35 customers, whereas instances S7 and S8 have 25 facilities and 25 customers. The instances S2 and S7 demand 75% of customers to

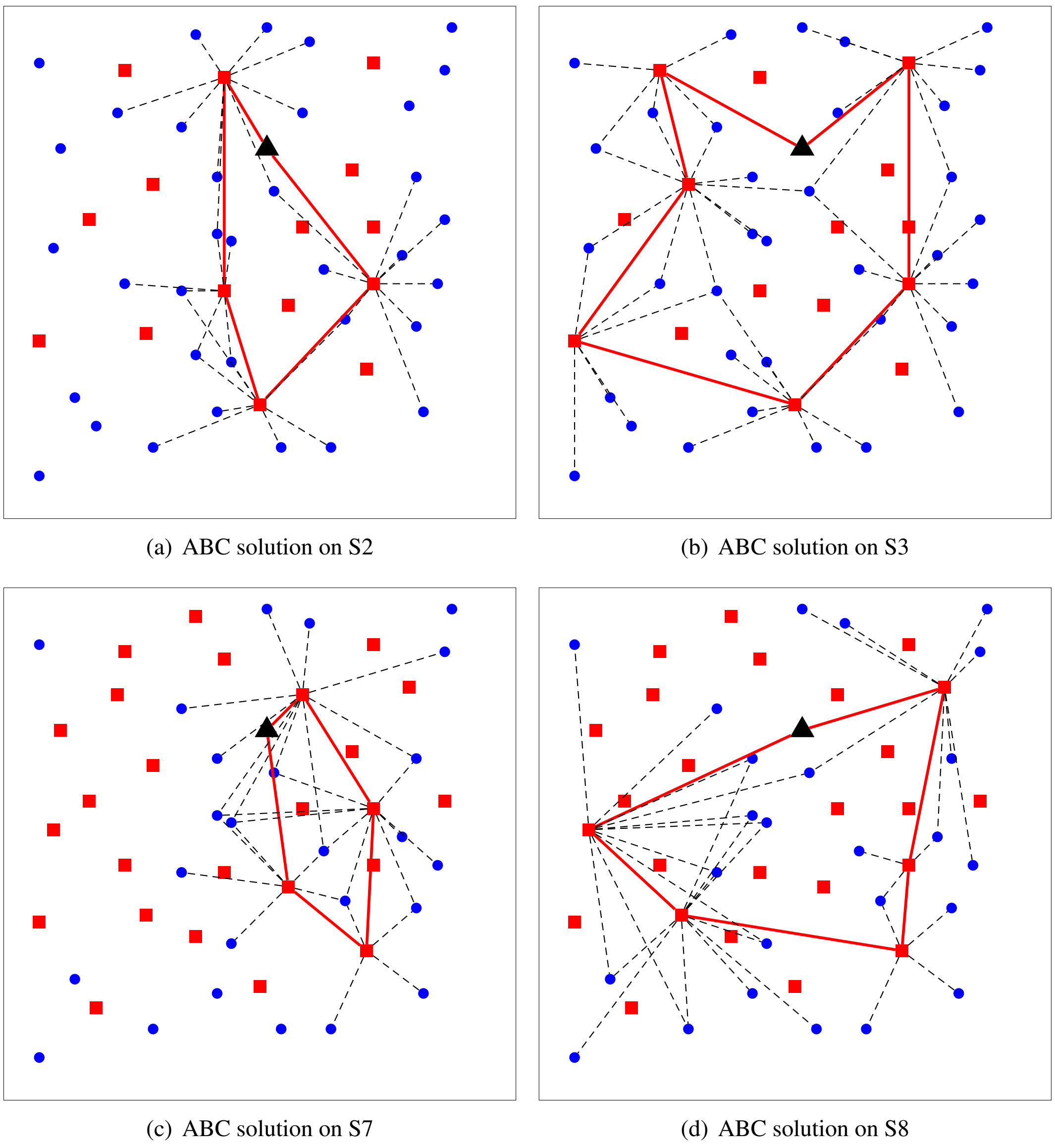

(a) ABC solution on S2
(b) ABC solution on S3
(c) ABC solution on S7
(d) ABC solution on S8

Figure 6.2: Solutions obtained by ABC on four different instances

be covered, whereas the instances S3 and S8 demand 100% of customers to be covered. From these plots, it can be clearly seen that tour length increases when there is a higher percentage of demand (i.e., more number of customers need to be covered) even though total number of facilities and total number of customers remain the same.

6.4.2 Wilcoxon signed rank test

To rule out the possibility of attributing the better performance of our approach to random fluctuations, Wilcoxon signed rank test [94] has been used. The calculator available online [1] is used to perform the two tailed Wilcoxon signed rank test with significance criteria set to 5% (i.e. p-value $\leq$ 0.05). In this test, the difference between the normalized values of *'Average Obj.'* obtained by ABC and the compared approach is ranked. Table 6.6 summarizes the results of this test. In the Table 6.6, the '$NWT/Total$' denotes the number of instances without tie out of the total number of instances compared. The R^+ denotes the sum of ranks for the instances where ABC performed better than its competitor, whereas the R^- denotes the sum of ranks for the instances where ABC performed worse than its competitor. As the number of instances are more than thirty ($NWT > 30$), the test statistic Z is used to compare with the critical value Z_{Cri} according to Wilcoxon signed rank test [94]. If $Z \leq Z_{Cri}$, then there is a significant difference between the performance of the two compared approaches, otherwise the difference is not significant. Table 6.6 clearly shows that the better results of ABC are significant with respect to four other algorithms, viz. Flow-based formulation, Node-based formulation, MA and VNS with $Z \leq Z_{Cri}$ and $R^+ > R^-$.

Table 6.6: Results of Wilcoxon signed rank test between various approaches

	ABC vs...					
	$NWT/Total$	R^+	R^-	Z	Z_{Cri}	Significant
FBF	36/101	666	0	-5.232	-1.960	yes
NBF	39/101	780	0	-5.442	-1.960	yes
MA	52/143	1273	105	-5.318	-1.960	yes
VNS	52/143	1332	46	-5.856	-1.960	yes

6.5 Conclusions

In this chapter, we have proposed a swarm intelligence based approach, viz. an artificial bee colony algorithm for the generalized covering traveling salesman problem where we have incorporated a variable degree of perturbation strategy and redundant facility removal in neighboring

[1] `https://mathcracker.com/wilcoxon-signed-ranks.php`

solution generation. We have evaluated and compared our proposed approach with the state-of-the-art approaches on the benchmark instances available in the literature. Computational results on these benchmark instances show the effectiveness of our approach over all the other state-of-the-art approaches in terms of solution quality.

Chapter 7

Colored Traveling Salesman Problem

7.1 Introduction

The multiple traveling salesman problem (MTSP) is a well-known TSP variant where one has to find the tours for multiple salesmen instead of one. Given a set of n cities, and m salesmen, the MTSP is to find the tours for all m salesmen though each city must be visited exactly once by only one salesman and the sumtotal of tour lengths of all m salesmen is minimized. In the MTSP, any city can be accessed by any salesman, i.e., there is no restriction that a particular city has to be visited by a particular salesman only. The colored traveling salesman problem (CTSP) is another TSP variant that is much similar to the MTSP as it also involves multiple salesmen and has the same objective as the MTSP. However, in the CTSP, there are m different colored salesmen, and each salesman have access to his own colored cities, and there is a shared city set that can be accessed by different salesmen though each city must be visited exactly once by only one salesman. Both the CTSP and the MTSP, being generalization of the TSP are $\mathcal{NP}$-hard. Li *et al.* [2] highlighted the limitation of the MTSP in modelling the scheduling of a typical multi-machine engineering system (MES), where the individual machines have access to their exclusive workspaces, and some workspace may partially overlap with each other. Thus, each machine has to perform not only the operations independently in its exclusive workspace, but also complete all the tasks with other machine(s) together in the overlapping workspace. To get rid of this limitation of the MTSP, Li *et al.* [2] proposed the CTSP.

Another TSP variant that is closely related to MTSP and CTSP is vehicle routing problem (VRP). The VRP consists in finding the optimal set of routes for a fleet of vehicles in order to deliver the goods to a given set of customers subject to capacity restrictions on each vehicle.

There are many variants of the VRP in the literature. Unlike the VRP, in the CTSP, salesmen do not have capacity restrictions on the amount of goods that they can carry, but have the constraints of color. There exists a vast literature on VRP and its variants . For a survey on VRP and its variants, the readers are referred to [133, 134, 135, 136, 137, 138].

As the CTSP is recently introduced, only a few studies exist in literature on the CTSP. Li *et al.* [139] called the CTSP as the MTSP* and presented a solution based on a genetic algorithm (GA). Later Li *et al.* [2] formulated the CTSP in a mathematically rigorous way after providing the necessary background for introducing the CTSP and proposed three GAs, viz. GA with greedy initialization (GAG), hill-climbing GA (HCGA), and simulated annealing GA (SAGA). Comparison of the results suggested that SAGA can achieve best solution quality in computational time comparable to other GAs. Earlier versions of GA, GAG and HCGA can be found in [140]. Li *et al.* [141] presented a decomposition approach for the CTSP, where the CTSP is solved by decomposing it into several TSPs and one MTSP. Li *et al.* [142] investigated a CTSP model for planning dual-bridge waterjet cutting paths under a situation where individual bridges must handle their own exclusive jobs, and cooperate with one another to handle some shared jobs. They proposed a hybrid approach combining genetic algorithm with a simulated annealing operation to solve this problem. Li *et al.* [143] investigated the job scheduling and coordination problems in multi-bridge machining systems (MBMS). They proposed a serial colored traveling salesman problem (S-CTSP) and modeled the job scheduling and coordination problems in MBMS as S-CTSP. Further, they proposed two approaches based on a greedy algorithm and evaluated them on a triple-bridge waterjet cutting system.

The CTSP is quite complex problem in comparison to the TSP. Unlike the TSP, which is a permutation problem, the CTSP has aspects of grouping and permutation problems both. For solving the CTSP, not only cities need to be divided into different groups, but need to be arranged in a suitable order within each group. In this chapter, we have proposed a new approach for the CTSP, which is based on the artificial bee colony (ABC) algorithm. Our ABC approach is designed keeping in mind this dual nature of the CTSP. The performance of the proposed approach is compared with the state-of-the-art approaches. Computational results clearly show the superiority of our proposed approach over existing approaches in terms of solution quality and execution times both.

The rest of this chapter is organized as follows: Section 7.2 provides the formal definition of the CTSP. Our proposed ABC approach to the CTSP is presented in Section 7.3. Section 7.4 presents the computational results and their analysis. Finally, Section 7.5 concludes the chapter.

7.2 Problem definition

Suppose there are n cities, and $m < n$ salesmen, where $n, m \in Z_+$. Consider a complete graph $G = (V, E)$, where vertices are numbered from 0 to $n - 1$. Each edge $(i, j) \in E, i \neq j$ has an associated weight w_{ij} that represents the distance between the two cities i and j. There are $m + 1$ disjoint non-empty sets, viz. m exclusive city sets $C_1, C_2, ..., C_m$, and a shared city set S such that $S \cup C_1 \cup C_2 \cup \cdots \cup C_m = V$. A vertex $d_i \in C_i \cup S$ represents depot for the salesman i where it starts and ends its tour. Salesman i is of color $i \in \{1, 2, ..., m\}$. Each city $a \in C_i$ has color i, and can only be visited by salesman i. Each city $a \in S$, has color set $col(a) \subseteq \{1, 2, ..., m\}$, and $1 < |col(a)| \leq m$. If $i \in col(a)$, then a can be visited by salesman i. Usually, $|col(a)| = m$, $\forall a \in S$, i.e., S is shared among all m salesmen. The CTSP seeks m tours with minimum sumtotal of tour costs such that all vertices of each exclusive set must be visited exactly once by a particular salesman and all vertices of the shared set must be visited exactly once by one of the salesmen.

The composition of exclusive and shared city sets can vary in the CTSP, and accordingly the CTSP may reduce into different problems.

- If for each salesman i, C_i has only one city which is also the depot for salesman i and all remaining cities are in S, then the CTSP reduces to the multi-depot MTSP.
- If $|S| = 1$ such that the single city in S is common depot for all salesmen then the CTSP reduces to multiple independent TSPs. If we waive the requirement of S to be non-empty and take $S = \emptyset$ then also the CTSP reduces to multiple independent TSPs.
- If we waive the requirement of each exclusive city set to be non-empty and take $C_i = \emptyset$, $\forall i \in \{1, 2, ..., m\}$ and $S = V = \{0, 1, ..., n - 1\}$, and a common single depot for all m salesmen, then the CTSP reduces to the single depot MTSP.

7.3 Artificial bee colony algorithm for the CTSP

We have developed an artificial bee colony algorithm based approach for the CTSP. Subsequent subsections describe the salient features of our ABC approach for the CTSP.

1^{st} salesman tour	1	2	7	8
2^{nd} salesman tour	3	4	9	
3^{rd} salesman tour	5	6	10	

(a) A CTSP solution represented using m-tour encoding

city chromosome	1	2	7	8	3	4	9	5	6	10
salesman chromosome	1	1	1	1	2	2	2	3	3	3

(b) One way of representing the same solution as in Figure 7.1(a) using dual-chromosome encoding for the CTSP

city chromosome	5	6	10	3	4	9	1	2	7	8
salesman chromosome	3	3	3	2	2	2	1	1	1	1

(c) Another way of representing the same solution as in Figure 7.1(a) using dual-chromosome encoding for the CTSP

city chromosome	1	2	3	4	5	6	7	8	9	10
salesman chromosome	1	1	2	2	3	3	1	1	2	3

(d) One more way of representing the same solution as in Figure 7.1(a) using dual-chromosome encoding for the CTSP.

Figure 7.1: Illustrating m-tour encoding and dual-chromosome encoding

7.3.1 Solution encoding

There are several encoding schemes available in literature for the TSP and the MTSP [144, 145, 146, 147, 148, 149]. Due to the complex nature and city visit requirement of the CTSP, the single chromosome [146] and two-part chromosome representations [148] are not appropriate [2]. Li *et al.* [2] used dual-chromosome scheme (also known as two chromosome scheme) [144, 147] to represent a solution for the CTSP. The two chromosomes are city and salesman chromosomes having individual length N (excluding depots). The city chromosome has a permutation of N cities, whereas the salesman chromosome dictates which city is assigned to which salesman. We have used a different encoding, which we call m-tour encoding, to represent a solution for the CTSP. This encoding can be considered as a natural extension of encoding scheme used in [149] for the single depot MTSP. However, [149] did not present any theoretical analysis regarding the size of the search space resulting from the encoding used. This encoding represents a solution as an ordered list of m tours, i.e., every tour is associated with a salesman, and there is an ordering among the tours so that the tour associated with a particular salesman can be identified. Each tour is represented as a linear permutation of its constituent cities excluding depot. The depots (common or individual) of salesmen are not included in the tours, but they are included into the final solution. In reality, a tour is a circular permutation including depot. Excluding depots from the tours enables the use of a linear permutation to represent a circular permutation without increasing the size of the solution space for an individual tour. Suppose $\forall i \in \{1, 2, ..., m\}$, if there are n_i cities (excluding depot) to be visited by the salesman i, then its associated tour is some linear permutation of n_i cities and the size of the search space for this tour is $n_i!$ with this encoding. However, actual search space is $\frac{n_i!}{2}$ in case distance matrix is symmetric, whereas in case distance matrix is asymmetric, size of search space is $n_i!$. Therefore, some inherent redundancy is there in any permutation based representation of the CTSP.

Consider a CTSP instance with $n = 11$, $m = 3$, $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$, $C_3 = \{5, 6\}$, $S = \{0, 7, 8, 9, 10\}$ and city 0 is depot for all salesmen. Figure 7.1 illustrates m-tour encoding and dual-chromosome encoding. A solution for this CTSP instance in m-tour representation is shown in Figure 7.1(a). Cities $(1, 2, 7, 8), (3, 4, 9), (5, 6, 10)$ are visited by salesmen $1, 2, 3$ respectively in that order in addition to the depot 0. The same solution is represented in Figure 7.1(b), 7.1(c), and 7.1(d) by using dual-chromosome encoding. Thus the dual-chromosome representation suffers from the problem of redundancy, i.e., different chromosomes represent the same CTSP solution. Because of the redundancy, the solution space (space of all possible solution representations) becomes larger than the problem space (space of all feasible solutions to the problem). As any approach has to work in solution space instead of problem space, in the presence of redundancy, it has to search a larger space. This can adversely affect the performance of the approach.

It is interesting to note that we can use m-tour encoding without any modification to represent a solution of the MTSP also. If each salesman starts and ends their tour in a depot of its own then m-tours has to be ordered so that the tour corresponding to each depot can be identified. However, if all salesmen start and end their tours in the same depot then even the requirement of m-tours to form an ordered list can be dispensed with, and instead a set of m-tours can be used to represent a solution.

Theorem 1: The solution space size of the MTSP with m-tour encoding is smaller than that of dual-chromosome encoding.

Proof: For a MTSP with N cities (excluding depot(s)) and m salesmen, the solution space size for dual-chromosome encoding is $N!m^N$ as given in [148]. Please note that every salesman has to visit at least one city in addition to the depot and there are ${}^{(N-1)}C_{(m-1)}$ distinct positive integer-valued vectors of length m with sum exactly equal to N ([150], page 13). Let us denote each such vector by X_i, $i = 1 \ldots {}^{(N-1)}C_{(m-1)}$ and its j^{th} element by $X_i(j)$, $j = 1 \ldots m$. $\forall i \in \{1, 2, ..., m\}$ if there are n_i cities to be visited by the salesman i such that $\sum_{i=1}^{m} n_i = N$, then the solution space size by using m-tour encoding in case each salesman starts and ends their tour in a depot of its own is

$$\sum_{i=1}^{{}^{(N-1)}C_{(m-1)}} \left(\prod_{j=1}^{m} \left(\left({}^{(N-\sum_{k=1}^{j-1} X_i(k))}C_{(X_i(j))} \right) X_i(j)! \right) \right)$$

which simplifies to

$${}^{(N-1)}C_{(m-1)}N!$$

as each term in the summation of first expression is equivalent to $N!$. So the size of solution space is exactly the same as two-part encoding proposed in [148].

The size of solution space in the case of common single depot for all salesmen as there is no ordering among tours is

$$\frac{{}^{(N-1)}C_{(m-1)}N!}{m!}$$

This is so because m given tours can be arranged in $m!$ different ways and when there is no ordering among tours, each of these $m!$ arrangements represents the same MTSP solution. Clearly, in this case size of the solution space is less than two-part encoding. The $N!$ term in the last two expressions can be $\frac{N!}{2}$ in case the distance matrix is symmetric.

It has already been shown in [148] that the size of the solution space using two-part encoding is less than that of dual-chromosome encoding. Same holds for m-tour encoding in comparison to dual-chromosome encoding as the size of solution with m-tour encoding is exactly the same as two-part encoding in case of multi-depot case and less than that of two-part encoding in case of common single depot.

$\square$

Theorem 2: The solution space size of the CTSP with m-tour encoding is smaller than that of dual-chromosome encoding.

Proof: For a CTSP with N cities (excluding depot(s)), m salesmen, n_0 shared cities, and $\forall i \in \{1, 2, ..., m\}$ if there are e_i exclusive cities to be visited by the salesman i such that $\sum_{i=1}^{m} e_i + n_0 = N$, then the solution space size by dual-chromosome encoding is $N!m^{n_0}$ as shown in [2]. Please note that a salesman may not visit any of the shared cities and there are ${}^{(n_0+m-1)}C_{(n_0)}$ distinct non-negative integer-valued vectors of length m with sum exactly equal to n_0 ([150], page 13). Let us denote each such vector by Y_i, $i = 1 \ldots {}^{(n_0+m-1)}C_{(n_0)}$ and its j^{th} element by $Y_i(j)$, $j = 1 \ldots m$. So, the solution space size by using m-tour encoding is

$$\sum_{i=1}^{{}^{(n_0+m-1)}C_{(n_0)}} \left(\prod_{j=1}^{m} \left(\left({}^{(n_0 - \sum_{k=1}^{j-1} Y_i(k))} C_{(Y_i(j))} \right) (e_j + Y_i(j))! \right) \right)$$

which simplifies to

$$\sum_{i=1}^{{}^{(n_0+m-1)}C_{(n_0)}} \left(n_0! \prod_{j=1}^{m} \left(\frac{(e_j+Y_i(j))!}{Y_i(j)!} \right) \right)$$

In the above expression, the product of numerator terms is product of factorials of m positive integers whose sum is equal to N and product of denominator terms is product of factorials of m non-negative integers whose sum is equal to n_0. So the product of numerator terms is less than $N!$ and product of denominator terms is less than or equal to $n_0!$. As $n_0 < N$ and each numerator term ($(e_j + Y_i(j))!$) is greater than or equal to the corresponding denominator term ($Y_i(j)!$), the whole product term in the above expression is less than $\frac{N!}{n_0!}$. Therefore, the above expression is less than

$$\sum_{i=1}^{{}^{(n_0+m-1)}C_{(n_0)}} N!$$

Now, the last expression can be further simplified to

$${}^{(n_0+m-1)}C_{(n_0)} N!$$

So to show that solution space with m-tour encoding is less than dual-chromosome encoding, it is enough to show that ${}^{(n_0+m-1)}C_{(n_0)}$ is less than or equal to m^{n_0}. We will prove this by induction on n_0. If $n_0 = 1$ then ${}^{(n_0+m-1)}C_{(n_0)} = m$ and $m^{n_0} = m$, so basis step is complete. Let us assume that result holds for $n_0 = k$, i.e., $m^k \geq^{(k+m-1)} C_{(k)}$. Obviously, $m(k+1) \geq k+m$ or $m \geq \frac{k+m}{k+1}$, so $mm^k \geq \frac{k+m}{k+1}{}^{(k+m-1)}C_{(k)}$, which is nothing but $m^{k+1} \geq^{(k+1+m-1)} C_{(k+1)}$. This completes the induction step. Hence by principal of mathematical induction $m^{n_0} \geq^{(n_0+m-1)} C_{(n_0)}$. This then proves that the solution space size of the CTSP by using m-tour encoding, is smaller than that of dual-chromosome coding. Please note that $(e_j + Y_i(j))!$ term in the first expression of this theorem can be $\frac{(e_j+Y_i(j))!}{2}$ in case the distance matrix is symmetric.

$\square$

Corollaries: Following corollaries can be easily derived from the previous two theorems:

- The solution space size of the multi-depot MTSP is at least as large as that of the multi-depot CTSP
- The solution space size of the common single depot MTSP in comparison to the common single depot CTSP depends on the proportion of shared cities with respect to all cities in case of the CTSP. If this proportion is nearly 1 then solution space of the single depot MTSP is less than that of the single depot CTSP. If this proportion is near to 0 then solution space of the single depot MTSP is grater than that of the single depot CTSP.

7.3.2 Fitness

The fitness of each solution of the CTSP is the sumtotal of the distances traveled by all the salesmen, i.e., the objective function is itself the fitness function. Hence, a solution with lower value of the fitness function is more fit than a solution with higher value of the fitness function.

7.3.3 Initialization

Each initial solution is generated in an iterative manner. Assuming U to be the set of unassigned cities. Initially, every city is unassigned so $U = V$, i.e., U is initialized to the set containing all the cities. Then an iterative process ensues. Each iteration starts by selecting a city $a \in U$ uniformly at random, and then it is checked whether this city is a colored city or a shared city. If a is colored with $col(a) = i$, then it is inserted into the tour of salesman i at the best place. By best place, we mean the place where inserting the city increases the tour cost by least amount. For finding this best place, all possible insertion positions in the tour of salesman i have to be checked. And if a is a shared city, then it is inserted into the tour of a salesman at the place where cost increases least by this insertion. For finding this place, all possible insertion positions in every salesman's tour need to be checked. After assigning city a to one of the salesmen tours, that city is removed from the set U and another iteration begins. This process is repeated until the set U becomes empty.

Each generated initial solution is uniquely associated with an employed bee, and fitness value of each solution is calculated.

7.3.4 Mechanism for choosing a food source

Binary tournament selection method (Section 5.3.4) is used for choosing an employed bee solution for an onlooker bee, where the probability of selection of the better solution is P_{bts}.

7.3.5 Determination of a neighboring food source

An effective neighboring solution generation process should exploit the problem's aspects in such a manner so that the search process is guided towards optimal. Our neighboring solution generation process is designed keeping in mind the grouping and the permutation nature of the CTSP. Two methods INTER and INTRA have been devised for neighboring solution generation. These two methods have been used one after the other, i.e., INTER is used after INTRA. In both these methods, in order to generate a neighboring solution S' of solution S, each city existing in a tour of S is copied to the corresponding tour of S' with probability P_{cp}. The remaining cities constitute a set of unassigned cities. INTRA and INTER differ in the manner these unassigned cities are assigned to the tours.

INTRA assigns unassigned cities to the same tour from which they got deleted. To implement this operation efficiently, for each tour we maintain a set of cities deleted from it. Each tour is considered one-by-one. While considering a tour, an unassigned city is chosen at random from the corresponding set of unassigned cities and inserted at the best possible place in the tour in the same manner as done for colored cities in Section 7.3.3. Once all the cities from the set of unassigned cities got assigned to the tour in this manner, resulting cost of the tour is compared with the cost of the original tour before deleting the cities. If this cost is better then tour configuration is retained, otherwise it is restored to the original configuration before deleting the cities. After this the next tour is considered. Clearly, INTRA makes use of only the permutation nature of the problem.

INTER assigns each unassigned city to the best possible tour at the best possible position while respecting all the constraints. INTER considers each city in the set of unassigned cities one-by-one in some random order and the city under consideration is assigned to a tour using the same procedure as used in the Section 7.3.3. Hence, this method makes use of both the grouping and the permutation nature of the problem, and as a result, it is costlier than INTRA.

