Swarm Intelligence Techniques for Some Combinatorial Optimization Problems

A thesis submitted during 2012 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in Department of Computer & Information Sciences

by

Shyam Sundar

Department of Computer & Information Sciences

School of Mathematics & Computer/Information Sciences

University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500046
Andhra Pradesh
India

CERTIFICATE

This is to certify that the thesis entitled **"Swarm Intelligence Techniques for Some Combinatorial Optimization Problems"** submitted by **Shyam Sundar** bearing **Reg. No. 08MCPC17** 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.

The thesis 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.

(Dr. Alok Singh)
Supervisor
Department of Computer & Information Sciences
School of Mathematics & Computer/Information Sciences
University of Hyderabad
Hyderabad – 500046, India

Head
Department of Computer &
Information Sciences
University of Hyderabad
Hyderabad – 500046, India

Dean
School of Mathematics &
Computer/Information Sciences
University of Hyderabad
Hyderabad – 500046, India

DECLARATION

I, **Shyam Sundar**, hereby declare that this thesis entitled **"Swarm Intelligence Techniques for Some Combinatorial Optimization Problems"** submitted by me under the guidance and supervision of **Dr. Alok Singh** is a bonafide research work. 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.

Date :	**Name: Shyam Sundar**
	Signature of the Student:
	Reg. No.: 08MCPC17

Abstract

Over the last few decades, combinatorial optimization problems have been attracting massive attention from the research community due to their practical and theoretical importance. Most of combinatorial optimization problems are $\mathcal{NP}$-Hard. $\mathcal{NP}$-Hardness of a problem means that the execution time of any known exact algorithm for the problem will grow exponentially with the size of problem instance and no polynomial time algorithm is known even to verify the optimality of a proposed solution. Under these circumstances, one has to abandon the search for provably optimal solutions and look towards other methods such as heuristics and metaheuristics which can find high quality solutions in a reasonable amount of time. This thesis is focused primarily on solving some $\mathcal{NP}$-Hard combinatorial optimization problems using two metaheuristics techniques – artificial bee colony (ABC) algorithm and ant colony optimization (ACO) algorithm – which belong to the broad class of swarm intelligence techniques. Both techniques are inspired by intelligent foraging behavior of social insects, i.e., honey bees in case of ABC algorithm and ants in case of ACO algorithm. ABC algorithm, which was originally designed for optimization in continuous domains, is under-explored in case of combinatorial optimization problems, whereas ACO algorithm is one among most successful techniques for solving combinatorial optimization problems.

In this thesis, we have considered 8 problems which are as follows: minimum routing cost spanning tree problem, quadratic minimum spanning tree problem, set covering problem, dominating tree problem, two bounded-degree spanning tree problems, early/tardy scheduling problem and block-model problem. Out of these 8 problems, first 6 problems are subset selection problems, next problem is a permutation problem, whereas the last one is a grouping problem. Apart from being theoretically challenging, these problems have many practical applications in diverse fields such as networks, computational biology, transportation, scheduling, production, social network analysis etc. We have developed ABC algorithm based approaches for all the aforementioned problems except two bounded-degree spanning tree problems. We have also developed ACO approaches for dominating tree problem and two bounded-degree spanning tree problems. Computational results show the effectiveness of our proposed approaches. Ideas presented in this thesis can be applied to a wide range of problems.

Acknowledgements

Words cannot express how deeply grateful I am to my supervisor, Dr. Alok Singh, for his perseverance in nurturing me to become a worthy researcher from a novice through his thoughtful direction. I consider myself privileged to have worked with my supervisor. I am obliged to him for having faith in me and giving me the freedom to pursue my ideas and for being most generous with his time. It has been a delightful learning experience for me to work under my supervisor. Without an iota of exaggeration, it would be befitting to say that this thesis would not have been possible without his continuous support, both professionally and personally.

I am indebted to Dr. André Rossi who helped me in my research work. I learned a lot from him and whenever I needed him he was always there to help me.

I am thankful to the Head of the Department Prof. P.N. Girija and the Dean of the School Prof. T. Amaranath for providing the necessary facilities.

I am thankful to Dr. Rajiv Wankar and Dr. Vineet Nair Padmanabhan who as the members of my Doctoral Review Committee always facilitated a smooth process of review.

Special thanks to Prof. Arun Kumar Pujari who played a pivotal role towards realization of my aspiration for the research.

My innermost thanks to Ms. Anupama Potluri for her constant counseling and foresight.

Equally grateful I am to Dr. S. Durga Bhavani, Dr. S.K. Udgata and other faculty members of the department for their constant encouragement.

My sincere thanks to a number of researchers in my field who furnished me with indispensable research data. Prof. B. A. Julstrom provided me the random data set for minimum routing cost spanning tree problem. Dr. S.-M. Soak provided me the Euclidean instances for quadratic minimum spanning tree problem. Dr. J.M.S. Valente provided me instances for early/tardy scheduling problem. Prof. L. Proll provided me test instances for blockmodel problem and also clarified some of my doubts regarding the results of some instances. I am extremely grateful to all of them.

The quest for a doctoral degree would not have been plausible, had it not been for the unstinting support from the research infrastructure and the effervescent ambiance of the University. Due credit to university for building a research oriented Computer Science Department, a library rich in wide range of research books, and most importantly a healthy campus atmosphere. These pillars of education helped me during the different phases of my life in the campus as a research scholar.

I am also thankful to my friends specially Tarak Patra, Abhay Kumar, Sachchidanand Chaurasia, Hemant Taram, Satpal Singh, Rakesh Kumar, Mathew Sinu Simon and Satveer Singh who played an important role in supporting me through thick and thin.

A major part of this thesis was supported by two fellowships – Junior Research Fellowship funded by the Department of Science & Technology, Government of India, and Senior Research Fellowship (Direct) funded by the Council of Scientific and Industrial Research, Government of India. I am thankful to both organizations for their financial support.

Finally, I would like to thank my family for all their love, patience, and constant encouragement.

(Shyam Sundar)

Contents

List of Figures

List of Tables

LIST OF TABLES

Chapter 1

Introduction

Since time immemorial, Nature has offered itself as a matrix of infinite mysteries for observers, especially with regard to its elements and their functions. Researchers have always looked up to Nature as an inexhaustible source of inspiration that allows them to understand its elements, a perspective that has afforded them various models through which one can seek to solve many complex problems of the real world. It goes without saying that to a certain extent researchers have been successful in addressing and answering some of these problems. To take an example, the foraging behaviors of swarms such as honey bees and ants have inspired researchers to develop swarm intelligence techniques for the solution of combinatorial optimization problems.

Combinatorial optimization problems, over the last few decades, have been attracting massive attention from the research community owing to the practical and theoretical importance they entail. A combinatorial optimization problem, having discrete and finite nature, seeks an optimal solution among its large, nevertheless, finite set of feasible solutions. Characteristics of such problems lie in their combinatorial structures and computational aspects. The beauty of such problems and their results is easy to state and explain. In spite of these characteristics, the most intriguing part of many such problems lies in their inherent combinatorial explosion properties. This explosion property describes an explosive growth of the number of possible feasible solutions to such a problem with the increase in the size of instance. Because of this inherent property, it has been observed in many combinatorial optimization problems that searching an optimal solution to such a problem, in practice, is really hard.

1. INTRODUCTION

Existing literature on combinatorial optimization reveals the fact that combinatorial optimization might not be a very old discipline but its trajectories cannot be traced to the point of provenance. The elementary combinatorial concepts related to manipulating finite discrete objects have been known to the world since time immemorial. A problem of finding a shortest path from source to destination, problems arising from practical experiences such as traveling salesman problem, which crops up while finding a shortest route through a given set of cities for a salesman that starts from salesman's home city and visits each city exactly once and then back to his home city, are some of the well-known combinatorial optimization problems. Earlier, these problems were considered in isolation and ad hoc solutions were offered in particular mathematical context. There was an age when many disciplines of Mathematics were used for the purposes of generality and abstraction. However, nowadays, there is a lot of appreciation for and emphasis on problems that are practical, concrete and complex. The first scientific breakthrough took place in 1735 when Euler's paper on the *Seven Bridges of Königsberg* was published. It is considered to be the first scientific paper that laid the foundation of graph theory which eventually helped in pushing combinatorial optimization to the capacity of an independent discipline of Mathematics. The contributions of Kantorovich [5] and Dantzig [6] to the development of mathematical linear programming further gave the required impetus to combinatorial optimization.

Since the mid-20th century, evolution and continuous improvement in computers have done, more often than not, a great deal to reduce the effort that is required on part of the researchers to solve these computational problems. Towards the late 1960s, it was observed that while many combinatorial optimization problems, such as minimum spanning tree problem, were solvable in polynomial time, a wide range of other combinatorial optimization problems still continue to escape solutions in polynomial time. Problems that could not be solved in polynomial time are intriguing in nature and pose inherent difficulties to any approach attempting to solve them. The main difficulty with such a problem is that the number of possible solutions grows exponentially as the size of its instance increases and there is no definitive approach to obtain an optimal solution in polynomial time. Such problems are widely recognized as difficult to solve, and the number of such complex problems, like those arising in networks due to the advent of computers, is increasing day-by-day in real world. Researchers find themselves in a confounding situation in the wake of such problems. Solving such difficult problems remains a formidable challenge for researchers even after the enormous advancement in computing technology.

1.1 Combinatorial Optimization

Combinatorial optimization - a subfield of optimization - is concerned with finding an optimal solution to problems which are of combinatorial nature, i.e., problems having a large, but finite set of feasible solutions. Such problems are called combinatorial optimization problems and they involve both maximization and minimization problems. Each problem, having computational aspects, consists of an infinite set of instances. Typically, a *problem* refers to an abstract question to be answered and an *instance* of a particular problem refers to an input for this problem. Such problem can be regarded as an infinite set of instances together with a solution for each and every instance. For example, the task in traveling salesman problem is to find a minimum cost Hamiltonian circuit in a connected and weighted graph, while a particular instance as an input for this problem is a specified number of vertices and specified edge weights.

A formal definition of a combinatorial optimization problem is as follows: a combinatorial optimization problem Π, on a given instance, is a triple (S, Ω, f) where

- S is the set of feasible solutions,
- Ω is the set of constraints, and
- f is the objective function which assigns a value $f(s)$ to each feasible solution $s \in S$.

The objective of Π is to find an optimal solution $s^* \in S$. The term *feasible solution* refers to a solution that satisfies all the constraints in Ω. In Π, variables, which take part in constructing solutions, are discrete quantities whose values are used in optimizing an objective function. Note that an optimal feasible solution to a particular problem may not be unique for a given instance to that problem, and any minimization problem can be transformed into a maximization problem simply by changing the sign of the objective function, and vice versa.

There is no denying that combinatorial optimization problems have been drawing a tremendous amount of research, as these problems, arising in network, scheduling, production, planning, transportation, and so on, are of practical and theoretical importance and occur in various walks of life.

The combinatorial optimization problems can be broadly divided into following three classes:

1. *Subset selection problems:* The objective of these types of problems is to select a subset of items from a set of items under certain constraints in such a way that optimizes a given cost function. Knapsack problem, maximum clique problem, minimum spanning tree problem are common examples of subset selection problems.

2. *Permutation problems:* The objective of these types of problems is to arrange a given set of items in a particular order which optimizes a given cost function without violating any constraint. Traveling salesman problem, single processor scheduling are common examples of permutation problems.

3. *Grouping problems:* The objective of these types of problems is to find an optimal assignment of objects according to a given cost function into different groups subject to some constraint. Bin packing problem, graph coloring problem are common examples of grouping problems.

These classes are not disjoint, i.e., there exist problems which possess characteristics of more than one class. For example, consider multiple traveling salesman problem (MTSP) which is an extension of traveling salesman problem. Given $m > 1$ salesmen and $n > m$ cities to visit, MTSP seeks a partition of cities into m groups as well as an ordering among cities in each group so that each group of cities is visited by exactly one salesman in their specified order in such a way that each city is visited exactly once and sum of total distance traveled by all the salesmen is minimized. Clearly, MTSP involves aspects of grouping as well as permutation problem.

As already mentioned, only some combinatorial optimization problems can be optimally solved in polynomial time. On the other hand, most combinatorial optimization problems are $\mathcal{NP}$-Hard. $\mathcal{NP}$-Hardness of a problem means that the execution time of any known exact algorithm for the problem will grow exponentially with the size of problem instance and no polynomial time algorithm is known even to verify the optimality of a proposed solution. Hence, exact algorithms cannot be used for solving every arbitrary instance of arbitrary size of a $\mathcal{NP}$-Hard problem and their use is restricted to solving only instances of small size. Here, it is pertinent to mention that we may find an exact algorithm fast enough to produce the solution in a reasonable amount of time for arbitrary sized instance of a specific type. For example, several $\mathcal{NP}$-Hard graph problems, such as maximum clique problem, are known to be solvable in polynomial time on planar graphs. However, in many real world problems, the nature and the size of the instances make the use of exact algorithms impractical. Under these circumstances, one has to abandon the search for provably optimal solutions and look towards those methods which can find high quality solutions in a reasonable amount of time. *Approximate algorithms* [7] is the umbrella term for such methods.

1.2 Approximate Algorithms

Approximate algorithms, though do not provide guarantee of returning an optimal solution, are nevertheless good enough in most of the cases to return near-optimal solutions. These algorithms usually take polynomial time. It is to be noted that the solutions obtained by these algorithms are near to the optimal solution in terms of cost function, but may be far away from the optimal solution in the search space. Generally, the search space in $\mathcal{NP}$-Hard problems is rugged and consists of a large number of locally optimal solutions, many of which are of quality comparable to the optimal solutions. Approximate algorithms can be broadly divided into following three categories:

- *Heuristic* simply refers to rules originated from an intelligent guesswork based on the structure of a particular problem under consideration. It usually incorporates problem-specific knowledge of the problem under consideration. Heuristic generally finds a good solution quickly, but there is no proof that the solutions can never be arbitrarily bad.

- *Metaheuristic* is a set of algorithmic concepts that can be used to define heuristic methods for a wide range of problems. In other words, metaheuristic can be seen as a general algorithmic framework which can be adapted to different optimization problems without much modifications [7]. However, many metaheuristics require that the problem be represented in a form which is suitable for their applications. Some prominent examples of metaheuristics are genetic algorithms

[8, 9], simulated annealing [10, 11], ant colony optimization [7, 12], artificial bee colony algorithm [13], tabu search [14, 15] etc.

- *Approximation algorithm* is a heuristic with proven solution quality. By proven solution quality, we mean that solution returned by an approximation algorithm is guaranteed to be always within a certain factor of optimal solution.

It is to be noted that many combinatorial optimization problems are so hard that it is impossible to obtain good polynomial time approximation algorithms for them. In these situations, heuristics and metaheuristics are the only available options. In many such cases, a hybrid approach, where a metaheuristic is combined with a problem-specific heuristic, is used to obtain a good approximate solution. This thesis is particularly concerned with two metaheuristics - *artificial bee colony algorithm* and *ant colony optimization* - which fall under the broad class of techniques based on *swarm intelligence.*

1.3 Swarm Intelligence

Swarm intelligence inspired metaheuristic techniques have become an active area of research in the field of optimization over the last two decades. These techniques are based on modeling the behavior of the self-organizing individuals showing collective intelligence. Swarm refers to a collection of natural agents, such as honey bees and ants, or artificial agents that perform collective behavior. All agents perform stochastic behavior in order to interact locally with one another and with their environment. They follow simple local rules without any relation to the global pattern in parallel (all agents perform tasks simultaneously) and distributed (no centralized control) manner that exploit only local information that the individual agents exchange. Interactions between such self-organized agents lead to the emergence of intelligent global behavior unknown to the individual agents. Such a collective intelligence is termed *swarm intelligence.* Beni and Wang [16] coined the term swarm intelligence in the context of cellular robotic systems.

Self-organization and division of labor are the important characteristics of swarm in Nature. Self-organization is a characteristic of a coherent global behavior that emerges from the local interaction of the agents of a swarm system. Bonabeau *et al.* [17] explained self-organization through four characteristics:

1. *Positive feedback* promotes the purposeful formation of structures composed of many individual agents. Recruitment and reinforcement, which can be respectively seen in dances in bees and pheromone trail laying and following in ants, are the examples of positive feedback.

2. *Negative feedback* is required in order to avoid the saturation, such as food source exhaustion during bee foraging, pheromone evaporation in ants. It counterbalances positive feedback and helps to stabilize the collective pattern.

3. *Fluctuations* refer to random fluctuations in behavior of swarm individuals which often helps in finding new solutions.

4. *Opportunities for multiple interactions* must exist in a swarm. Actually, in a single interaction, exchange of information takes place between two or among at most a few individuals. Therefore, multiple interactions are required to spread the information globally.