Like the previous two chapters, here also the degree of perturbation varies over iterations. The parameter P_{cp} controls the degree of perturbation. However, the way P_{cp} is defined, the degree of perturbation is $1 - P_{cp}$. Hence, the value of P_{cp} varies over iterations from P_{mincp} to P_{maxcp}. The value of P_{cp} in an iteration $iter$ is computed using the following formula:

$$P_{cp} := \begin{cases} \left(\frac{P_{maxcp}-P_{mincp}}{iter_{max}}\right)(iter) + P_{mincp} & \text{if } iter \leq iter_{max} \\ P_{maxcp} & \text{otherwise} \end{cases}$$

Here $iter_{max}$ is maximum number of iterations upto which P_{cp} can be varied. Beyond $iter_{max}$, P_{cp} assumes the constant value P_{maxcp}. The pseudo-code for generating a neighboring solution can be seen in Algorithm 20.

7.3.6 Other features

Like the previous two chapters, we have used different number of employed bees and onlooker bees. If an employed bee solution does not improve over a fixed number of trials $limit_{scout}$, then that employed bee abandons that food source and becomes a scout. However, unlike previous two chapters, a scout bee is again turned into an employed bee by generating a neighboring solution of the neighboring solution of just abandoned solution and associating it with the scout bee under consideration. We have observed that in case of CTSP when we assign scout bee a neighboring solution of just abandoned solution, then from that neighboring solution, it often jumps to the abandoned solution.

The pseudo-code of our ABC approach is given in Algorithm 21, where n_{eb} and n_{ob} are, respectively, the number of employed and onlooker bees. *Generate_Neighboring_Solution(S)* function returns a solution in the neighborhood of the solution S as per the procedure described in Section 7.3.5. *Select_Solution($S_1, S_2, \ldots, S_{n_{eb}}$)* function selects a solution from solutions $S_1, S_2, \ldots, S_{n_{eb}}$ for an onlooker bee using binary tournament selection method (Section 7.3.4) and returns the solution selected.

Algorithm 20: Neighboring solution generation

```
Input: A solution $S$
Output: A neighboring solution $S'$
begin
    $S' \leftarrow \emptyset$;
    foreach tour $t$ in $S$ do
        foreach city $c$ in $t$ as per their order in $t$ do
            Generate a random number $r$ such that $0 \leq r \leq 1$;
            if $r < P_{cp}$ then
                copy $c$ to tour $t$ in $S'$;
            else
                Add $c$ to set of unassigned cities;
    // INTRA method
    foreach unassigned city $c$ in the set of unassigned cities in some random order do
        Insert $c$ into $S'$ using the INTRA method;
    $S \leftarrow S'$;
    $S' \leftarrow \emptyset$;
    foreach tour $t$ in $S$ do
        foreach city $c$ in $t$ as per their order in $t$ do
            Generate a random number $r$ such that $0 \leq r \leq 1$;
            if $r < P_{cp}$ then
                copy $c$ to tour $t$ in $S'$;
            else
                Add $c$ to set of unassigned cities;
    // INTER method
    foreach unassigned city $c$ in the set of unassigned cities in some random order do
        Insert $c$ into $S'$ using the INTER method;
    return $S'$;
```

7.4 Computational results

For testing our ABC approach, we have used the same test-bed as used in [2]. Li *et al.* [2] used 21 instances to test the performance of their approaches. These instances have cities ranging from 21 to 101. In addition, we have also used 8 more instances of large size from the standard TSPLIB leading to a total of 29 instances. These instances have upto 666 cities. These large instances are available publicly for download[1]. All the 29 instances have symmetric $n \times n$ distance matrix. In these instances, first city is common depot for all salesmen. Table 7.1 lists the characteristics of these instances. Instances numbered from 1 to 10, 11 to 21, 22 to 29 are categorized as small, medium, and large instances respectively.

[1]`scis.uohyd.ac.in/~alokcs/ctsp_new.zip`

Algorithm 21: Pseudo code of our ABC approach

```
Input: Set of parameters for the ABC Algorithm and a CTSP instance
Output: Best solution found
generate $n_{eb}$ random solutions $S_1, S_2, \ldots, S_{n_{eb}}$;
$best$ := best solution among $S_1, S_2, \ldots, S_{n_{eb}}$;
while Termination condition not satisfied do
    for $i := 1$ to $n_{eb}$ do
        $S'$ := Generate_Neighboring_Solution($S_i$);
        if $S'$ is better than $S_i$ then
            $S_i := S'$;
    for $i := 1$ to $n_{ob}$ do
        $S_p :=$ Select_Solution($S_1, S_2, \ldots, S_{n_{eb}}$);
        $S' :=$ Generate_Neighboring_Solution($S_p$);
        if $S'$ is better than $S_p$ then
            $S_p := S'$;
    for $i := 1$ to $n_{eb}$ do
        if $S_i$ is better than $best$ then
            $best := S_i$;
    Find the solution $S_{scout}$ that has not improved over maximum number of trials ;
    if $S_{scout}$ has not improved for at least $limit_{scout}$ trials then
        replace $S_{scout}$ with a neighboring solution of the neighboring solution;
return $best$;
```

We have implemented our ABC approach in C and executed it on a Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM. To show that superior performance of our ABC approach is not due to the way an initial solution is generated, we have implemented another version of our ABC approach where an initial solution is generated in a purely random manner, i.e., cities are inserted randomly while maintaining the constraint imposed by colored cities. If a city is shared, it is inserted at a random position in a randomly chosen tour. If a city is colored, it is inserted into its specific tour at a random position. This version will be referred to as RAND-ABC. Except for initialization, RAND-ABC is same as ABC in all other aspects.

For benchmarking, we have compared ABC and RAND-ABC with previously proposed genetic algorithm based approaches, viz. GA [139], GAG, HCGA and SAGA [2]. To ensure precise comparison of execution times and to ensure comparison on large instances which were not used in [2], we have re-implemented GA, GAG, HCGA and SAGA approaches of [2] and executed them on the same system as ABC and RAND-ABC for the same amount of time. While executing these approaches, we have utilized the same parameter values as reported in

Table 7.1: Test-bed for the CTSP.

Instance	City count	Salesman count	Shared city count	Exclusive city count
Small				
1	21	2	11	5 per salesman
2	21	3	9	4 per salesman
3	31	2	19	6 per salesman
4	31	3	16	5 per salesman
5	31	4	15	4 per salesman
6	41	2	21	10 per salesman
7	41	3	23	6 per salesman
8	41	4	21	5 per salesman
9	51	2	21	15 per salesman
10	51	3	21	10 per salesman
Medium				
11	51	3	21	10 per salesman
12	51	4	21	7, 7, 8 and 8
13	51	5	21	6 per salesman
14	76	3	31	15 per salesman
15	76	4	36	10 per salesman
16	76	5	26	10 per salesman
17	76	6	40	6 per salesman
18	101	4	21	20 per salesman
19	101	5	53	10 per salesman
20	101	6	41	10 per salesman
21	101	7	21	10 per salesman
Large				
22	229	10	79	15 per salesman
23	229	15	79	10 per salesman
24	229	20	69	8 per salesman
25	229	30	79	5 per salesman
26	666	10	266	40 per salesman
27	666	15	216	30 per salesman
28	666	20	266	20 per salesman
29	666	30	216	15 per salesman

[2]. We have performed two sets of experiments. In the first set of experiments, execution times are determined as per the convergence behavior of ABC and RAND-ABC, whereas in the latter set of experiments, it is determined as per the convergence behavior of GA, GAG, HCGA and SAGA. Actually, ABC and RAND-ABC converge very fast, whereas GA, GAG, HCGA and SAGA require some time to converge. Therefore, we have performed two sets of experiments to

Table 7.2: Candidate parameter values used in parameter settings of ABC

ABC	
Parameter	Value
n_{eb}	$\in \{75, 100, 125\}$
n_{ob}	$\in \{125, 150, 175\}$
$limit_{scout}$	$\in \{75, 100, 125\}$
P_{bts}	$\in \{0.7, 0.8, 0.9\}$
P_{mincp}	$\in \{0, 0.1, 0.2\}$
P_{maxcp}	$\in \{0.7, 0.8, 0.9\}$
$iter_{max}$	$\in \{75, 100, 125\}$

show the superiority of ABC and RAND-ABC under both the termination conditions. In the first set of experiments, all the approaches were executed for 1 second, whereas in the latter set of experiments all the approaches were executed for 1, 5 and 60 seconds for small, medium, and large instances respectively. In both sets of experiments, all the approaches were executed on each test instance ten times independently like Li *et al.* [2].

7.4.1 Parameter settings

Since ABC is a stochastic approach, therefore, selection of appropriate parameter values is vital for its success. In order to set different parameters of ABC, we have followed the Taguchi method [131, 132]. There are seven parameters in our ABC approach. We have identified three candidate parameter values for each parameter. These potential candidate parameter values are shown in Table 7.2. These candidate parameter values are chosen based on available literature, our own previous experience with artificial bee colony algorithm, and some preliminary experimentations. As per Taguchi method, for determining the value of seven parameters each having three candidate values, at least 18 sets of experiments involving various combinations of candidate parameter values need to be done. The combination of parameter values for each of these set of experiments has to be chosen according to the rows of $L18$ orthogonal array [131]. Based on the these experiments, following parameter values have been used in subsequent experiments: We have used a population of 100 employed bees ($n_{eb} = 100$), 150 onlooker bees ($n_{ob} = 150$), $P_{bts} = 0.7$, $limit_{scout} = 125$ and the value of P_{cp} can be varied upto 125 iterations ($iter_{max} = 125$). We have varied the P_{cp} from 0.1 to 0.7, i.e. $P_{mincp} = 0.1$ and $P_{maxcp} = 0.7$.

7.4.2 Comparison of our approaches with approaches of [2]

Table 7.3 and Table 7.4 compare the performance of ABC and RAND-ABC with different approaches proposed in [2] on each of the 29 test problems. Table 7.3 reports the results of first set of experiments having the termination condition as 1 second for all instances, whereas Table 7.4 reports the results of second set of experiments having the termination condition as 1, 5, 60 seconds for small, medium, and large instances respectively. Li *et al.* [2] reported the solution obtained by solving the integer programming formulation of the CTSP through LINGO on small instances. Though it does not make much sense to compare the metaheuristic approaches with an exact method like LINGO, we can always utilize LINGO results for verifying the optimality of the solutions obtained through metaheuristic approaches. Further, the time taken by LINGO to find an optimal solution also provides an idea about the difficulty of the problem from an exact method perspective. Because of these two reasons, we have included LINGO results in these tables which are taken directly from [2]. Further, we have found that the reported results of LINGO are not optimal and discussed about it in subsequent paragraphs. Li *et al.* [2] reported the results of LINGO for those instances only for which they got the solution within 2 days. For LINGO, we have reported the solution obtained (if any) (column $Distance$) and execution time in seconds (column $Time$) only. On the other hand, for GA, GAG, HCGA, SAGA, ABC and RAND-ABC approaches, we have reported the $Best$, $Worst$ and $Mean$ solutions obtained over 10 runs. For each of these three categories of solutions ($Best$, $Worst$ and $Mean$), the best results over all the approaches are shown in bold font in these tables. In addition, for each of these six methods, the average number of iterations (column $iter$) in allotted time is also reported. These tables clearly show the superiority of ABC and RAND-ABC in terms of solution quality and execution time both. Our approaches obtained solutions of better quality in much shorter time. Claim regarding execution time is justified as the results obtained by our approaches in Table 7.3 are better than the results obtained by GA, GAG, HCGA and SAGA in Table 7.4.

If we compare the results of ABC with those obtained by RAND-ABC in Table 7.3 and Table 7.4, it is clear that initialization has little impact and that too only on large instances on the final solution obtained. As can be seen in these tables, ABC and RAND-ABC approaches obtained the same solution in all 10 runs for small and medium instances except for instance numbered 18 in both sets of experiments and hence, the values of best, mean and worst solutions

Table 7.3: Termination condition of 1 sec for all instances.

Instance	n	m	LINGO		GA				GAG				HCGA				SAGA				RAND-ABC				ABC			
			Distance	Time	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter
Small																												
1	21	2	**144.92**	26	148.44	160.89	154.72	46751	149.37	151.98	150.70	47195	150.38	154.87	152.00	17416	**144.92**	**144.92**	**144.92**	1444	**144.92**	**144.92**	**144.92**	4436	**144.92**	**144.92**	**144.92**	4479
2	21	3	**157.48**	48	**157.48**	193.57	166.41	45109	**157.48**	176.00	163.41	43715	164.65	184.17	170.20	11438	**157.48**	**157.48**	**157.48**	1190	**157.48**	**157.48**	**157.48**	4508	**157.48**	**157.48**	**157.48**	4480
3	31	2	**259.36**	94	278.71	360.81	311.03	35748	268.71	305.12	290.73	33440	282.63	313.19	296.86	12870	269.89	277.41	273.76	935	**259.36**	**259.36**	**259.36**	2523	**259.36**	**259.36**	**259.36**	2505
4	31	3	**295.31**	900	305.60	333.91	316.05	32383	298.68	342.63	320.89	32532	311.39	359.94	331.30	8542	296.42	305.95	302.44	952	**295.31**	**295.31**	**295.31**	2588	**295.31**	**295.31**	**295.31**	2571
5	31	4	316.06	5400	320.14	333.99	328.66	31276	322.89	351.40	331.63	30804	326.90	363.76	340.23	6095	317.30	322.07	319.70	834	**315.97**	**315.97**	**315.97**	2597	**315.97**	**315.97**	**315.97**	2633
6	41	2	355.69	23040	379.44	454.22	417.64	26807	384.83	411.01	393.84	26452	381.16	424.71	397.84	7993	393.65	409.22	398.94	913	**346.24**	**346.24**	**346.24**	1639	**346.24**	**346.24**	**346.24**	1655
7	41	3	371.89	65880	397.58	473.64	428.14	26018	411.25	472.56	444.14	25624	414.17	472.63	444.83	6899	404.20	419.74	410.29	691	**367.84**	**367.84**	**367.84**	1641	**367.84**	**367.84**	**367.84**	1652
8	41	4	N/A	N/A	417.54	458.63	434.42	23839	420.29	472.19	444.78	24438	444.46	504.46	454.99	4883	409.20	428.55	420.37	696	**392.14**	**392.14**	**392.14**	1699	**392.14**	**392.14**	**392.14**	1702
9	51	2	N/A	N/A	559.38	654.11	596.74	21187	509.20	555.97	526.76	20628	507.73	552.13	529.60	8198	537.01	595.95	568.66	774	**478.08**	**478.08**	**478.08**	1202	**478.08**	**478.08**	**478.08**	1226
10	51	3	N/A	N/A	511.86	584.81	553.54	19913	491.52	572.71	521.89	19789	508.85	574.28	542.96	4576	547.54	561.64	555.70	642	**469.50**	**469.50**	**469.50**	1287	**469.50**	**469.50**	**469.50**	1299
Medium																												
11	51	3	-	-	509.38	578.64	546.57	15761	518.58	551.71	532.67	15573	510.17	574.97	537.27	5237	555.26	587.28	572.51	721	**469.50**	**469.50**	**469.50**	1281	**469.50**	**469.50**	**469.50**	1281
12	51	4	-	-	517.18	586.51	551.30	14743	535.03	607.80	562.96	14477	531.02	599.76	572.22	3159	561.29	582.13	574.18	654	**489.99**	**489.99**	**489.99**	1365	**489.99**	**489.99**	**489.99**	1372
13	51	5	-	-	541.72	591.32	568.72	13721	546.27	608.79	570.58	13140	549.69	634.51	590.97	2871	583.97	606.89	595.87	518	**525.98**	**525.98**	**525.98**	1365	**525.98**	**525.98**	**525.98**	1359
14	76	3	-	-	785.37	867.03	815.91	10949	682.08	789.01	738.51	10797	712.23	752.26	731.18	3644	759.81	795.78	780.36	524	**593.28**	**593.28**	**593.28**	652	**593.28**	**593.28**	**593.28**	658
15	76	4	-	-	740.46	864.93	801.52	10574	723.76	809.08	759.16	10326	719.62	809.04	768.68	2687	823.53	858.54	844.15	472	**603.79**	**603.79**	**603.79**	643	**603.79**	**603.79**	**603.79**	645
16	76	5	-	-	759.97	821.84	790.21	9579	719.81	792.75	773.84	9251	775.33	852.83	816.07	1992	811.45	869.53	848.28	410	**651.99**	**651.99**	**651.99**	761	**651.99**	**651.99**	**651.99**	754
17	76	6	-	-	820.15	909.59	856.93	9190	774.46	899.63	842.38	9209	858.18	932.36	896.75	1576	921.29	948.55	930.09	384	**672.73**	**672.73**	**672.73**	599	**672.73**	**672.73**	**672.73**	602
18	101	4	-	-	946.47	1087.70	1030.56	7593	844.91	924.53	892.02	7268	802.13	877.29	851.60	2033	933.59	961.77	950.07	351	**726.82**	**728.01**	**727.57**	524	**726.82**	**728.01**	727.73	525
19	101	5	-	-	1076.74	1187.05	1140.37	7644	1035.39	1183.49	1089.65	7456	1102.08	1166.20	1119.77	1455	1191.25	1247.71	1216.81	329	**779.15**	**779.15**	**779.15**	354	**779.15**	**779.15**	**779.15**	356
20	101	6	-	-	966.40	1123.16	1028.76	6963	961.19	1104.86	1021.05	6843	1013.65	1104.03	1058.25	1234	1110.55	1154.18	1128.35	290	**759.55**	**759.55**	**759.55**	428	**759.55**	**759.55**	**759.55**	431
21	101	7	-	-	999.14	1069.10	1035.66	6392	956.10	1029.22	998.82	6224	994.45	1089.78	1039.71	965	1077.16	1114.98	1095.31	255	**798.85**	**798.85**	**798.85**	508	**798.85**	**798.85**	**798.85**	513
Large																												
22	229	10	-	-	565124.00	634691.00	603977.80	2423	416277.00	441125.00	428059.60	2309	410224.00	432574.00	420931.50	246	320587.00	347764.00	338865.50	98	223632.00	225189.00	224297.90	70	**223359.00**	**224265.00**	**223836.90**	62
23	229	15	-	-	610663.00	676775.00	648840.60	2005	508116.00	550784.00	530422.30	1906	526517.00	550201.00	538016.50	124	379482.00	424706.00	405952.60	71	265842.00	267290.00	266677.80	68	**265205.00**	**266733.00**	**266154.90**	61
24	229	20	-	-	630220.00	695149.00	665540.90	1657	584525.00	615003.00	602009.40	1609	586741.00	624936.00	606885.50	77	431792.00	486648.00	458452.80	56	320541.00	**321825.00**	321262.60	71	**320010.00**	322306.00	**321028.30**	65
25	229	30	-	-	782092.00	837051.00	814308.50	1282	721737.00	791847.00	763015.20	1268	763635.00	820069.00	803882.50	37	550217.00	599689.00	569106.00	40	408076.00	409260.00	408640.30	63	**407154.00**	**408597.00**	**408018.90**	54
26	666	10	-	-	3084948.00	3350667.00	3224139.80	499	764045.00	804063.00	788548.00	626	747060.00	781800.00	768323.50	74	680151.00	725824.00	705796.30	31	**408034.00**	419314.00	414880.30	6	409924.00	**413483.00**	**411643.00**	6
27	666	15	-	-	3015053.00	3191828.00	3079307.00	471	916956.00	952118.00	940496.00	583	888932.00	923121.00	909046.20	41	824978.00	861522.00	841683.10	23	464952.00	475867.00	470787.20	7	**461740.00**	**471176.00**	**466757.60**	8
28	666	20	-	-	3073809.00	3343527.00	3184430.70	420	1106944.00	1129919.00	1118942.70	471	1074327.00	1095955.00	1087026.20	26	979916.00	999511.00	991724.10	19	542632.00	553063.00	545526.90	6	**537057.00**	**543850.00**	**539879.70**	6
29	666	30	-	-	3026767.00	3246605.00	3142572.80	345	1327860.00	1393886.00	1363615.10	374	1293208.00	1336973.00	1322972.60	13	1176090.00	1242411.00	1202727.80	14	667513.00	675507.00	672471.10	7	**664291.00**	**669050.00**	**666936.40**	7

Table 7.4: Termination condition of 1 sec for all instances.

Instance	n	m	LINGO		GA				GAG				HCGA				SAGA				RAND-ABC				ABC			
			Distance	Time	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter	Best	Worst	Mean	Iter
Small																												
1	21	2	**144.92**	26	148.44	160.89	154.72	46751	149.37	151.98	150.70	47195	150.38	154.87	152.00	17416	**144.92**	**144.92**	**144.92**	1444	**144.92**	**144.92**	**144.92**	4436	**144.92**	**144.92**	**144.92**	4479
2	21	3	**157.48**	48	**157.48**	193.57	166.41	45109	**157.48**	176.00	163.41	43715	164.65	184.17	170.20	11438	**157.48**	**157.48**	**157.48**	1190	**157.48**	**157.48**	**157.48**	4508	**157.48**	**157.48**	**157.48**	4480
3	31	2	**259.36**	94	278.71	360.81	311.03	35748	268.71	305.12	290.73	33440	282.63	313.19	296.86	12870	269.89	277.41	273.76	935	**259.36**	**259.36**	**259.36**	2523	**259.36**	**259.36**	**259.36**	2505
4	31	3	**295.31**	900	305.60	333.91	316.05	32383	298.68	342.63	320.89	32532	311.39	359.94	331.30	8542	296.42	305.95	302.44	952	**295.31**	**295.31**	**295.31**	2588	**295.31**	**295.31**	**295.31**	2571
5	31	4	316.06	5400	320.14	333.99	328.66	31276	322.89	351.40	331.63	30804	326.90	363.76	340.23	6095	317.30	322.07	319.70	834	**315.97**	**315.97**	**315.97**	2597	**315.97**	**315.97**	**315.97**	2633
6	41	2	355.69	23040	379.44	454.22	417.64	26807	384.83	411.01	393.84	26452	381.16	424.71	397.84	7993	393.65	409.22	398.94	913	**346.24**	**346.24**	**346.24**	1639	**346.24**	**346.24**	**346.24**	1655
7	41	3	371.89	65880	397.58	473.64	428.14	26018	411.25	472.56	444.14	25624	414.17	472.63	444.83	6899	404.20	419.74	410.29	691	**367.84**	**367.84**	**367.84**	1641	**367.84**	**367.84**	**367.84**	1652
8	41	4	N/A	N/A	417.54	458.63	434.42	23839	420.29	472.19	444.78	24438	444.46	504.46	454.99	4883	409.20	428.55	420.37	696	**392.14**	**392.14**	**392.14**	1699	**392.14**	**392.14**	**392.14**	1702
9	51	2	N/A	N/A	559.38	654.11	596.74	21187	509.20	555.97	526.76	20628	507.73	552.13	529.60	8198	537.01	595.95	568.66	774	**478.08**	**478.08**	**478.08**	1202	**478.08**	**478.08**	**478.08**	1226
10	51	3	N/A	N/A	511.86	584.81	553.54	19913	491.52	572.71	521.89	19789	508.85	574.28	542.96	4576	547.54	561.64	555.70	642	**469.50**	**469.50**	**469.50**	1287	**469.50**	**469.50**	**469.50**	1299
Medium																												
11	51	3	-	-	497.42	535.78	520.54	79308	495.38	539.94	519.27	78729	493.34	574.97	523.34	21395	524.00	582.17	560.76	3652	**469.50**	**469.50**	**469.50**	6677	**469.50**	**469.50**	**469.50**	6752
12	51	4	-	-	506.85	564.04	525.27	74985	507.31	554.73	527.99	72937	518.60	567.71	543.77	19229	555.74	575.10	561.51	3286	**489.99**	**489.99**	**489.99**	7106	**489.99**	**489.99**	**489.99**	7205
13	51	5	-	-	532.11	579.34	555.31	69483	536.34	570.92	555.05	68143	537.02	599.40	573.83	14510	565.84	592.36	583.45	2906	**525.98**	**525.98**	**525.98**	7034	**525.98**	**525.98**	**525.98**	7171
14	76	3	-	-	710.84	751.01	730.45	55043	680.19	726.55	700.45	55295	683.18	748.64	712.81	15767	759.81	795.78	780.36	2608	**593.28**	**593.28**	**593.28**	3511	**593.28**	**593.28**	**593.28**	3576
15	76	4	-	-	693.68	754.78	720.19	53414	702.21	757.93	716.15	52056	711.83	785.59	739.46	13303	823.53	858.54	843.98	2339	**603.79**	**603.79**	**603.79**	3500	**603.79**	**603.79**	**603.79**	3504
16	76	5	-	-	684.33	745.58	723.16	47791	686.88	756.72	729.36	47410	693.96	809.02	755.14	10021	811.45	853.07	838.14	2056	**651.99**	**651.99**	**651.99**	4077	**651.99**	**651.99**	**651.99**	4120
17	76	6	-	-	751.49	804.23	778.52	46893	755.97	831.67	789.70	47214	796.70	857.74	828.97	7959	895.25	928.05	913.62	1925	**672.73**	**672.73**	**672.73**	3275	**672.73**	**672.73**	**672.73**	3315
18	101	4	-	-	846.94	969.96	904.14	38744	833.78	872.20	852.05	36905	799.79	864.17	837.38	10269	933.59	961.77	950.07	1751	**726.82**	727.56	727.08	2865	**726.82**	**727.20**	**726.93**	2903
19	101	5	-	-	928.27	1067.12	1013.05	38611	938.17	1061.10	995.89	37851	962.74	1092.40	1048.77	7952	1191.25	1247.71	1216.81	1649	**779.15**	**779.15**	**779.15**	2040	**779.15**	**779.15**	**779.15**	2072
20	101	6	-	-	872.88	962.21	912.73	34959	887.53	1002.17	941.08	34437	970.60	1045.37	998.84	6195	1102.86	1140.62	1118.58	1461	**759.55**	**759.55**	**759.55**	2410	**759.55**	**759.55**	**759.55**	2445
21	101	7	-	-	872.51	964.37	921.38	32331	891.51	993.35	924.34	31455	948.85	1015.44	984.36	4892	1062.86	1094.71	1080.07	1295	**798.85**	**798.85**	**798.85**	2791	**798.85**	**798.85**	**798.85**	2838
Large																												
22	229	10	-	-	340999.00	382231.00	358953.70	148844	343504.00	378217.00	363592.40	134612	359934.00	399254.00	386235.20	14715	306107.00	328500.00	320510.30	5995	222180.00	**222584.00**	**222407.20**	7282	**222167.00**	**222584.00**	222408.40	7242
23	229	15	-	-	382093.00	410775.00	398223.30	121506	400928.00	446816.00	420310.80	112939	463818.00	515091.00	478458.50	7534	351344.00	390192.00	374736.60	4232	**264146.00**	264592.00	264384.40	7293	**264146.00**	**264556.00**	**264225.20**	7256
24	229	20	-	-	406107.00	460290.00	440030.80	103353	443670.00	481398.00	465306.30	97632	507915.00	556687.00	538917.50	4646	400909.00	457257.00	424575.60	3375	**319669.00**	**319678.00**	**319669.90**	7743	**319669.00**	**319678.00**	**319669.90**	7757
25	229	30	-	-	542670.00	577547.00	557586.60	80056	545360.00	620099.00	585112.70	76535	702254.00	753080.00	716464.30	2250	511926.00	545545.00	530949.40	2410	**406664.00**	**407194.00**	406956.50	6682	**406664.00**	407226.00	**406768.50**	6678
26	666	10	-	-	1333791.00	1460320.00	1385983.50	42792	763479.00	793465.00	779782.90	40240	734424.00	762101.00	751810.90	4674	549857.00	608474.00	573863.30	1836	391885.00	**395492.00**	394352.30	450	**391831.00**	395549.00	**393949.00**	463
27	666	15	-	-	1293871.00	1449643.00	1370313.10	37480	887416.00	942529.00	913237.90	33958	879510.00	911000.00	894909.50	2506	661930.00	702904.00	679841.10	1359	**448624.00**	452704.00	451189.50	566	448981.00	**450725.00**	**449978.60**	585
28	666	20	-	-	1438875.00	1567500.00	1498059.90	32569	1014583.00	1096827.00	1055167.30	30975	1044149.00	1078822.00	1067620.60	1571	764280.00	805419.00	781635.20	1106	524243.00	525324.00	524798.00	442	**522403.00**	**524042.00**	**523358.60**	450
29	666	30	-	-	1466851.00	1614448.00	1561156.70	25326	1221293.00	1296674.00	1249769.90	24117	1284411.00	1326217.00	1308931.10	781	919270.00	974089.00	948262.50	808	**652714.00**	655906.00	654442.90	547	653224.00	**654294.00**	**653857.20**	559

over ten runs are same for our approaches on above mentioned instances in Table 7.3 and Table 7.4.