Division of labor is another important characteristic of a swarm that helps in performing a variety of tasks simultaneously by specialized agents. It can be seen clearly in honey bees and ants.

1.3.1 The Behavior of Honey Bees

In Nature, the behavior of honey bees is one among the most widely studied social organisms behaviors. They are social as they live in colonies and work together in cooperation. They are industrious (working to death) and selfless (producing honey and dying to defend their hive). They are intelligent creatures. They are assumed to be insightful enough to participate in collective decision making process when faced with the problems of selecting new sites for nesting. They are equipped with eidetic memories, navigation systems and so many things. They show division of labor as they specialize in multiple tasks, such as storing, retrieving and distributing honey, communication and foraging, and so on to fulfill their needs simultaneously.

At the centre of all activities in the hive, it is the queen who is the only egg-laying female. The queen is bound to lay eggs, as she lays around 2000 eggs a day. She produces fertilized and unfertilized eggs. Workers are female bees produced from fertilized eggs, whereas drones are male bees produced from unfertilized eggs. The queen stops laying eggs when the size of the colony becomes large. An egg becomes an adult bee after going through several phases, such as larva and pupate. The queen usually mates only once in her life. She goes on mating flight, away from the hive and mates with many drones before returning back to the hive. She remains fertile for two or more years by the sperm she collects during mating flight. When sperm are exhausted, she begins to produce unfertilized eggs, and then one of her daughters is selected (by worker bees) as a queen who succeeds her in the function of laying eggs. Worker bees help in developing a new queen bee by feeding her royal jelly exclusively. In order to decide a new nest site, the queen bee with a large group of worker bees move to the new nest site scouted by worker bees beforehand. Information about new sites of nesting already explored by scout bees is communicated to each other by dancing. The intensity of dance, which is proportional to the quality of nest site, attracts more scout bees that helps in making group decision during selection of the most suitable new nest site [18].

In a typical hive of 30,000 bees, only about hundreds of bees are males called drones. They mate with the queen bee, and then they die. These drones are responsible for fertilizing the queen.

Worker bees are female bees who literally work themselves to death. Worker bees live for 4-9 months during winter, but only 6 weeks during summer. They build wax honeycomb cells for the colony in which the queen lays eggs, and honey is stored. They feed the larvae, drones, and the queen. They maintain the hive neat and clean, ventilate the hive, guard the hive from intruders among other things. When a worker bee is 3 weeks old, she becomes a forager and spends the rest of her life in collecting nectar which is regarded to as the most important task in the hive. [19, 20, 21] have investigated the foraging behavior of the individual bee. A worker bee flies up to 3 miles away from her hive in search of food sources and amazingly returns to the same hive with the help of her spatial memory [22]. After discovering a food source, i.e., a flower, she stores the nectar in her honey stomach. Honey making process starts with the secretion of an enzyme on the nectar in her honey stomach. When she returns to the hive, she unloads the nectar to empty honeycomb cells. After unloading the nectar, she recruits other worker bees through most amusing form of communication - a dance called waggle dance. She informs other worker bees about the richness of the food source, its direction and distance from the hive through dancing. These pieces of information are encoded symbolically in movements and sounds that she produces [23, 24].

A good survey on the behaviors of honey bee swarm can be found in [25].

1.3.2 Swarm Intelligence Techniques Based on the Behaviors of Honey Bee Swarm

As described above, honey bee swarm exhibit a variety of intelligent behaviors such as dance and communication, role of queen bee, mating flights, nest site selection, navigation, task selection, foraging and so on that can be used as models for intelligent systems to solve complex problems [25]. A series of approaches based on these behaviors has been developed. For example, approaches based on simulating the role of queen bee are addressed in [26, 27, 28, 29, 30, 31]; frameworks based on dance and communication of honey bees are introduced in [32, 33, 34, 35, 36]; frameworks based on task allocation of honey bee foragers are modeled in [37, 38, 39, 40]; the works on the principles of collective decisions and nest site selection of honey bees are presented in [41, 42, 43]; the works on simulating the behaviors of mating, marriage and reproduction are addressed in [44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54]; a study about floral constancy is addressed in [55]; necrophoric bee behavior in a robot swarm by pheromone communication is introduced in [56]; paradigms based on the navigation behavior of bees are proposed in [57, 58, 59]; various models based on the intelligent foraging behavior of honey bee swarm are introduced in [13, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79], and so on.

A major portion of this thesis is devoted to artificial bee colony algorithm [13] which is based on the the intelligent foraging behavior of honey bee swarm.

1.3.3 Overview of Artificial Bee Colony Algorithm

Artificial bee colony (ABC) algorithm is a population-based new metaheuristic technique motivated by the intelligent foraging behavior of honey bee swarm. Usually, the foraging bees are divided into three classes – employed, onlookers and scouts. "Employed" bees are those bees that are presently exploiting the food sources. Employed bees are responsible for bringing loads of nectar from the food sources to the hive and sharing the information about the food sources with onlooker bees. "Onlookers" are those bees that are waiting in the hive for the information to be shared by the employed bees about their food sources and “scouts” are those bees that are presently searching new food sources in the vicinity of the hive. Employed bees share information about the food sources by dancing in a common area in the hive called dance area. The nature and duration of a dance depends on the nectar content of the food source currently being exploited by the dancing bee. Onlooker bees observe numerous dances before choosing a food source. The onlookers have a tendency to choose a food source with a probability proportional to the nectar content of that food source. Therefore, good food sources attract more onlookers than the bad ones. Whenever a bee, whether it is scout or onlooker, finds a food source it becomes employed. Whenever a food source is completely exhausted, all the employed bees associated with it leave, and again become scouts or onlookers. So in a way, scout bees can be perceived as performing the job of exploration, whereas employed bees and onlooker bees can be perceived as performing the job of exploitation. This foraging behavior of honey bee swarm can be illustrated through Figure 1.1, where employed bee, onlooker bee and scout bee are denoted by **E**, **O** and **S** respectively, while **F** represents a food source. In this figure, a dancing employed bee is marked by a solid red point and those onlooker bees who are watching the dance are marked by solid black points.

Inspired by this foraging behavior of honey bee swarm, Karaboga [13] introduced the ABC algorithm which has been extended further by [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90]. Like real bees, in ABC algorithm, artificial bees are also classified into scout, employed and onlooker bees with similar functions. In ABC algorithm, each food source represents a possible solution to the problem being investigated and the nectar content of a food source corresponds to the fitness of the solution being represented by that food source. ABC algorithm assumes that there is only one employed bee for every food source, i.e., the number of employed bees is same as the number of food sources. Usually, the number of onlooker bees is also taken to be equal to the number of employed bees. The employed bee of an exhausted food source becomes a scout and as soon as it finds a new food source it again becomes employed. The action of a scout bee is simulated by generating a new food source (solution) randomly and associating the scout bee in consideration with this newly generated food source to make it employed again. We will use food source and solution interchangeably throughout this thesis in the context of ABC algorithm.

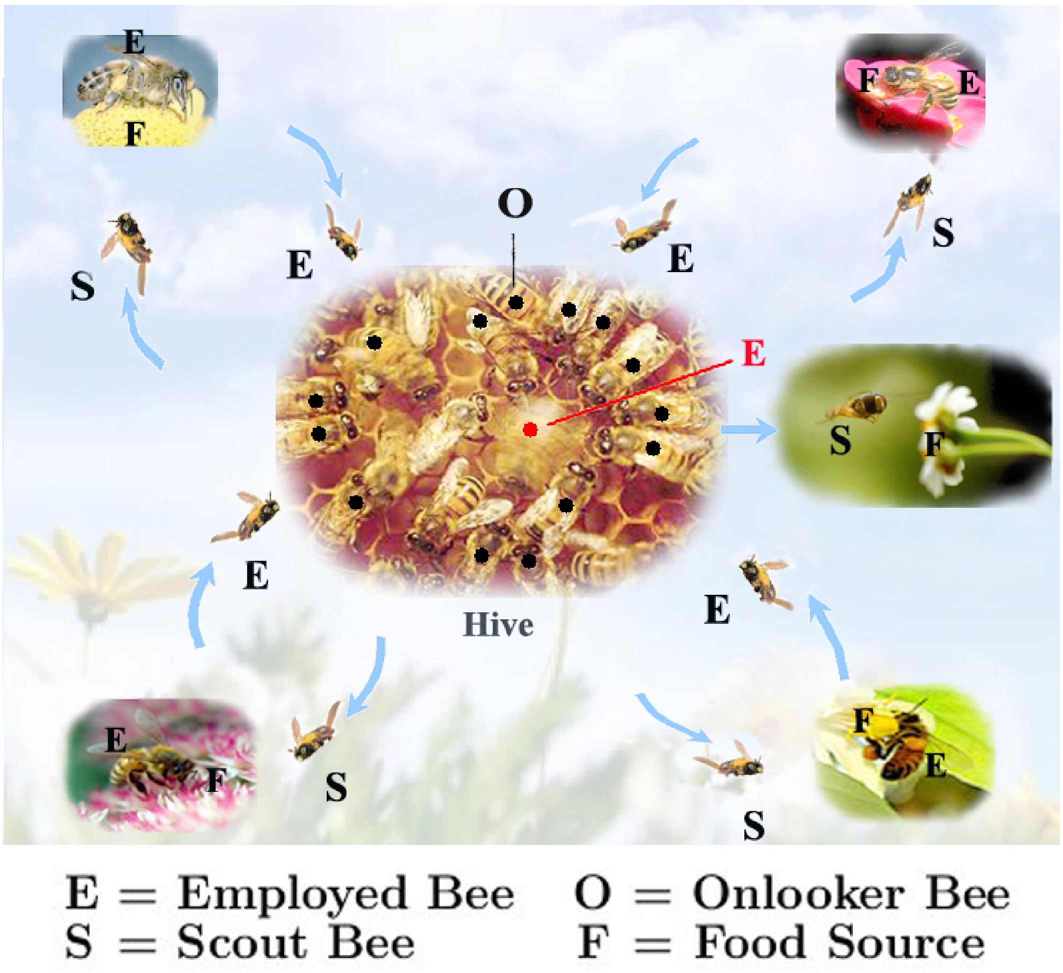


Figure 1.1: Foraging behavior of honey bee swarm

ABC algorithm follows an iterative process. It begins by generating a certain number of food sources randomly and associating each of these food sources with an employed bee. Then at each iteration, each employed bee determines a new food source in the neighborhood of its currently associated food source and evaluates its nectar content (fitness). If the nectar content of this new food source is higher than that of its currently associated food source, this employed bee associates itself with this new food source abandoning the old one, otherwise it continues with the old one. The actual process of determining a new food source in the neighborhood of a particular food source depends on the problem under consideration. This phase, where every employed bee determines a neighboring food source, can be termed as *employed bee phase*. Once this phase is over, *onlooker bee phase* begins.

At the beginning of onlooker bee phase, each onlooker bee selects a food source using some probabilistic selection criterion. Usually, the selection criterion prefers good food sources over bad ones so that more onlooker bees will be assigned to good food sources which in turn results in more exploration in the vicinity of these food sources. After all onlookers have selected their food sources, each of them determines a food source in the neighborhood of their selected food source in a manner similar to employed bee phase and computes its fitness. Among all the neighboring food sources determined by the onlooker bees associated with a particular food source i and the food source i itself, the best food source is determined. This best food source becomes the new location for the food source i in the next iteration. Once the new locations of all food sources are determined, the onlooker bee phase ends and the next iteration of ABC algorithm begins. The whole process is repeated until the termination condition is satisfied. Hence, each iteration of ABC algorithm consists of two phases viz. employed bee phase and onlooker bee phase. If a solution corresponding to a particular food source does not improve for a predetermined number of iterations *limit*, then that food source is assumed to be exhausted and its associated employed bee abandons it to become a scout. A new food source is randomly generated to handle this scout. This scout bee is associated with the newly generated food source and its status is again changed from scout to employed. Note that the parameter *limit* is an important control parameter of ABC algorithm, as it is responsible for maintaining the delicate balance between exploration and exploitation. A small value of *limit* parameter favors exploration over exploitation, whereas reverse is true for large value.

In essence, ABC algorithm is based on the fact that in the neighborhood of a good solution, chances of finding even better solutions are high. That is why, more onlookers are deputed for good solutions so that their neighborhood can be explored more thoroughly in comparison to worse solutions. Though in the employed bee phase, every solution is given a chance to improve itself, the onlooker bee phase is biased towards good solutions which get more chances to improve themselves. However, if a solution is locally optimal with respect to the whole neighborhood then it cannot be improved. Therefore, any attempt to improve it is futile. This is where the concept of scout bee comes to our rescue. As it is computationally expensive to determine whether a solution is locally optimal or not, ABC algorithm assumes that if a solution does not improve for certain number of iterations, then it is locally optimal and therefore, replaces this solution with a randomly generated solution.

The actual process for determination of a food source in the neighborhood of a particular food source depends on the nature of the problem under investigation. Karaboga's original ABC algorithm was proposed for continuous optimization. In this model, the food source in the neighborhood of a particular food source is determined by altering the value of one randomly chosen solution parameter and keeping other parameters unchanged. This is done by adding to the current value of the chosen parameter the product of a uniform variate in [-1, 1] and the difference in values of this parameter for this food source and some other randomly chosen food source. This method cannot be applied for combinatorial optimization problems for which it produces at best a random effect.

In the domain of combinatorial optimization problems, Singh [85] was the first to present a variant of ABC algorithm that is applicable for subset selection problems. To generate a neighboring solution, in this method, an object is randomly removed from the solution and in its place another object, which is not already present in the solution, is inserted. The object to be inserted is selected from another randomly chosen solution. If there are more than one candidate objects for insertion, then ties are broken arbitrarily. This method is based on the idea that if an object is present in one good solution, then it is highly likely that this object is present in many good solutions. Another advantage of this method is that it helps in keeping a check on the number of duplicate solutions. If, during the employed bee phase, the method fails to find an object in the randomly chosen solution different from the objects in the original solution, then that means that the two solutions are identical. Such a situation was called *collision* and it is resolved by making the employed bee associated with the original solution scout. This eliminates one duplicate solution. Therefore, here there is another way through which a bee can become a scout. However, if, during the onlooker bee phase, a collision happens while generating a neighboring solution for an onlooker bee, then another solution is chosen randomly for selecting an object from it to insert in the original solution. This process is repeated till the neighborhood procedure finds a solution that is different from the original solution. Here, it is to be noted that in case of a collision it is worthless to generate a random solution for an onlooker bee.

The reason is that for survival this randomly generated solution has to compete with the original solution as well as with the solution of all other onlooker bees which are also associated with the same original solution. Hence, it is highly unlikely that such a randomly generated solution survives.

Later, Pan *et al.* [86] proposed another variant of ABC algorithm which is suitable for permutation problems. In this approach, generating a neighboring solution depends on insert and swap operators. Moreover, this approach, in case of scout bee, always generates a solution in the neighborhood of the best solution in the population instead of generating a solution randomly.

For grouping problems, Sundar and Singh [87] introduced one more variant of ABC algorithm which was the first extension of ABC algorithm for a grouping problem. This method was applied for quadratic multiple knapsack problem that has a fixed number of groups. In this approach, generating a neighboring solution depends on two procedures, i.e., knapsack replacement and perturbation which are applied in a mutually exclusive manner. knapsack replacement strategy tries to preserve grouping information as far as possible, while perturbation strategy helps in providing diversity in the population and also prevents the algorithm from converging prematurely.

The *roulette wheel selection method* [9] and the *binary tournament selection method* [91] are the two most commonly used methods for selecting a food source for an onlooker bee. In roulette wheel selection method, a candidate food source is selected randomly from among all candidate food sources in such a way that the probability of selecting a food source is proportional to its relative fitness in the population, i.e., the probability p_i of selecting a food source i is as follows:

$$p_i = \frac{f_i}{\displaystyle\sum_{j=1}^{N} f_j}$$

where f_i is the fitness of the candidate solution represented by the food source i and N is the total number of food sources. In the original ABC model, Karaboga [13] suggested the use of this selection method.

Algorithm 1: Pseudo-code of ABC Algorithm

```
Generate $n_e$ random solutions $E_1, E_2, ..., E_{n_e}$;
$best\_sol \leftarrow$ best solution among $E_1, E_2, ..., E_{n_e}$;
while (termination criteria is not satisfied) do
    for ($i \leftarrow 1$ to $n_e$) do
        $E' \leftarrow$ Generate_Neighboring_Solution($E_i$);
        if ($E' = \emptyset$) then
            Scout bee processing;
        else if ($E'$ is better than $E_i$) then
            $E_i \leftarrow E'$;
        else if ($E_i$ has not changed over last limit iterations) then
            Scout bee processing;
        if ($E_i$ is better than best_sol) then
            $best\_sol \leftarrow E_i$;
    for ($i \leftarrow 1$ to $n_o$) do
        $p_i \leftarrow$ Select_and_Return_Index($E_1, E_2, ..., E_{n_e}$);
        $On_i \leftarrow$ Generate_Neighboring_Solution($E_{p_i}$);
        if ($On_i = \emptyset$) then
            Apply a procedure, which depends on the problem under
            consideration, for $On_i$;
        if ($On_i$ is better than best_sol) then
            $best\_sol \leftarrow On_i$;
    for ($i \leftarrow 1$ to $n_o$) do
        if ($On_i$ is better than $E_{p_i}$) then
            $E_{p_i} \leftarrow On_i$;
```

In binary tournament selection method, two different food sources are picked uniformly at random and the better of the two candidate food sources (according to their fitness) is selected with fixed probability p_{bt}, otherwise the worse of the two candidate solutions is selected.

The pseudo-code of ABC algorithm is given in Algorithm 1, where n_e and n_o are respectively the number of employed and onlooker bees. *Generate_Neighboring_Solution (X)* is a function that returns either a solution in the neighborhood of the solution X or $\emptyset$, if it fails to find a neighboring solution. Exact implementation of this function depends on the problem under consideration. If this function returns $\emptyset$ in the employed bee phase, then the employed bee associated with the solution E_i becomes a scout and how this scout is dealt with depends on the problem under consideration. However, if this function returns $\emptyset$ in the onlooker bee phase, then again a problem-specific procedure is applied. *Select_and_Return_Index*(X_1, X_2,...,X_{n_e}) is another function that selects a solution from solutions X_1, X_2,...,X_{n_e} for an onlooker and returns the index of the solution selected. Exact implementation of this function depends on the selection policy used.

An excellent survey on the ABC algorithm can be found in [25].

1.3.4 The Behavior of Ants

In Nature, ants are also social insects as they live in colonies and their behavior for the weal survival of the colony is collective rather than individualistic. They exhibit division of labor. As a whole, the colony of ants presents a highly structured social organization that helps in accomplishing complex tasks such as foraging and transport which are beyond the capability of an individual ant. In all these tasks, ants coordinate their tasks via *stigmergy* - a form of indirect communication mediated by modifications of the environment.

The colony of ants is divided into three types of ants - queen, male and worker. The queen does the job of laying eggs. The males are responsible for mating with the queen, whereas female ants are workers. For the survival of the colony, workers perform a variety of tasks such as foraging, defending, food preparing, attending to the safety of the queen and nest construction among many other things [92].

Foraging, which is based on self-organization and division of labor, is one of the most important and crucial tasks performed by the colony of worker ants (in short ants) especially about the manner in which ants find the shortest path from their nest to food source and vice versa. While walking from their nest to food sources and vice versa, ants lay down on the ground a chemical substance called *pheromone*, forming in this way, a pheromone trail which is used for stigmergetic communication. Other ants are able to smell this pheromone, and its presence influences the choice of their paths, i.e., ants choose probabilistically the paths marked by strong pheromone concentrations. Pheromone is a volatile chemical substance and evaporates over time, thereby decreasing the intensity of pheromone trails and favoring path exploration. While searching for a path from the nest to the food source, ants tracing the shortest path will return sooner, hence immediately after their return, the concentration of

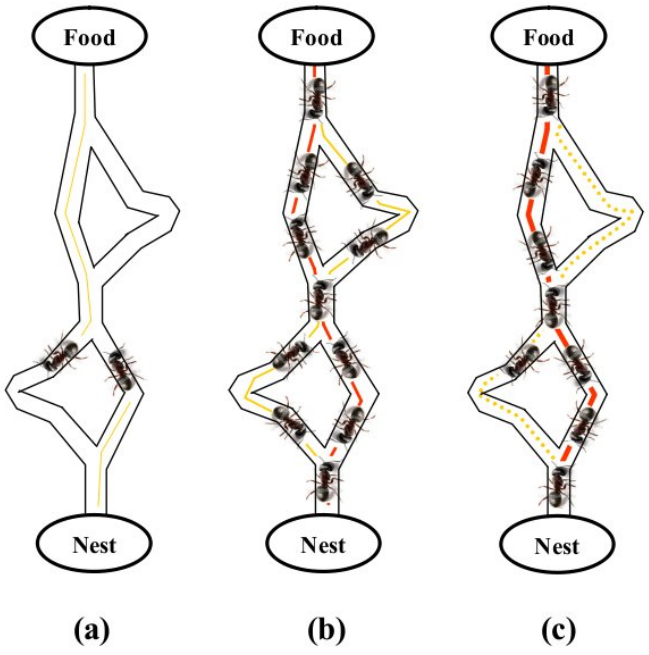


Figure 1.2: Foraging behavior of ants

pheromone will be more on this path influencing other ants to follow this path which in turn leads to accumulation of more pheromones. After some time, this will result in almost the whole colony of ants following the same path. Thus, pheromone trails allow the ants to find the shortest paths between their nest and food source and the formation of pheromone trails is the result of the cooperation among individuals of the whole colony.

Foraging behavior of ants can be illustrated through Figure 1.2. Initially, there is no pheromone on the path and ants walk stochastically without any bias for a particular path between nest and food source and vice versa (see Figure 1.2 (a)). Due to stochastic walk without any bias, it is expected that, on an average, half of the ants will pass through the shortest path and half of the ants will pass through the longest path (see Figure 1.2 (b)). Since ants lay down pheromone on the path while walking and ants selecting the shortest paths are the first to reach the food and to start their return to the nest, therefore, it results in a large amount of pheromone on this shortest path compared to the largest path. It can be seen in Figure 1.2 (c) that the pheromone trail in the shortest path between nest and food source is in solid line while the longest path in thin line. This large amount of pheromone starts influencing other ants to select the shortest path again. After some time, almost the whole colony of ants converges to this shortest path (see Figure 1.2 (c)).

Gross *et al.* [93] and Deneubourg *et al.* [94] thoroughly examined the pheromone trail behavior of foraging ants with the help of double-bridge experiments. They have concluded from their controlled experiments that pheromones on the ground, deposited by foraging ants, are used for marking paths in search of a food source and foraging ants are capable of finding the shortest path between their nest and food source through indirect and mutual interaction aided by pheromone trails.

1.3.5 Overview of Ant Colony Optimization

Ant colony optimization (ACO) is a generic name given to a family of swarm intelligence techniques based on the cooperative foraging behavior of real ants. The basic idea behind these techniques is same. These techniques have been successfully applied to solve many hard combinatorial optimization problems. The basic idea behind ACO is to model the problem under consideration as a graph. The ACO algorithm iteratively distributes a set of artificial ants (ants for short) on this graph to perform randomized walks in search of high quality solutions. Generally, individual ants do not find high quality solutions on their own. High quality solutions emerge from the cooperative interaction among ants which are achieved by using and updating pheromone values associated with components of the graph. The components can be vertices or edges or both. What constitutes a component varies from problem to problem. The cooperative interaction among ants through pheromones is governed by following two rules:

1. *Probabilistic decision rule*: In order to construct a solution, each ant follows an incremental procedure. At each step of this incremental procedure, a component is selected from among all candidate components using a probabilistic decision rule and added to the solution. This rule is biased by pheromone values and heuristic information values. Heuristic information value is known a priori. So we can say that the pheromone value associated with a component is a measure of dynamic or learned desirability of that component, whereas heuristic information value is a measure of the static desirability of a component. The probabilistic

decision rule favors the components with the higher pheromone and heuristic values. Such components have higher probability of selection.

2. *Pheromone Update Rule*: Pheromone update rule changes the degree of preference of a component by changing its associated pheromone value, depending on the quality of the solutions containing that component. Pheromone update rule inserts both positive and negative feedback in ACO. Pheromone values on those components, which are part of a solution constructed by an ant, are enhanced. This is done to encourage ants in future iterations to construct solutions containing these components. So this is a positive feedback mechanism. While pheromone evaporation, which can be considered as a negative feedback mechanism, is used to prevent the search process from trapping in local optima. During pheromone evaporation, pheromone values on some or all components are reduced by a constant factor. Pheromone evaporation on components helps in forgetting the bad choices made in the past, thereby facilitating a better exploration of the search space.

Variations in these two rules lead to different ACO algorithms.

In short, ACO is an iterative, stochastic and incremental approach for finding high quality solutions to combinatorial optimization problems. Initially, the pheromone value on each component is set to some value. At each iteration, a colony of (artificial) ants constructs solutions to a particular problem under consideration with the help of pheromone model. Construction of a solution starts from an empty solution. At each step, the selection of a component from all candidate components is done by applying a probabilistic decision rule that makes use of pheromone trails and heuristic information. When a component is selected, it is added to the current partial solution. Once an ant has constructed a solution, the ant evaluates the solution that will be used for pheromone update. In pheromone update, artificial pheromone trails are modified. The pheromone trails value can either increase, as the ants lay down pheromones on the components they use, or decrease due to pheromone evaporation.

Since the development of the first ACO algorithm called Ant System by Dorigo and colleagues [12, 95, 96], many different ACO algorithms have been reported in the literature, some of which are explained subsequently. For a comprehensive survey on different ACO algorithms and their applications, see [7].

1.3.5.1 Ant System

Ant System (AS) [12, 95, 96] was the first ACO algorithm. In AS, the pheromone value on each component is initialized with a value slightly higher than the expected amount of pheromone laid down by the ants in one iteration. The reason behind this policy is that if the pheromone values are initialized with a low value, then the search becomes biased by the first solutions constructed by the ants and as a result, the search space cannot be explored fully. On the other hand, if the pheromone values are initialized with a high value, then the biased search by ants will not start until enough pheromone values are decreased by pheromone evaporation, thereby wasting many iterations.

All ants construct solutions in a concurrent and asynchronous manner. The roulette wheel selection method is used as probabilistic decision rule in AS.

Once all ants have constructed their solutions, pheromone values are updated. First pheromone evaporation is performed where pheromone values on all components are reduced by a constant factor. After pheromone evaporation, all ants augment pheromone values associated with components present in the solution constructed by them. Components that are selected by more than one ant receive more pheromone and therefore, it is highly likely that these components will be selected by ants in future iterations of the algorithm.

1.3.5.2 Elitist Ant System

Elitist Ant System (EAS) [12, 95] was the first improved version of the original AS algorithm. EAS is similar to AS except for the fact that in each iteration, the components of the best solution found since the beginning of the algorithm receive additional pheromone reinforcement.

1.3.5.3 Ant Colony System

Ant Colony System (ACS) [97, 98] introduces new mechanisms for better performance which are not in AS. The probabilistic decision rule, which is called pseudo-random-proportional rule in ACS, is governed by a parameter $q_0 \in [0, 1]$. With probability q_0, the best component is selected from all the candidate components, otherwise a component is selected using the roulette wheel selection method. There are two pheromone update rules in ACS viz. global pheromone update rule and local pheromone update rule.

The global pheromone update rule is used at the end of an iteration when all ants have constructed their solutions. In this rule, the best solution found since the beginning of the algorithm is used for updating the pheromone. Pheromone evaporation and augmentation both take place only on the components belonging to this best solution.

The local pheromone update rule is used by each ant while constructing a solution. According to this rule, each time an ant selects a component for the solution construction, the pheromone value associated with this component is reduced so that the probability of selecting this component becomes less for the following ants. This rule promotes exploration as it prefers components that have not yet been visited.

In addition, ACS uses a *restricted candidate list* to restrict the number of available components at each step of solution construction. The restricted candidate list consists of a number of preferred components according to some heuristic criterion. Only the components belonging to restricted candidate list can take part in selection at each construction step. If restricted candidate list is empty at any step, then all candidate components take part in selection.

1.3.5.4 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) proposed by Stützle and Hoos [99, 100] is among the most successful ACO variants. Stützle and Hoos [99, 100] modified AS for $\mathcal{MMAS}$ by introducing some important additional features. The important features of $\mathcal{MMAS}$ are following:

1. In $\mathcal{MMAS}$, the best solution found is strongly exploited. The best solution found can either be current iteration's best solution or best-so-far solution (global best solution). In general, the pheromone augmentation is performed only on components present in iteration best solution or global best solution or both (in an alternate way).

2. In order to avoid search stagnation due to excessive growth of pheromone values associated with the components of a good suboptimal solution, $\mathcal{MMAS}$ explicitly restricts all pheromone values between minimum and maximum values.

Algorithm 2: Pseudo-code of a general ACO algorithm

```
Initialize pheromones;
while (termination criteria is not satisfied) do
    for ($i \leftarrow 1$ to $n_a$) do
        Iteratively construct a solution for ant $i$ where at each iteration,
        a component is selected from all candidate components using
        probabilistic decision rule;
    Update pheromones using pheromone update rules;
```

3. In $\mathcal{MMAS}$, pheromone values are initialized to maximum, which, along with a small pheromone evaporation rate, helps in achieving better exploration of the search space at the beginning of the algorithm.

4. Pheromone trails are reinitialized each time when the best solution does not improve over a certain number of consecutive iterations.

It should be noted that in AS and ACS, pheromone values are restricted implicitly, whereas in $\mathcal{MMAS}$, all pheromone values are explicitly constrained between minimum and maximum values.

Since there are so many variants of ACO algorithm, it is not possible to give a generic pseudo-code for all variants. The pseudo-code of a representative ACO algorithm is given below in Algorithm 2, where n_a is the number of ants. All ACO algorithms described in this thesis follow this pseudo-code.

1.4 Scope of the Thesis

This thesis is focused primarily on solving some $\mathcal{NP}$-Hard combinatorial optimization problems through two swarm intelligence techniques viz. artificial bee colony algorithm and ant colony optimization algorithm. Actually, artificial bee colony (ABC) algorithm was originally designed for continuous optimization and was under-explored in the domain of combinatorial optimization at the time when the work was started on this thesis. [85] was the only reference available in the literature that described the application of ABC algorithm to the leaf constrained minimum spanning tree problem which is a subset selection problem. With the intention of further exploring the capabilities of ABC algorithm for combinatorial optimization, we chose to work on ABC algorithm. The choice of ant colony optimization (ACO) was more straightforward as it is one among the most successful techniques for solving combinatorial optimization problems.

This thesis is divided into 9 chapters beginning with this introductory chapter. We have considered 8 problems in this thesis. Out of these 8 problems, first 6 problems are subset selection problems, next problem is a permutation based problem, whereas the last one is a grouping problem. Apart from being theoretically challenging, these problems have many practical applications in diverse fields such as networks, computational biology, transportation, scheduling, production, social network analysis, and so on. Chapters 2 to 8 describe the approaches that we have developed for these problems, whereas chapter 9, which is the last chapter, provides some concluding remarks and directions for future research. In the following, we outline the content of each of these chapters.

In chapter 2, we have proposed an ABC algorithm for minimum routing cost spanning tree problem (MRCST), where the best solution obtained through ABC algorithm is improved further by a local search. Given a connected, weighted and undirected graph, MRCST seeks a spanning tree of minimum routing cost on this graph where routing cost of a spanning tree is defined as the sum of the costs of the paths connecting all possible pairs of distinct vertices in that spanning tree. We have compared our approach with the best heuristic approaches reported in the literature for MRCST. On the benchmark instances considered, our approach outperforms other approaches in terms of solution quality. Except for two previously proposed approaches, our approach is also faster than other approaches.

Chapter 3 deals with quadratic minimum spanning tree problem (Q-MST). Q-MST is an extension of well-known minimum spanning tree problem in graphs. In Q-MST, costs are associated not only with edges of the graph, but also with ordered pairs of distinct edges and the objective is to find a spanning tree that minimizes the sumtotal of the costs associated with individual edges and the costs resulting from ordered pairs consisting of those edges present in the spanning tree. Our approach to this problem is an extension of the artificial bee colony algorithm based approach developed in the previous chapter for MRCST problem. We have compared our approach with the best approaches known so far. Our approach obtains better quality solutions than all the other approaches. Barring one previously proposed approach, our approach is also faster than all the previously proposed approaches.

Chapter 4 presents a hybrid approach combining an artificial bee colony algorithm with a local search to solve non-unicost set covering problem (SCP). Given an $m \times n$ 0-1 matrix $A = (a_{ij})$ and non-negative n-dimensional vector $C = (c_j)$ where each element c_j of C gives the cost of selecting the column j of matrix A. If a_{ij} is equal to 1, then it indicates that row i is covered by column j, otherwise it is not. The objective of SCP is to find a minimum cost subset of columns of A such that each row of A is covered by at least one column in the subset. We have extended an already existing greedy local search and used the extended version in our hybrid approach effectively. We have compared our approach with the best population-based approaches and the overall best approaches. Its results are comparable with all these methods in terms of solution quality, though the overall best methods are much faster. As far as comparison with other population-based methods is concerned, except for one method, it outperforms all the other population-based methods in terms of solution quality. However, it is slower than some population-based methods.