Except for two small instances, where ABC and SAGA performed similar, our ABC approach performed much better than GA, GAG, HCGA and SAGA, and it is nearly impossible that this superior performance is due to random fluctuations. However, to rule out completely even this scarcest possibility, we have performed two tailed Mann-Whitney U-test between our ABC approach and other approaches on each instance. Tables 7.5 and 7.6 present the results of this test on each instance under our two different termination conditions. These tables clearly show that the results obtained by our ABC approach are statistically significant in comparison to GA, GAG, HCGA and SAGA even at 1% significance criteria (*p-value* $\leq$ 0.01) on each instance, except for two small instances, where ABC and SAGA obtained the same solution in all the runs. However, except for few large instances, results of ABC are not statistically significant with respect to RAND-ABC again stressing the point made in the previous paragraph.

For instances 5, 6 and 7, our approaches even improved the solution obtained through LINGO, and as such, the results of LINGO reported in [2] were not optimal for these 3 cases at least. Figure 7.2 plots the solution obtained by our ABC approach and the solution obtained by LINGO [2] for instance 7 with $n = 41$ and $m = 3$. In the plots of this figure, shared cities are shown as circles and exclusive cities for each of the 3 salesmen are shown as triangles, squares, and stars. The plot of solution obtained by LINGO is adapted from the plot of this solution in Figure 8 of [2]. There are only four edges which are present in plot of one solution, but not in the plot of other. These four edges are shown by dashed lines in these plots and their costs are also shown. Figure 7.2 clearly establishes that there exists a solution better than the one obtained by LINGO, and hence the solution of LINGO is not optimal for instance 7. However, as per the plot of LINGO solution in Figure 8 of [2], this solution has the cost of 368.10, whereas its cost is reported as 371.89 in Table II of [2].

7.4.3 Convergence behavior of various approaches

For showing the convergence behavior of various approaches, instances numbered 22 and 26 have been considered. Figure 7.3(a) and Figure 7.3(b) present the plots of convergence behavior of various approaches on instances numbered 22 and 26. As in Table 7.4, all the approaches were executed for 60 seconds. These plots clearly show that our approaches not only converge very fast in comparison to approaches of [2], but to a better objective function value. Except for few initial iterations, there is no difference between the convergence plots of ABC and

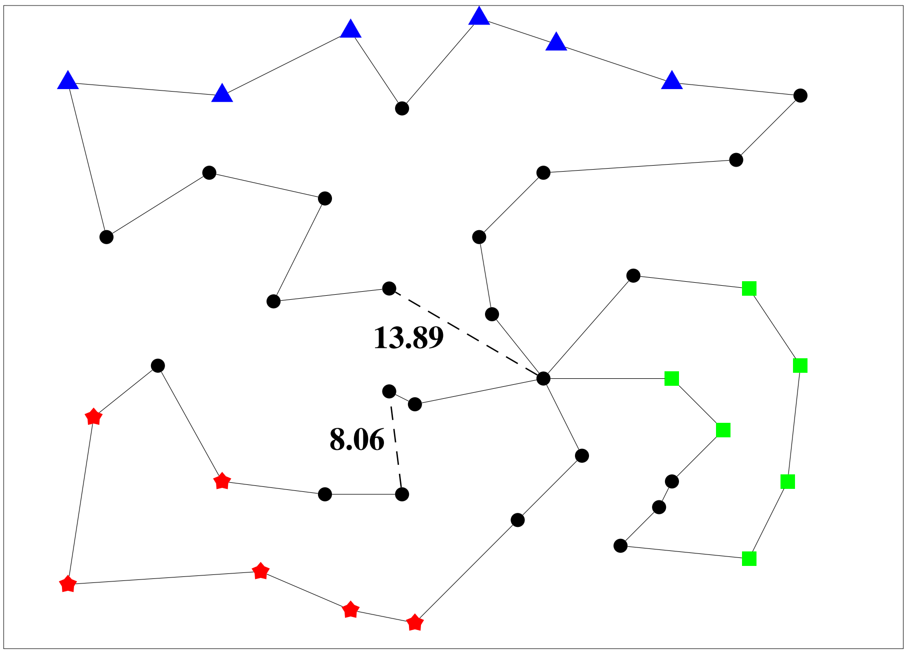
Table 7.5: Results of Mann-Whitney U-test between ABC and other approaches on each instance for termination condition of 1 sec.

ABC vs.			**GA**		**GAG**		**HCGA**		**SAGA**		**RAND-ABC**	
Instance	n	m	U-value	p-value	U-value	p-value	U-value	p-value	U-value	p-value	U-value	p-value
Small												
1	21	2	0	0.00018	0	0.00018	0	0.00018	50	0.96810	50	0.96810
2	21	3	10	0.00278	0	0.00018	0	0.00018	50	0.96810	50	0.96810
3	31	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
4	31	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
5	31	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
6	41	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
7	41	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
8	41	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
9	51	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
10	51	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
Medium												
11	51	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
12	51	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
13	51	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
14	76	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
15	76	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
16	76	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
17	76	6	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
18	101	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	45	0.72786
19	101	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
20	101	6	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
21	101	7	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
Large												
22	229	10	0	0.00018	0	0.00018	0	0.00018	0	0.00018	18	0.01732
23	229	15	0	0.00018	0	0.00018	0	0.00018	0	0.00018	22	0.03752
24	229	20	0	0.00018	0	0.00018	0	0.00018	0	0.00018	39	0.42952
25	229	30	0	0.00018	0	0.00018	0	0.00018	0	0.00018	13	0.00578
26	666	10	0	0.00018	0	0.00018	0	0.00018	0	0.00018	14	0.00736
27	666	15	0	0.00018	0	0.00018	0	0.00018	0	0.00018	18	0.01732
28	666	20	0	0.00018	0	0.00018	0	0.00018	0	0.00018	3	0.00044
29	666	30	0	0.00018	0	0.00018	0	0.00018	0	0.00018	5	0.00078

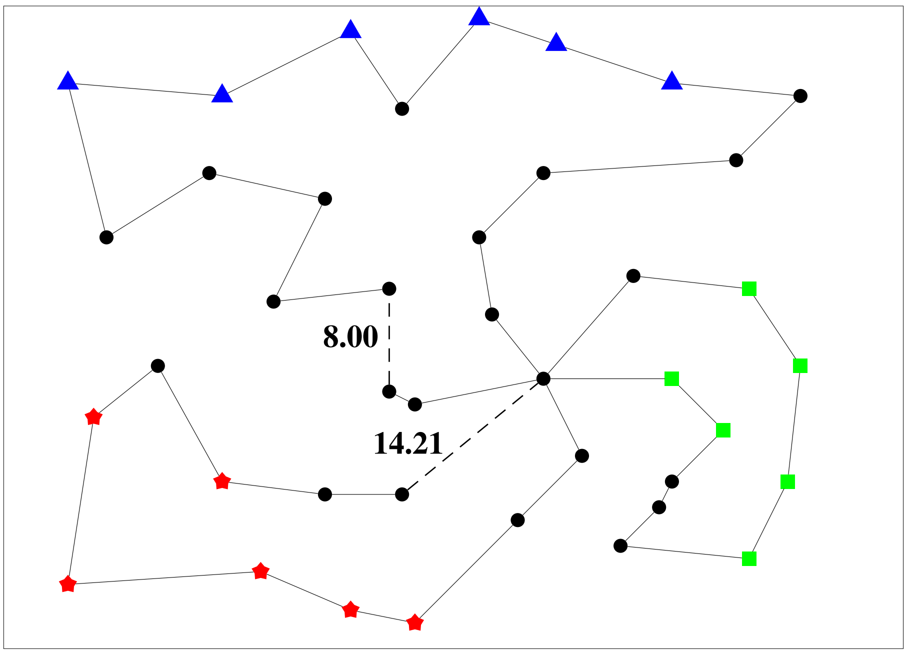
RAND-ABC. Further, these plots clearly show the advantage of greedy initialization as ABC starts with much better solutions than other methods. RAND-ABC also catches up ABC very fast due to higher degree of perturbation at initial iterations and effectiveness of INTRA and INTER procedures. Higher degree of perturbation makes a large number of cities unassigned, thereby providing opportunity to INTRA and INTER procedures to insert them at their proper positions. Hence, higher degree of perturbation at the beginning helps in rapidly transforming a purely randomly generated solution with a very bad fitness into a good solution through INTRA and INTER procedures. The problem specific greedy strategies along with m-tour encoding technique incorporated in ABC/RAND-ABC facilitate their rapid convergence. On the other hand, GA, GAG, HCGA and SAGA do not make use of any problem specific greedy strategy and employ a less efficient solution encoding scheme, and hence, they converge very slowly and that too to either inferior or same values.

Table 7.6: Results of Mann-Whitney U-test between ABC and other approaches on each instance for termination condition of 1, 5, 60 sec for small, medium, large instances respectively.

ABC vs.			**GA**		**GAG**		**HCGA**		**SAGA**		**RAND-ABC**	
Instance	n	m	U-value	p-value	U-value	p-value	U-value	p-value	U-value	p-value	U-value	p-value
Small												
1	21	2	0	0.00018	0	0.00018	0	0.00018	50	0.96810	50	0.96810
2	21	3	10	0.00278	0	0.00018	0	0.00018	50	0.96810	50	0.96810
3	31	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
4	31	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
5	31	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
6	41	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
7	41	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
8	41	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
9	51	2	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
10	51	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
Medium												
11	51	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
12	51	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
13	51	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
14	76	3	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
15	76	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
16	76	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
17	76	6	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
18	101	4	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
19	101	5	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
20	101	6	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
21	101	7	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
Large												
22	229	10	0	0.00018	0	0.00018	0	0.00018	0	0.00018	48.5	0.93624
23	229	15	0	0.00018	0	0.00018	0	0.00018	0	0.00018	30.5	0.14986
24	229	20	0	0.00018	0	0.00018	0	0.00018	0	0.00018	50	0.96810
25	229	30	0	0.00018	0	0.00018	0	0.00018	0	0.00018	30	0.14156
26	666	10	0	0.00018	0	0.00018	0	0.00018	0	0.00018	32	0.18684
27	666	15	0	0.00018	0	0.00018	0	0.00018	0	0.00018	16	0.01140
28	666	20	0	0.00018	0	0.00018	0	0.00018	0	0.00018	0	0.00018
29	666	30	0	0.00018	0	0.00018	0	0.00018	0	0.00018	33	0.21130

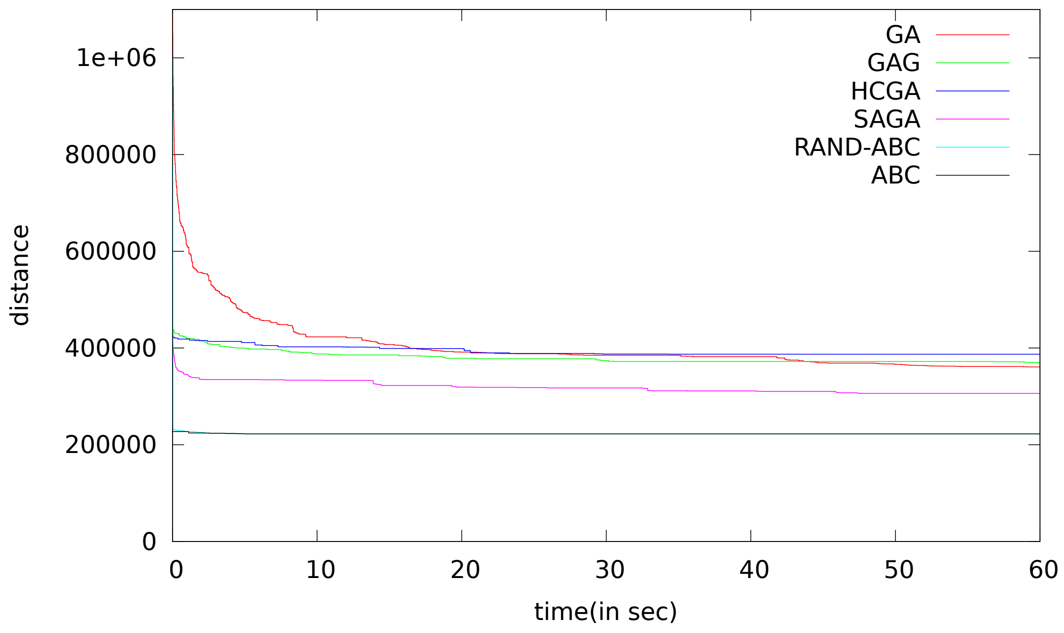


(a) ABC solution with cost 367.84

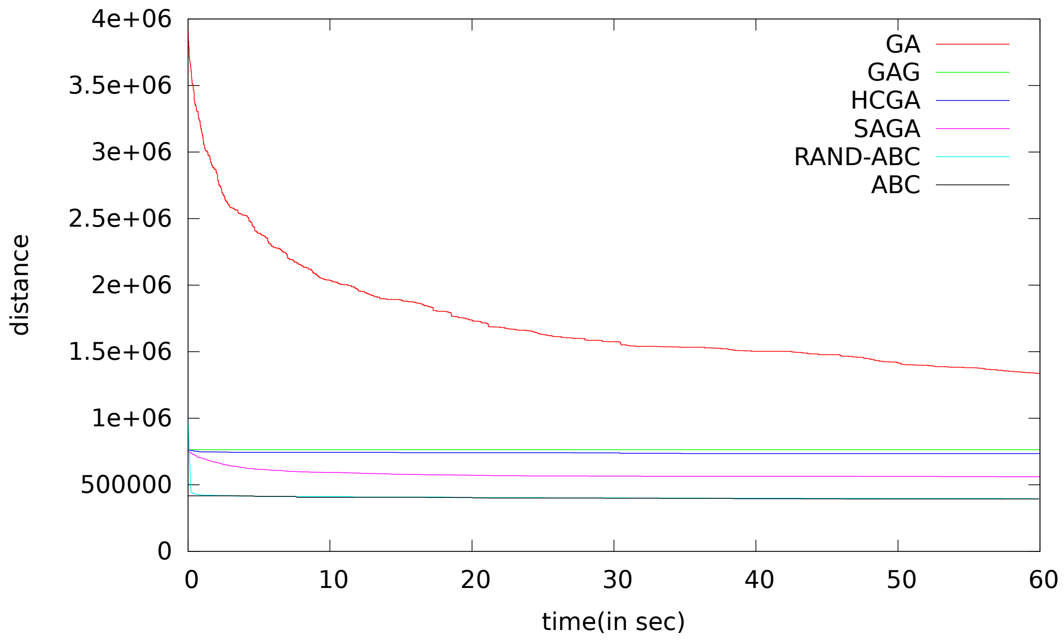


(b) LINGO solution with cost 368.10

Figure 7.2: Solution of instance 7 with $n = 41$ and $m = 3$ obtained by ABC and LINGO.



(a) Convergence behavior on instance having 229 cities and 10 salesmen



(b) Convergence behavior on instance having 666 cities and 10 salesmen

Figure 7.3: Convergence behavior of various approaches on two instances

7.5 Conclusions

In this chapter, we have proposed a swarm intelligence based approach, viz. an artificial bee colony algorithm for the colored traveling salesman problem. We have evaluated and compared our proposed approach with the state-of-the-art approaches on 21 benchmark instances available in [2] and 8 large instances derived from instances available on TSPLIB. Computational results on these instances clearly demonstrate the superiority of our approach over all the other state-of-the-art approaches in terms of solution quality and execution time both. Moreover, a new solution encoding scheme is proposed for representing a CTSP solution inside ABC algorithm which yields a considerably smaller solution space when compared to encoding schemes used previously for this problem.

Chapter 8

k-Interconnected Multi-Depot Multi-Traveling Salesman Problem

8.1 Introduction

Last TSP variant that we consider in this thesis has all the three aspects of a combinatorial optimization problem. This variant, which is introduced by Andrade *et al.* [3] is called k-Interconnected Multi-Depot Multi-Traveling Salesman Problem (k-IMDMTSP). This problem seeks a subset of the depots of size k from a given set of cities, where from each depot a salesman will start and end its tour to visit the remaining cities. Each city should be visited exactly once by only one salesman. The k depots also need to be connected via a cycle which is termed as inner cycle, whereas each salesman's tour is termed as outer cycle. The objective of k-IMDMTSP is to minimize the sum total of the costs of inner and outer cycles. There are several practical applications of k-IMDMTSP such as designing cheaper ring networks with link-failure tolerance, hub location problems in telecommunication networks, transportation networks where multiple echelons are used, location-routing problems. k-IMDMTSP possess all the three aspects of a combinatorial optimization problem in the following manner:

- *Subset selection:* Choosing a subset of depots from the set of all the cities.
- *Grouping:* Cities need to be partitioned into different groups to be visited by each salesman (outer cycles).
- *Permutation:* Both depots and cities need to be arranged in inner and outer cycles respectively to get cycles of least costs.

Basically, any problem that involves locating (depots, hubs etc.), planning tours for salesmen etc., has a k-IMDMTSP aspect to it. However, the problems that are closely related to k-IMDMTSP are multi-depot multi-traveling salesman problem (MDMTSP), location-routing problems (LRP), 2-echelon vehicle routing problem (2E-VRP). In the MDMTSP, the set of cities have to be visited using tours of unrestricted number of salesmen that can be based at available depots. A 2-approximation algorithm is proposed in [151] for the generalized MDMTSP (GMDMTSP), a variant where there are m salesmen based at each depot but have to choose at most $p \leq m$ salesmen. A Lagrangean-based algorithm is proposed in [152] to solve a variation of GMDMTSP where at least three vertices should be present in each tour. The MDMTSP differs from k-IMDMTSP, in having only pre-specified depots, whereas in k-IMDMTSP every city has the possibility of becoming a depot (terminal), and it also requires an inner cycle connecting all the depots. The LRP is a combination of two well known problems as the name itself suggests that it involves locating the depots and planning the routes of the vehicles with limited capacity based at depots to visit the cities. There are opening costs for using a depot, and each depot has some capacity limitation to serve its routes. There are many applications and variants of the LRP and interested readers may refer to [153] for a recent survey. In [154], LRP is solved by using a hybrid heuristic having two phases, where Lagrangean relaxation is used in the first phase and a granular tabu search is used in the second phase to improve the solution obtained by the first phase. This heuristic is the best heuristic till date for the LRP. An exact algorithm based on a branch-and-cut procedure with a series of new inequalities is proposed in [155] for the LRP. This approach provides the best results compared to any other exact algorithm for the LRP. Compared to LRP, k-IMDMTSP requires an extra inner cycle connecting all the depots, and for each depot, it permits only a single outer cycle. The 2E-VRP is first introduced in [156]. In this problem, first the freight is transported from origin to the intermediate depots on long-haul vehicles, and then the freight is delivered to customers from intermediate depots on small vehicles. In the first echelon, the routes are planned between origin and intermediate depots, whereas in the second echelon, routes are planned between intermediate depots and customers. Each route should start at origin and visit a subset of intermediate depots to serve required demand of customers. Difference between 2E-VRP and the k-IMDMTSP is that the latter permits only one single cycle that connects all the depots and only single outer cycle for each depot. The k-IMDMTSP is a general problem that is applicable to many real world scenarios related to locating, routing, scheduling, etc. Still it is an under-studied problem and yet to gather attention from researchers of both theory and practice. While introducing k-IMDMTSP, Andrade *et al.* [3] proposed a biased random key genetic algorithm (BRKGA) and a multi-start heuristic (MSH) for this problem. Empirical observations found BRKGA to be significantly better than MSH. These are the only approaches available in the literature for this problem.

In this chapter, we present an ABC algorithm based approach for k-IMDMTSP where a hyper-heuristic approach is utilized for neighboring solution generation. Owing to k-IMDMTSP possessing all the three aspects of a combinatorial optimization problem, a heuristic algorithm that one designs for k-IMDMTSP may not perform well over all the instances as aspects of the instances can vary a lot depending on their constituent parameter values (one may refer to next section for further details). To effectively solve such a problem within a short time, a heuristic needs to quickly adapt itself according to the aspects of the instance at hand, thereby providing good quality solutions over a wide range of instances of the problem. Hyper-heuristics are tailor-made for such situations. Hence, our ABC algorithm employs a hyper-heuristic for generating neighboring solutions. This hyper-heuristic is designed keeping in mind the specific needs of k-IMDMTSP and utilizes three low level heuristics each catering to a particular aspect of k-IMDMTSP. Our hyper-heuristic based ABC algorithm will be referred to as *HH-ABC* subsequently. We have also proposed a new solution encoding scheme for representing a k-IMDMTSP solution within HH-ABC which yields a search space that is considerably smaller in comparison to search spaces associated with encoding schemes used previously. Further, a solution is represented directly in our encoding scheme and hence, our encoding scheme does not incur the overhead of sorting to decode a solution like the schemes used previously. Comparison among HH-ABC and other state-of-the-art approaches on k-IMDMTSP benchmark instances available in the literature shows that our proposed approach outperformed the other approaches.

The remainder of this chapter is organized as follows: Section 8.2 provides the formal definition of the k-IMDMTSP. Our proposed approach to k-IMDMTSP is presented in Section 8.3. Section 8.4 presents the computational results and their analysis. Finally, Section 8.5 outlines some concluding remarks.

8.2 Problem definition

k-Interconnected Multi-Depot Multi-Traveling Salesman Problem (k-IMDMTSP) was formulated by Andrade *et al.* [3]. This problem can be defined formally as follows: Suppose $G = (V, E)$ is an undirected, edge-weighted complete graph having vertex set V and edge set

E, where vertices are numbered from 1 to n, i.e., $|V| = n$. Each edge $e \in E$ connecting two vertices i and j has an associated weight $w_e \in \mathbb{R}^+$ that represents the distance between the two vertices i and j. The k-IMDMTSP involves finding the following:

- A subset D of V with cardinality k, i.e., $D = \{d_1, d_2, \ldots, d_i, \ldots, d_k\}$ such that $d_i \in V$, $\forall d_i \in D$. Each element of D is called a *depot* or a *terminal*.
- A Hamiltonian cycle I over the subgraph induced by vertices in D. This cycle connects all depots and is termed as *inner cycle*.
- For each depot d_i, a cycle O_i involving d_i and some of the vertices in $V \setminus D$. Each such cycle is termed as an *outer cycle*. The formation of these k outer cycles is subjected to following two constraints:
    1. Each of the vertices in $V \setminus D$ should belong to exactly one outer cycle, i.e., each outer cycle should be disjoint with respect to all other outer cycles.
    2. Each outer cycle can have a maximum of C vertices including the depot. The number of distinct vertices in a cycle is called its *size*. So this constraint can be restated as the size of an outer cycle can not exceed C.