Chapter 5 is concerned with dominating tree problem (DTP). Given an undirected, connected and weighted graph, DTP seeks on this graph a tree with the minimum total edge weight such that each vertex of the graph is either in this tree or adjacent to a vertex in this tree. This chapter describes three approaches for this problem. First, a problem-specific heuristic has been proposed which produces much better results in comparison with existing problem-specific heuristics for this problem. With the intent of improving the solution quality even further, two swarm intelligence techniques viz. an artificial bee colony algorithm and an ant colony optimization algorithm have been proposed. These two techniques have obtained much better results at the expense of increased computation time. Performance of these two techniques are comparable to each other in terms of solution quality. Though ant colony optimization algorithm produces slightly better results, it is several times slower than artificial bee colony algorithm on large instances for this problem.

Chapter 6 deals with two bounded-degree spanning tree problems. Given a connected graph, a vertex of this graph is called a branch vertex, if its degree is greater than two. Pertaining to branch vertices, we have studied two bounded-degree spanning tree problems – first problem seeks a spanning tree of this graph with the minimum number of branch vertices (MBV), whereas the second problem seeks a spanning tree of this graph with the minimum degree sum of branch vertices (MDS). MBV and MDS are related problems, but not the same. Two heuristics approaches are presented for each problem. The first approach is a problem-specific heuristic, whereas the latter approach is a hybrid ant colony optimization algorithm. A significant feature of our ACO approaches is the use of two pheromones. Our problem-specific heuristic approaches outperform the previously proposed best problem-specific heuristic approaches for these problems. As the results obtained by all these problem-specific heuristics are inferior to exact approaches and the difference in solution quality grows with instance size, we have proposed the hybrid ACO approaches. Computational results show the effectiveness of our hybrid ACO approaches.

Chapter 7 addresses a single machine scheduling problem with earliness and tardiness costs and no unforced machine idle time. We have proposed an artificial bee colony (ABC) algorithm for this permutation-based problem. A local search is used inside ABC algorithm to further improve the schedules obtained through it. A variant of this basic ABC approach, referred to as ABC-MNAI, is also considered in this chapter, where the best solution obtained through ABC algorithm is improved further via an exhaustive local search. We have compared these two approaches with 16 heuristic approaches reported in the literature on existing set of benchmark instances as well as on a new set of large instances. The basic ABC approach performs better than all the 16 approaches on existing set of instances, but it performs worse in comparison to one approach on two largest groups of instances containing 750 and 1000 jobs. ABC-MNAI outperforms all other approaches in terms of solution quality on both existing as well as on new groups of large instances. It also requires less execution time in comparison to other best performing approaches on most group of instances. We have shown that for all groups of instances, even if we compare the results of ABC-MNAI approach against the best results among all the approaches, our results are still better.

Chapter 8 describes two approaches viz. a steady-state grouping genetic algorithm (SSGGA) and a grouping-based artificial bee colony (GABC) algorithm for blockmodel problem. The objective of this problem is to find small number of large blocks containing structural similarities or equivalences among entities in a given graph representing a complex network. We have compared our approaches against the best approaches reported in the literature. Computational results clearly demonstrate the superiority of our approaches. Our approaches have obtained better quality solutions in shorter time. To our knowledge, GABC is the first approach for a grouping problem with variable number of groups. Therefore, the success of GABC in solving the blockmodel problem demonstrates the applicability of ABC approach in a new domain.

Chapter 9 concludes the thesis by summarizing the contribution made in different chapters of the thesis. It also outlines some directions for future research.

Chapter 2

Minimum Routing Cost Spanning Tree Problem

2.1 Introduction

Given a connected, weighted, and undirected graph, the routing cost of any of its spanning tree is defined as the sum of the costs of the paths connecting all possible pairs of distinct vertices in that spanning tree. Minimum routing cost spanning tree problem (MRCST) consists in finding a spanning tree with minimum routing cost among all spanning trees of the graph. It is an $\mathcal{NP}$-Hard problem [101]. To find the routing cost of a spanning tree, it is not necessary to enumerate every possible path in that spanning tree. Instead, routing cost can be computed more easily by first determining for each edge of the spanning tree, the count of the paths containing that edge, and then summing the product of this count and edge weight for every edge [102]. Formally, let $G = (V, E)$ be a connected undirected graph where V denotes the set of vertices and E denotes the set of edges. Given a non-negative weight function $w : E \to \mathbb{R}^+$ associated with its edges, MRCST seeks on this graph a spanning tree $T \subseteq E$ that minimizes

$$W = \sum_{e \in T} c_e.w(e) \tag{2.1}$$

where c_e is the count of those paths in T which contains edge e. c_e can be determined by calculating the product of the number of vertices in the two components arising as a result of temporarily removing e from the spanning tree. This is due to the fact that any path connecting a vertex in one component to a vertex in another has to contain e.

Therefore, the total number of paths containing e is simply the product of the number of vertices in one component with the number of vertices in other component. Here, it is to be noted that in a spanning tree, exactly one path exists between any pair of distinct vertices. Also note that we have used vertex and node interchangeably throughout this thesis.

MRCST has several important applications in communication network design, where the cost of an edge may represent the cost incurred in routing messages between its endpoints. For example, consider a situation [103], where the cost of an edge represents the delay in routing a message between its endpoints and one has to find a spanning tree that minimizes the average delay of communicating between any two vertices via the spanning tree. The delay between any two vertices is the sum of the delays of the edges lying on the unique tree path connecting the two vertices. Clearly, this is an instance of MRCST as minimizing the average delay is equivalent to minimizing the total delay between all pair of vertices [103]. The problem gains importance in heterogeneous computer networks, where different subnetworks are connected through bridges [104]. Bridging mandates that the active network topology should be a spanning tree. In such a situation, a MRCST is always an optimal spanning tree from the routing cost point of view provided the probability of communication between any pair of nodes is the same. MRCST also finds application in multiple sequence alignment problems in computational biology [103].

Most of the work on MRCST is focused on designing approximation algorithms for it. Wong [105] proposed an approximation scheme that computes n shortest path trees, where n is the number of vertices in the graph, and returns the tree with smallest routing cost among these trees. Wu *et al.* [106] proposed an approximation algorithm that is applicable to metric graphs only. Grout [107] described an algorithm that gives better results in shorter time on homogeneous graphs in comparison to the approach of Wong, but performs poorly on non-homogeneous graphs. Campos and Ricardo [104] proposed another algorithm that gives results comparable to Wong's algorithm in lesser time for practical cases. Fischetti *et al.* [108] presented an exact branch-and-price algorithm for MRCST.

Though the approximation algorithms provide solutions that are guaranteed to be within a certain factor of optimum, solutions obtained through these algorithms on many problems cannot compete with the solutions provided by the state-of-the-art metaheuristics. Therefore, on many problems, the use of these approximation algorithms is limited mainly to either dynamic environment where a quick solution is desired or providing an initial solution to a metaheuristic technique.

Among the metaheuristic techniques, Julstrom [102] proposed two genetic algorithms and a stochastic hill climber for the problem. One genetic algorithm uses Blob code [109] to represent a spanning tree, whereas other is edge-set-coded, i.e., it represents a spanning tree by the set of its edges [110]. Edge-set-coded genetic algorithm uses a crossover operator that is derived from Kruskal's algorithm [111] for finding the minimum spanning tree. The mutation operator for this genetic algorithm replaces, with small probability, each edge with some randomly chosen edge connecting the two components resulting from deletion of the original edge. Blob-coded genetic algorithm uses two-point crossover and position-by-position mutation operator. However, Blob-code requires a decoder to convert the code into an equivalent spanning tree. Starting from a random initial solution, stochastic hill climber repeatedly generates a new solution that differs from the current solution on exactly one edge. If this new solution is better than the current solution, then it becomes the current solution, otherwise, it is discarded, and the whole process is repeated again and again. On the test instances considered, stochastic hill climber performed best in terms of solution quality followed by edge-set-coded genetic algorithm, though Blob-coded genetic algorithm is fastest among the three followed by stochastic hill climber. Earlier versions of edge-set-coded genetic algorithm and stochastic hill climber are appeared in [112] along with an exhaustive hill climber.

Singh [113] proposed a perturbation based local search PB-LS that also encodes a spanning tree by the set of its edges. Starting from an initial solution to MRCST, PB-LS repeatedly generates a new solution by applying a local search. This local search randomly deletes an edge and tries all edges in G connecting the two resulting components. The edge that yields the routing cost spanning tree of least cost is included into the tree. If the resulting solution is better than the current solution, it becomes the new current solution. After this, the local search is again applied to the current solution. The newly included edge is prohibited from deletion in the next iteration of the local search because this edge represents the best possible edge connecting the two components at that iteration. If this procedure fails to improve the best solution for a specific number of iterations, then the current solution is perturbed by randomly removing and inserting some edges. The whole process is repeated for a fixed number of iterations.

In this chapter, we have proposed an ABC algorithm based approach for MRCST. The best solution obtained by ABC algorithm is improved further through a local search. We have compared our approach against PB-LS [113] and the three methods proposed by Julstrom [102] which are the best methods known so far for the problem. Our approach have obtained better quality solutions in comparison to these approaches.

The remainder of this chapter is organized as follows: Section 2.2 describes our approach for MRCST, whereas Section 2.3 presents the performance of our ABC approach on a set of 35 benchmark instances and compares it with other approaches. Finally, Section 2.4 provides some concluding remarks.

2.2 ABC Approach for MRCST

Our approach (referred to as ABC+LS) is a combination of an ABC algorithm and a local search heuristic. The best solution obtained through ABC algorithm is improved further by using the local search. The main features of ABC+LS are as follows:

2.2.1 Solution Encoding

We have used edge-set encoding [110] to represent a spanning tree. Edge-set encoding represents a spanning tree by the set of its $n-1$ edges, where n is the number of vertices in the graph. Approaches proposed in [102, 113] for MRCST also used this encoding.

Advantage of using this encoding lies in the fact that it offers high locality, and the problem-specific heuristics can be easily incorporated within the metaheuristic.

2.2.2 Initial Employed Bee Solutions

The algorithm is initialized by assigning a randomly generated solution to every employed bee. Starting from a random start vertex, each initial solution is generated in a manner similar to Prim's algorithm [114] for generating a spanning tree. However, at each stage, instead of selecting a least cost edge among all edges connecting a vertex in the partially constructed tree to a vertex not in the partially constructed tree, it selects an edge using the roulette wheel selection method from all candidate edges where the probability of selecting an edge is inversely proportional to either its weight or square of its weight. With probability p_{sq}, we generate the entire spanning tree by setting the probability of selecting an edge inversely proportional to square of its weight and with probability $(1 - p_{sq})$, we generate the entire spanning tree by setting the probability of selecting an edge inversely proportional to its weight.

2.2.3 Probability of Selecting a Food Source

Instead of using the usual roulette wheel selection method, we have used the binary tournament selection for selecting a food source for an onlooker. In the binary tournament selection method, two food sources are randomly chosen and the better of the two food sources are selected with probability p_{bt} and the worse of the two with probability $(1 - p_{bt})$. We have tried the roulette wheel selection method also, but the binary tournament selection method gives better results. Besides, the binary tournament selection method is computationally more efficient than the roulette wheel selection method.

2.2.4 Determination of a Food Source in the Neighborhood of a Food Source

The method used here is derived from the method used in [85]. It is based on the concept that if an edge is present in one good solution, then it is highly likely that the same edge is present in many good solutions. In order to generate a solution in the neighborhood of a particular solution, say solution i, first we create a copy i' of solution i. An edge e is randomly deleted from i', thereby creating two components. Another employed bee solution is chosen randomly and all edges from this solution connecting the two components, which are different from the deleted edge e, are tried one-by-one for insertion. The edge that results in routing cost spanning tree (RCST) of least cost is selected for insertion. If we are not able to find any edge in the randomly chosen solution different from e connecting the two components, then the deleted edge is reinserted into the solution i' and the method starts afresh. Note that there exist $\mathcal{O}(n^2)$ edges connecting the two components and our method tries at the maximum only $n - 1$ edges. Normally, the number of edges that are tried for insertion is much less than $n - 1$. Therefore, our method, in effect, curtails the search space from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Algorithm 3: Method for Determining a Neighboring Food Source

```
Create a copy $E'$ of food source $E$;
$i \leftarrow 0$;
$nosolution \leftarrow True$;
while ($nosolution = False \ or \ i = t_k$) do
    Randomly delete an edge $e$ of $E'$;
    Randomly select another food source $F$ different from $E'$;
    $C \leftarrow Find\_Candidate\_Edges \ (E', e, F)$;
    if ($C = \emptyset$) then
        $i \leftarrow i + 1$;
        Reinsert the deleted edge $e$ to $E'$;
    else
        $nosolution \leftarrow False$;
        Find the edge $e'$ in $C$ that will result in the RCST of least cost;
        Add $e'$ to $E'$;
if ($nosolution = True$) then
    $E' \leftarrow \emptyset$;
return $E'$;
```

If the method fails for consecutive t_k trials while generating a neighboring solution for an employed bee, then instead of again starting afresh, we simply make the corresponding employed bee a scout. Actually, repeated failures of the method to find an edge, different from the one deleted, indicates the lack of diversity in the employed bee solutions, and therefore, the employed bee associated with the food source i' is made a scout to increase the diversity. However, we keep on trying in case we are generating a neighboring solution for an onlooker, because there is no point in generating a solution for the onlooker randomly as for the survival this randomly generated solution has to compete with the original solution as well as with the solutions of all those onlookers which are associated with the same original solution. Hence, it is highly likely that such a randomly generated solution dies immediately. This is analogous to the concept of "collision" introduced in [85]. The pseudo-code for determining a neighboring food source is given in Algorithm 3, where $Find\_Candidate\_Edges(X, Y, Z)$ is a function that returns all the edges of the solution Z which can connect the two components of the solution X resulting from the deletion of the edge Y and are different from the edge Y.

2.2.5 Cost Evaluation

As described in Section 2.1, the routing cost of a spanning tree can be computed by first determining for each edge e of the spanning tree the count c_e of the paths containing e, and then summing the product of this count and edge weight for every edge. The computation of these c_e values has to be efficient because every time a new solution is created, we have to recompute c_e values from scratch. Even a change of one edge can drastically change c_e values. This is true in case of neighboring solutions which differ on only one edge from an already existing solution. It was suggested in [102] that these c_e values, and hence, the routing cost can be determined by doing a traversal of the spanning tree beginning at any node and keeping track of the number of nodes in each subtree. If the number of nodes in a subtree is known, then the number of nodes in the rest of the spanning tree is also known, and the product of these two values gives the value of c_e for the edge joining that particular subtree to the rest of the spanning tree. Therefore, the routing cost of a spanning tree can be computed in $\mathcal{O}(n)$ time provided adjacency list representation is used [115], as there are only $n-1$ edges in a spanning tree.

2.2.6 Other Features

Unlike the usual practice, we have used different number of employed bees and onlooker bees. If the solution associated with an employed bee does not improve for *limit* number of iterations, then it becomes a scout. Like [85], here also there is a second possibility in which an employed bee can become a scout as described previously while discussing the method for generating a neighboring solution. This second possibility is introduced as an auto-correction measure, as it forces ABC algorithm to move towards exploration whenever there is a lack of diversity in the population. Also, there is no upper limit on the number of scouts in a single iteration. The number of scouts in a particular iteration depends on the number of times the aforementioned two conditions hold. The solution for a scout is generated in the same manner as the initial solutions.

2.2.7 Local Search

The best routing cost spanning tree obtained through ABC algorithm is further improved by applying a local search. It is an iterative approach. At each iteration, local

Table 2.1: Results of ESCGA, BCGA, SHC, PB-LS, ABC, and ABC+LS on 21 Euclidean Instances

Instance	**ESCGA**			**BCGA**			**SHC**			**PB-LS**			**ABC**			**ABC+LS**		
	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD
e50.1	984.8	998.1	17.1	987.6	1008.1	15.6	985.1	1003.5	23.9	983.5	983.6	0.2	983.5	983.6	0.1	983.5	983.6	0.1
e50.2	901.4	907.8	6.7	902.3	912.2	12.5	902.0	912.6	13.3	901.3	901.5	0.1	901.3	901.3	0.0	901.3	901.3	0.0
e50.3	888.3	913.1	21.8	889.9	935.5	31.7	888.3	908.7	18.4	888.3	888.3	0.1	888.3	888.9	0.4	888.3	888.7	0.4
e50.4	778.2	793.8	18.8	777.1	797.3	17.4	776.9	792.5	18.5	776.9	777.0	0.4	776.9	776.9	0.3	776.9	776.9	0.0
e50.5	847.9	858.3	15.1	848.1	865.5	20.4	847.9	860.8	30.2	847.9	847.9	0.0	847.9	848.0	0.0	847.9	848.0	0.0
e50.6	818.4	825.5	6.1	819.3	843.3	34.4	818.1	829.0	22.7	818.1	818.1	0.0	818.1	818.2	0.0	818.1	818.2	0.0
e50.7	865.6	881.5	14.8	865.9	889.1	16.6	865.9	886.3	16.7	865.6	865.7	0.2	865.6	866.2	0.5	865.6	866.1	0.5
e100.1	3538.4	3585.5	56.9	3525.5	3592.8	65.0	3513.2	3553.5	47.1	3507.0	3510.4	2.7	3507.0	3508.6	1.4	3507.0	3507.9	1.1
e100.2	3315.2	3400.2	48.5	3342.7	3407.3	46.3	3310.6	3359.7	52.8	3308.0	3310.0	2.0	3307.9	3309.0	1.2	3307.9	3308.7	1.1
e100.3	3576.0	3641.8	56.7	3573.9	3665.6	76.1	3566.9	3610.7	38.4	3566.3	3567.7	0.9	3566.3	3566.5	0.8	3566.3	3566.5	0.8
e100.4	3464.5	3541.3	57.0	3468.1	3551.2	57.6	3458.4	3504.0	39.9	3448.2	3451.3	4.4	3448.1	3451.4	2.5	3448.1	3451.0	2.2
e100.5	3652.8	3764.3	83.7	3641.9	3737.8	63.0	3639.9	3707.6	62.9	3637.7	3643.3	6.4	3637.0	3640.4	2.3	3637.0	3639.4	1.9
e100.6	3455.0	3487.1	19.9	3443.3	3501.3	37.5	3436.8	3461.3	18.8	3437.6	3442.0	2.3	3436.5	3438.4	2.7	3436.5	3438.0	2.7
e100.7	3730.1	3783.5	61.2	3733.8	3819.5	76.2	3711.6	3741.8	29.0	3703.7	3706.4	2.6	3703.5	3704.9	2.4	3703.5	3704.6	2.0
e250.1	22545.7	23225.3	556.1	22543.0	23013.6	306.9	22177.2	22568.3	359.6	22137.4	22199.2	77.1	22110.1	22163.0	28.9	22089.6	22144.8	33.0
e250.2	23286.6	24310.7	590.7	23149.1	23834.0	563.1	22961.9	23433.1	464.9	22797.9	22970.8	117.4	22775.2	22864.7	56.8	22775.2	22838.6	54.6
e250.3	22394.7	23094.2	519.7	22237.0	22821.4	420.6	22055.7	22473.7	310.7	21888.8	22069.4	161.2	21888.1	21945.0	24.9	21886.1	21927.7	16.5
e250.4	23725.8	24824.2	743.9	23837.3	24647.0	607.1	23598.3	23903.2	293.8	23456.6	23581.4	101.3	23454.4	23486.7	17.0	23428.5	23467.6	19.9
e250.5	22604.0	23352.8	455.0	22739.4	23264.8	306.3	22458.1	22880.5	256.3	22420.1	22492.9	50.7	22396.7	22446.8	33.4	22386.9	22423.8	30.0
e250.6	22662.0	23326.6	448.8	22507.4	23102.6	407.4	22334.3	22559.3	171.1	22312.6	22397.2	86.6	22285.5	22323.3	25.0	22285.3	22314.2	22.3
e250.7	23390.0	23853.7	538.7	23228.2	23781.2	328.5	23039.9	23226.9	190.5	22936.5	23003.3	34.6	22931.4	22986.6	31.5	22923.9	22966.2	30.4

search deletes each edge of the spanning tree one-by-one and examines all graph edges connecting the two resulting components for a possible inclusion. The edge that results in a spanning tree of least routing cost will be selected for inclusion. The local search is applied repeatedly until a complete iteration fails to improve the solution. We have also tried this local search inside ABC algorithm, but it had made the resulting approach too slow to be of any practical use.

2.3 Computational Results

ABC+LS has been implemented in C and has been executed on a Pentium 4 system with 512 MB RAM running at 3.0 GHz under Red Hat Linux 9.0. We have used a colony of 200 bees. 50 of these bees are employed, whereas remaining bees are onlookers, i.e., $n_e = 50$ and $n_o = 150$. In all our experiments with ABC+LS, we have used $p_{bt} = 0.95$, $p_{sq} = 0.25$, $limit = 5n$ and $t_k = 5$. On a problem instance of size n, ABC component of ABC+LS terminates when the best solution does not improve over $20n$ iterations. All these parameter values are chosen empirically after a large number of trials, though they are in no way optimal for all instances. To test the effectiveness of ABC algorithm alone, we have also implemented a version of our approach where local search is not used. We call this approach ABC. Except for the use of local search, ABC is same as ABC+LS.

To test ABC+LS and ABC, we have used the same 35 test instances as used in [102, 113]. Out of these 35 instances, 21 are Euclidean, whereas the rest are randomly generated. The Euclidean instances were originally designed for the Euclidean Steiner tree problem. These instances consist of points in the unit square. These points can be considered as the vertices of a complete graph, whose edge-weights are the Euclidean distances between them. These instances are available from Beasley's OR-library(`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`). There are 15 Euclidean instances for each of 50, 100 and 250 vertices and first 7 instances of each size were used in [102, 113]. The random instances were generated by Julstrom [102]. There are 7 random instances for each of 100 and 300 vertices. The edge-weights of these random instances are uniformly distributed in $[0.01, 0.99]$. Each Euclidean instance has the name of the form $en.i$, where n is the number of vertices in the instance and i is its number. Similarly, each random instance has the name of the form $rn.i$.

Table 2.2: Results of ESCGA, BCGA, SHC, PB-LS, ABC, and ABC+LS on 14 Random Instances

Instance	**ESCGA**			**BCGA**			**SHC**			**PB-LS**			**ABC**			**ABC+LS**		
	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD	Best	Mean	SD
r100.1	598.5	628.0	33.8	625.8	724.4	86.9	597.9	609.8	18.4	597.9	597.9	0.0	597.9	597.9	0.0	597.9	597.9	0.0
r100.2	586.0	640.7	37.1	623.0	730.5	53.2	586.0	601.6	26.6	586.0	586.0	0.0	586.0	586.0	0.0	586.0	586.0	0.0
r100.3	607.7	668.9	49.5	678.2	756.5	48.1	607.0	637.3	55.8	607.0	607.0	0.0	607.0	607.0	0.0	607.0	607.0	0.0
r100.4	607.3	636.8	20.2	628.3	715.9	66.3	598.4	610.9	16.9	598.4	602.1	3.0	598.4	598.4	0.0	598.4	598.4	0.0
r100.5	628.4	668.4	34.9	641.7	757.1	74.8	624.4	643.1	19.1	624.4	624.4	0.0	624.4	624.4	0.0	624.4	624.4	0.0
r100.6	615.6	655.1	30.2	639.1	732.8	54.9	615.5	615.5	0.0	615.5	615.5	0.0	615.5	615.5	0.0	615.5	615.5	0.0
r100.7	514.9	539.0	24.7	531.3	646.8	66.4	514.7	514.8	0.1	514.7	514.7	0.0	514.7	514.7	0.0	514.7	514.7	0.0
r300.1	4926.3	5444.7	316.9	5361.7	6048.2	618.6	4131.1	4259.8	119.4	4131.1	4196.1	91.3	4131.1	4131.4	0.8	4131.1	4131.1	0.0
r300.2	4957.5	5488.0	314.2	5236.3	6017.8	413.5	4040.7	4237.6	180.7	4040.7	4131.2	138.2	4040.7	4040.9	0.5	4040.7	4040.7	0.0
r300.3	5013.1	5452.4	300.8	5093.5	5917.3	396.3	4134.8	4259.5	87.3	4134.8	4220.6	82.5	4134.8	4136.5	2.1	4134.8	4134.8	0.0
r300.4	5012.4	5773.5	378.4	5320.0	6135.4	472.9	4229.3	4397.1	163.8	4229.3	4272.6	31.6	4229.3	4229.5	0.5	4229.3	4229.3	0.0
r300.5	4622.4	5433.1	362.7	5118.2	6010.7	423.7	3951.9	4132.3	177.4	3951.9	4041.6	69.8	3951.9	3951.9	0.0	3951.9	3951.9	0.0
r300.6	5259.9	5641.2	222.3	5408.8	6083.7	360.4	4314.4	4517.6	80.2	4314.4	4458.8	52.9	4314.4	4319.3	7.5	4314.4	4314.5	0.0
r300.7	4868.3	5468.5	342.1	5342.6	5990.2	442.9	4093.9	4268.5	144.9	4093.9	4299.4	159.7	4093.9	4094.1	0.7	4093.9	4093.9	0.0

ABC+LS and ABC were executed 30 times on each instance.

We have compared ABC+LS and ABC with edge-set-coded genetic algorithm [102], Blob-coded genetic algorithm [102], stochastic hill climber [102] and PB-LS [113]. Hereafter, edge-set-coded genetic algorithm, Blob-coded genetic algorithm, stochastic hill climber will be respectively referred as ESCGA, BCGA and SHC. Table 2.1 reports the results of ABC+LS and ABC along with those of ESCGA, BCGA, SHC and PB-LS on 21 Euclidean instances, whereas Table 2.2 does the same for 14 random instances. Data for ESCGA, BCGA and SHC are taken from [102], whereas data for PB-LS are taken from [113]. These tables report for each instance the best and average solution obtained by each of the 6 methods as well as standard deviation of their solution values. On Euclidean instances, except for 4 instances of size 50 where average solution quality of PB-LS is better, ABC+LS always obtains solutions of same or better average quality than other methods. The best solution found by ABC+LS is always as good as or better than the other methods. On larger instances of size 250, it performs much better than other methods. As far as the performance on 14 random instances is concerned, ABC+LS, ABC, SHC and PB-LS all obtained the same best solution on every instance, but the average solution quality of ABC+LS is as good as or better than the other methods. On instances of size 300, though PB-LS and SHC also obtained the same best solution, they performed much worse than ABC+LS in terms of average solution quality. Standard deviation of solution values for ABC+LS is also less for most of the instances.

As far as the performance of ABC is concerned, barring ABC+LS, it outperforms the remaining 4 methods in terms of both best as well as average solution quality except for 5 Euclidean instances where average solution quality of PB-LS is better than ABC. This shows that ABC is an effective heuristic in itself. ABC+LS, anyway, will perform as good as or better than ABC, because the best solution obtained by ABC algorithm is serving as the input to the local search.

To test the statistical significance of the results of ABC+LS vis-à-vis other approaches, we have performed t-tests on ABC+LS in conjunction with each of the other approaches except ABC. Actually, ABC+LS is nothing but ABC with a local search that improves the best solution obtained through ABC. Therefore, improvements, if any, in results of ABC+LS over ABC can never be due to random fluctuations and hence t-test does not have any significance here. Table 2.3 shows the t-test results of

Table 2.3: *t*-test Results of ESCGA, BCGA, SHC, PB-LS in Conjunction with ABC+LS

Instance	ESCGA		BCGA		SHC		PB-LS	
	t value	*p* value	*t* value	*p* value	*t* value	*p* value	*t* value	*p* value
e50.1	4.644352	0.000020	8.601876	0.000000	4.560495	0.000027	0.000000	1.000000
e50.2	5.313726	0.000002	4.776141	70.000013	4.653583	0.000019	11.254629	0.000000
e50.3	6.129441	0.000000	8.085607	0.000000	5.952100	0.000000	-3.104930	0.003588
e50.4	4.923676	0.000007	6.421575	0.000000	4.618633	0.000022	1.406829	0.167613
e50.5	3.736121	0.000429	4.698600	0.000017	2.321473	0.023800	—*	—
e50.6	6.554713	0.000000	3.996464	0.000184	2.605905	0.011625	—*	—
e50.7	5.696026	0.000000	7.585487	0.000000	6.622180	0.000000	-2.447880	0.019104
e100.1	7.468424	0.000000	7.153075	0.000000	5.301346	0.000002	4.205791	0.000153
e100.2	10.330666	0.000000	11.660952	0.000000	5.289354	0.000002	2.602924	0.013111
e100.3	7.273263	0.000000	7.132235	0.000000	6.303147	0.000000	3.984496	0.000296
e100.4	8.670623	0.000000	9.521148	0.000000	7.264478	0.000000	0.285538	0.776812
e100.5	8.171198	0.000000	8.551017	0.000000	5.936033	0.000000	3.026252	0.004428
e100.6	13.391462	0.000000	9.221685	0.000000	6.719321	0.000000	4.195810	0.000157
e100.7	7.057558	0.000000	8.256123	0.000000	7.009309	0.000000	2.285097	0.027984
e250.1	10.623538	0.000000	15.416553	0.000000	6.423523	0.000000	3.148515	0.003190
e250.2	13.592007	0.000000	9.636974	0.000000	6.956299	0.000000	4.864401	0.000020
e250.3	12.287793	0.000000	11.629183	0.000000	9.611706	0.000000	4.865150	0.000020
e250.4	9.984873	0.000000	10.634776	0.000000	8.102196	0.000000	5.961882	0.000001
e250.5	11.158941	0.000000	14.967060	0.000000	9.693668	0.000000	5.257341	0.000006
e250.6	12.340264	0.000000	10.583677	0.000000	7.780299	0.000000	4.895672	0.000018
e250.7	9.009311	0.000000	13.531039	0.000000	7.401949	0.000000	3.231070	0.002548
r100.1	4.877648	0.000009	7.973176	0.000000	3.542336	0.000791	—*	—
r100.2	8.075586	0.000000	14.877051	0.000000	3.212207	0.002151	—*	—
r100.3	6.849298	0.000000	17.023809	0.000000	2.974192	0.004276	—*	—
r100.4	10.412152	0.000000	9.706999	0.000000	4.051202	0.000153	6.940354	0.000000
r100.5	6.905385	0.000000	9.716950	0.000000	5.362519	0.000001	—*	—
r100.6	7.182057	0.000000	11.702706	0.000000	—*	—	—*	—
r100.7	5.388526	0.000001	10.896709	0.000000	5.477226	0.000001	—*	—
r300.1	22.703956	0.000000	16.974441	0.000000	5.903844	0.000000	4.006303	0.000277
r300.2	25.229754	0.000000	26.188688	0.000000	5.968266	0.000000	3.685036	0.000710
r300.3	23.991996	0.000000	24.635767	0.000000	7.823712	0.000000	5.852407	0.000001
r300.4	22.351828	0.000000	22.076844	0.000000	5.610980	0.000001	7.710845	0.000000
r300.5	22.367980	0.000000	26.614378	0.000000	5.569851	0.000001	7.231663	0.000000
r300.6	32.688417	0.000000	26.887646	0.000000	13.870630	0.000000	15.350122	0.000000
r300.7	22.008168	0.000000	23.451034	0.000000	6.599887	0.000000	7.241159	0.000000

*t-test cannot analyze perfect data

Table 2.4: Average Execution Times for ESCGA, BCGA, SHC, PB-LS, ABC, and ABC+LS

Instance	**Execution time (Seconds)**					
	ESCGA[1]	BCGA[1]	SHC[1]	PB-LS	ABC	ABC+LS
e50.1	–	–	–	7.5	4.7	4.7
e50.2	–	–	–	7.9	3.6	3.6
e50.3	–	–	–	7.6	4.4	4.4
e50.4	–	–	–	8.2	3.2	3.2
e50.5	–	–	–	8.7	3.2	3.2
e50.6	–	–	–	8.2	4.1	4.1
e50.7	–	–	–	8.0	4.1	4.1
Typical time	21.2	9.2	9.6	–	–	–
e100.1	–	–	–	54.7	29.5	29.7
e100.2	–	–	–	53.9	28.7	28.9
e100.3	–	–	–	60.6	25.0	25.1
e100.4	–	–	–	53.5	24.8	25.1
e100.5	–	–	–	50.5	29.3	29.5
e100.6	–	–	–	56.5	28.6	28.9
e100.7	–	–	–	55.4	28.3	28.5
Typical time	91.3	36.6	40.0	–	–	–
e250.1	–	–	–	590.5	253.3	266.9
e250.2	–	–	–	573.3	265.6	277.2
e250.3	–	–	–	590.7	291.1	303.0
e250.4	–	–	–	573.0	296.5	309.5
e250.5	–	–	–	563.3	283.9	296.5
e250.6	–	–	–	605.0	363.8	377.0
e250.7	–	–	–	585.4	307.7	321.0
Typical time	618.2	231.1	258.4	–	–	–
r100.1	–	–	–	52.9	16.1	16.3
r100.2	–	–	–	54.5	16.2	16.3
r100.3	–	–	–	52.6	11.0	11.1
r100.4	–	–	–	51.6	16.4	16.5
r100.5	–	–	–	54.9	16.3	16.5
r100.6	–	–	–	54.4	15.9	16.0
r100.7	–	–	–	53.1	14.6	14.8
Typical time	92.6	37.4	40.1	–	–	–
r300.1	–	–	–	709.1	465.1	472.4
r300.2	–	–	–	674.3	302.0	307.9
r300.3	–	–	–	697.1	460.3	467.8
r300.4	–	–	–	668.2	359.0	364.7
r300.5	–	–	–	681.1	267.8	272.6
r300.6	–	–	–	681.3	587.9	600.0
r300.7	–	–	–	689.4	377.9	383.5
Typical time	905.5	351.7	377.8	–	–	–

[1]Execution time on Pentium 4, 2.53 GHz

ESCGA, BCGA, SHC, PB-LS in conjunction with ABC+LS. For each instance and for each method in conjunction with ABC+LS, this table reports the t value and the (two-tailed) p value. Data for ESCGA, BCGA and SHC are taken from [102], whereas data for PB-LS are taken from [113]. Even if we use the 1% significance criterion (p value $\leq$ 0.01), except for two instances in case of SHC and six instances in case of PB-LS, the results of ABC+LS are statistically significant. In most cases, p values are quite small showing the statistical significance of the results obtained through ABC+LS.