Therefore, a feasible solution for the k-IMDMTSP consists of a set of depots $D = \{d_1, d_2, \ldots, d_k\}$, an inner cycle I over D, and a collection of k outer cycles O_i with C or fewer vertices based at depot d_i $\forall i = 1, \ldots, k$. The cost of a feasible solution is

$$\sum_{e \in E[I]} w_e + \sum_{i=1}^{k} \sum_{e \in E[O_i]} w_e \tag{8.1}$$

where $E[I]$ and $E[O_i]$ represent the set of edges present in the inner cycle I and the outer cycle O_i respectively. If $|E[I]| = 1$ or $|E[O_i]| = 1$ then that means the cycle concerned involves a single edge, i.e., cycle concerned has size 2. In such cases, the cost of the single edge involved is doubled for calculating the round trip cost. The k-IMDMTSP seeks a solution with minimum cost among all its feasible solutions. It is to be noted that for a k-IMDMTSP instance to have any feasible solution, the relation $\lceil \frac{n}{k} \rceil \leq C$ must hold.

Some special cases for the k-IMDMTSP exist due to different constituent parameter values [3]. With respect to the size of an outer cycle, there are two possible cases.

1. If the size of an outer cycle is one then that means only depot is in this cycle. Such a cycle is called an *empty cycle*.
2. If the size of an outer cycle is two then that means this cycle has a vertex in addition to depot and this case is called fur.

With respect to the size of inner cycle, i.e., parameter k, three special cases can occur.

1. If $k = 2$, then the inner cycle is a bridge between two vertices that connects two outer cycles.
2. If $k = 1$, then the inner cycle is empty and there is a single outer cycle, so k-IMDMTSP reduces to TSP.
3. If $k = n$, then only the inner cycle exists and all the outer cycles are empty, so k-IMDMTSP reduces to TSP.

If $\frac{k}{n}$ is small (near to 0) then there will be few depots and outer cycles, and hence, less effort is needed in selecting depots among vertices, finding the inner cycle comprising all the depots and grouping the remaining vertices into outer cycles. However, there will be a large number of vertices in each outer cycle, and hence, most of the effort is focussed on determining the best permutation of vertices for each outer cycle. On the other hand, if $\frac{k}{n}$ is high (near to 1) then a large number of vertices will be depots, and hence, less effort is required in selecting depots among vertices. However, as there are large number of depots, more effort is required in determining the inner cycle. As the number of outer cycles is equal to the number of depots, more effort is spent in grouping the remaining vertices into outer cycles. However, less effort is required in determining the best permutation of vertices for each outer cycle as relatively less number of vertices is available for each outer cycle. If the search process give attention to different aspects one-by-one in a sequential manner then we may end up with a very bad solution even at the expense of consuming much time as each aspect in itself is an $\mathcal{NP}$-Hard problem. Therefore, to get a good solution, the search process needs to be carried out by taking into consideration all the aspects at the same time. Throughout this chapter, we will use vertex and city interchangeably.

8.3 Hyper-heuristic based artificial bee colony algorithm for the k-IMDMTSP

We have developed an artificial bee colony algorithm based approach for k-IMDMTSP where a hyper-heuristic is employed for neighboring solution generation. As mentioned already, aspects of the k-IMDMTSP instances can vary a lot depending on their constituent parameter values, and hence, its natural to use a hyper-heuristic for neighboring solution generation, so that HH-ABC can quickly adapt as per the aspects of the instance at hand.

Following subsections describe the main features of our HH-ABC algorithm for the k-IMDMTSP.

8.3.1 Solution encoding

There is abundant literature on variants of multiple traveling salesperson problem, and various encoding schemes have been used [145, 148, 149, 157, 158] to represent a solution. In [3], k-IMDMTSP is addressed through a biased random key genetic algorithm (BRKGA) that uses random keys based single chromosome representation. First, we will briefly describe this encoding and then introduce the encoding used by us. Recalling k, n and C are number of depots, number of vertices and maximum number of vertices permitted in an outer cycle respectively, a chromosome is represented in BRKGA using an integer vector v = $[v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_{k+n}]$ of length $(k + n)$. The first k alleles represent the size of k outer cycles, i.e., $v_i \in \{1, \ldots, C\}, \forall i \in \{1, \ldots, k\}$. The remaining alleles have random keys which determines the position of a vertex in outer cycles, i.e., $v_i \in \mathbb{N}, \forall i \in \{k + 1, \ldots, k + n\}$. For extracting a solution from this chromosome, the n alleles $v_{k+1}, \ldots, v_{k+n}$ are assumed to have indices $1, \ldots, n$ respectively, and these n alleles are sorted in non-decreasing order with their indices occupying their position in the sort order. So these indices form a permutation π of vertices. Out of these first k vertices, $\pi_1, \ldots, \pi_k$ are the depots. Each depot $\pi_i, i = \{1, \ldots, k\}$, has an outer cycle formed by v_i vertices including depot π_i and other $v_i - 1$ vertices $\pi_{k+o(i)+j}$, such that $j = 1, \ldots, v_i - 1$, where $o(i)$ indicates the offset in the alignment of cycles in π and can be computed using following recursive formula.

$$o(i) = \begin{cases} o(i-1) + v_{i-1} & : \quad if \quad i > 1 \\ 0 & : \quad otherwise \end{cases} \tag{8.2}$$

It is to be noted that $v_i = 1$ means an empty cycle having only depot d_i. Advantage of this encoding scheme is that even if some random keys (alleles at positions $k+1, \dots, k+n$) are modified, it will not effect the feasibility of the solution. However, any random modification in first k alleles affects feasibility and requires a repair operation to restore feasibility.

Theorem 1: The search space size of k-IMDMTSP on T vertices using single chromosome encoding with random keys is

1. infinite when any positive integer can be used as random keys
2. $Q \times T! \times \binom{\alpha}{T}$ when integers in the interval [1, α] only can be used as random keys where $Q = \sum_{i=0}^{\lfloor \frac{N}{C} \rfloor} (-1)^i \binom{k}{i} \binom{T-1-iC}{k-1}$

Proof. Consider a k-IMDMTSP with N vertices (excluding depot(s)) and k depots, and total number of vertices $T = N + k$. The single chromosome is a positive integer-vector of length $T + k$. The subvector containing first k values can be any one of the Q distinct vectors of length k with values between $[1, \dots, C]$ having sum exactly equal to T. The expression for Q is derived below. The subvector containing remaining T integer values (random keys) resolves into a permutation of T cities after sorting. There are infinitely many ways in which a particular permutation can be represented using random keys as set of integers is infinite. So the size of the search space is infinite when any positive integer can be used as a random key. However, Andrade *et al.* [3] used only integers in the interval [1, α] for random keys and a decoder ensured the uniqueness of random keys, where α is an integer sufficiently large ($>> T$). So, there are $T! \times \binom{\alpha}{T}$ ways in which these T random keys can be selected and arranged. Hence, the size of the search space when integers in the interval [1, α] only can be used as random keys is

$$Q \times T! \times \binom{\alpha}{T} \tag{8.3}$$

where Q can be derived as follows. Our aim is to get the number of ways to distribute T vertices into k tours (cycles) with a restriction of having at least one vertex (i.e. depot) and at most C (capacity constraint) vertices in each tour. When there is no capacity constraint, the number of ways that are possible for distributing T vertices to k tours with at least one vertex allocated to each tour is $\binom{T-1}{k-1}$ ([159], page 14). From these we have to subtract the ways which violate the capacity constraint. The number of distributions that violate the capacity constraint for at least the first tour can be calculated in the following manner: assign C vertices directly to the first tour and distribute remaining $T - C$ vertices to k tours with the same restriction of assigning at least one vertex to each tour. Having assigned C vertices already to first tour, it will get at least one more vertex from remaining vertices and hence its capacity will be violated. So number of distributions that violate the capacity constraint for at least the first tour is

$$\binom{T-C-1}{k-1} \tag{8.4}$$

Like the first tour, any of the k tours can be given definite chance to violate. So this results in

$$\binom{T-1}{k-1} - \binom{k}{1}\binom{T-C-1}{k-1} \tag{8.5}$$

However, this is over subtraction. Distributions where at least the capacity for two tours get violated have been subtracted twice. So, we add back the distributions that violate the capacity for at least two tours. The distributions that violate capacity for at least two tours can be calculated by assigning C vertices to each of two tours and remaining $T-2C$ cities are distributed to k tours with the same restriction of having at least one city.

$$\binom{T-2C-1}{k-1} \tag{8.6}$$

Like this way, any combination of two out of the k tours can be given definite chance to violate. so this results in

$$\binom{T-1}{k-1} - \binom{k}{1}\binom{T-C-1}{k-1} + \binom{k}{2}\binom{T-2C-1}{k-1} \tag{8.7}$$

Now, the distributions that violate capacity for at least three tours have been added twice and need to be subtracted once. Continuing this argument and observing that this violation of capacity happens to at most $\lfloor \frac{N}{C} \rfloor$ number of tours as at least one vertex need to be assigned to each of the k tours leaving N vertices only to take part in violations if any which happens when C vertices out of these N get assigned to any tour as each tour already contains one vertex. So Q can be formulated as follows.

$$Q = \sum_{i=0}^{\lfloor \frac{N}{C} \rfloor} (-1)^i \binom{k}{i}\binom{T-1-iC}{k-1} \tag{8.8}$$

$\square$

We have represented a chromosome directly by $(k+1)$ tours. The first tour represents the inner cycle and remaining k tours represent the k outer cycles. Hence, a solution is represented directly, and there is no overhead to decode a solution like the random key encoding used in [3] which requires sorting. We have represented the inner cycle of length k by linear permutation of length k. So inner cycle representation has redundancy as k linear permutations of k vertices corresponds to the same tour which is a circular permutation. However, this much redundancy is inherent and it helps in eliminating redundancy in representing each outer cycle. The second tour represents the outer cycle corresponding to first depot in the permutation, the third tour represent the outer cycle corresponding to second depot in the permutation and so on. However, while representing each outer cycle, the corresponding depot is not part of the permutation, so each outer cycle is represented without any redundancy. We call this encoding $(k+1)$-tour encoding.

Theorem 2: The search space size of k-IMDMTSP by using our $(k+1)$-tour encoding is $Q \times T!$ where value of Q is same as in previous theorem.

Proof. Here the first tour represents the inner cycle and search space size of this tour is $\binom{T}{k} \times k!$. Please note that an outer cycle contains at least a depot and the number of vertices in it can not exceed C. In our representation, as depot is not the part of outer cycle so outer cycle in our representation can have no vertex and the number of vertices in an outer cycle can not exceed $C-1$. There are R distinct vectors of length k with values between $[0, \ldots, C-1]$ having sum exactly equal to N. The expression for R is derived below. Let us denote each such vector by X_d, $d = 1 \ldots R$ and its e^{th} element by $X_d(e)$, $e = 1 \ldots k$. The search space size of remaining k-tours is

$$\sum_{d=1}^{R} \left(\prod_{e=1}^{k} \left(\binom{(N - \sum_{f=1}^{e-1} X_d(f))}{X_d(e)} X_d(e)! \right) \right) \tag{8.9}$$

As each summation term reduces to $N!$, therefore above expression simplifies to

$$R \times N! \tag{8.10}$$

Therefore the search space size of all $(k+1)$ tours is

$$\binom{T}{k} \times k! \times R \times N! \tag{8.11}$$

As $T = N + k$, the above expression can be further simplified to

$$R \times T! \tag{8.12}$$

The value of R can de derived as follows: Here our aim is to get the number of ways to distribute N vertices into k tours (cycles) with only constraint of having at most $C-1$ (capacity constraint) vertices in each tour. As mentioned already, as depots are not part of outer cycles in our representation, so a tour may not contain any vertex in our representation. When there is no capacity constraint, the number of ways that are possible for distributing N cities to k tours is $\binom{N+k-1}{N}$ ([159], page 14). From these we have to subtract the ways which are violating the capacity constraint. The number of distributions that violate the capacity constraint for at least the first tour can be calculated in the following manner: Assign C vertices directly to the first tour and distribute remaining $N-C$ vertices to k tours without any lower limit on the number of cities in a tour. Please note the difference with the proof of first theorem. Here only when we assign C cities to first tour, we can be sure of capacity ($C-1$) violation as first tour may not get any vertex from remaining vertices. Hence, the number of distributions that violate the capacity constraint for the first tour is

$$\binom{N-C+k-1}{N-C} \tag{8.13}$$

Like the first tour, any of the k tours can be given definite chance to violate. So this results in

$$\binom{N+k-1}{N} - \binom{k}{1}\binom{N-C+k-1}{N-C} \tag{8.14}$$

However, this is over subtraction. Distributions that violate capacity for at least two tours have been subtracted twice. So, we add back the distributions that violate the capacity for at least two tours. The distributions that violate capacity for at least two tours can be calculated by assigning C vertices to each one of two tours and remaining $N-2C$ cities are distributed to k tours with the same restrictions.

$$\binom{N-2C+k-1}{N-2C} \tag{8.15}$$

Like this way, any combination of two out of the k tours can be given definite chance to violate. So this results in

$$\binom{N+k-1}{N} - \binom{k}{1}\binom{N-C+k-1}{N-C} + \binom{k}{2}\binom{N-2C+k-1}{N-2C} \tag{8.16}$$

Now, building the argument in the same way as in previous theorem and observing that this violation of capacity can happen in $\lfloor \frac{N}{C} \rfloor$ number of tours, the R can be formulated as follows.

$$R = \sum_{i=0}^{\lfloor \frac{N}{C} \rfloor} (-1)^i \binom{k}{i}\binom{N+k-1-iC}{N-iC} \tag{8.17}$$

Please note that here all N cities can take part in capacity violations as tours need not contain any city in our representation and capacity gets violated when C cities get assigned to any tour. Recalling, $T = N + k$ and $\binom{X}{Y} = \binom{X}{X-Y}$, we can see that $R = Q$. □

Theorem 3: The search space size of k-IMDMTSP with $(k + 1)$-tour encoding is smaller than that of a single chromosome encoding with random keys.

Proof. Proof is obvious as $T! \times \binom{\alpha}{T}$ is always greater that $T!$. In fact, for random key encoding to be effective $\alpha >> T$ and hence search space size of k-IMDMTSP with $(k+1)$-tour encoding is much smaller than that of a single chromosome encoding with random keys □

8.3.2 Fitness

The fitness function is same as the objective function given in Equation (8.1) , i.e., a solution's fitness is the sum total of the costs of inner and outer cycles. Hence, a solution with lower value of the fitness function is considered to be better than a solution with higher value.

Algorithm 22: Pseudo-code for generating an initial solution

```
Input: A $k$-IMDMTSP instance
Output: An initial solution $S$
function Generate_Initial_Solution()
/**********Initializing inner cycle by using $k$-means clustering**********/
$n := no.\ of\ cities$;
$k := no.\ of\ depots$;
Initialize $k$ clusters with $k$ randomly selected cities as centroids
for $i := 1$ to 20 do
    Assign remaining $(n - k)$ cities to the cluster corresponding to nearest centroids;
    For each cluster, compute the mean and choose a city nearest to the mean as new centroid;
    Reinitialize $k$ clusters with these $k$ centroids;
foreach city $c$ in the set of $k$ centroids do
    Insert the city $c$ in best possible position in inner cycle;
/**********Initializing outer cycles**********/
foreach city $c$ in the set of remaining cities in some random order do
    Follow the procedure described in Section 8.3.3 to insert $c$ into $S$;
return $S$;
end function
```

8.3.3 Initialization

Each initial solution is constructed in an iterative manner by using the following two constructive heuristics.

1. The first heuristic is responsible for identifying the depots and constructing the inner cycle. It is based on k-means clustering algorithm to select the set D of k depots from the set V consisting of n vertices. Initially, k vertices are selected at random out of n, where each vertex will constitute a cluster with itself as the centroid of the cluster. Obviously, for each such cluster with single vertex, the mean of each cluster will coincide with this vertex. The remaining vertices are assigned one by one in some random order to the nearest cluster (cluster whose centroid is nearest to the vertex in consideration) [160]. Once all vertices are assigned in this way, mean for each cluster is updated to reflect its composition, and the vertices which are nearest to the updated mean of each cluster are updated as new centroids. The vertices other than centroids are now reassigned by following afore-mentioned procedure. This process iterates for a limited number of times (20 times). The centroids identified after last iteration are chosen as depots. The main idea behind this heuristic is to select those vertices as depots, around whom there is a heavy concentration of vertices. To construct the inner cycle from these identified depots, we have followed an iterative procedure where during each iteration a depot is selected at random from the depots not yet assigned to the inner cycle and assigned to the inner cycle at the best possible insertion position. This process is repeated till all the depots are part of the inner cycle.

2. The second heuristic constructs the outer cycle for each of the k depots identified by the first heuristic. Let U be the set of unassigned vertices after applying the first heuristic, i.e., $U = V \setminus \{D\}$ initially. This heuristic also follows an iterative process that begins by selecting a vertex $a \in U$ uniformly at random and then it is inserted into the cycle associated with a depot (with out violating the capacity constraint ,i.e., number of cities must not exceed C) at a position that yields the least increase in the cost. For this, all possible insertion positions in every depot's outer cycle need to be checked. Once a is assigned to an outer cycle, that city is removed from the set of unassigned cities, i.e., $U = U \setminus \{a\}$ and the next iteration starts. This process is repeated until the set U becomes empty.

Algorithm 22 provides the pseudo-code for generating an initial solution. Each constructed initial solution is uniquely associated with an employed bee, and fitness of each solution is evaluated.

8.3.4 Mechanism for choosing a food source

In order to choose a food source for an onlooker bee, we have used the binary tournament selection method (Section 5.3.4) where the probability of selection of the better solution is P_{bts}.

8.3.5 Determination of a neighboring food source

An effective neighboring solution determination procedure should consider all the aspects of a problem, and should also maintain the appropriate balance among the considerations given to different aspects. Keeping this in mind, our neighboring solution determination procedure follows a hyper-heuristic based approach. The hyper-heuristic generates a neighboring solution S' from the current solution S. The hyper-heuristic is provided with a three low-level heuristics IN_HEUR, SS_HEUR, and OUT_HEUR. These three heuristics are explained below:

1. *IN_HEUR:* This heuristic is designed to deal with permutation aspects of vertices in the inner cycle. Each vertex in the inner cycle of S is copied to the inner cycle of S' with probability P_{cp}. As a result, some vertices will be left unassigned. These unassigned vertices will be assigned in the following manner: An unassigned vertex is picked uniformly at random and inserted at a position in the inner cycle of S' that yields the least increase in the cost of the inner cycle. This process continues until no unassigned vertices remain.

2. *SS_HEUR:* This heuristic caters to subset selection aspect, i.e., it identifies a non-depot vertex to be a new depot in place of an existing depot so that the cost of the inner cycle can be reduced. Each vertex d in the inner cycle is a depot and has an associated outer cycle O. Assume, d_s, and d_p to be the successor and the predecessor of d in the inner cycle respectively. Each vertex in outer cycle O is treated as a potential candidate to become a depot. A vertex $d' \in O$ that has least sum of edge costs, i.e., $cost(d_p, d') + cost(d', d_s)$, is chosen as depot, and d (in case it is different from d') is made a vertex of the outer cycle O in neighboring solution S'. This is done for every depot one-by-one in the order in which these depots occur in our encoding for the inner cycle (Please recall that we have used a linear permutation of depots to represent the inner cycle). This is similar to the second neighborhood operator used in [3].

3. *OUT_HEUR:* This heuristic deals with two aspects grouping, and permutation of cities in outer cycle. Each city in an outer cycle of S is copied to corresponding outer cycle of S'

with probability P_{cp}. The unassigned cities that are left out after copying are assigned in the same manner as in Section 8.3.3.

For each initial employed bee solution, the hyper-heuristic generates ten distinct vectors of size ten containing at each position a randomly chosen value from the set $\{1, 2, 3\}$ with one restriction mentioned below. The value 1, 2 and 3 respectively corresponds to IN_HEUR, SS_HEUR and OUT_HEUR. Each vector of length ten dictates the order in which these three heuristics can be applied on a solution, i.e., beginning at first position in a vector, the heuristic corresponding to the value at each position in the vector is applied one after the other on the solution under consideration. If after applying a heuristic, we get an improved solution then solution under consideration is replaced by this solution before applying the next heuristic, otherwise the next heuristic is applied on the same solution. Two consecutive positions in a vector can not be 2 as the solution obtained after SS_HEUR is locally optimal with respect to SS_HEUR, and hence, two consecutive applications of SS_HEUR is worthless. Further, there is no need to apply SS_HEUR again if the solution under consideration has not been improved by some other heuristic since the previous application of SS_HEUR. Clearly, the order and the number of times a heuristic repeats in a vector matter.

Ten neighboring solutions are generated for each initial solution utilizing the ten vectors associated with each of them. The vector yielding the neighboring solution of highest fitness for each initial solution is retained. For generating a neighboring solution for a solution, always its associated vector is utilized. This association between an employed bee solution and a vector continues till the solution fails to improve over certain number of iterations. In this situation, the current solution is discarded and replaced with a new solution. A new vector will also be associated with the new solution by using the same procedure as above.

Here also the degree of perturbation varies over iterations. Similar to previous chapter, the parameter P_{cp} controls the degree of perturbation and the degree of perturbation is $1 - P_{cp}$. The value of P_{cp} varies according to iteration from P_{mincp} to P_{maxcp}. The value of P_{cp} in an iteration $iter$ is computed using the following formula:

$$P_{cp} := \left(\frac{P_{maxcp} - P_{mincp}}{iter_{max}}\right)(iter) + P_{mincp} \tag{8.18}$$

Here $iter_{max}$ is the maximum number of iterations allowed for our approach.

Algorithm 23 provides the pseudo code for generating a neighboring solution.

Algorithm 23: Pseudo-code for generating a neighboring solution

```
Input: A solution $S$ and its associated vector
Output: A neighboring solution $S'$
function Create_Neighbor($S$, $S.vector$)
for $i := 1$ to vector length do
    if $S.vector[i] := 1$ then
        /**********apply $IN\_HEUR$**********/
        foreach city c in inner cycle as per their order do
            Generate a random number $r$ such that $0 \leq r \leq 1$;
            if $r < P_{copy}$ then
                Copy $c$ to inner cycle in $S'$;
            else
                Add $c$ to a set of unassigned cities;
        Computer the fitness of $S'$;
        foreach city c in the set of unassigned cities in some random order do
            Follow the procedure described in $IN\_HEUR$ (Section 8.3.5-Item 1) to insert $c$ into inner
            cycle of $S'$;
        return $S'$;
    else if $S.vector[i] := 2$ then
        /**********apply $SS\_HEUR$**********/
        $S' := S$;
        foreach outer cycle $t$ in $S'$ do
            $d :=$ depot of outer cycle $t$;
            $d_s :=$ successor of $d$ in inner cycle of $S'$;
            $d_p :=$ predecessor of $d$ in inner cycle of $S'$;
            foreach city $d'$ in $t$ as per their order in $t$ do
                if $cost(d_p, d') + cost(d', d_s) < cost(d_p, d) + cost(d, d_s)$ then
                    $d := d'$;
        return $S'$;
    else
        /**********apply $OUT\_HEUR$**********/
        foreach outer cycle $t$ in $S$ do
            foreach city $c$ in $t$ as per their order in $t$ do
                Generate a random number $r$ such that $0 \leq r \leq 1$;
                if $r < P_{copy}$ then
                    Copy $c$ to outer cycle $t$ in $S'$;
                else
                    Add $c$ to a set of unassigned cities;
        Compute the fitness of $S'$;
        foreach city c in the set of unassigned cities in some random order do
            Follow the procedure described in Section 8.3.3 to insert $c$ into $S'$;
        return $S'$;
end function
```

8.3.6 Other features

We have utilized different number of employed bees and onlooker bees in our approach. If an employed bee solution does not improve over a fixed number of trials $limit_{scout}$, then employed bee associated with that food source can abandon it to become a scout. However, to control the diversification, we have put the restriction that at most *one* bee can be a scout bee in an iteration, i.e., in an iteration the number of scout bees can be either zero or one. In our approach, the number of scout bees in an iteration is equal to one if there exists some food sources in that iteration which have not been improved over last $limit_{scout}$ trials, otherwise it is equal to zero. Among all food sources which have not been improved over last $limit_{scout}$ trials, the employed bee associated with the food source that has not been improved for highest number of trials becomes the scout bee.

Like the previous chapter, a scout bee is turned into an employed bee by generating a neighboring solution of the neighboring solution of just abandoned solution and associating it with the scout bee under consideration. For doing this, we have utilized the vector associated with the abandoned solution with the modification that solution obtained after applying a heuristic as per the vector always replaces the original solution before applying the next heuristic dictated by the vector.

The pseudo-code of our HH-ABC approach is given in Algorithm 24, where n_{eb} and n_{ob} are the number of employed bees and the number of onlooker bees respectively. *Associate_Vector(S_i)* is a function that associates a vector $S_i.vector$ with a solution S_i as per the procedure described in Section 8.3.5. As mentioned in this section, $S_i.vector$ dictates the order in which the low level heuristics can be applied on S_i. *Create_Neighbor(S, S.vector)* is a function that returns a solution in the neighborhood of the solution S using its associated vector $S.vector$ as per the procedure described in Section 8.3.5. A solution for an onlooker bee is selected from employed bee solutions through the function *Select_Selection($S_1, S_2, \ldots, S_{n_{eb}}$)* which implements the binary tournament selection method (Section 8.3.4). This function returns the solution selected.