Table 2.4 shows the execution time in seconds for all the 6 methods. Instead of reporting the execution time on each instance, Julstrom [102] reported only the typical execution times on instances of each size and type for ESCGA, BCGA and SHC. Moreover, ESCGA, BCGA and SHC were executed on a Pentium 4 processor with 256MB RAM, running at 2.53 GHz under Red Hat Linux 9.0, which is different from the one used to execute ABC+LS and ABC. Because of these two reasons, it is not possible to exactly compare the execution time of ABC+LS and ABC with those of ESCGA, BCGA and SHC. However, it can be safely concluded that ABC+LS and ABC are faster than BCGA, ESCGA and SHC on instances of size 50 and 100, whereas on instances of size 250 and 300, ABC and ABC+LS are slower than BCGA and SHC, but faster than ESCGA. PB-LS [113] was executed on the same system as ABC+LS and ABC. ABC+LS and ABC are faster than PB-LS on all instances.

2.4 Conclusions

In this chapter, we have proposed a new approach ABC+LS combining an artificial bee colony algorithm with a local search heuristic for MRCST and compared it with best heuristic approaches reported in the literature. On the benchmark instances considered, our approach have outperformed other approaches in terms of solution quality. Except for two previously proposed approaches, our approach is faster than other approaches. We have also shown that artificial bee colony algorithm without local search is able to outperform the existing heuristics, thereby establishing artificial bee colony algorithm to be an effective approach for MRCST.

Chapter 3

Quadratic Minimum Spanning Tree Problem

3.1 Introduction

Quadratic minimum spanning tree problem (Q-MST) is an extension of the well-known minimum spanning tree problem (MST) in graphs. In Q-MST, costs are associated not only with edges of the graph, but also with ordered pairs of distinct edges and the objective is to find a spanning tree that minimizes the sumtotal of the costs associated with individual edges of the spanning tree and the costs resulting from ordered pairs consisting of those edges present in the spanning tree. Formally, let $G = (V,\, E)$ be a connected undirected graph where V denotes the set of nodes and E denotes the set of edges. Given a non-negative cost function $w : E \rightarrow \mathbb{R}^+$ associated with edges of G and a non-negative cost function $c : (E \times E - \{(e, e), \forall e \in E\}) \rightarrow \mathbb{R}^+$ associated with ordered pairs of distinct edges, the Q-MST seeks a spanning tree $T \subseteq E$ that minimizes

$$\sum_{e_1 \in T} \sum_{\substack{e_2 \in T \\ e_2 \neq e_1}} c(e_1, e_2) + \sum_{e \in T} w(e) \tag{3.1}$$

Q-MST has several practical applications. It occurs when transferring oil from one pipe to another in a situation, where the cost depends on the type of interface between two pipes. This quadratic cost structure also arises in the connection of overground and underground cables or in a transportation or road network with turn penalties. The presence of quadratic costs in all these cases leads to the minimum spanning tree problem with quadratic cost instead of the usual linear cost [116]. Q-MST was introduced and proved $\mathcal{NP}$-Hard by Asad and Xu [117, 118].

Assad and Xu [117, 118] proposed a branch-and-bound based exact method and two heuristics for Q-MST and applied them on graph instances with number of nodes between 6 and 15. Even on these small instances, the solutions obtained by the two heuristics were far from optimal. Zhou and Gen [116] proposed a genetic algorithm based on Prüfer encoding [119] for Q-MST and observed that this genetic algorithm is superior to two heuristic algorithms of [117, 118] on random instances with number of nodes between 6 and 50. Soak *et al.* [120] proposed edge-window-decoder encoding and showed that a genetic algorithm based on this encoding performs much better than genetic algorithms based on other encodings on Q-MST and other problems. All algorithms were tested by Soak *et al.* [120] on Euclidean instances with number of nodes between 50 and 100.

In this chapter, we have proposed an ABC algorithm based approach to solve Q-MST. Our approach is an extension of the approach described in the previous chapter. Like the previous chapter, here also the best solution obtained through ABC algorithm is improved further by a local search. We have compared our ABC approach with the best approaches. Computational results demonstrate the effectiveness of our approach.

The rest of this chapter is organized as follows: Section 3.2 describes our ABC approach for Q-MST. Computational results are reported in Section 3.3, whereas Section 3.4 contains some concluding remarks.

3.2 ABC Algorithm for Q-MST

The main features of our ABC algorithm for Q-MST are described below:

3.2.1 Solution Encoding

Like the approach described in the previous chapter, edge-set encoding [110] has also been used here to represent a spanning tree.

3.2.2 Initial Employed Bee Solutions

The algorithm is initialized by associating a randomly generated solution to each employed bee. For the purpose of initialization phase, we have defined a special cost, called potential cost, which is associated with each edge. The potential cost of an edge e is the sum of all costs that can be attributed to e, i.e., $w(e) + \sum_{\substack{e_1 \in E \\ e_1 \neq e}} (c(e, e_1) + c(e_1, e))$. Each initial random solution is generated by an iterative process that is similar to Prim's algorithm [114]. However, instead of picking a least cost edge connecting a node in the partially constructed tree to a node not in the tree, an edge is picked at random from all the candidate edges using the roulette wheel selection method where the probability of selecting an edge is inversely proportional to the potential cost of that edge.

3.2.3 Probability of Selecting a Food Source

Like the approach described in the previous chapter, we have also used the binary tournament selection method for selecting a food source for an onlooker bee where the candidate with better fitness is selected with probability p_{bt}. The roulette wheel selection method was also tried, but computational experiments showed that the performance of the binary tournament selection method is better than the roulette wheel selection method in terms of solution quality.

3.2.4 Determination of a Food Source in the Neighborhood of a Food Source

The method used here for determining a neighboring food source is an extension of the method used in previous chapter. In order to generate a solution in the neighborhood of a particular solution say solution i, first we create a copy i' of solution i. An edge e is randomly deleted from i'. This delete operation results into partitioning of the spanning tree into two components and makes solution i' infeasible. To make i' feasible again, we randomly select another solution j different from the solution i'. An edge e', different from edge e is searched in j, which can connect the two components of i' and has the minimum cost among all such candidate edges in solution j. Here the cost of the edge e' means the sum of the cost of the edge e' itself and the intercost with remaining edges of i'. If there is no edge e' in solution j different from the edge e, which can connect the two components of solution i', then the solution i' is restored by adding the deleted edge e into it. The edge e is placed in a tabu list so that it cannot be selected again in subsequent trials and another edge is deleted randomly and the whole procedure is repeated. If the above procedure fails even after t_t trials, then it

Algorithm 4: Method for Determining a Neighboring Food Source

```
input  : A food source $F_i$
output: A neighboring food source $F_i'$ or $\emptyset$ if the procedure fails to generate a
        neighboring food source even after $t_t$ trials
Create a copy $F_i'$ of food source $F_i$;
trial ← 0;
Clear the tabu list;
while ($trial < t_t$) do
    Delete an edge $e$ not belonging to tabu list from the food source $F_i'$;
    Add $e$ to tabu list;
    Randomly select another food source $F_j$ different from $F_i'$;
    Find the least cost candidate edge $e'$ in $F_j$ different from $e$;
    // see Section 3.2.4
    if (no candidate edge exists) then
        Restore $F_i'$ by adding back the deleted edge $e$;
        trial ← trial+1;
    else
        Add $e'$ to $F_i'$;
        break;
if ($trial = t_t$) then
    $F_i' \leftarrow \emptyset$;
```

shows a lack of diversity in the population. If such a situation arises while determining a new neighboring solution for an employed bee, then the employed bee associated with this food source i' abandons it to become a scout so that the population can be made more diversified. This scout is immediately made employed by associating it with a new randomly generated food source. However, if such a situation arises while determining a new neighboring solution for an onlooker bee, then this onlooker bee is associated with a dummy solution which is discarded by artificially assigning it an objective function value higher than the associated employed bee solution so that it cannot be selected for next iteration.

Algorithm 4 gives the pseudo-code of the aforementioned procedure for determining a neighboring food source. The algorithm returns the newly determined neighboring

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	84	7	18	16	14	16	7	13	10	2	3	8	11	20	4
2	7	41	7	13	17	12	9	8	10	3	11	3	4	8	16
3	10	3	23	19	10	8	14	17	12	3	10	14	2	20	5
4	18	19	5	16	11	14	7	12	1	17	14	3	11	17	2
5	6	6	5	8	37	6	7	10	14	18	5	16	3	6	15
6	8	15	5	4	11	88	9	17	19	9	5	4	12	15	20
7	13	1	17	9	20	13	27	7	17	20	16	11	15	19	8
8	2	18	3	18	13	13	17	2	1	7	2	6	10	5	20
9	1	10	12	18	18	12	2	16	10	8	8	17	18	14	7
10	6	7	4	20	5	9	12	13	10	4	20	11	9	9	16
11	1	10	17	4	19	6	7	12	2	16	80	9	5	9	2
12	11	14	1	15	5	5	15	8	17	4	12	28	6	20	17
13	13	12	18	9	16	8	15	2	19	16	10	18	36	14	19
14	9	4	12	9	10	17	5	4	4	14	19	7	1	5	19
15	9	9	10	18	18	17	5	4	11	4	11	20	13	6	45

(a)

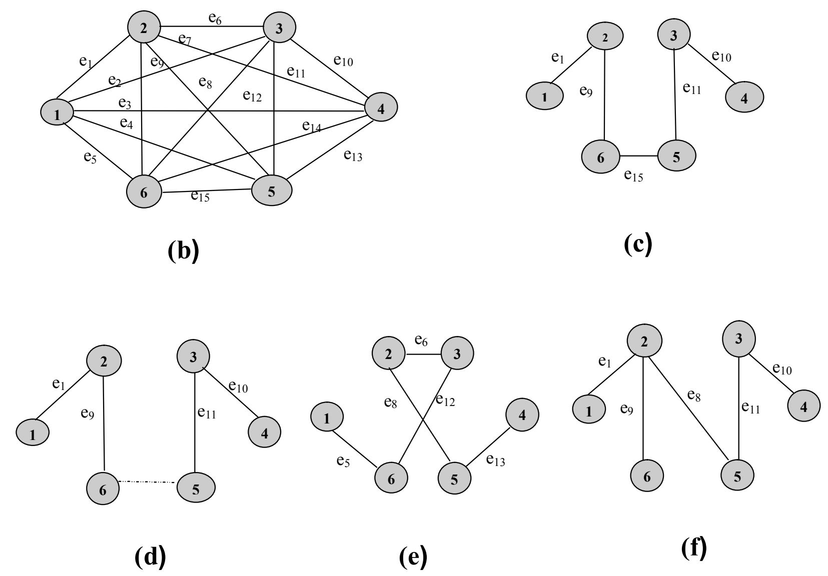


Figure 3.1: An example illustrating the determination of a neighboring food source

food source if it succeeds in finding one within t_t trials, otherwise it returns $\emptyset$.

Figure 3.1 explains the procedure for determining a neighboring food source with the help of an example. Suppose we have a complete graph with 6 nodes. Figures 3.1(a) and 3.1(b) respectively show the intercost matrix and the graph itself. The diagonal elements of this intercost matrix represent the costs of individual edges, whereas off diagonal elements represent the intercosts among edges. Consider the spanning tree shown in Figure 3.1(c). It has a cost of 374. In order to generate a solution in the neighborhood of this solution, we first delete an edge, say e_{15} (as shown in Figure 3.1(d) by dashed line). This creates two components. The one component contains nodes 1, 2, 6, whereas the other contains nodes 3, 4, 5. Now, another solution is chosen randomly, say the solution shown in Figure 3.1(e). This solution contains 3 candidate edges e_6, e_8 and e_{12}, which can be inserted in place of e_{15}. The total costs of e_6, e_8, e_{12} are 172 (88+16+12+9+6+8+19+9+5), 68 (2+13+16+13+12+2+1+7+2), 117 (28+8+17+11+9+11+17+4+12) respectively. Among these three candidate edges, the edge e_8 has the least cost. Therefore, the edge e_8 is selected for insertion in place of e_{15}. This leads to a tree shown in Figure 3.1(f), which has a total cost of 333.

3.2.5 Other Features

Rules governing the scout bees are same as described in the previous chapter.

3.2.6 Local Search

Except for fitness function, the local search and the manner in which it is coupled with ABC algorithm are same as described in the previous chapter.

3.3 Computational Results

Our ABC algorithm for Q-MST has been implemented in C and executed on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. In all our computational experiments with our ABC algorithm, we have used a population of 400 bees, 200 of these bees are employed and the remaining 200 are onlookers, i.e., $n_e = 200$ and $n_o = 200$. We have set $limit = 150$, $t_t = 5$ and $p_{bt} = 0.8$. All these parameter values are set empirically after a large number of trials. These parameter values provide good results though they may not be optimal for all instances. In subsequent subsections, we compare the performance of our ABC approach with genetic algorithms proposed in [116, 120] which are the best heuristics known for Q-MST.

Table 3.1: Results of ABC Algorithm without Local Search and ZG-GA without Local Search on 18 Random Instances

Instance	**ZG-GA**				**ABC**			
	Min	Avg	SD	TET	Min	Avg	SD	TET
25.1	5301	5460.05	81.37	1.14	5085	5085.85	3.71	0.91
25.2	5293	5431.10	73.57	1.00	5081	5101.50	6.81	1.02
25.3	5273	5389.50	63.58	1.16	4962	4962.00	0.00	1.05
50.1	22737	23042.70	197.05	3.89	21126	21160.60	35.05	8.68
50.2	22520	22987.10	188.91	3.53	21123	21187.55	38.91	8.83
50.3	22522	22924.10	187.30	3.92	21059	21095.10	60.29	9.51
100.1	94436	95269.90	460.75	28.17	89098	89430.30	175.02	116.52
100.2	93901	95134.55	486.44	20.51	89202	89550.00	224.87	115.80
100.3	94582	95264.10	382.27	35.58	89007	89272.55	186.83	98.71
150.1	216274	217621.65	632.34	106.62	205638	206470.30	549.31	444.13
150.2	216499	217807.55	758.47	106.70	205874	206311.65	241.59	373.85
150.3	216440	217633.30	569.42	110.11	205634	206168.55	316.51	432.44
200.1	390370	391159.85	833.92	392.66	371797	372558.45	389.85	1139.51
200.2	388929	390574.50	905.05	400.40	371951	372339.90	317.48	1153.35
200.3	388664	390690.50	1103.59	361.57	372156	373029.90	1425.28	1274.43
250.1	612520	614215.20	1118.04	1915.19	587928	588809.60	589.13	2557.50
250.2	611079	614537.65	1402.57	1767.05	588068	588781.70	386.55	2834.89
250.3	611826	614209.00	1219.67	1819.58	587927	588584.10	483.73	2322.16

3.3.1 Comparison of Our ABC Approach with Genetic Algorithm of Zhou and Gen

First, we will compare ABC approach with genetic algorithm of Zhou and Gen [116]. As the test instances used in [116] were not available, therefore, to compare our ABC algorithm with genetic algorithm proposed in [116], we have reimplemented this genetic algorithm. This also facilitated comparison on larger instances, as Zhou and Gen [116] originally tested their genetic algorithm on smaller instances with the number of nodes between 6 and 50. Hereafter, genetic algorithm of Zhou and Gen [116] will be referred to as ZG-GA. All parameter settings for ZG-GA are same as used in [116] except the termination condition.

We have compared our ABC approach with ZG-GA on a set of 18 instances that were generated exactly in the manner as in [116]. All instances represent complete graphs with integer edge-costs uniformly distributed in [1, 100]. The intercosts between

Table 3.2: Results of ABC Algorithm with Local Search and ZG-GA with Local Search on 18 Random Instances

Instance	**ZG-GA**				**ABC**			
	Min	Avg	SD	TET	Min	Avg	SD	TET
25.1	5085	5219.40	76.57	1.14	5085	5085.85	3.71	0.91
25.2	5121	5217.80	59.72	1.00	5081	5101.20	6.64	1.02
25.3	4962	5078.80	83.35	1.16	4962	4962.00	0.00	1.05
50.1	21363	21699.30	165.83	3.90	21126	21157.25	34.40	8.68
50.2	21311	21721.85	136.93	3.54	21123	21179.85	32.47	8.84
50.3	21393	21681.90	177.81	3.92	21059	21091.95	59.67	9.51
100.1	90051	90530.60	273.26	28.93	89098	89404.60	167.87	116.66
100.2	90248	90820.95	332.93	21.17	89202	89520.45	190.05	115.95
100.3	90132	90645.70	260.72	36.36	89007	89242.60	138.59	98.88
150.1	208174	208713.05	436.44	111.50	205619	206404.30	405.97	444.87
150.2	207907	208623.70	414.48	112.32	205874	206300.55	243.36	374.33
150.3	207622	208601.55	421.41	114.54	205634	206160.10	316.07	432.93
200.1	375468	376233.55	650.49	409.82	371797	372527.60	381.44	1141.42
200.2	375119	376205.70	486.18	420.46	371864	372306.60	311.74	1155.60
200.3	375181	376347.15	577.42	381.04	372156	372842.90	735.21	1276.71
250.1	592265	593658.70	929.13	1970.74	587924	588785.10	578.65	2563.41
250.2	592084	593539.45	837.63	1822.70	588068	588731.45	368.08	2840.91
250.3	591841	593443.30	704.45	1870.19	587883	588534.95	463.20	2328.29

edges are also integers and uniformly distributed in [1, 20]. For each value of $n \in \{25, 50, 100, 150, 200, 250\}$, there are 3 instances leading to a total of 18 instances. We have experimented only upto instances with $n = 250$ due to the prohibitive size of intercost matrix at higher values of n.

We have allowed our ABC approach to execute till the best solution fails to improve over $max(10n, 1000)$ iterations. To allow a fair comparison, ZG-GA terminates when the best solution does not improve over an equivalent number of solution generated. ZG-GA uses a population of 300, therefore, it terminates when the best solution does not improve over $max(13.33n, 1333)$ iterations. Each approach is executed 20 times on each instance. The local search described in Section 3.2.6 can be used to improve the best solution of ZG-GA also. Therefore, to compare ABC algorithm with local search, we have incorporated local search with ZG-GA also.

Tables 3.1 and 3.2 compare the performance of ZG-GA with ABC. Table 3.1 reports the performance of two approaches without using the local search, whereas Table 3.2 reports the same when the local search described in Section 3.2.6 is used to improve the best solution obtained by the two approaches. For each instance, these tables report the best and average solution obtained by each of the two methods, standard deviation of solution values and average execution time in seconds. These tables clearly show the superiority of ABC over ZG-GA in terms of solution quality. Best and average solutions obtained by ABC are always better than ZG-GA. Moreover, standard deviations of solution values are also less for ABC. This shows its robustness. Except for small instances, ZG-GA is faster than ABC approach. This is due to the premature convergence of ZG-GA at suboptimal solutions in many cases.

3.3.2 Comparison of Our ABC Approach with the Two Best Genetic Algorithms of Soak *et al.*

We have used the same set of Euclidean instances as used in [120]. This set contains 6 instances each representing a complete graph with the number of nodes between 50 and 100. In all these instances, nodes are distributed uniformly at random on a 500×500 grid. The edge costs are the integer Euclidean distance between these points. The intercost between edges are uniformly distributed between $[1, 20]$. Soak *et al.* [120] presented a number of genetic algorithms and we have taken the two best performing genetic algorithms (EWD+CGPX with dK-TCR and EWD+ANX with dK-TCR) for comparison with our ABC approach. We have also included ZG-GA in this comparison. To allow a fair comparison we have not used the local search of Section 3.2.6 with our ABC approach or with ZG-GA. Genetic algorithms in Soak *et al.* [120] generates a total of 1000000 solutions, therefore, we have also allowed our ABC approach and ZG-GA to generate only 1000000 solutions. Like EWD+CGPX with dK-TCR and EWD+ANX with dK-TCR, we have also executed our ABC approach and ZG-GA 20 times on each instance.

Performance of ZG-GA, EWD+CGPX with dK-TCR, EWD+ANX with dK-TCR along with our ABC approach are reported in Table 3.3. The results reported for EWD+CGPX with dK-TCR, EWD+ANX with dK-TCR are taken from the results presented in Table IX of [120]. As Table IX of [120] presents the results in terms of gap, we have converted the results presented there to our format before reporting. As

Table 3.3: Results of ABC Algorithm, EWD+CGPX with dK-TCR, EWD+ANX with dK-TCR, and ZG-GA on 6 Euclidean Instances

Instance	**ZG-GA**				**EWD+CGPX (dK-TCR)**[1]				**EWD+ANX (dK-TCR)**[1]				**ABC**			
	Min	Avg	SD	TET	Min	Avg	SD	TET	Min	Avg	SD	TET	Min	Avg	SD	TET
50	27042	27734.85	438.57	3.49	25339	25488.50	78.55	302.95	25339	25455.56	70.94	343.00	25200	25201.30	2.37	7.89
60	38150	38640.70	435.31	4.80	36014	36237.29	147.66	448.40	36086	36262.50	126.05	495.70	35544	35617.40	33.57	10.24
70	51660	52854.85	530.68	6.72	48601	48838.94	174.74	699.70	48538	48877.77	155.32	716.60	48125	48189.25	39.51	18.69
80	68064	68853.80	437.71	8.67	63831	64162.39	216.06	1044.45	63546	63933.63	235.12	1086.70	63066	63186.70	50.32	16.73
90	84840	86072.30	743.66	10.75	80033	80407.34	199.07	1323.75	79627	80168.46	222.96	1337.20	78879	79093.95	117.02	26.38
100	103949	105769.35	709.83	13.26	98774	99325.42	304.86	1773.45	98342	98932.05	226.19	1828.90	96783	97100.50	153.14	30.89

[1]Executed on a 1.6 GHz Pentium 4 System

Table 3.4: Results of ABC Algorithm without Local Search and ZG-GA without Local Search on 6 Euclidean Instances using the Termination Criterion of Section 3.3.1

Instance	**ZG-GA**				**ABC**			
	Min	Avg	SD	TET	Min	Avg	SD	TET
50	27229	27821.15	391.84	2.87	25200	25201.30	2.37	4.35
60	37921	38698.10	456.02	4.41	35487	35615.35	42.91	8.44
70	51760	52556.15	515.70	7.04	48125	48173.10	31.28	16.84
80	67732	68751.25	496.28	10.53	63057	63150.85	46.24	20.83
90	84154	85886.20	797.53	16.02	78879	79090.15	129.69	37.49
100	103976	105257.60	645.33	19.37	96750	97038.70	154.05	76.92

Table 3.5: Results of ABC Algorithm with Local Search and ZG-GA with Local Search on 6 Euclidean Instances using the Termination Criterion of Section 3.3.1

Instance	**ZG-GA**				**ABC**			
	Min	Avg	SD	TET	Min	Avg	SD	TET
50	25262	25414.30	99.48	2.87	25200	25201.30	2.37	4.35
60	35447	35860.55	129.93	4.43	35466	35603.10	48.31	8.45
70	48365	48609.95	195.97	7.12	48125	48166.20	27.71	16.86
80	63248	63664.60	215.92	10.72	63022	63130.30	47.55	20.89
90	79361	79828.60	282.68	16.40	78879	79076.65	122.44	37.59
100	97414	98021.35	397.91	20.07	96750	97018.75	139.48	77.12

shown by this table, ABC approach has performed best in terms of solution quality followed by EWD+ANX with dK-TCR and EWD+CGPX with dK-TCR. ZG-GA has performed the worst. Best and average solution quality of our approach are always better than other approaches. Moreover, its standard deviation of solution values is also the smallest among all the four approaches. EWD+CGPX with dK-TCR and EWD+ANX with dK-TCR were executed on Pentium 4, 1.6 GHz, therefore, it is not possible to compare their execution times with our ABC approach and ZG-GA. However, we can safely say that even after compensating for processing speed, ABC approach and ZG-GA are faster than the two genetic algorithms. Though ZG-GA is fastest, it has performed the worst.

We have also executed our ABC approach and ZG-GA on these 6 Euclidean instances with the same termination condition as used in Section 3.3.1. Table 3.4 reports the results obtained without using the local search, whereas Table 3.5 reports the results obtained while using the local search. Best and average solutions obtained by our ABC approach improved slightly in comparison to Table 3.3 at the expense of increased computation times. Except for one instance where the best solution obtained by ZG-GA with local search is better than ABC with local search, solution quality of our ABC approach is always better. However, on the same instance ABC without local search have obtained the better best solution than ZG-GA without local search. Therefore, even on this instance the superiority of ZG-GA with local search over ABC with local search is purely due to the local search.

3.4 Conclusions

In this chapter, we have developed an artificial bee colony (ABC) algorithm for Q-MST. We have compared our approach against genetic algorithms proposed in [116, 120]. Our approach have obtained better quality solutions than other approaches. Though genetic algorithm of [116] is faster than our ABC approach, it has performed worst in terms of solution quality among all the approaches considered in this chapter.

Chapter 4

Set Covering Problem

4.1 Introduction

Set Covering Problem (SCP) is an important combinatorial optimization problem. SCP is defined as follows: Given an $m \times n$ $0-1$ matrix $A = (a_{ij})$ and non-negative n-dimensional vector $C = (c_j)$, where each element c_j of C gives the cost of selecting the column j of matrix A. If a_{ij} is equal to 1, then it indicates that row i is covered by column j, otherwise it is not. The objective of SCP is to find a minimum cost subset of columns of A such that each row of A is covered by at least one column in the subset. Mathematically, SCP can be formulated as follows:

$$minimize \sum_{j=1}^{n} c_j x_j \tag{4.1}$$

$$subject\, to \sum_{j=1}^{n} a_{ij} x_j \geq 1, \; i = 1, 2, ..., m \tag{4.2}$$

$$x_j \in \{0, 1\}, \; j = 1, 2, ..., n \tag{4.3}$$

where x_j is 1 if column j is in the solution, 0 otherwise. Constraint (4.2) ensures that each row i is covered by at least one column. In addition to the notational convention already introduced, the following notational conventions have been used throughout this chapter:

I = set of all rows, i.e., $\{1, 2, ..., m\}$

J = set of all columns, i.e., $\{1, 2, ..., n\}$

α_i = set of columns covering row i, $i \in I$

β_j = set of rows covered by column j, $j \in J$

S = set of columns in a solution

u_i = number of columns in S covering row i

SCP is an $\mathcal{NP}$-Hard problem in the strong sense [121] and it has many practical applications such as airline and bus crew scheduling [122, 123], location of emergency facilities [124], logical analysis of numerical data [125], steel production [126], vehicle routing [127], ship scheduling [128] etc.

Many exact and heuristic algorithms have been proposed in the literature for solving SCP. Among exact algorithms, Fisher and Kedia [129] presented a branch and bound technique based on a dual heuristic which can solve SCP instances with up to 200 rows and 2000 columns, Beasley and JØrnsten [130] used Lagrangian heuristic, feasible solution exclusion constraints, Gomory f-cuts and an improved branching strategy to improve the previous algorithms proposed in [131, 132] and solved SCP instances with up to 400 rows and 4000 columns. Balas and Carrera [133] presented a dynamic subgradient-based branch-and-bound procedure for SCP.

Among the metaheuristic techniques, a number of population-based metaheuristics such as genetic algorithm [134], an indirect genetic algorithm [135], and ant colony optimization algorithms [1, 136] have been developed for solving SCP.

There are also non-population-based methods such as simulated annealing algorithm [137], Lagrangian relaxation based heuristic [138], a combination of Lagrangian relaxation based heuristic and a greedy algorithm [123], a 3-flip neighborhood local search using Lagrangian relaxation approach [139], and a new metaheuristic based method Meta-RaPS [140] have been developed for solving SCP. Haouari and Chaouachi [141] proposed a probabilistic greedy search algorithm for SCP. Caserata [142] developed an efficient algorithm based on tabu search. For a good survey of SCP, the reader is referred to [143, 144, 145].

In this chapter, we have proposed a hybrid artificial bee colony algorithm ABC_SCP to solve the non-unicost SCP. We have compared ABC_SCP with the best population-based approaches and the overall best approaches. Computational results show that it is competitive in terms of solution quality with all these approaches.

The rest of this chapter is organized as follows: Section 4.2 describes our hybrid approach for SCP. Computational results are presented in Section 4.3, whereas Section 4.4 outlines some concluding remarks.

4.2 Hybrid ABC Algorithm for SCP

We have adopted the ideas of ABC algorithm as presented in [81, 85]. We have incorporated a problem-specific heuristic for SCP with suitable randomness and a local search in our ABC algorithm. While designing our algorithm, like [134], we have also assumed that the columns of SCP are arranged in increasing order of their costs and columns of equal cost are arranged in decreasing order of the number of rows that they cover. By assuming so, no generality has been lost, as it is always possible to rearrange columns of a matrix in the above mentioned order. The main features of our hybrid ABC algorithm (ABC_SCP) for SCP are described below.

4.2.1 Initialization

ABC_SCP is initialized by associating each employed bee with a randomly generated solution. Each solution is generated by following the procedure described by the pseudo-code in Algorithm 5. To generate a feasible candidate solution, for each row i, a column is selected randomly from the restricted candidate list r_c that contains the least cost columns of α_i (step 3). The size of the restricted candidate list r_c is chosen such that it should not exclude any column occurring in an optimal or a near-optimal solutions. The idea of a restricted candidate list is first used in [134]. The feasible solution obtained after step 3 may contain redundant columns. Therefore, some redundant columns are removed randomly as described in step 5. It should be noted that step 5 may not remove all redundant columns in the solution. We learned from our initial experiments that removing all redundant columns in the initial solution makes our program not only computationally expensive, but the solution quality also deteriorates for some instances. This is due to the fact that the presence of some redundant columns provides additional options in the search process and reduces the cost of repair operator described in Section 4.2.3.

Algorithm 5: Initial Solution Generation

```
1 Initialize $S \leftarrow \phi$;
2 Initialize $u_i \leftarrow 0, \forall i \in I$;
3 for (each row i in I) do
      Randomly select a column $j$ from the set $r_c$ of row $i$;
      $S \leftarrow S \cup \{j\}$;
      $u_i \leftarrow u_i + 1, \forall i \in \beta_j$;
4 $t \leftarrow |S|$;
5 while ($t > 0$) do
      Randomly select a column $j$ from first $t$ columns of $S$;
      if ($u_i \geq 2, \forall i \in \beta_j$) then
          $S \leftarrow S - \{j\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$;
      $t \leftarrow t - 1$;
6 return $S$;
```

4.2.2 Probability of Selecting a Food Source

In ABC_SCP, we have used the roulette wheel selection method for selecting a food source for onlookers. We have also tried the binary tournament selection method, but the roulette wheel selection method gave better results.

4.2.3 Determination of a Food Source in the Neighborhood of a Food Source

It is based on the fact that if a column is present in one good solution, then the likelihood that this column is present in many other good solutions is high. In order to determine a food source in the neighborhood of a particular food source, say i, first we create a copy i' of food source i. Another food source j (different from i') is selected randomly. Then we randomly select a maximum of *col_add* distinct columns from j, which are not present in i' and add them to the food source i'. If this procedure fails to find even one column in j different from the columns of i', then it testifies that i' and j are the same and a "collision" occurs [85]. If a collision occurs while determining a new neighboring food source for an employed bee, then the employed bee abandons its associated food source to become a scout. This scout is again made employed by associating it with a new randomly generated food source. However, if a collision occurs while determining a new neighboring food source for an onlooker, then another food source j is selected randomly. This process is continued until we find a food source j which is different from the food source i'.

If there is no collision, then we proceed to the next phase which drops *col_drop* columns randomly from the food source. We have also experimented with the greedy approach, i.e., dropping columns having maximum ratio of its cost to the number of rows covered by it from the food source. The empirical observation suggested that for ABC_SCP, randomness always gives better results than the greedy approach and is computationally more efficient. After drop phase, if the resulting solution is found to be infeasible, i.e., some rows are not covered, then a repair operator is used to convert this infeasible solution into a feasible one. The repair operator begins by computing the set of uncovered rows. Then it considers one-by-one row from this set in their natural order. With probability p_a, it selects a column, which covers the row in consideration and has the minimum ratio of its cost to the number of yet uncovered rows covered by it and add it to the solution. Otherwise, a column covering the row in consideration is selected randomly from a restricted candidate list r_c of columns of that uncovered row (as described in Section 4.2.1) and added to the solution. After this the set of uncovered rows are updated to reflect the addition of the new column. This process is repeated till the set of uncovered rows becomes empty. The concept of selecting a column based on minimum ratio is first used in the heuristic feasibility operator of [134]. The repaired feasible solution may contain some redundant columns which are removed by iteratively deleting the redundant column having the maximum ratio of its cost to the number of rows covered by it. A somewhat similar idea of first dropping and then adding columns was used in simulated annealing method of [137]. Once all redundant columns are removed, we try to further improve the quality of solution by using a local search that is described in the next subsection.