8.4 Computational results

For testing our HH-ABC approach, we have used the same instances as used in [3]. These instances are actually derived from the standard TSPLIB. Andrade *et al.* [3] used 63 instances with five different scenarios to test the performance of their approaches. These instances have cities ranging from 14 to 1379, and have symmetric $n \times n$ distance matrix. Each of the five

Algorithm 24: Pseudo code of HH-ABC approach for k-IMDMTSP

```
Input: Set of parameters for ABC Algorithm and a $k$-IMDMTSP instance
Output: Best solution found
/**********Initialization phase**********/
for $i := 1$ to $n_{eb}$ do
    $S_i$ := Generate_Initial_Solution();
$best$ := best solution among $S_1, S_2, \ldots, S_{n_{eb}}$;
for $i := 1$ to $n_{eb}$ do
    $S_i.vector$ := Associate_Vector($S_i$);
$P_{cp} := P_{mincp}$;
$gap := \frac{(P_{maxcp} - P_{mincp})}{iter_{max}}$;
for $iter := 1$ to $iter_{max}$ do
    $P_{cp} := P_{cp} + gap$;
    /**********Employed bee phase**********/
    for $i := 1$ to $n_{eb}$ do
        $S'$ := Create_Neighbor($S_i$, $S_i.vector$);
        if $S'$ is better than $S_i$ then
            $S_i := S'$;
    /**********Onlooker bee phase**********/
    for $i := 1$ to $n_{ob}$ do
        $S_p$ := Select_Selection($S_1, S_2, \ldots, S_{n_{eb}}$);
        $S'$ := Create_Neighbor($S_p$, $S_p.vector$);
        if $S'$ is better than $S_p$ then
            $S_p := S'$;
    /**********Memorize best solution**********/
    for $i := 1$ to $n_{eb}$ do
        if $S_i$ is better than $best$ then
            $best := S_i$;
    /**********Scout bee phase**********/
    Find the solution $S_{scout}$ that has not improved over maximum number of trials;
    if $S_{scout}$ has not improved over $limit_{scout}$ trials then
        replace $S_{scout}$ as per the procedure described in Section 8.3.6;
        $S_{scout}.vector$ := Associate_Vector($S_{scout}$);
return $best$;
```

scenarios considered consists of different sets of values for k and C. These five scenarios are described below:

1. **ST** – small inner cycle and tight outer cycles: $k = \lceil 0.2n \rceil$, $C = \lceil n/k \rceil$

2. **SL** – small inner cycle and loose outer cycles: $k = \lceil 0.2n \rceil$, $C = \lceil 2n/k \rceil$

3. **LT** – large inner cycle and tight outer cycles: $k = \lceil 0.5n \rceil$, $C = \lceil n/k \rceil$

Table 8.1: Candidate parameter values used in parameter settings of ABC

Parameter	Description	Candidate Values
n_{eb}	Number of employed bees	$\{75, 100, 125\}$
n_{ob}	Number of onlooker bees	$\{125, 150, 175\}$
$limit_{scout}$	Number of iterations without improvement after which a solution can be discarded	$\{25, 50, 75\}$
P_{bts}	Probability of selecting better of the two solutions in binary tournament slection	$\{0.7, 0.8, 0.9\}$
P_{mincp}	Minimum probability of copying a city belonging to a cycle of the solution to the corresponding cycle of neighboring solution being generated	$\{0, 0.1, 0.2\}$
P_{maxcp}	Maximum probability of copying a city belonging to a cycle of the solution to the corresponding cycle of neighboring solution being generated	$\{0.7, 0.8, 0.9\}$

4. **LL** – large inner cycle and loose outer cycles: $k = \lceil 0.5n \rceil$, $C = \lceil 2n/k \rceil$

5. **SQ** – inner and outer cycles of same size: $k = C = \sqrt{n}$

We have implemented HH-ABC in C and executed it on a Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM. We have allowed our approach to execute for 100 iterations, i.e., $iter_{max}$ is set to 100.

For benchmarking, we have compared our HH-ABC approach with biased random key genetic algorithm (BRKGA) and multi-start heuristic (MSH) proposed in [3]. Each test instance has been solved through HH-ABC thirty independent times like these previously proposed approaches.

8.4.1 Parameter settings for HH-ABC

Since HH-ABC is a stochastic approach, therefore, selection of appropriate parameter values is important for its success. In order to set different parameters of HH-ABC, we have followed the Taguchi method [131, 132]. There are six parameters in HH-ABC approach. We have identified three candidate parameter values for each of these six parameters. These six parameters along with their description and candidate values are listed in Table 8.1. These candidate parameter values are chosen based on available literature, our own previous experience with the artificial bee colony algorithm, and some preliminary experimentations. As per the Taguchi method, for determining the value of seven parameters, each having three candidate values, at least 18 sets of experiments involving various combinations of candidate parameter values need to be done. The combination of parameter values for each of these set of experiments has to be chosen according to the rows of $L18$ orthogonal array [131]. Based on these experiments, following parameter values have been used in subsequent experiments: We have used a population of 125 employed bees ($n_{eb} = 125$), 175 onlooker bees ($n_{ob} = 175$), $P_{bts} = 0.7$, $limit_{scout} = 50$ and the value of P_{cp} can be varied from 0.1 to 0.9, i.e. $P_{mincp} = 0.1$ and $P_{maxcp} = 0.9$.

8.4.2 Comparison of our approach on test problems of [3]

Tables 8.2, 8.3, 8.4, 8.5, and 8.6 compare the performance of HH-ABC with BRKGA and MSH approaches proposed in [3] on ST, SL, TT, TL and SQ scenarios respectively. In all these tables, the first column shows the instance name. The digits at the last in the name of an instance indicate the number of vertices it contains. The second and third columns report the size of the inner cycle (k) and outer cycles (C) respectively. The fourth column reports the best value found for an instance (v_b) by both BRKGA and MSH (best of BRKGA and MSH). The fifth and sixth columns respectively report the average solution quality ($\bar{v}$) and average execution time in seconds (t) of BRKGA over 30 runs, whereas seventh and eighth columns do the same for MSH. For HH-ABC, we have reported the best solution obtained (v_b), average solution quality ($\bar{v}$), and average execution time in seconds (t) over 30 runs for each instance. In addition, we have reported the %-improvement in quality of best solution obtained through BRKGA/MSH by HH-ABC (%-imp-v_b), %-improvement in average solution quality obtained through BRKGA by HH-ABC (%-imp-$\bar{v}$.BRKGA) and %-improvement in average solution quality obtained through MSH by HH-ABC (%-imp-$\bar{v}$.MSH). The %-improvement in solution quality of a method A by method B is defined as $100 \times \frac{S_A - S_B}{S_A}$ where S_A and S_B are the solutions obtained by methods A and B respectively. The %-improvement values are in bold face whenever HH-ABC obtained as good as or better solution in comparison to BRKGA or MSH as the case may be. It is to be noted that the detailed results for BRKGA and MSH are taken from supplementary results of [3] available online[1] where they have reported only the best result among BRKGA and MSH (and not the separate best results for BRKGA and MSH). Further, %-deviation of BRKGA and MSH from best is also reported. From the best result and %-deviation, we have calculated the average solution quality for BRKGA and MSH.

[1]`www.loco.ic.unicamp.br/results/kimdmtsp`

Table 8.2: Comparison of various approaches for ST scenario

			Previous approaches					**Proposed approach**					
Instances	k	C	v_b	**BRKGA**		**MSH**		**HH-ABC**					
				$\bar{v}$	t	$\bar{v}$	t	v_b	$\bar{v}$	t	%-imp-v_b	%-imp-$\bar{v}.BRKGA$	%-imp-$\bar{v}.MSH$
ali535	107	5	1403446.00	1406393.24	904.79	1406393.24	1232.10	465537.00	491392.20	70.30	**66.83**	**65.06**	**65.06**
att48	10	5	16776.00	19423.25	73.72	31361.05	35.25	15909.00	16111.04	1.33	**5.17**	**17.05**	**48.63**
att532	107	5	175972.00	176235.96	695.18	176253.56	1010.67	57671.00	61189.80	71.26	**67.23**	**65.28**	**65.28**
bayg29	6	5	2223.00	2306.36	17.75	2877.01	13.19	2210.00	2212.73	0.42	**0.58**	**4.06**	**23.09**
bays29	6	5	2759.00	2888.40	16.00	4048.28	23.87	2751.00	2751.00	0.41	**0.29**	**4.76**	**32.05**
berlin52	11	5	12409.00	13481.14	107.85	16566.02	43.77	10820.00	10940.63	1.03	**12.81**	**18.84**	**33.96**
bier127	26	5	310919.00	323169.21	144.38	320215.48	218.67	181802.00	184915.90	4.06	**41.53**	**42.78**	**42.25**
brazil58	12	5	43250.00	52955.30	167.66	84977.60	113.28	34911.00	35968.80	1.04	**19.28**	**32.08**	**57.67**
brg180	36	5	345320.00	368905.36	492.51	421014.14	555.89	8860.00	16387.00	6.83	**97.43**	**95.56**	**96.11**
burma14	3	5	4428.00	4436.86	2.47	4447.48	4.38	4428.00	4428.00	0.14	**0.00**	**0.20**	**0.44**
ch130	26	5	20812.00	20934.79	114.25	20930.63	163.79	9928.00	10263.97	4.45	**52.30**	**50.97**	**50.96**
ch150	30	5	21111.00	21210.22	131.31	21214.44	189.38	11012.00	11303.87	5.77	**47.84**	**46.71**	**46.72**
d657	132	5	322367.00	323043.97	812.64	322947.26	1217.09	105693.00	111889.03	128.10	**67.21**	**65.36**	**65.35**
dantzig42	9	5	1153.00	1230.14	51.60	1644.52	27.17	1046.00	1061.60	0.75	**9.28**	**13.70**	**35.45**
eil101	21	5	1558.00	1670.96	92.04	1669.24	125.50	895.00	908.03	2.94	**42.55**	**45.66**	**45.60**
eil51	11	5	631.00	730.00	70.90	1068.60	36.23	590.00	600.21	0.90	**6.50**	**17.78**	**43.83**
eil76	16	5	944.00	1227.01	148.03	1365.87	70.51	771.00	776.60	1.90	**18.33**	**36.71**	**43.14**
fri26	6	5	1296.00	1380.76	14.38	1795.87	18.43	1280.00	1280.80	0.35	**1.23**	**7.24**	**28.68**
gil262	53	5	10774.00	10823.56	245.04	10822.48	348.93	4086.00	4266.00	16.44	**62.08**	**60.59**	**60.58**
gr120	24	5	23853.00	25000.33	125.75	25076.66	172.45	10550.00	10812.67	3.78	**55.77**	**56.75**	**56.88**
gr137	28	5	255489.00	255770.04	138.09	255770.04	217.29	110910.00	113588.97	4.97	**56.59**	**55.59**	**55.59**
gr17	4	5	2481.00	2535.09	5.28	2769.79	8.36	2481.00	2488.44	0.30	**0.00**	**1.84**	**10.16**
gr202	41	5	127079.00	130573.67	233.69	130090.77	372.27	62221.00	63954.43	9.21	**51.04**	**51.02**	**50.84**
gr21	5	5	3870.00	4047.63	9.71	4943.93	11.72	3870.00	3875.42	0.41	**0.00**	**4.25**	**21.61**
gr229	46	5	549056.00	552789.58	256.23	552405.24	413.57	231656.00	242970.20	11.79	**57.81**	**56.05**	**56.02**
gr24	5	5	1716.00	1738.65	13.65	2269.07	15.59	1716.00	1716.00	0.50	**0.00**	**1.30**	**24.37**
gr431	87	5	747273.00	752802.82	608.29	751681.91	971.99	317746.00	330286.20	42.28	**57.48**	**56.13**	**56.06**
gr48	10	5	7555.00	8812.15	69.31	14845.58	58.66	7085.00	7104.73	0.93	**6.22**	**19.38**	**52.14**
gr666	134	5	1724806.00	1730325.38	975.89	1731015.30	1345.69	640721.00	666681.30	125.98	**62.85**	**61.47**	**61.49**
gr96	20	5	133823.00	163531.71	114.87	173836.08	118.47	82975.00	84130.17	2.83	**38.00**	**48.55**	**51.60**
hk48	10	5	18765.00	20585.21	71.15	33771.37	62.95	16702.00	17029.00	0.90	**10.99**	**17.28**	**49.58**
kroA100	20	5	62746.00	62777.37	88.70	62777.37	130.27	34830.00	35923.77	2.93	**44.49**	**42.78**	**42.78**
kroA150	30	5	112327.00	112338.23	137.26	112338.23	195.04	44201.00	45720.87	5.78	**60.65**	**59.30**	**59.30**
kroA200	40	5	125867.00	126735.48	182.46	126685.14	266.02	51875.00	54056.17	9.47	**58.79**	**57.35**	**57.33**
kroB100	20	5	55864.00	72947.21	113.30	73902.49	134.26	35459.00	36294.50	2.90	**36.53**	**50.25**	**50.89**
kroB150	30	5	106388.00	108643.43	137.91	108728.54	195.52	42697.00	44355.73	5.63	**59.87**	**59.17**	**59.21**
kroB200	40	5	156552.00	156755.52	183.24	156739.86	271.08	51831.00	53861.90	9.51	**66.89**	**65.64**	**65.64**
kroC100	20	5	70945.00	71242.97	88.64	71228.78	129.43	34394.00	35287.57	2.91	**51.52**	**50.47**	**50.46**
kroD100	20	5	82294.00	82450.36	92.08	82442.13	131.82	33598.00	34967.73	2.92	**59.17**	**57.59**	**57.59**
kroE100	20	5	52419.00	74068.05	113.92	74822.88	133.18	35916.00	36747.20	2.85	**31.48**	**50.39**	**50.89**
lin105	21	5	41678.00	49105.02	158.02	49538.47	183.26	25451.00	26157.37	3.04	**38.93**	**46.73**	**47.20**
lin318	64	5	260132.00	260756.32	343.81	260600.24	492.89	80173.00	84314.07	23.04	**69.18**	**67.67**	**67.65**
nrw1379	276	5	430369.00	430842.41	2596.44	430928.48	3601.68	129789.00	139128.07	887.54	**69.84**	**67.71**	**67.71**
pa561	113	5	15471.00	15498.85	618.36	15506.58	895.15	5755.00	5982.40	80.71	**62.80**	**61.40**	**61.42**
pr107	22	5	146993.00	229529.57	447.07	247859.60	232.40	66649.00	67697.67	3.36	**54.66**	**70.51**	**72.69**
pr124	25	5	279457.00	328389.92	155.85	330485.85	183.14	108829.00	113227.57	4.14	**61.06**	**65.52**	**65.74**
pr136	28	5	374920.00	376382.19	136.25	376307.20	190.53	147515.00	151314.23	4.88	**60.65**	**59.80**	**59.79**
pr144	29	5	329380.00	329874.07	150.21	330005.82	196.16	100654.00	109565.53	5.22	**69.44**	**66.79**	**66.80**
pr152	31	5	408929.00	409378.82	262.20	409215.25	322.14	120041.00	124902.40	5.94	**70.65**	**69.49**	**69.48**
pr226	46	5	707932.00	709560.24	289.28	709277.07	372.61	148258.00	157302.67	12.00	**79.06**	**77.83**	**77.82**
pr264	53	5	411099.00	411921.20	630.10	411921.20	971.31	93946.00	99824.23	15.39	**77.15**	**75.77**	**75.77**
pr299	60	5	303013.00	304043.24	294.61	304225.05	442.30	93255.00	98360.07	20.42	**69.22**	**67.65**	**67.67**
pr439	88	5	654763.00	656138.00	601.34	656072.53	858.01	244524.00	255653.53	45.24	**62.65**	**61.04**	**61.03**
pr76	16	5	221016.00	265462.32	69.81	267827.19	74.56	158543.00	161186.27	1.93	**28.27**	**39.28**	**39.82**
rd100	20	5	22100.00	22159.67	84.77	22164.09	120.85	12475.00	12713.40	2.89	**43.55**	**42.63**	**42.64**
rd400	80	5	75434.00	75562.24	405.51	75554.69	577.44	31551.00	32815.07	38.95	**58.17**	**56.57**	**56.57**
si175	35	5	37171.00	40166.98	861.10	46385.69	446.40	28551.00	29070.80	6.37	**23.19**	**27.63**	**37.33**
si535	107	5	119258.00	131851.64	3381.47	146568.08	2141.13	70538.00	71629.73	55.46	**40.85**	**45.67**	**51.13**
st70	14	5	1165.00	1510.77	161.36	1823.34	68.49	990.00	1011.03	1.61	**15.02**	**33.08**	**44.55**
swiss42	9	5	1912.00	2209.12	51.12	3388.83	50.55	1839.00	1846.60	0.77	**3.82**	**16.41**	**45.51**
ts225	45	5	635554.00	637587.77	203.99	637587.77	289.16	276450.00	285473.60	12.04	**56.50**	**55.23**	**55.23**
ulysses16	4	4	8873.00	8873.00	4.38	9081.52	9.33	8873.00	8873.00	0.23	**0.00**	**0.00**	**2.30**
ulysses22	5	5	8461.00	8618.37	8.95	10776.78	12.63	8461.00	8461.00	0.29	**0.00**	**1.83**	**21.49**

Table 8.3: Comparison of various approaches for SL scenario

			Previous approaches					**Proposed approach**					
				BRKGA		**MSH**		**HH-ABC**					
Instances	k	C	v_b	$\bar{v}$	t	$\bar{v}$	t	v_b	$\bar{v}$	t	%-imp-v_b	%-imp-$\bar{v}.BRKGA$	%-imp-$\bar{v}.MSH$
ali535	107	10	478199.00	480590.00	907.05	480255.26	1223.84	276771.00	281410.83	82.96	**42.12**	**41.44**	**41.40**
att48	10	10	13985.00	14438.11	34.73	14793.33	30.16	12667.00	12972.10	0.79	**9.42**	**10.15**	**12.31**
att532	107	10	56985.00	57281.32	678.73	57315.51	969.71	37920.00	38317.03	82.11	**33.46**	**33.11**	**33.15**
bayg29	6	10	1805.00	1866.91	12.81	2483.50	10.09	1805.00	1841.40	0.39	**0.00**	**1.37**	**25.85**
bays29	6	10	2264.00	2328.07	15.28	3108.92	10.70	2264.00	2286.41	0.73	**0.00**	**1.79**	**26.46**
berlin52	11	10	9642.00	11253.18	72.34	12151.81	40.51	9443.00	9683.80	0.80	**2.06**	**13.95**	**20.31**
bier127	26	10	217101.00	218447.03	143.87	218360.19	209.83	152126.00	155000.20	3.82	**29.93**	**29.04**	**29.02**
brazil58	12	10	32668.00	36594.69	180.00	69968.32	104.70	30991.00	31263.72	0.95	**5.13**	**14.57**	**55.32**
brg180	36	10	122070.00	142345.83	408.41	267235.64	625.74	3500.00	3581.00	6.08	**97.13**	**97.48**	**98.66**
burma14	3	10	3540.00	3548.50	1.73	3587.79	3.38	3540.00	3579.65	0.22	**0.00**	-0.88	**0.23**
ch130	26	10	8858.00	8928.86	98.92	8912.92	137.35	8011.00	8198.00	4.11	**9.56**	**8.19**	**8.02**
ch150	30	10	9745.00	9816.14	111.96	9871.69	156.64	8638.00	8817.90	5.13	**11.36**	**10.17**	**10.67**
d657	132	10	120426.00	121220.81	784.22	121136.51	1093.58	67354.00	68055.00	138.80	**44.07**	**43.86**	**43.82**
dantzig42	9	10	876.00	954.14	40.36	1087.03	21.59	836.00	859.74	1.27	**4.57**	**9.89**	**20.91**
eil101	21	10	905.00	914.50	74.55	915.77	105.41	760.00	772.40	2.56	**16.02**	**15.54**	**15.66**
eil51	11	10	562.00	608.48	45.58	714.25	29.23	495.00	513.56	0.76	**11.92**	**15.60**	**28.10**
eil76	16	10	798.00	817.71	42.09	816.99	59.35	645.00	658.30	1.61	**19.17**	**19.49**	**19.42**
fri26	6	10	1095.00	1122.59	10.35	1486.68	15.60	1095.00	1095.99	0.59	**0.00**	**2.37**	**26.28**
gil262	53	10	3948.00	4002.09	211.88	4001.30	293.30	3167.00	3227.87	14.27	**19.78**	**19.35**	**19.33**
gr120	24	10	10205.00	10313.17	105.29	10300.93	155.72	8689.00	8810.83	3.40	**14.86**	**14.57**	**14.47**
gr137	28	10	118373.00	119627.75	129.28	119627.75	199.09	93665.00	95760.07	4.41	**20.87**	**19.95**	**19.95**
gr17	4	10	2155.00	2155.22	3.29	2363.17	6.93	2155.00	2176.33	0.30	**0.00**	-0.98	**7.91**
gr202	41	10	66222.00	67063.02	224.94	67089.51	355.92	51216.00	52170.33	8.61	**22.66**	**22.21**	**22.24**
gr21	5	10	3186.00	3206.39	6.47	3854.10	10.00	3186.00	3252.59	0.43	**0.00**	-1.44	**15.61**
gr229	46	10	214706.00	218162.77	252.70	217754.83	383.42	175203.00	177055.27	10.95	**18.40**	**18.84**	**18.69**
gr24	5	10	1470.00	1485.73	7.60	1845.14	14.14	1460.00	1474.02	0.54	**0.68**	**0.79**	**20.11**
gr431	87	10	269601.00	272917.09	631.74	274130.30	962.22	226474.00	228759.87	43.32	**16.00**	**16.18**	**16.55**
gr48	10	10	6245.00	6996.27	55.91	11868.00	40.61	5900.00	6100.01	1.63	**5.52**	**12.81**	**48.60**
gr666	134	10	621541.00	625705.32	953.58	627259.18	1277.53	401729.00	406667.53	135.91	**35.37**	**35.01**	**35.17**
gr96	20	10	79811.00	80896.43	73.57	80752.77	100.96	72583.00	73866.27	2.45	**9.06**	**8.69**	**8.53**
hk48	10	10	14029.00	15481.00	62.26	27433.71	52.32	13579.00	13809.84	1.53	**3.21**	**10.79**	**49.66**
kroA100	20	10	31405.00	31552.60	80.24	31480.37	112.19	28123.00	28788.77	2.59	**10.45**	**8.76**	**8.55**
kroA150	30	10	43554.00	43571.42	121.59	43575.78	168.96	34689.00	35338.77	5.15	**20.35**	**18.89**	**18.90**
kroA200	40	10	48249.00	48495.07	165.48	48437.17	228.21	38649.00	39407.40	8.62	**19.90**	**18.74**	**18.64**
kroB100	20	10	40344.00	40424.69	84.73	40432.76	119.26	28992.00	29393.33	2.65	**28.14**	**27.29**	**27.30**
kroB150	30	10	37335.00	37394.74	123.59	37391.00	170.20	34262.00	34575.37	5.15	**8.23**	**7.54**	**7.53**
kroB200	40	10	67539.00	67674.08	168.75	67667.32	236.96	39134.00	39806.93	8.70	**42.06**	**41.18**	**41.17**
kroC100	20	10	31616.00	31856.28	79.57	31875.25	111.31	27815.00	28504.47	2.59	**12.02**	**10.52**	**10.57**
kroD100	20	10	35187.00	35218.67	81.10	35239.78	114.50	27444.00	27819.43	2.62	**22.01**	**21.01**	**21.06**
kroE100	20	10	32259.00	32375.13	85.50	32391.26	115.00	29167.00	29642.30	2.68	**9.58**	**8.44**	**8.49**
lin105	21	10	30369.00	30472.25	130.81	30454.03	168.36	18957.00	19365.70	2.71	**37.58**	**36.45**	**36.41**
lin318	64	10	101604.00	102294.91	313.05	102142.50	423.90	56396.00	56827.00	20.91	**44.49**	**44.45**	**44.36**
nrw1379	276	10	86506.00	87241.30	2469.87	87120.19	3505.54	78384.00	78971.40	733.28	**9.39**	**9.48**	**9.35**
pa561	113	10	6337.00	6378.82	560.86	6377.56	793.51	3802.00	3848.07	88.01	**40.00**	**39.67**	**39.66**
pr107	22	10	92314.00	93384.84	194.37	93440.23	214.31	53035.00	53173.00	2.87	**42.55**	**43.06**	**43.09**
pr124	25	10	198997.00	213702.88	127.67	214299.87	154.59	82513.00	83664.77	3.71	**58.54**	**60.85**	**60.96**
pr136	28	10	142115.00	143109.81	117.91	143365.61	163.18	114980.00	117002.07	4.35	**19.09**	**18.24**	**18.39**
pr144	29	10	92727.00	93079.36	136.76	93023.73	165.79	73821.00	74303.53	4.68	**20.39**	**20.17**	**20.12**
pr152	31	10	170696.00	171464.13	243.39	171412.92	298.28	89773.00	91964.93	5.39	**47.41**	**46.36**	**46.35**
pr226	46	10	433901.00	434682.02	262.33	434595.24	324.59	106824.00	108134.97	11.33	**75.38**	**75.12**	**75.12**
pr264	53	10	170593.00	170831.83	636.78	170729.47	940.68	63742.00	64534.80	14.52	**62.64**	**62.22**	**62.20**
pr299	60	10	134869.00	135489.40	284.95	135502.88	397.05	67428.00	68181.43	19.47	**50.00**	**49.68**	**49.68**
pr439	88	10	171725.00	172995.77	599.86	173150.32	806.42	145333.00	148100.37	47.60	**15.37**	**14.39**	**14.47**
pr76	16	10	165235.00	165532.42	44.98	165499.38	69.48	141853.00	146327.13	1.70	**14.15**	**11.60**	**11.58**
rd100	20	10	13735.00	13744.61	73.32	13747.36	99.20	10292.00	10479.97	2.65	**25.07**	**23.75**	**23.77**
rd400	80	10	27035.00	27156.66	355.50	27159.36	496.06	20705.00	20923.67	37.71	**23.41**	**22.95**	**22.96**
si175	35	10	33659.00	35173.66	418.61	41063.98	338.96	24911.00	25108.77	6.93	**25.99**	**28.61**	**38.85**
si535	107	10	108002.00	110140.44	3196.19	129386.40	2112.14	56794.00	57044.77	66.79	**47.41**	**48.21**	**55.91**
st70	14	10	944.00	958.25	38.52	959.01	50.14	842.00	849.40	1.45	**10.81**	**11.36**	**11.43**
swiss42	9	10	1545.00	1596.29	35.67	2785.02	37.10	1535.00	1581.05	2.09	**0.65**	**0.95**	**43.23**
ts225	45	10	226870.00	227210.31	174.24	227459.86	240.28	205998.00	209714.77	10.04	**9.20**	**7.70**	**7.80**
ulysses16	4	8	7081.00	7092.33	3.14	7658.10	5.92	7081.00	7147.56	0.28	**0.00**	-0.78	**6.67**
ulysses22	5	10	7392.00	7409.00	9.48	9246.65	12.80	7392.00	7414.18	0.46	**0.00**	-0.07	**19.82**

Table 8.4: Comparison of various approaches for LT scenario

			Previous approaches					**Proposed approach**					
				BRKGA		**MSH**		**HH-ABC**					
Instances	k	C	v_b	$\bar{v}$	t	$\bar{v}$	t	v_b	$\bar{v}$	t	%-imp-v_b	%-imp-$\bar{v}.BRKGA$	%-imp-$\bar{v}.MSH$
ali535	268	2	1976741.00	1978322.39	1966.12	1978322.39	1966.12	389290.00	431664.77	45.04	**80.31**	**78.18**	**78.18**
att48	24	2	17290.00	18955.03	49.72	18955.03	49.72	16046.00	16413.45	1.03	**7.19**	**13.41**	**13.41**
att532	266	2	215450.00	234689.69	1537.56	225188.34	2205.06	55759.00	59666.13	45.16	**74.12**	**74.58**	**73.50**
bayg29	15	2	2251.00	2378.63	6.66	3370.65	10.13	2251.00	2261.27	0.30	**0.00**	**4.93**	**32.91**
bays29	15	2	2810.00	2967.08	5.40	4155.99	11.46	2681.00	2710.43	0.29	**4.59**	**8.65**	**34.78**
berlin52	26	2	12439.00	14062.29	107.19	18716.96	42.31	11711.00	12016.53	0.65	**5.85**	**14.55**	**35.80**
bier127	64	2	388875.00	397624.69	202.04	405791.06	315.61	183370.00	187985.70	2.70	**52.85**	**52.72**	**53.67**
brazil58	29	2	39324.00	49556.10	400.48	89894.66	239.93	36641.00	37689.03	0.77	**6.82**	**23.95**	**58.07**
brg180	90	2	517910.00	576226.67	737.10	596476.95	1047.88	3060.00	3345.00	5.07	**99.41**	**99.42**	**99.44**
burma14	7	2	4552.00	4560.19	1.54	4598.89	3.48	4552.00	4552.00	0.10	**0.00**	**0.18**	**1.02**
ch130	65	2	14347.00	23385.61	244.52	26336.79	136.99	9879.00	10514.40	2.84	**31.14**	**55.04**	**60.08**
ch150	75	2	16256.00	29644.44	156.56	31487.87	140.63	11444.00	11889.30	3.72	**29.60**	**59.89**	**62.24**
d657	329	2	361342.00	372977.21	2579.54	367448.68	3602.22	103510.00	107686.90	79.33	**71.35**	**71.13**	**70.69**
dantzig42	21	2	1181.00	1292.96	24.65	1938.02	16.84	1051.00	1084.00	0.84	**11.01**	**16.16**	**44.07**
eil101	51	2	1295.00	1822.71	121.55	2077.96	76.01	977.00	1000.43	1.84	**24.56**	**45.11**	**51.86**
eil51	26	2	700.00	761.60	26.20	1241.80	22.41	622.00	644.07	0.65	**11.14**	**15.43**	**48.13**
eil76	38	2	963.00	1121.90	94.86	1498.33	37.26	823.00	836.77	1.14	**14.54**	**25.41**	**44.15**
fri26	13	2	1429.00	1472.44	8.04	1914.57	11.94	1429.00	1433.70	0.24	**0.00**	**2.63**	**25.12**
gil262	131	2	13100.00	13119.65	264.27	13258.51	378.17	4237.00	4548.80	10.28	**67.66**	**65.33**	**65.69**
gr120	60	2	14374.00	17621.09	570.09	26587.59	139.36	10762.00	11225.70	2.44	**25.13**	**36.29**	**57.78**
gr137	69	2	155397.00	175458.75	1112.15	358531.96	204.97	113737.00	117793.93	3.19	**26.81**	**32.87**	**67.15**
gr17	9	2	2733.00	2750.49	3.26	2967.76	6.34	2733.00	2734.33	0.14	**0.00**	**0.59**	**7.87**
gr202	101	2	166493.00	168957.10	387.86	169556.47	651.88	66848.00	69371.97	6.17	**59.85**	**58.94**	**59.09**
gr21	11	2	4112.00	4334.05	4.31	5056.53	7.65	4080.00	4084.27	0.19	**0.78**	**5.76**	**19.23**
gr229	115	2	443023.00	691293.09	487.01	719735.17	524.67	224602.00	232558.07	8.02	**49.30**	**66.36**	**67.69**
gr24	12	2	1848.00	1979.21	4.66	2516.98	8.39	1848.00	1859.50	0.22	**0.00**	**6.05**	**26.12**
gr431	216	2	913594.00	1209233.02	1838.97	1290360.17	1890.32	312155.00	324045.77	27.76	**65.83**	**73.20**	**74.89**
gr48	24	2	7713.00	8488.93	21.64	15802.39	29.38	7077.00	7367.40	0.58	**8.25**	**13.21**	**53.38**
gr666	333	2	2479425.00	2482648.25	2750.84	2518599.92	3601.83	622408.00	646129.53	82.54	**74.90**	**73.97**	**74.35**
gr96	48	2	105491.00	118107.72	475.19	221404.51	95.03	84801.00	91195.53	1.67	**19.61**	**22.79**	**58.81**
hk48	24	2	20223.00	21679.06	34.77	36423.65	37.21	18499.00	18850.17	0.59	**8.52**	**13.05**	**48.25**
kroA100	50	2	45063.00	49663.93	435.33	97660.53	85.90	34726.00	36737.23	1.81	**22.94**	**26.03**	**62.38**
kroA150	75	2	65467.00	74874.61	827.95	161932.62	165.54	45908.00	48407.33	3.70	**29.88**	**35.35**	**70.11**
kroA200	100	2	82026.00	97299.24	1532.72	186871.63	268.27	52358.00	55883.27	6.23	**36.17**	**42.57**	**70.10**
kroB100	50	2	47210.00	50330.58	415.20	108394.16	92.05	35341.00	37440.97	1.81	**25.14**	**25.61**	**65.46**
kroB150	75	2	64545.00	71070.50	803.40	141172.82	159.02	44361.00	46079.37	3.73	**31.27**	**35.16**	**67.36**
kroB200	100	2	84458.00	112118.00	1334.21	181922.53	274.71	51543.00	54363.93	6.24	**38.97**	**51.51**	**70.12**
kroC100	50	2	44181.00	48422.38	418.13	115104.76	83.42	35246.00	36648.57	1.80	**20.22**	**24.31**	**68.16**
kroD100	50	2	43451.00	49012.73	469.82	94084.45	94.59	34778.00	36844.10	1.80	**19.96**	**24.83**	**60.84**
kroE100	50	2	43988.00	57136.01	344.34	82640.26	85.80	34525.00	37008.87	1.81	**21.51**	**35.23**	**55.22**
lin105	53	2	32814.00	38077.37	797.12	70855.27	146.82	21625.00	23218.70	2.00	**34.10**	**39.02**	**67.23**
lin318	159	2	297594.00	298308.23	542.40	298337.99	797.74	80325.00	85904.77	14.79	**73.01**	**71.20**	**71.21**
nrw1379	690	2	554094.00	554592.68	3609.57	554481.87	3609.13	128751.00	134539.70	761.61	**76.76**	**75.74**	**75.74**
pa561	281	2	18754.00	18785.88	1457.39	18787.76	1960.84	5569.00	5834.47	51.82	**70.31**	**68.94**	**68.95**
pr107	54	2	68663.00	83789.46	3406.54	281827.28	621.73	61142.00	64725.90	2.10	**10.95**	**22.75**	**77.03**
pr124	62	2	154141.00	223982.29	947.11	308328.24	255.65	109899.00	117027.13	2.63	**28.70**	**47.75**	**62.04**
pr136	68	2	204334.00	340604.34	519.45	401761.51	199.38	163613.00	169638.20	3.08	**19.93**	**50.19**	**57.78**
pr144	72	2	312468.00	312842.96	295.62	312874.21	376.49	108336.00	119773.67	3.50	**65.33**	**61.71**	**61.72**
pr152	76	2	464736.00	557032.57	1055.17	560332.20	1358.87	122202.00	136512.37	3.86	**73.71**	**75.49**	**75.64**
pr226	113	2	341413.00	623078.73	1472.57	714474.99	933.10	144344.00	165123.23	7.98	**57.72**	**73.50**	**76.89**
pr264	132	2	243113.00	274085.60	3600.64	548001.01	2211.72	90020.00	97727.27	10.53	**62.97**	**64.34**	**82.17**
pr299	150	2	186292.00	256039.72	2549.49	388437.45	576.01	92631.00	97017.07	13.33	**50.28**	**62.11**	**75.02**
pr439	220	2	886444.00	887507.73	1175.74	887862.31	1619.01	212502.00	229947.17	28.78	**76.03**	**74.09**	**74.10**
pr76	38	2	232517.00	303643.95	121.43	325965.58	71.76	170832.00	177815.33	1.16	**26.53**	**41.44**	**45.45**
rd100	50	2	16096.00	17974.40	397.66	32647.52	85.99	12944.00	13489.40	1.81	**19.58**	**24.95**	**58.68**
rd400	200	2	105561.00	105751.01	699.33	105772.12	972.36	30490.00	31693.40	23.46	**71.12**	**70.03**	**70.04**
si175	88	2	38717.00	41245.22	1103.57	54230.90	406.61	32976.00	33559.17	4.93	**14.83**	**18.64**	**38.12**
si535	268	2	125285.00	130647.20	3601.30	170525.41	3577.29	77890.00	79317.60	45.24	**37.83**	**39.29**	**53.49**
st70	35	2	1205.00	1358.40	90.33	2051.27	35.03	1038.00	1083.10	1.03	**13.86**	**20.27**	**47.20**
swiss42	21	2	2070.00	2351.93	17.91	3682.32	23.84	1927.00	1954.27	0.48	**6.91**	**16.91**	**46.93**
ts225	113	2	761726.00	763782.66	194.58	764087.35	292.83	253398.00	264282.93	7.75	**66.73**	**65.40**	**65.41**
ulysses16	8	2	9098.00	9157.14	3.71	9471.02	5.84	9098.00	9106.60	0.13	**0.00**	**0.55**	**3.85**
ulysses22	11	2	9420.00	10072.81	9.72	11532.91	17.05	9420.00	9472.70	0.19	**0.00**	**5.96**	**17.86**

Table 8.5: Comparison of various approaches for LL scenario

			Previous approaches					Proposed approach					
				BRKGA		MSH		HH-ABC					
Instances	k	C	v_b	$\bar{v}$	t	$\bar{v}$	t	v_b	$\bar{v}$	t	%-imp-v_b	%-imp-$\bar{v}.BRKGA$	%-imp-$\bar{v}.MSH$
ali535	268	4	493588.00	495414.28	2017.00	495315.56	2954.24	260873.00	264513.50	90.69	**47.15**	**46.61**	**46.60**
att48	24	4	14624.00	14947.19	47.51	15097.82	40.86	12978.00	13349.90	0.75	**11.26**	**10.69**	**11.58**
att532	266	4	81028.00	81344.01	1633.64	81344.01	2333.58	36187.00	36940.10	92.20	**55.34**	**54.59**	**54.59**
bayg29	15	4	1959.00	2102.20	14.64	2364.90	8.07	1932.00	1989.77	0.63	**1.38**	**5.35**	**15.86**
bays29	15	4	2443.00	2579.08	11.97	3028.34	8.49	2432.00	2476.67	0.35	**0.45**	**3.97**	**18.22**
berlin52	26	4	10492.00	12339.64	53.57	12834.86	48.37	9246.00	9464.21	0.75	**11.88**	**23.30**	**26.26**
bier127	64	4	164116.00	166200.27	217.26	166151.04	343.62	144453.00	147699.77	3.76	**11.98**	**11.13**	**11.11**
brazil58	29	4	31719.00	38291.18	539.32	73521.47	256.67	29318.00	29897.03	1.02	**7.57**	**21.92**	**59.34**
brg180	90	4	333670.00	363633.57	794.45	437908.51	1105.76	2600.00	2659.00	6.82	**99.22**	**99.27**	**99.39**
burma14	7	4	3819.00	3870.17	2.84	4111.92	2.91	3819.00	3871.70	0.21	**0.00**	-0.04	**5.84**
ch130	65	4	9363.00	9480.04	117.89	9462.25	168.35	7743.00	7845.23	3.99	**17.30**	**17.24**	**17.09**
ch150	75	4	13620.00	13780.72	124.24	13771.18	177.42	8639.00	8737.97	5.10	**36.57**	**36.59**	**36.55**
d657	329	4	102304.00	102856.44	2999.82	102856.44	3602.05	65107.00	65742.23	151.32	**36.36**	**36.08**	**36.08**
dantzig42	21	4	932.00	1001.06	33.47	1078.04	21.12	852.00	891.19	1.13	**8.58**	**10.98**	**17.33**
eil101	51	4	991.00	1003.39	71.12	1001.41	100.34	807.00	820.57	2.48	**18.57**	**18.22**	**18.06**
eil51	26	4	579.00	612.70	51.34	776.55	22.16	535.00	545.53	0.82	**7.60**	**10.96**	**29.75**
eil76	38	4	768.00	811.32	41.16	810.16	51.62	697.00	703.23	1.53	**9.24**	**13.32**	**13.20**
fri26	13	4	1157.00	1210.92	12.04	1656.13	14.20	1116.00	1148.92	0.53	**3.54**	**5.12**	**30.63**
gil262	131	4	4647.00	4676.74	321.32	4670.23	445.47	3128.00	3145.77	16.01	**32.69**	**32.74**	**32.64**
gr120	60	4	10887.00	11024.18	107.55	11003.49	168.25	8831.00	8922.03	3.36	**18.88**	**19.07**	**18.92**
gr137	69	4	103672.00	104884.96	150.00	104947.17	237.89	91186.00	92043.93	4.42	**12.04**	**12.24**	**12.29**
gr17	9	4	2273.00	2336.64	5.17	2677.59	6.23	2273.00	2304.82	0.27	**0.00**	**1.36**	**13.92**
gr202	101	4	73467.00	74429.42	444.68	74289.83	706.44	50356.00	50956.63	8.80	**31.46**	**31.54**	**31.41**
gr21	11	4	3256.00	3425.31	6.86	4366.30	8.52	3256.00	3435.08	0.38	**0.00**	-0.29	**21.33**
gr229	115	4	239960.00	242383.60	387.24	242023.66	577.72	169278.00	173513.77	11.41	**29.46**	**28.41**	**28.31**
gr24	12	4	1559.00	1605.30	10.10	2152.20	10.99	1512.00	1559.63	0.46	**3.01**	**2.84**	**27.53**
gr431	216	4	405363.00	407551.96	1296.35	407227.67	1994.08	220785.00	224159.43	53.98	**45.53**	**45.00**	**44.95**
gr48	24	4	6220.00	6913.53	61.59	12886.60	35.23	6073.00	6228.13	0.75	**2.36**	**9.91**	**51.67**
gr666	333	4	658983.00	662409.71	2845.84	662146.12	3602.04	390047.00	396706.23	150.15	**40.81**	**40.11**	**40.09**
gr96	48	4	77422.00	77716.20	79.12	77677.49	117.14	68749.00	70074.53	2.33	**11.20**	**9.83**	**9.79**
hk48	24	4	15338.00	17296.66	77.50	29493.44	49.33	14363.00	14858.52	1.42	**6.36**	**14.10**	**49.62**
kroA100	50	4	38543.00	38701.03	72.91	38693.32	110.31	27114.00	27474.10	2.50	**29.65**	**29.01**	**29.00**
kroA150	75	4	52564.00	52995.02	139.62	52963.49	202.57	34290.00	34631.97	5.08	**34.77**	**34.65**	**34.61**
kroA200	100	4	50933.00	51075.61	213.85	51075.61	320.16	38160.00	38626.03	8.97	**25.08**	**24.37**	**24.37**
kroB100	50	4	39313.00	39635.37	80.50	39639.30	116.06	27685.00	28048.27	2.52	**29.58**	**29.23**	**29.24**
kroB150	75	4	48190.00	48556.24	130.35	48565.88	197.29	33601.00	33877.13	5.20	**30.27**	**30.23**	**30.24**
kroB200	100	4	56982.00	57187.14	221.19	57227.02	324.15	38005.00	38632.23	8.77	**33.30**	**32.45**	**32.49**
kroC100	50	4	39277.00	39434.11	71.57	39390.90	106.68	26929.00	27204.80	2.51	**31.44**	**31.01**	**30.94**
kroD100	50	4	32942.00	32955.18	77.25	32968.35	119.05	27268.00	27595.83	2.50	**17.22**	**16.26**	**16.30**
kroE100	50	4	40033.00	40045.01	74.50	40041.01	108.99	27658.00	28099.77	2.48	**30.91**	**29.83**	**29.82**
lin105	53	4	29518.00	29571.13	110.74	29559.33	170.96	17618.00	17969.97	2.69	**40.31**	**39.23**	**39.21**
lin318	159	4	77281.00	78223.83	605.66	78208.37	875.51	53813.00	54274.53	24.63	**30.37**	**30.62**	**30.60**
nrw1379	690	4	114921.00	115323.22	3608.99	115357.70	3608.96	78971.00	79999.33	1060.87	**31.28**	**30.63**	**30.65**
pa561	281	4	6793.00	6849.38	1558.99	6841.91	2098.61	3825.00	3857.50	103.30	**43.69**	**43.68**	**43.62**
pr107	54	4	77943.00	78075.50	436.16	78122.27	645.35	51303.00	51782.50	2.85	**34.18**	**33.68**	**33.72**
pr124	62	4	84521.00	85577.51	188.34	85467.64	284.48	77351.00	79426.73	3.60	**8.48**	**7.19**	**7.07**
pr136	68	4	222771.00	223639.81	150.64	223617.53	232.28	122637.00	126426.70	4.10	**44.95**	**43.47**	**43.46**
pr144	72	4	88436.00	88745.53	310.47	88683.62	409.82	70777.00	71370.00	4.80	**19.97**	**19.58**	**19.52**
pr152	76	4	123344.00	125527.19	1003.86	125033.81	1383.43	88164.00	89767.83	5.41	**28.52**	**28.49**	**28.21**
pr226	113	4	193582.00	194685.42	673.41	194666.06	983.87	98598.00	100040.47	11.81	**49.07**	**48.61**	**48.61**
pr264	132	4	95447.00	95905.15	1492.84	95838.33	2271.21	65802.00	66440.07	15.71	**31.06**	**30.72**	**30.67**
pr299	150	4	90494.00	91371.79	436.06	91136.51	647.96	65700.00	66465.27	21.87	**27.40**	**27.26**	**27.07**
pr439	220	4	262339.00	263676.93	1231.71	263860.57	1725.28	142129.00	143665.87	56.02	**45.82**	**45.51**	**45.55**
pr76	38	4	148851.00	149669.68	54.95	149758.99	85.37	136512.00	138699.97	1.54	**8.29**	**7.33**	**7.38**
rd100	50	4	14816.00	14941.94	77.24	14968.60	110.12	10056.00	10181.03	2.48	**32.13**	**31.86**	**31.98**
rd400	200	4	35972.00	36173.44	777.65	36191.43	1072.71	20441.00	20578.33	45.60	**43.18**	**43.11**	**43.14**
si175	88	4	33123.00	36488.30	769.50	43897.91	476.97	26474.00	26702.23	5.98	**20.07**	**26.82**	**39.17**
si535	268	4	101823.00	106384.67	3601.41	136778.84	3573.52	60873.00	61242.40	76.41	**40.22**	**42.43**	**55.23**
st70	35	4	1039.00	1241.61	39.11	1253.24	49.91	821.00	844.20	1.35	**20.98**	**32.01**	**32.64**
swiss42	21	4	1656.00	1863.00	39.88	3046.21	34.84	1548.00	1593.77	0.61	**6.52**	**14.45**	**47.68**
ts225	113	4	241754.00	244002.31	240.54	244413.29	350.15	196061.00	197378.43	10.70	**18.90**	**19.11**	**19.24**
ulysses16	8	4	7692.00	7800.46	4.52	8425.05	7.06	7692.00	7809.69	0.25	**0.00**	-0.12	**7.30**
ulysses22	11	4	7654.00	8052.77	11.01	8181.36	10.81	7654.00	7898.16	0.41	**0.00**	**1.92**	**3.46**

Table 8.6: Comparison of various approaches for SQ scenario

			Previous approaches					Proposed approach					
Instances	k	C	v_b	BRKGA		MSH		HH-ABC					
				$\bar{v}$	t	$\bar{v}$	t	v_b	$\bar{v}$	t	%-imp-v_b	%-imp-$\bar{v}.BRKGA$	%-imp-$\bar{v}.MSH$
ali535	24	24	495241.00	502620.09	1349.58	502620.09	1349.58	270581.00	276845.53	76.97	**45.36**	**44.92**	**44.92**
att48	7	7	15685.00	16906.86	42.02	16906.86	42.02	14870.00	15088.83	0.86	**5.20**	**10.75**	**10.75**
att532	24	24	78677.00	79188.40	1067.56	79164.80	1049.25	35802.00	36387.47	72.71	**54.49**	**54.05**	**54.04**
bayg29	6	6	2086.00	2156.51	16.45	2489.64	11.54	2072.00	2076.63	0.40	**0.67**	**3.70**	**16.59**
bays29	6	6	2596.00	2691.79	15.02	3773.29	11.17	2581.00	2589.13	0.39	**0.58**	**3.81**	**31.38**
berlin52	8	8	10496.00	11070.13	72.96	14986.19	35.91	9690.00	9864.17	1.02	**7.68**	**10.89**	**34.18**
bier127	12	12	191417.00	234619.82	293.45	247023.64	192.95	146703.00	148970.50	4.75	**23.36**	**36.51**	**39.69**
brazil58	8	8	38758.00	41722.99	101.72	69055.13	83.25	32993.00	33652.70	1.21	**14.87**	**19.34**	**51.27**
brg180	14	14	5210.00	33419.55	521.98	83754.92	381.60	3590.00	3711.33	8.21	**31.09**	**88.89**	**95.57**
burma14	4	4	4320.00	4339.44	3.80	4520.02	6.87	4320.00	4320.00	0.12	**0.00**	**0.45**	**4.43**
ch130	12	12	13119.00	13154.42	101.89	13151.80	115.05	8014.00	8109.93	4.60	**38.91**	**38.35**	**38.34**
ch150	13	13	14393.00	16531.80	134.28	16559.15	130.07	8428.00	8525.83	5.76	**41.44**	**48.43**	**48.51**
d657	26	26	115829.00	116234.40	834.84	116164.90	656.48	62369.00	63841.60	140.45	**46.15**	**45.08**	**45.04**
dantzig42	7	7	1019.00	1078.92	35.34	1255.71	23.06	962.00	976.97	0.72	**5.59**	**9.45**	**22.20**
eil101	11	11	926.00	939.52	74.34	940.45	92.65	765.00	773.27	2.89	**17.39**	**17.70**	**17.78**
eil51	8	8	548.00	604.12	52.14	871.27	30.53	532.00	537.53	0.97	**2.92**	**11.02**	**38.31**
eil76	9	9	725.00	766.47	129.77	914.81	54.47	680.00	686.13	1.86	**6.21**	**10.48**	**25.00**
fri26	6	6	1211.00	1250.48	12.05	1732.94	16.52	1151.00	1161.59	0.59	**4.95**	**7.11**	**32.97**
gil262	17	17	5847.00	5859.86	230.68	5861.62	218.67	3011.00	3048.07	15.28	**48.50**	**47.98**	**48.00**
gr120	11	11	11319.00	13873.70	260.88	14819.97	141.04	9102.00	9301.17	3.47	**19.59**	**32.96**	**37.24**
gr137	12	12	141900.00	142609.50	149.61	142822.35	188.44	94292.00	95994.90	4.60	**33.55**	**32.69**	**32.79**
gr17	5	5	2403.00	2439.29	4.39	2857.89	7.10	2403.00	2430.39	0.29	**0.00**	**0.36**	**14.96**
gr202	15	15	84018.00	85328.68	243.62	85127.04	309.85	48180.00	49204.90	10.59	**42.66**	**42.33**	**42.20**
gr21	5	5	3870.00	4039.89	7.99	5001.59	11.06	3870.00	3875.42	0.42	**0.00**	**4.07**	**22.52**
gr229	16	16	274928.00	275312.90	309.49	275120.45	360.87	168099.00	169845.47	11.46	**38.86**	**38.31**	**38.27**
gr24	5	5	1716.00	1740.54	12.10	2277.13	15.55	1716.00	1716.00	0.49	**0.00**	**1.41**	**24.64**
gr431	21	21	392557.00	395108.62	1089.82	395344.15	1363.97	218543.00	223636.37	42.13	**44.33**	**43.40**	**43.43**
gr48	7	7	7553.00	8222.95	49.85	12594.63	40.05	6771.00	6912.51	1.52	**10.35**	**15.94**	**45.12**
gr666	26	26	716090.00	720100.10	1341.77	720458.15	1224.26	404877.00	416887.13	144.37	**43.46**	**42.11**	**42.14**
gr96	10	10	125671.00	126513.00	73.02	126374.76	89.21	77254.00	78163.40	2.71	**38.53**	**38.22**	**38.15**
hk48	7	7	16699.00	17973.13	45.00	28503.52	46.70	15787.00	15902.93	0.86	**5.46**	**11.52**	**44.21**
kroA100	10	10	38157.00	52160.62	180.99	56075.53	101.95	31640.00	32013.93	2.44	**17.08**	**38.62**	**42.91**
kroA150	13	13	67920.00	67981.13	138.59	67981.13	158.14	35224.00	35689.10	5.21	**48.14**	**47.50**	**47.50**
kroA200	15	15	73659.00	73968.37	207.24	73946.27	206.51	38880.00	39356.40	8.88	**47.22**	**46.79**	**46.78**
kroB100	10	10	40591.00	51347.62	280.29	58946.25	106.18	31987.00	32518.13	2.43	**21.20**	**36.67**	**44.83**
kroB150	13	13	53275.00	72400.73	411.24	78522.02	159.79	34629.00	35086.73	5.17	**35.00**	**51.54**	**55.32**
kroB200	15	15	84398.00	84575.24	201.44	84600.56	214.20	38972.00	39317.00	9.00	**53.82**	**53.51**	**53.53**
kroC100	10	10	39666.00	52620.92	276.62	60209.02	102.01	31050.00	31663.50	2.42	**21.72**	**39.83**	**47.41**
kroD100	10	10	36779.00	42494.46	408.15	55760.64	103.92	30688.00	31378.60	2.45	**16.56**	**26.16**	**43.73**
kroE100	10	10	41415.00	57011.89	197.13	61849.16	98.22	32322.00	33085.40	2.45	**21.96**	**41.97**	**46.51**
lin105	11	11	29364.00	39438.79	126.25	39946.79	135.75	20249.00	20568.80	2.92	**31.04**	**47.85**	**48.51**
lin318	18	18	117651.00	117839.24	353.27	117815.71	292.21	56461.00	58288.60	21.00	**52.01**	**50.54**	**50.53**
nrw1379	38	38	154675.00	155262.77	2494.68	155432.91	1686.39	68029.00	68709.03	1012.26	**56.02**	**55.75**	**55.80**
pa561	24	24	6719.00	6737.14	621.97	6737.81	644.24	3489.00	3578.23	98.23	**48.07**	**46.89**	**46.89**
pr107	11	11	88417.00	134685.62	265.51	144968.51	132.77	58896.00	59033.17	3.09	**33.39**	**56.17**	**59.28**
pr124	12	12	136011.00	136187.81	120.76	136283.02	125.54	87320.00	88071.67	3.71	**35.80**	**35.33**	**35.38**
pr136	12	12	188640.00	244194.48	370.79	268944.05	128.42	123410.00	124993.57	4.57	**34.58**	**48.81**	**53.52**
pr144	12	12	175679.00	175696.57	132.21	175714.14	130.59	104952.00	107608.67	4.54	**40.26**	**38.75**	**38.76**
pr152	13	13	237141.00	238255.56	186.17	238255.56	188.62	104437.00	105980.67	5.52	**55.96**	**55.52**	**55.52**
pr226	16	16	306104.00	439044.97	492.24	448503.58	200.53	113988.00	115879.10	10.79	**62.76**	**73.61**	**74.16**
pr264	17	17	246782.00	247275.56	633.65	247176.85	698.15	66160.00	67403.43	14.82	**73.19**	**72.74**	**72.73**
pr299	18	18	170702.00	171265.32	463.88	171282.39	450.14	64381.00	65516.13	18.20	**62.28**	**61.75**	**61.75**
pr439	21	21	321399.00	321881.10	756.75	322009.66	679.41	162891.00	174016.93	37.88	**49.32**	**45.94**	**45.96**
pr76	9	9	163369.00	201564.67	167.34	246295.10	59.29	144401.00	147016.23	1.86	**11.61**	**27.06**	**40.31**
rd100	10	10	13302.00	16349.49	174.27	17880.55	86.60	11043.00	11276.90	2.46	**16.98**	**31.03**	**36.93**
rd400	20	20	37937.00	38084.95	408.13	38103.92	352.28	22808.00	23545.07	27.51	**39.88**	**38.18**	**38.21**
si175	14	14	28666.00	30449.03	737.94	35520.04	272.18	24217.00	24401.93	10.30	**15.52**	**19.86**	**31.30**
si535	24	24	82273.00	84922.19	3278.62	90278.16	1788.84	52908.00	53568.83	123.74	**35.69**	**36.92**	**40.66**
st70	9	9	1019.00	1084.22	126.20	1465.02	49.06	898.00	899.90	1.54	**11.87**	**17.00**	**38.57**
swiss42	7	7	1810.00	1931.27	34.39	2972.38	36.83	1726.00	1760.00	1.30	**4.64**	**8.87**	**40.79**
ts225	15	15	350432.00	351728.60	176.02	351553.38	181.83	203159.00	210747.83	9.31	**42.03**	**40.08**	**40.05**
ulysses16	4	4	8873.00	8873.00	4.17	9101.92	7.66	8873.00	8873.00	0.23	**0.00**	**0.00**	**2.52**
ulysses22	5	5	8533.00	8666.97	9.43	10810.46	11.83	8461.00	8461.00	0.46	**0.84**	**2.38**	**21.73**

These tables clearly show the superiority of our approach in terms of both solution quality and execution time. Our approach obtained solutions of better quality in much shorter time. In fact, on most of the instances, our results are much better than BRKGA and MSH in terms of best as well as average solution quality as indicated clearly by various percentage improvement values. In no case, the best solution quality of HH-ABC is inferior to those of BRKGA and MSH. However, there are few cases where best solution quality obtained through HH-ABC is equal to that of BRKGA and MSH. There are six cases (five for scenario SL and one for scenario TL) where average solution quality of HH-ABC is slightly inferior to BRKGA. In no case, average solution quality of HH-ABC is inferior to MSH. There are also few cases where average solution quality of HH-ABC is equal to that of BRKGA and/or MSH. Please note that the BRKGA and MSH approaches were executed on a GNU/Linux based quad-core Intel Xeon 2.4 GHz system with 8 GB RAM, which is different from the system used to execute our HH-ABC approach (Linux based 3.10 GHz Core-i5-2400 system with 4 GB RAM). Further, chromosome decoding in BRKGA/MSH is done in parallel utilizing the 4 cores available. We have not utilized any parallel processing feature. Due to these reasons, the execution times of HH-ABC can not be compared precisely with BRKGA/MSH. However, considering the information available in the public domain regarding the performances of the two processors on various benchmarks, a Core-i5-2400 based system can be anywhere between 1.25 to 1.5 times faster than a Xeon 2.4 GHz based system. Even after compensating for this difference in processing speeds from average execution times reported in Tables 8.2, 8.3, 8.4, 8.5, and 8.6, we can safely say that our approach is much faster on almost all the instances.

8.5 Conclusions

In this chapter, we have proposed a hyper-heuristic based artificial bee colony algorithm for the k-inter connected multi-depot multi-traveling salesman problem. A new solution encoding scheme is proposed for representing a solution inside our algorithm. This encoding scheme is theoretically analyzed and it is proved that this encoding scheme yields a search space that is considerably smaller in comparison to encoding schemes used previously. Further, a solution is represented directly in our encoding scheme and there is no decoding overhead. We have evaluated and compared our proposed approach with the state-of-the-art approaches on the benchmark instances available in the literature ([3]). Computational results on these benchmark instances show the superiority of our approach over all the other state-of-the-art approaches in terms of both solution quality and execution time. On most of the instances, our approach has obtained much better results. Thus, we have clearly demonstrated the benefit of using an approach that considers all the aspects (subset selection, permutation and grouping) of k-IMDMTSP at the same time .

Chapter 9

Conclusions and Directions for Future Research

In this thesis, we have considered seven $\mathcal{NP}$-hard TSP variants. These variants are solved through various heuristic approaches based on artificial bee colony algorithm, variable neighborhood search, large neighborhood search, iterated local search, hyper-heuristic and genetic algorithm. Among the heuristic approaches that we have developed, approaches based on artificial bee colony algorithm and hyper-heuristic have major shares as they have been used to solve four and three problems respectively, whereas each of the remaining techniques is used to solve a single problem only. All our approaches make use of problem specific knowledge wherever possible. Actually, the field of addressing combinatorial optimization problems in general and TSP & its variants in particular through heuristic approaches has been advanced to a level where any new approach for a problem has to utilize the knowledge particular to that problem in an appropriate manner in order to be competitive with respect to the state-of-the-art approaches for that problem. Our approaches are no exception. The problem specific knowledge can be incorporated into various components of an approach. Our approaches make use of problem specific knowledge in solution encoding, metaheuristic operators, repair procedures, local search and initial solution generation. Developing these heuristic approaches, which are able to perform as good as or better than the state-of-the-art approaches for their respective problems, is the main contribution of this thesis.

In the following, we summarize the contributions made by various chapters along with the possible directions in which future research can be carried out based on the work reported in these chapters.

In Chapter 2, we have proposed a multi-start iterated local search (MS-ILS) algorithm for the maximum scatter traveling salesman problem (MSTSP). Two local search heuristics are proposed based on insertion and 2-opt moves. Three versions of MS-ILS are presented by executing MS-ILS with both the heuristics individually and together. The MS-ILS based on insertion move (viz. MS-ILS (h_1)) performed worse than the other two variants (MS-ILS based on 2-opt move (viz. MS-ILS(h_2)) & MS-ILS based on both insertion and 2-opt moves (viz. MS-ILS($h_1 + h_2$)). There is no significant difference between the performances of the MS-ILS(h_1+h_2) and MS-ILS(h_2). Our MS-ILS approach being the first metaheuristic approach for the MSTSP will serve as the baseline approach for any future metaheuristic approaches for this problem. To facilitate this baselining, we have made use of publicly available TSPLIB instances to assess the performance of MS-ILS. The proposed MS-ILS approach for MSTSP can be extended to other related variants of TSP such as bottleneck TSP, balanced TSP and maximum TSP. Another possible future work is to develop a hybrid approach combining a population-based metaheuristic with the components of the MS-ILS in a bid to further improve the results.

Chapter 3 presented two approaches based on general variable neighborhood search (GVNS) and hyper-heuristic (HH) for the k-traveling salesman problem (k-TSP). The main component of the proposed GVNS approach is the variable neighborhood descent which uses two neighborhood structures based on exchange and swap operations. In exchange operation, a visited city in the tour is exchanged with an unvisited city, whereas in swap move, positions of two visited cities in the tour are swapped. Both the neighborhoods are explored according to a first improvement strategy, i.e., once an improved solution is encountered in the neighborhood, current solution is replaced with the improved solution, and the search process starts from this newly improved solution. These two neighborhood structures effectively handle both the aspects of the k-TSP, viz. subset selection and permutation. Likewise, in the proposed HH also, two low level heuristics are used to handle subset selection and permutation. Two versions of HH, viz. HH_RAND and HH_GREEDY are proposed based on random and greedy selection mechanisms. To evaluate the performance of the proposed approaches (viz. GVNS, HH_RAND and HH_GREEDY), various k-TSP test instances are derived from the instances in TSPLIB. All the details regarding how these instances are derived from TSPLIB instances have been provided in this chapter to facilitate reuse of these instances. Comparison among the proposed approaches shows that both HH_RAND and HH_GREEDY performed better than GVNS. As far as comparison between HH_RAND and HH_GREEDY is concerned, the former performed better than the latter. Approaches analogous to the proposed approaches for k-TSP can be developed for other problems where different neighborhoods and/or heuristics need to be explored as per the aspects of the problem. Interested researchers can develop additional problem specific strategies to improve the performance of our approaches.

Chapter 4 addressed the family traveling salesman problem (FTSP) with a hyper-heuristic(HH) approach. Three large neighborhood search methods based on destroy and repair mechanism are used as low level heuristics in the HH framework. These three large neighborhood methods effectively address both the aspects of the FTSP, viz. subset selection and permutation. Three versions of HH (HH(Random), HH(Greedy) and HH(Rand2)) have been presented for the FTSP. In HH(Random) and HH(Greedy), one of the three low level heuristics is selected based on random and greedy selection mechanisms respectively. In HH(Rand2), two of the three low level heuristics are selected randomly, and one of those two heuristics is selected according to greedy mechanism. The solutions obtained by these three approaches are further improved by a local search phase containing four local search heuristics based on swap, relocate, exchange and 2-opt moves. The proposed three versions of HH have been evaluated on 21 benchmark instances and compared with state-of-the-art approaches available in the literature ([112] and [113]). On most of the instances, our approaches got much better results and they are several times faster in comparison to other approaches. Further, Wilcoxon signed rank test shows the significance of the results obtained by our approaches with respect to all the other approaches.

We have generated 60 new benchmark instances for the FTSP based on the instances from TSPLIB, and reported the results of our approaches on them to facilitate future researchers to evaluate the performance of their approaches on a larger benchmark set comprising 81 instances in total. Wilcoxon signed rank test on our approaches on these 81 instances shows that both both HH(Greedy) and HH(Rand2) performed significantly better than HH(Random). However, there is no significant difference between the performances of HH(Greedy) and HH(Rand2). Through this work, we have demonstrated the advantage of using a hyper-heuristic approach that simultaneously takes into account both the aspects of the FTSP, viz. subset selection and permutation as well as the advantage of using large neighborhood search methods as low level heuristics in a hyper-heuristic framework. Analogous approaches can be designed for other related problems also. Three large neighborhood search methods that we have developed for the FTSP can also be used in other heuristic or metaheuristic frameworks in different ways, and it will be interesting to compare the performance of such approaches with the approaches presented in this chapter.

Chapter 5 described two hybrid metaheuristic approaches based on artificial bee colony algorithm (ABC) and genetic algorithm (GA) for the covering salesman problem (CSP). The proposed approaches are compared with the state-of-the-art approaches on the benchmark instances available in the literature. Computational results and their analysis show our approaches to perform as good as or better than the state-of-the-art approaches in terms of solution quality. Further, GA improved the best known solution values on majority of large instances. As far as comparison between our two proposed approaches is concerned, GA found solutions of same or better quality than the ABC in shorter times on most of the instances.

In both the approaches, operators (neighboring solution generation in ABC, genetic operators in GA) are designed by keeping in mind the subset selection and permutation aspects of CSP. In the ABC approach, neighboring solutions are generated by using variable degree of perturbation. In both the approaches, a local search with two neighborhood structures (LS2N) is used. The first neighborhood structure is based on exchanging a city i with a pair of cities which can cover all the cities covered by i. These pair of cities are precomputed for each city. As there are many pairs possible for each city, first improvement strategy is used while replacing a solution with an improved solution. The second neighborhood structure is based on exchanging a city with a city that it can cover. As there are limited number of cities that a city can cover, best improvement strategy is used in this neighborhood structure. Our proposed approaches can be extended to other related problems such as ring star problem and generalized traveling salesman problem. The LS2N like local search can be used in those situations where there is a need of exploring the neighborhood space of two neighborhoods in an alternating manner, i.e., when finding an improved solution in one neighborhood creates the possibility of finding an improved solution in other neighborhood.

Chapter 6 extended the ABC approach developed in the previous chapter to the generalized covering traveling salesman problem (GCTSP). The neighboring solution generation procedure is designed to cater to subset selection and permutation aspects of GCTSP. Initial solution generation and addition procedures proposed in [1] have been modified to enhance their performance. The proposed approach is compared with the state-of-the-art approaches on the benchmark instances available in the literature. The proposed approach performed significantly better than the state-of-the-art approaches. GCTSP was introduced with the sole intention to minimize the tour length while satisfying the given demand. A possible future work is to study the trade-off by considering a bi-objective version of GCTSP, where the first objective is same as objective function of GCTSP (i.e., minimize the tour length) and the second objectve is to maximize the sumtotal of the demands of the customers covered.

Both Chapter 5 and Chapter 6 addressed the problems based on the concept of coverage. Approaches presented in these chapters can be extended to other problems where the concept of coverage is used. In addition, hyper-heuristic approaches can be developed for CSP, GCTSP and related problems by utilizing the components of the approaches proposed in these two chapters.

Chapter 7 proposed an ABC based approach for solving the colored traveling salesman problem (CTSP). A new solution encoding scheme is proposed for representing a CTSP solution inside ABC algorithm. The encoding scheme is theoretically analyzed and it is proved that this encoding scheme yields a solution space that is considerably smaller in comparison to encoding schemes used previously [2]. The neighboring solution generation procedure in the proposed ABC approach is designed by keeping in mind the grouping and permutation aspects of CTSP. The proposed approach is compared with the state-of-the-art approaches on the 21 benchmark instances available in the literature and 8 large instances derived from instances available on TSPLIB. The proposed approach obtained solutions of much better quality than other state-of-the-art approaches. A possible future work is to extend our approach to the related variants of multiple traveling salesman problem. The solution encoding scheme introduced in this chapter for representing a CTSP solution can be used to encode solutions for other related problems also. Ideas presented in this chapter can be used for other problems also containing the aspects of grouping and permutation.

In Chapter 8, we have proposed a hyper-heuristic based artificial bee colony algorithm (HH-ABC) for the k-interconnected multi-depot multi-traveling salesman problem (k-IMDMTSP). A new solution encoding scheme is proposed for representing a solution inside our approach which yields a search space considerably smaller than that of the encoding scheme used in [3]. The neighboring solution generation procedure in the HH-ABC is designed by keeping in mind all the three aspects of k-IMDMTSP, viz. permutation, subset selection and grouping. The proposed approach is compared with state-of-the-art approaches on the benchmark instances available in the literature and found to be superior in terms of solution quality and execution time both. In fact, proposed approach obtained solutions of much better quality on a vast majority of instances.

A possible future work is to extend our approach to related variants of the location-routing problem. Since many practical problems can be casted as a k-IMDMTSP, our approach can be applied to these problems also. Approaches similar to our approach can be developed for other problems possessing multiple aspects. Similar approaches can be designed for problems that are combination of several different problems, and proportion of constituent problems vary from one instance to another. The solution encoding scheme introduced in this chapter for representing a k-IMDMTSP solution can be used to encode solutions for other related problems also.

Like the multi-objective version of TSP [161], the multi-objective versions of these seven TSP variants can be studied where there can be more than one objective function each specifying distance, travel time, travel expense etc.

Together the seven problems considered in this thesis cover all possible combinations of combinatorial optimization aspects that can occur in a TSP variant and all types of TSP variants discussed in Chapter 1. So these seven problems cover entire spectrum of TSP variants excluding those having roots in vehicle routing problem. This is essential for considering TSP variants in totality and obtaining vital insights about how to solve such problems by means of heuristic approaches. These insights will be useful for solving other TSP variants as solution encoding and design of metaheuristic operators depend to a large extent on the type of TSP variant.

Among the heuristic techniques used in this thesis, hyper-heuristics seem to be most promising for TSP variants possessing more than one aspect of combinatorial optimization as the approaches utilizing them are able to beat the existing approaches by far. Another advantage is their simplicity. They also seem to be under-explored as far as their application to TSP variants is concerned. Hyper-heuristic approaches either alone or in combination with other approaches can be explored not only for other TSP variants, but also for other combinatorial optimization problems having more than one aspect. Reinforcement learning based hyper-heuristics [162, 163] is a recent variant of hyper-heuristics that can also be tried for such problems.

Presently, the approaches that we have developed as part of this thesis are the best/one among the best heuristic approaches for their respective problems. However, the knowledge about a field is continuously advancing due to the effort of various researchers in that field. The field of solving combinatorial optimization problems through heuristic techniques is no exception, and as a result, better heuristic approaches can be developed in future exploiting this advancement in knowledge. All our approaches make use of some properties of the problem at hand. Discovering and studying new properties for each problem, and using them either in isolation or in combination with those already exploited may pave the way for development of even better heuristic approaches for these problems.

References

[1] MH Shaelaie, M Salari, and Z Naji-Azimi. **The generalized covering traveling salesman problem**. *Applied Soft Computing*, **24**:867–878, 2014. (xiv, 107, 108, 110, 111, 112, 113, 114, 116, 117, 121, 123, 179)

[2] J Li, MC Zhou, Q Sun, X Dai, and X Yu. **Colored traveling salesman problem**. *IEEE transactions on cybernetics*, **45**(11):2390–2401, 2015. (xiv, 127, 128, 130, 133, 137, 138, 139, 140, 141, 144, 149, 180)

[3] CE Andrade, FK Miyazawa, and MGC Resende. **Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem**. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 463–470. ACM, 2013. (xv, 150, 151, 152, 153, 155, 156, 158, 162, 165, 167, 168, 174, 180)

[4] P Cowling, G Kendall, and E Soubeiga. **A hyperheuristic approach to scheduling a sales summit**. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 176–190. Springer, 2000. (xvi, 16, 17)

[5] JH Drake, E Özcan, and EK Burke. **An improved choice function heuristic selection for cross domain heuristic search**. In *International Conference on Parallel Problem Solving from Nature*, pages 307–316. Springer, 2012. (xvi, 17)

[6] G Morton and AH Land. **A Contribution to the "Travelling-Salesman" Problem**. *Journal of the Royal Statistical Society: Series B (Methodological)*, **17**(2):185–194, 1955. (1, 2)

[7] J Robinson. **On the Hamiltonian game: a traveling salesman problem. Tech- nical report, RAND Research Memorandum RM-303**. Technical report, RAND PROJECT AIR FORCE ARLINGTON VA, 1949. (1)

[8] MM FLOOD. **Merrill Flood (with Albert Tucker)**. *Interview of Merrill Flood in San Francisco on 14 May 1984. The Princeton Mathematics Commu- nity in the 1930s, Transcript Number 11 (PMC11). Princeton University*, 1984. (2)

[9] K MENGER. **Das botenproblem**. *Ergebnisse eines mathematischen kolloquiums*, **2**:11–12, 1932. (2)

[10] DL APPLEGATE, RE BIXBY, V CHVATAL, AND WJ COOK. *The traveling salesman problem: a computational study*. Princeton university press, 2006. (2, 3)

[11] G GUTIN AND AP PUNNEN. *The traveling salesman problem and its variations*, **12**. Springer Science & Business Media, 2006. (2, 4, 27)

[12] AP PUNNEN. **The traveling salesman problem: Applications, formulations and variations**. In *The traveling salesman problem and its variations*, pages 1–28. Springer, 2007. (2)

[13] RJ JESSEN. **Statistical investigation of a sample survey for obtaining farm facts**. *Research Bulletin (Iowa Agriculture and Home Economics Experiment Station)*, **26**(304):1, 1942. (2)

[14] ES MARKS. **A lower bound for the expected travel among m random points**. *The Annals of Mathematical Statistics*, **19**(3):419–422, 1948. (2)

[15] MN GHOSH. **Expected travel among random points in a region**. *Calcutta Statistical Association Bulletin*, **2**(2):83–87, 1949. (2)

[16] PC MAHALANOBIS. **A sample survey of the acreage under jute in Bengal**. *Sankhyā: The Indian Journal of Statistics*, pages 511–530, 1940. (2)

[17] G DANTZIG, R FULKERSON, AND S JOHNSON. **Solution of a large-scale traveling-salesman problem**. *Journal of the operations research society of America*, **2**(4):393–410, 1954. (2, 3, 4)

[18] G REINELT. **TSPLIB—A traveling salesman problem library**. *ORSA journal on computing*, **3**(4):376–384, 1991. (2)

[19] M HELD AND RM KARP. **The traveling-salesman problem and minimum spanning trees: Part II**. *Mathematical programming*, **1**(1):6–25, 1971. (3)

[20] PM Camerini, L Fratta, and F Maffioli. **On improving relaxation methods by modified gradient techniques**. In *Nondifferentiable optimization*, pages 26–34. Springer, 1975. (3)

[21] M Padberg and G Rinaldi. **Optimization of a 532-city symmetric traveling salesman problem by branch and cut**. *Operations Research Letters*, **6**(1):1–7, 1987. (3)

[22] M Grötschel and O Holland. **A cutting plane algorithm for minimum perfect 2-matchings**. *Computing*, **39**(4):327–344, 1987. (3)

[23] D Applegate, R Bixby, V Chvátal, and W Cook. **Finding cuts in the TSP (A preliminary report)**. *DIMACS Technical Report*, **95**(5), 1995. (3)

[24] D Applegate, R Bixby, W Cook, and V Chvátal. **On the solution of traveling salesman problems**. *Technical Report*, 1998. (3)

[25] P Toth and D Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014. (3)

[26] AO Adewumi and OJ Adeleke. **A survey of recent advances in vehicle routing problems**. *International Journal of System Assurance Engineering and Management*, **9**(1):155–172, 2018. (3)

[27] T Bektas. **The multiple traveling salesman problem: an overview of formulations and solution procedures**. *Omega*, **34**(3):209–219, 2006. (4)

[28] A Schrijver. **On the history of combinatorial optimization (till 1960)**. *Handbooks in operations research and management science*, **12**:1–68, 2005. (4)

[29] LV Kantorovich. **Mathematical methods of organizing and planning production**. *Management Science*, **6**:366–422, 1939. (4)

[30] M Dorigo, VO Maniezzo, and A Colorni. **Ant system: Optimization by a colony of cooperating agents**. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, **26**:29–41, 1996. (7)

[31] JH HOLLAND. *Adaptation in natural and artificial systems: An introductory analysis with applications in biology, control and artificial intelligence*. University of Michigan Press, Ann Arbor, MI, 1975. (7, 17, 20)

[32] D GOLDBERG. *Genetic algorithm in search, optimization and machine learning*. Reading, MA :Addison-Wesley, 1989. (7, 10)

[33] M DORIGO, V MANIEZZO, AND A COLORNI. **Positive feedback as a search strategy**, 1991. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy. (7)

[34] F GLOVER. **Tabu search - part 1**. *ORSA Journal on Computing*, **1**:190–206, 1989. (7)

[35] F GLOVER. **Tabu search - part 2**. *ORSA Journal on Computing*, **2**:4–32, 1990. (7)

[36] D KARABOGA. **An idea based on honey bee swarm for numerical optimization**, 2005. Computer Engineering Department, Erciyes University, Turkey. (7)

[37] N MLADENOVIĆ AND P HANSEN. **Variable neighborhood search**. *Computers & operations research*, **24**(11):1097–1100, 1997. (7, 12, 14)

[38] P HANSEN AND N MLADENOVIĆ. **Variable neighborhood search: Principles and applications**. *European journal of operational research*, **130**(3):449–467, 2001. (7)

[39] D KARABOGA. *An idea based on honey bee swarm for numerical optimization*. Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005. (8, 9)

[40] B BASTURK AND D KARABOGA. **An Artificial Bee Colony (ABC) algorithm for numeric function optimization**. In *IEEE Swarm Intelligence Symposium*, pages 12–14. IEEE, 2006. (8)

[41] D KARABOGA AND B BASTURK. **A powerful and efficient algorithm for numeric function optimization: Artificial Bee Colony (ABC) algorithm**. *Journal of Global Optimization*, **39**:459–471, 2007. (8)

[42] D KARABOGA AND B BASTURK. **Artificial Bee Colony (ABC) optimization algorithm for solving constrained optimization problems**. In *IFSA 2007, Lecture Notes in Artificial Intelligence*, **4529**, pages 789–798. Springer, 2007. (8)

[43] D Karaboga and B Basturk. **On the performance of artificial bee colony (ABC) algorithm**. *Applied Soft Computing*, **8**:687–697, 2008. (8)

[44] A Singh. **An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem**. *Applied Soft Computing*, **9**:625–631, 2009. (8)

[45] S Sundar and A Singh. **New heuristic approaches for the dominating tree problem**. *Applied Soft Computing*, **13**:4695–4703, 2013. (8)

[46] D Karaboga and B Akay. **A modified Artificial Bee Colony (ABC) algorithm for constrained optimization problems**. *Applied Soft Computing*, **11**:3021–3031, 2011. (8)

[47] QK Pan, MF Tasgetiren, PN Suganthan, and TJ Chua. **A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem**. *Information Sciences*, **181**:2455–2468, 2011. (8)

[48] W Gao and S Liu. **Improved artificial bee colony algorithm for global optimization**. *Information Processing Letters*, **111**:871–882, 2011. (8)

[49] S Sundar and A Singh. **A swarm intelligence approach to the early/tardy scheduling problem**. *Swarm and Evolutionary Computation*, **4**:25–32, 2012. (8)

[50] B Akay and D Karaboga. **A modified Artificial Bee Colony algorithm for real-parameter optimization**. *Information Sciences*, **192**:120–142, 2012. (8)

[51] B Jayalakshmi and A Singh. **A hybrid artificial bee colony algorithm for the cooperative maximum covering location problem**. *International Journal of Machine Learning & Cybernetics*, **8**:691–697, 2017. (8)

[52] W Xiang, X Meng, Y Li, R He, and M An. **An improved artificial bee colony algorithm based on the gravity model**. *Information Sciences*, **429**:49–71, 2018. (8)

[53] W Xiang, Y Li, R He, M Gao, and M An. **A novel artificial bee colony algorithm based on the cosine similarity**. *Computers & Industrial Engineering*, **115**:54–68, 2018. (8)

[54] DE GOLDBERG AND K DEB. **A comparative analysis of selection schemes used in genetic algorithms**. In *Foundations of Gentic Algorithms*, pages 69–93. Morgan Kaufmann, 1990. (10, 20)

[55] D KARABOGA, B GORKEMLI, C OZTURK, AND N KARABOGA. **A comprehensive survey: artificial bee colony (ABC) algorithm and applications**. *Artificial Intelligence Review*, **42**(1):21–57, 2014. (11)

[56] HR LOURENÇO, OC MARTIN, AND T STUTZLE. **Iterated local search**. *International series in operations research and management science*, pages 321–354, 2003. (11)

[57] HR LOURENÇO, OC MARTIN, AND T STUTZLE. **Iterated local search: Framework and applications**. In *Handbook of metaheuristics*, pages 363–397. Springer, 2010. (12)

[58] P HANSEN, N MLADENOVIĆ, AND JAM PÉREZ. **Variable neighbourhood search: methods and applications**. *Annals of Operations Research*, **175**(1):367–407, 2010. (14, 40)

[59] P SHAW. **Using constraint programming and local search methods to solve vehicle routing problems**. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998. (14)

[60] D AKSEN, O KAYA, FS SALMAN, AND Ö TÜNCEL. **An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem**. *European Journal of Operational Research*, **239**(2):413–426, 2014. (15)

[61] U BREUNIG, V SCHMID, RF HARTL, AND T VIDAL. **A large neighbourhood based heuristic for two-echelon routing problems**. *Computers & Operations Research*, **76**:208–225, 2016. (15)

[62] S CHEN, R CHEN, GG WANG, J GAO, AND AK SANGAIAH. **An adaptive large neighborhood search heuristic for dynamic vehicle routing problems**. *Computers & Electrical Engineering*, **67**:596–607, 2018. (15)

[63] Y LI, H CHEN, AND C PRINS. **Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests**. *European Journal of Operational Research*, **252**(1):27–38, 2016. (15)

[64] S Akpinar. **Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem**. *Expert Systems with Applications*, **61**:28–38, 2016. (15)

[65] M Eskandarpour, P Dejax, and O Péton. **A large neighborhood search heuristic for supply chain network design**. *Computers & Operations Research*, **80**:23–37, 2017. (15)

[66] VC Hemmelmayr. **Sequential and parallel large neighborhood search algorithms for the periodic location routing problem**. *European Journal of Operational Research*, **243**(1):52–60, 2015. (15)

[67] T Hintsch and S Irnich. **Large multiple neighborhood search for the clustered vehicle-routing problem**. *European Journal of Operational Research*, **270**(1):118–131, 2018. (15)

[68] MC Monçores, ACF Alvim, and MO Barros. **Large neighborhood search applied to the software module clustering problem**. *Computers & Operations Research*, **91**:92–111, 2018. (15)

[69] E Lizárraga, MJ Blesa, C Blum, and GR Raidl. **Large neighborhood search for the most strings with few bad columns problem**. *Soft Computing*, **21**(17):4901–4915, 2017. (15)

[70] EK Burke, M Gendreau, M Hyde, G Kendall, G Ochoa, E Özcan, and R Qu. **Hyper-heuristics: A survey of the state of the art**. *Journal of the Operational Research Society*, **64**(12):1695–1724, 2013. (15, 16, 17, 44, 45, 69)

[71] J Denzinger, M Fuchs, and M Fuchs. **High Performance ATP Systems by Combining Several AI Methods**. In *Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'97, pages 102–107, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. (15)

[72] H Fisher. **Probabilistic learning combinations of local job-shop scheduling rules**. *Industrial scheduling*, pages 225–251, 1963. (15)

[73] WB Crowston, F Glover, and JD Trawick. **Probabilistic and parametric learning combinations of local job shop scheduling rules**. Technical report, Research Memorandum, No. 117, GSIA, Carnegie Mellon University, Pittsburgh, 1963. (16)

[74] K CHAKHLEVITCH AND P COWLING. **Hyperheuristics: recent developments**. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer, 2008. (16)

[75] JH HOLLAND. **Adaptation in natural and artificial systems Ann Arbor**. *The University of Michigan Press*, **1**:975, 1975. (17)

[76] L DAVIS. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. (19)

[77] M MITCHELL. *An introduction to genetic algorithms*. Bradford Books, 1998. (20)

[78] JE BAKER. **Reducing bias and inefficiency in the selection algorithm**. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, 1987. (20)

[79] EM ARKIN, YJ CHIANG, JSB MITCHELL, SS SKIENA, AND TC YANG. **On the maximum scatter traveling salesperson problem**. *SIAM Journal on Computing*, **29**(2):515–544, 1999. (25)

[80] F SCHOLZ. **Coordination hole tolerance stacking**. Technical report, Technical Report BCSTECH-93-048, Boeing Computer Services, 1993. (25)

[81] F SCHOLZ. **Tolerance stack analysis methods**. *Research and technology boeing information & support services, boeing, seattle*, pages 1–44, 1995. (25)

[82] K PENAVIC. **Optimal firing sequences for CAT scans**. *Manuscript, Dept. Appl. Math., SUNY, Stony Brook*, 1994. (25)

[83] YJ CHIANG. **New approximation results for the maximum scatter TSP**. *Algorithmica*, **41**(4):309–341, 2005. (26)

[84] L KOZMA AND T MÖMKE. **Maximum scatter TSP in doubling metrics**. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 143–153. SIAM, 2017. (26)

[85] I HOFFMANN, S KURZ, AND J RAMBAU. **The maximum scatter TSP on a regular grid**. In *Operations Research Proceedings 2015*, pages 63–70. Springer, 2017. (26)

[86] W DONG, X DONG, AND Y WANG. **The Improved Genetic Algorithms for Multiple Maximum Scatter Traveling Salesperson Problems**. In *China Conference on Wireless Sensor Networks*, pages 155–164. Springer, 2017. (26)

[87] K Crammer, O Dekel, J Keshet, S Shalev-Shwartz, and Y Singer. **Online passive-aggressive algorithms**. *Journal of Machine Learning Research*, **7**(Mar):551–585, 2006. (26)

[88] EL Lawler. **The traveling salesman problem: a guided tour of combinatorial optimization**. *Wiley-Interscience Series in Discrete Mathematics*, 1985. (27)

[89] A Barvinok, SP Fekete, DS Johnson, A Tamir, GJ Woeginger, and R Woodroofe. **The geometric maximum traveling salesman problem**. *Journal of the ACM (JACM)*, **50**(5):641–664, 2003. (27)

[90] SY Fang, WY Chen, and YW Chang. **Graph-based subfield scheduling for electron-beam photomask fabrication**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **32**(2):189–201, 2013. (27)

[91] A Fischer and P Hungerländer. **The traveling salesman problem on grids with forbidden neighborhoods**. *Journal of Combinatorial Optimization*, **34**(3):891–915, 2017. (27)

[92] ZW Lin, SY Fang, YW Chang, WC Rao, and CH Kuan. **Provably Good Max–Min-m-Neighbor-TSP-Based Subfield Scheduling for Electron-Beam Photomask Fabrication**. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **26**(2):378–391, 2018. (27)

[93] A Fischer, P Hungerländer, and A Jellen. **The Traveling Salesperson Problem with Forbidden Neighborhoods on Regular 3D Grids**. In *Operations Research Proceedings 2017*, pages 213–219. Springer, 2018. (27)

[94] F Wilcoxon, SK Katti, and RA Wilcox. **Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test**. *Selected tables in mathematical statistics*, **1**:171–259, 1970. (36, 57, 58, 79, 80, 104, 125)

[95] E Balas. **The prize collecting traveling salesman problem**. *Networks*, **19**(6):621–636, 1989. (38)

[96] B Awerbuch, Y Azar, A Blum, and S Vempala. **New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen**. *SIAM Journal on computing*, **28**(1):254–262, 1998. (38)

[97] A Blum, R Ravi, and S Vempala. **A constant-factor approximation algorithm for the k MST problem**. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 442–448. ACM, 1996. (38)

[98] G Ausiello, V Bonifaci, S Leonardi, and A Marchetti-Spaccamela. **Prize-collecting traveling salesman and related problems**. *Handbook of Approximation Algorithms and Metaheuristics*, **40**:1–13, 2007. (38)

[99] N Garg. **A 3-approximation for the minimum tree spanning k vertices**. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 302–309. IEEE, 1996. (38)

[100] S Arora and G Karakostas. **A 2+ ε approximation algorithm for the k-MST problem**. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 754–759. Society for Industrial and Applied Mathematics, 2000. (38)

[101] N Garg. **Saving an epsilon: a 2-approximation for the k-MST problem in graphs**. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 396–402. ACM, 2005. (38)

[102] A Blum, S Chawla, DR Karger, T Lane, A Meyerson, and M Minkoff. **Approximation algorithms for orienteering and discounted-reward TSP**. *SIAM Journal on Computing*, **37**(2):653–670, 2007. (38)

[103] N Mladenović, D Urošević, and A Ilić. **A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem**. *European Journal of Operational Research*, **220**(1):270–285, 2012. (40)

[104] N Mladenović, R Todosijević, and D Urošević. **Two level general variable neighborhood search for attractive traveling salesman problem**. *Computers & Operations Research*, **52**:341–348, 2014. (40)

[105] N Mladenović, R Todosijević, and D Urošević. **An efficient general variable neighborhood search for large travelling salesman problem with time windows**. *Yugoslav Journal of Operations Research*, **23**(1):19–30, 2013. (40)

[106] R Todosijević, A Mjirda, M Mladenović, S Hanafi, and B Gendron. **A general variable neighborhood search variants for the travelling salesman problem with draft limits**. *Optimization Letters*, **11**(6):1047–1056, 2017. (40)

[107] G Kendall and J Li. **Competitive travelling salesmen problem: A hyper-heuristic approach**. *Journal of the Operational Research Society*, **64**(2):208–216, 2013. (42)

[108] V Pandiri and A Singh. **A hyper-heuristic based artificial bee colony algorithm for k-Interconnected multi-depot multi-traveling salesman problem**. *Information Sciences*, **463**:261–281, 2018. (42)

[109] ZA Aziz. **Ant colony hyper-heuristics for travelling salesman problem**. *Procedia Computer Science*, **76**:534–538, 2015. (42)

[110] SS Choong, LP Wong, and CP Lim. **An artificial bee colony algorithm with a modified choice function for the Traveling Salesman Problem**. *Swarm and evolutionary computation*, **44**:622–635, 2019. (42)

[111] M Fischetti, G Salazar, J Juan, and P Toth. **A branch-and-cut algorithm for the symmetric generalized traveling salesman problem**. *Operations Research*, **45**(3):378–394, 1997. (61, 84)

[112] LF Morán-Mirabal, JL González-Velarde, and MGC Resende. **Randomized heuristics for the family traveling salesperson problem**. *International Transactions in Operational Research*, **21**(1):41–57, 2014. (61, 71, 72, 73, 75, 80, 178)

[113] R Bernardino and A Paias. **Solving the family traveling salesman problem**. *European Journal of Operational Research*, **267**(2):453–466, 2018. (61, 71, 72, 73, 75, 80, 178)

[114] JR Current and DA Schilling. **The covering salesman problem**. *Transportation science*, **23**(3):208–213, 1989. (82)

[115] B Golden, Z Naji-Azimi, S Raghavan, M Salari, and P Toth. **The generalized covering salesman problem**. *INFORMS Journal on Computing*, **24**(4):534–553, 2012. (83, 98)

[116] S Lin and BW Kernighan. **An effective heuristic algorithm for the traveling-salesman problem**. *Operations research*, **21**(2):498–516, 1973. (83)

[117] M Salari and Z Naji-Azimi. **An integer programming-based local search for the covering salesman problem**. *Computers & Operations Research*, **39**(11):2594–2602, 2012. (83, 98)

[118] M Salari, M Reihaneh, and MS Sabbagh. **Combining ant colony optimization algorithm and dynamic programming technique for solving the covering salesman problem**. *Computers & Industrial Engineering*, **83**:244–251, 2015. (83, 87, 98, 102, 106)

[119] EM Arkin and R Hassin. **Approximation algorithms for the geometric covering salesman problem**. *Discrete Applied Mathematics*, **55**(3):197–218, 1994. (83)

[120] DJ Gulczynski, JW Heath, and CC Price. **The close enough traveling salesman problem: A discussion of several heuristics**. In *Perspectives in Operations Research*, pages 271–283. Springer, 2006. (83)

[121] R Shuttleworth, BL Golden, S Smith, and E Wasil. **Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network**. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501. Springer, 2008. (83)

[122] M Gendreau, G Laporte, and F Semet. **The covering tour problem**. *Operations Research*, **45**(4):568–576, 1997. (83, 84)

[123] M Hachicha, MJ Hodgson, G Laporte, and F Semet. **Heuristics for the multi-vehicle covering tour problem**. *Computers & Operations Research*, **27**(1):29–42, 2000. (84)

[124] L Motta, LS Ochi, and C Martinhon. **Grasp metaheuristics for the generalized covering tour problem**. In *Proceedings of IV metaheuristic international conference, Porto, Portugal*, **1**, pages 387–393, 2001. (84)

[125] JR Current and DA Schilling. **The median tour and maximal covering tour problems: Formulations and heuristics**. *European Journal of Operational Research*, **73**(1):114–126, 1994. (84)

[126] M LABBÉ, G LAPORTE, IR MARTÍN, AND JJS GONZÁLEZ. **The ring star problem: Polyhedral analysis and exact algorithm**. *Networks*, **43**(3):177–189, 2004. (84)

[127] R BALDACCI, M DELL'AMICO, AND JJS GONZÁLEZ. **The capacitated m-ring-star problem**. *Operations Research*, **55**(6):1147–1162, 2007. (84)

[128] Z NAJI-AZIMI, M SALARI, AND P TOTH. **A heuristic procedure for the capacitated m-ring-star problem**. *European Journal of Operational Research*, **207**(3):1227–1234, 2010. (84)

[129] M LABBÉ, G LAPORTE, IR MARTIN, AND JJS GONZÁLEZ. **Locating median cycles in networks**. *European Journal of Operational Research*, **160**(2):457–470, 2005. (84)

[130] L DAVIS. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. (93)

[131] C GÖKÇE AND RY HARUN. **Genetic algorithm parameter optimisation using Taguchi method for a flexible manufacturing system scheduling problem**. *International Journal of Production Research*, **53**(3):897–915, 2015. (116, 117, 140, 167, 168)

[132] J LI, X MENG, MC ZHOU, AND X DAI. **A Two-Stage Approach to Path Planning and Collision Avoidance of Multibridge Machining Systems**. *IEEE Transactions on Systems, Man and Cybernetics: Systems*, pages 1–11, 2016. (116, 140, 167)

[133] O BRÄYSY AND M GENDREAU. **Vehicle routing problem with time windows, Part I: Route construction and local search algorithms**. *Transportation science*, **39**(1):104–118, 2005. (128)

[134] O BRÄYSY AND M GENDREAU. **Vehicle routing problem with time windows, Part II: Metaheuristics**. *Transportation science*, **39**(1):119–139, 2005. (128)

[135] B EKSIOGLU, AV VURAL, AND A REISMAN. **The vehicle routing problem: A taxonomic review**. *Computers & Industrial Engineering*, **57**(4):1472–1483, 2009. (128)

[136] NA EL-SHERBENY. **Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods**. *Journal of King Saud University-Science*, **22**(3):123–131, 2010. (128)

[137] M GENDREAU AND CD TARANTILIS. *Solving large-scale vehicle routing problems with time windows: The state-of-the-art.* Technical Report CIRRELT-2010-04, CIRRELT, University of Montreal, Montreal, Canada, 2010. (128)

[138] K BRAEKERS, K RAMAEKERS, AND IV NIEUWENHUYSE. **The vehicle routing problem: State of the art classification and review**. *Computers & Industrial Engineering*, **99**:300–313, 2016. (128)

[139] L JUN, S QIRUI, Z MENGCHU, AND D XIANZHONG. **A new multiple traveling salesman problem and its genetic algorithm-based solution**. In *Proceedings of the 2013 IEEE International Conference on Systems Man and Cybernetics, Manchester, U.K.*, pages 1–6, 2013. (128, 138)

[140] J LI, Q SUN, MC ZHOU, X YU, AND X DAI. **Colored traveling salesman problem and solution**. *IFAC Proceedings Volumes*, **47**(3):9575–9580, 2014. (128)

[141] J LI, X DAI, H LIU, AND MC ZHOU. **A decomposition approach to colored traveling salesman problems**. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 51–56. IEEE, 2015. (128)

[142] J LI, MC ZHOU, X DAI, Q SUN, AND X YU. **A colored traveling salesman problem model for planning dual-bridge waterjet cutting paths**. *IEEE Transactions on Industrial Informatics*, 2015. (128)

[143] J LI, X MENG, AND X DAI. **Collision-free scheduling of multi-bridge machining systems: a colored traveling salesman problem-based approach**. *IEEE/CAA Journal of Automatica Sinica*, **5**(1):139–147, 2018. (128)

[144] C MALMBORG. **A genetic algorithm for service level based vehicle scheduling**. *European Journal of Operational Research*, **93**:121–134, 1996. (130)

[145] P JEAN-YVES. **Genetic algorithms for the traveling salesman problem**. *Annals of Operations Research*, **63**:339–370, 1996. (130, 155)

[146] L TANG, J LIU, A RONG, AND Z YANG. **A multiple traveling salesman problem model for hot rolling scheduling in Shangai Baoshan Iron and Steel Complex**. *European Journal of Operational Research*, **124**:1267–1282, 2000. (130)

[147] YB PARK. **A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines**. *International Journal of Production Economics*, **73**:175–188, 2001. (130)

[148] AE CARTER AND CT RAGSDALE. **A new approach to solving the multiple traveling salesperson problem using genetic algorithms**. *European Journal of Operational Research*, **175**:245–257, 2006. (130, 132, 155)

[149] A SINGH AND AS BAGHEL. **A new grouping genetic algorithm approach to the multiple traveling salesperson problem**. *Soft Computing*, **13**:95–101, 2009. (130, 131, 155)

[150] S ROSS. *A First Course in Probability*. Pearson Education, Upper Saddle River, NJ, 8 edition, 2010. (132, 133)

[151] W MALIK, S RATHINAM, AND S DARBHA. **An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem**. *Operations Research Letters*, **35**(6):747–753, 2007. (151)

[152] S YADLAPALLI, WA MALIK, S DARBHA, AND M PACHTER. **A Lagrangian-based algorithm for a multiple depot, multiple traveling salesmen problem**. *Nonlinear Analysis: Real World Applications*, **10**(4):1990–1999, 2009. (151)

[153] M DREXL AND M SCHNEIDER. **A survey of variants and extensions of the location-routing problem**. *European Journal of Operational Research*, **241**(2):283–308, 2015. (151)

[154] C PRINS, C PRODHON, A RUIZ, P SORIANO, AND RW CALVO. **Solving the capacitated location-routing problem by a cooperative Lagrangean relaxation-granular tabu search heuristic**. *Transportation Science*, **41**(4):470–483, 2007. (151)

[155] JM BELENGUER, E BENAVENT, C PRINS, C PRODHON, AND RW CALVO. **A branch-and-cut method for the capacitated location-routing problem**. *Computers & Operations Research*, **38**(6):931–941, 2011. (151)

[156] G PERBOLI, R TADEI, AND D VIGO. **The two-echelon capacitated vehicle routing problem: Models and math-based heuristics**. *Transportation Science*, **45**(3):364–380, 2011. (151)

[157] AE Carter and CT Ragsdale. **Scheduling pre-printed newspaper advertising inserts using genetic algorithms**. *Omega*, **30**:415–421, 2002. (155)

[158] S Yuan, B Skinner, S Huang, and D Liu. **A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms**. *European Journal of Operational Research*, **228**:72–82, 2013. (155)

[159] S Ross. *Introduction to Probability Model*. Macmillian, New York, NY, 1984. (156, 159)

[160] SP Lloyd. **Least squares quantization in PCM**. *IEEE Transactions on Information Theory*, **28**(2):129–137, 1982. (161)

[161] T Lust and J Teghem. **The multiobjective traveling salesman problem: a survey and a new approach**. In *Advances in Multi-Objective Nature Inspired Computing*, pages 119–141. Springer, 2010. (181)

[162] F Alanazi. **Reinforcement learning hyper-heuristics for optimisation**. *PhD Thesis*, 2017. (181)

[163] SS Choong, LP Wong, and CP Lim. **Automatic design of hyper-heuristic based on reinforcement learning**. *Information Sciences*, **436**:89–107, 2018. (181)

List of Publications

[1] **Venkatesh Pandiri** and Alok Singh. **A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem.** *Information Sciences, **463-464**: 261-281, 2018, Elsevier.*

[2] **Venkatesh Pandiri** and Alok Singh. **A swarm intelligence approach for the colored traveling salesman problem.** *Applied Intelligence, **48**: 4412-4428, 2018, Springer-Verlag.*

[3] **Venkatesh Pandiri** and Alok Singh. **An artificial bee colony algorithm with variable degree of perturbation for the generalized covering traveling salesman problem.** *Applied Soft Computing, **78**: 481-495, 2019, Elsevier.*

[4] **Venkatesh Pandiri**, Gaurav Srivastava and Alok Singh. **A general variable neighborhood search algorithm for the k-traveling salesman problem.** *Proceedings of the 2018 International Conference on Advances in Computing & Communications (ICACC-2018), Procedia Computer Science, **143**: 189-196, 2018, Elsevier.*

[5] **Venkatesh Pandiri**, Alok Singh and Rammohan Mallipeddi. **A multi-start iterated local search algorithm for the maximum scatter traveling salesman problem.** To appear in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC-2019), 2019, IEEE.*

[6] **Venkatesh Pandiri** and Alok Singh. **A large neighborhood search based hyper-heuristic approach for the family traveling salesman problem.** Communicated to *Information Sciences, Elsevier.*

[7] **Venkatesh Pandiri** and Alok Singh. **Multi-start heuristics for the k-traveling salesman problem.** Communicated to *Applied Intelligence, Springer-Verlag.*

[8] **VENKATESH PANDIRI**, ALOK SINGH AND ANDRÉ ROSSI. **Two hybrid heuristics for the covering salesman problem.** Communicated to *Information Sciences, Elsevier.*