4.2.4 Local Search

Our local search is an extension of the greedy local search proposed by Ren *et al.* [1]. The pseudo-code for the local search is given in Algorithm 6. This local search begins by computing for each row i, the number of columns u_i in the current solution covering it. Next it considers each column j in the solution one-by-one in non-increasing order

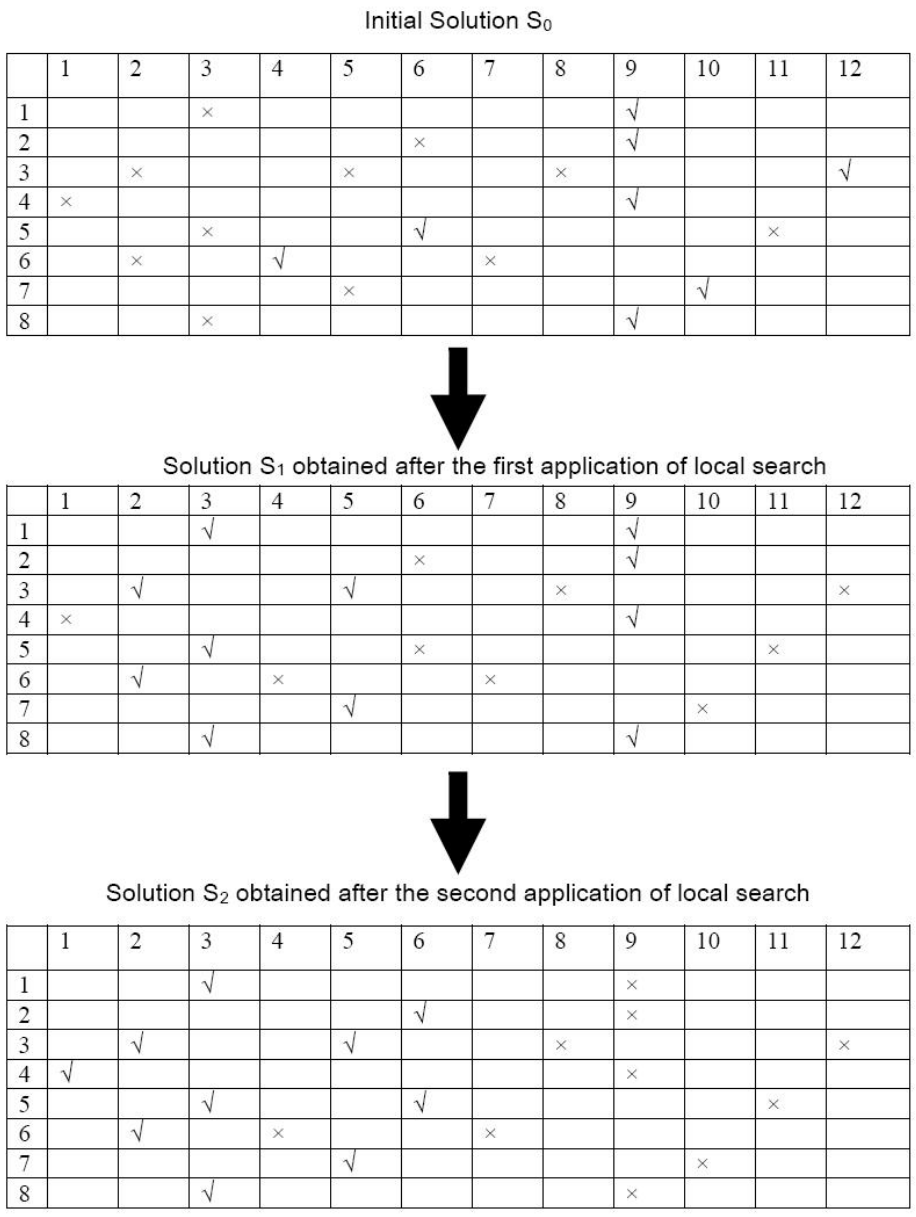
Algorithm 6: Pseudo-code of Local Search Procedure

```
Initialize $u_i \leftarrow |S \cap \alpha_i|, \forall i \in I$;
Improvement $\leftarrow 1$;
while (Improvement) do
    Improvement $\leftarrow 0$;
    for (each column $j$ in $S$ in non-increasing order of cost of columns) do
        $P_j \leftarrow \{i \in \beta_j | u_i = 1\}$;
        if $(|P_j| = 0)$ then
            $S \leftarrow S \setminus \{j\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 1$ and $P_j = \{i1\}$ and $j \neq li1)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 2$ and $P_j = \{i1, i2\}$ and $li1 \neq li2$ and $C_{li1} + C_{li2} < C_j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1, li2\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li2}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 2$ and $P_j = \{i1, i2\}$ and $li1 = li2 \neq j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 3$ and $P_j = \{i1, i2, i3\}$ and $li1 \neq li2 \neq li3$ and $C_{li1} + C_{li2} + C_{li3} < C_j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1, li2, li3\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li2}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li3}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 3$ and $P_j = \{i1, i2, i3\}$ and $li1 = li2 = li3 \neq j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 3$ and $P_j = \{i1, i2, i3\}$ and $li1 = li2$ and $li1 \neq li3$ and $C_{li1} + C_{li3} < C_j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1, li3\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li3}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 3$ and $P_j = \{i1, i2, i3\}$ and $li1 = li3$ and $li1 \neq li2$ and $C_{li1} + C_{li2} < C_j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1, li2\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li2}$; Improvement $\leftarrow 1$;
        else if $(|P_j| = 3$ and $P_j = \{i1, i2, i3\}$ and $li2 = li3$ and $li1 \neq li2$ and $C_{li1} + C_{li2} < C_j)$ then
            $S \leftarrow (S \setminus \{j\}) \cup \{li1, li2\}$; $u_i \leftarrow u_i - 1, \forall i \in \beta_j$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li1}$; $u_i \leftarrow u_i + 1, \forall i \in \beta_{li2}$; Improvement $\leftarrow 1$;
return $S$;
```

of its cost and computes the set of rows P_j solely covered by the column j. If the cardinality of P_j is 0, i.e., $|P_j| = 0$, then the column j is deleted from the solution and the next column is considered (Note that the case $|P_j| = 0$ will never occur in the very first iteration of our local search as there is no redundant columns initially, but it may occur in later iterations after column replacement). If $|P_j| = 1$, then it checks whether column j is the least cost column covering the only row $i1$ of P_j. If yes, then the column j is retained in the solution, otherwise it is replaced by the least cost column $li1$ covering row $i1$. If $|P_j| = 2$, then that means there are two rows, say $i1$ and $i2$, solely covered by the column j. Now we find the least cost column $li1$ covering $i1$ and the least cost column $li2$ covering $i2$. If $li1$ and $li2$ are distinct and the sum of the costs of $li1$ and $li2$ is less than or equal to the cost of column j, then column j is replaced by $li1$ and $li2$ in the solution. If $li1$ and $li2$ represent the same column that is different from j (see line (13) of pseudo-code), then the column j is replaced by the least cost column $li1$ (or $li2$). The u_i values are updated after each local search step before considering the next column.

Ren *et al.* [1] do not consider the case with $|P_j| \geq 3$. On the basis of limited computational experiments, they concluded that the case with $|P_j| \geq 3$ do not occur often and it is computationally too expensive to incorporate in the program. Inside our local search, we have considered the $|P_j| = 3$ case also, as we found that it occurs often and can be handled within reasonable computation time. If $|P_j| = 3$, then that means there exist three rows, say $i1, i2$ and $i3$, solely covered by the column j. Now, we find the least cost column $li1$ covering $i1$, the least cost column $li2$ covering $i2$ and the least cost column $li3$ covering $i3$. If $li1, li2$ and $li3$ are all distinct and the sum of the costs of $li1$, $li2$ and $li3$ is less than the cost of the column j, then the column j is replaced by $li1$, $li2$ and $li3$ in the solution. If $li1$, $li2$ and $li3$ represent the same column that is different from j, then the column j is replaced by the least cost column $li1$ (or $li2$ or $li3$). There are three more cases where any two of three columns are same, but the third column is different from the other two, i.e., we have only two distinct columns. In all these cases, we have to simply sum the costs of the two distinct columns, and if this sum is less than the cost of column j, then the column j is replaced with these two columns.

Unlike Ren *et al.* [1], which use the local search only once, we are using the local search repeatedly till it fails to improve the solution. Actually, the local search considers



Initial Solution S_0

	1	2	3	4	5	6	7	8	9	10	11	12
1			×						√			
2						×			√			
3		×			×			×				√
4	×								√			
5			×			√					×	
6		×		√			×					
7					×					√		
8			×						√			

Solution S_1 obtained after the first application of local search

	1	2	3	4	5	6	7	8	9	10	11	12
1			√						√			
2						×			√			
3		√			√			×				×
4	×								√			
5			√			×					×	
6		√		×			×					
7					√					×		
8			√						√			

Solution S_2 obtained after the second application of local search

	1	2	3	4	5	6	7	8	9	10	11	12
1			√						×			
2						√			×			
3		√			√			×				×
4	√								×			
5			√			√					×	
6		√		×			×					
7					√					×		
8			√						×			

Figure 4.1: An example explaining the possibility of a further improvement that is ignored by Ren *et al.* [1]

columns in non-increasing order of their costs and it may happen that when a column is considered it could not be replaced, but later such a replacement may be possible due to the replacement of other columns in the solution. Therefore, we have taken care of this possibility by applying the local search repeatedly.

Figure 4.1 explains the situation described in the previous paragraph with the help of an example. Consider a simple 8×12 SCP instance, where cost of a column j is j itself. A '$\checkmark$' or '$\times$' at the position (i, j) denotes that the row i is covered by the column j, but '$\checkmark$' means that the column j is in the current solution and '$\times$' means it is not. Suppose, we have an initial feasible solution $S_0 = \{4, 6, 9, 10, 12\}$ with cost 41. After applying the local search once, we get the solution $S_1 = 2, 3, 5, 9$ with cost 19. This solution will be returned as the final solution, if we use the local search only once. However, as Figure 4.1 shows, we can improve this solution further, if we apply the local search once again. Actually, the local search considers columns in the solution in non-increasing order of their costs. Therefore, when column 9 was considered during the first application of the local search, it has $|P_9| = 4$, and therefore, could not be replaced. But if we apply the local search once again, then $|P_9| = 2$, and therefore, column 9 can be replaced with column 1 as the cost of column 1 is less than the cost of column 9. This leads to an improved solution $S_2 = \{1, 2, 3, 5\}$ with total cost 11. Bigger instances will offer many such possibilities of improvements. Therefore, it is always better to apply local search repeatedly as long as it improves the solution.

Another difference with Ren *et al.* [1] is that we have performed all replacements only when the cost of the solution reduces, whereas in Ren *et al.* [1], replacements are performed even when the cost remains the same.

4.2.5 Fitness

We have used two fitness functions to evaluate the quality of a solution. The primary fitness function is same as the objective function, whereas the secondary fitness function is equal to the number of rows in the solution which are covered by a single column only. The secondary fitness function is used to distinguish between the two solutions having the same value of the primary fitness function. A solution is better than the other, if either its cost according to equation 4.1 is less or the two solutions have equal cost, but the secondary fitness function value of the first solution is less than the other. Actually, the secondary fitness function is required because there can be many solutions with the same cost. Clearly, among all these solutions, the solution having the least value of the secondary objective function offers the maximum possibility of further improvements.

4.2.6 Other Features

Rules governing the scout bees are same as described in previous chapters.

4.3 Computational Results

ABC_SCP has been implemented in C and executed on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. In all our computational experiments, we have allowed ABC_SCP to execute for 500 iterations. We have used a population of 200 bees. 50 of these bees are employed and remaining 150 are onlookers, i.e., $n_e = 50$ and $n_o = 150$. We have set $limit = 50$, $|rc| = 6$, $p_a = 0.9$, $col\_add = 5$ if $n > 35$, otherwise $col\_add = 3$, $col\_drop = 12$ if $n > 35$, otherwise $col\_drop = 5$. All these parameter values are chosen empirically after a large number of trials. These parameter values provide good results though they may not be optimal for all instances.

ABC_SCP has been tested on 65 standard non-unicost SCP instances available from OR-Library (`http://people.brunel.ac.uk/{~}mastjjb/jeb/info.html`). All these instances have column costs in the range $[1, 100]$. These instances are divided into 11 sets. Table 4.1 summarizes the characteristics of each of these sets where column labeled Density shows the percentage of non-zero entries in the matrix of each instance belonging to that particular set. The optimal solution values for each instance in the first seven sets are known. ABC_SCP was executed 10 independent times on each instance, each time with a different random seed.

Table 4.2 reports the solution quality of ABC_SCP along with 9 other heuristic methods viz. BeCh [134], IGA [135], BJT [137], RFKZ [1], CFT [123], BJ [130], CNS [138], Meta-RaPS [140] and YKI [139]. Among these 9 methods, BeCh, IGA and RFKZ are population-based metaheuristic approaches. CFT, BJ and Meta-RaPS were executed only once on each instance and that is why, their average solution quality is not reported. All other methods were executed 10 times on each instance. Though IGA was executed 10 times on each instance, Aickelin [135] reported only the best solution found over 10 trials, and therefore, no average solution quality is reported for IGA also. For each instance, this table reports their optimal/best known solution values (BKS),

Table 4.1: Characteristics of 11 Different Sets of Non-Unicost SCP Instances

Set name	**No. of instances**	**Rows (m)**	**Column (n)**	**Density**
4	10	200	1,000	2
5	10	200	2,000	2
6	5	200	1,000	5
A	5	300	3,000	2
B	5	300	3,000	5
C	5	400	4,000	2
D	5	400	4,000	5
NRE	5	500	5,000	10
NRF	5	500	5,000	20
NRG	5	1,000	10,000	2
NRH	5	1,000	10,000	5

the best solution found by each of these 9 methods. Except for IGA, CFT, BJ, and Meta-RaPS, the average solution quality of each method is also reported. Data for all the 9 methods are taken from their respective papers. Average solution quality of BeCh and YKI is computed by us based on the data provided in their respective papers. Some of these papers have not reported the results on all 65 instances as indicated by '—' in the table. Table 4.3 reports for different methods the number of instances solved to their optimal/best known solution values. We have included only those methods which were executed on all 65 instances. In addition to CFT, BeCh, IGA, Meta-RaPS, YKI, RFKZ and ABC_SCP, we have also reported the value for PROGRESS [141] and ANTS+LB [136]. Detailed results for PROGRESS are not provided, as its solution quality is inferior to other methods and Lessing *et al.* [136] did not report the detailed results for ANTS+LB.

ABC_SCP is able to find optimal/best known solution values for all but one instance. For 60 instances, it finds optimal/best known solution values in all 10 trials. Only CFT, Meta-RaPS, ANTS+LB and YKI are slightly better than our method as these methods

Table 4.2: Solution Quality of Different Methods on Each of the 65 Instances

Instance	**OPT/BKS**	**BeCh**		**IGA**	**BJT**		**RFKZ**		**CFT**	**BJ**	**CNS**	**Meta-RaPS**	**YKI**		**ABC_SCP**	
		Best	Avg	Best	Best	Avg	Best	Avg	Best	Best	Best	Best	Best	Avg	Best	Avg
4.1	429	429	429.7	429	—	—	429	429.0	429	429	—	429	429	429.0	429	429.0
4.2	512	512	512.0	512	—	—	512	512.0	512	512	—	512	512	512.0	512	512.0
4.3	516	516	516.0	516	—	—	516	516.0	516	516	—	516	516	516.0	516	516.0
4.4	494	494	494.8	494	—	—	494	494.0	494	494	—	494	494	494.0	494	494.0
4.5	512	512	512.0	512	—	—	512	512.0	512	512	—	512	512	512.0	512	512.0
4.6	560	560	560.0	560	—	—	560	560.0	560	560	—	560	560	560.0	560	560.0
4.7	430	430	430.2	430	—	—	430	430.0	430	430	—	430	430	430.0	430	430.0
4.8	492	492	492.1	492	—	—	492	492.0	492	492	—	492	492	492.0	492	492.0
4.9	641	641	643.1	641	—	—	641	641.0	641	641	—	641	641	641.0	641	641.0
4.10	514	514	514.0	514	—	—	514	514.0	514	514	—	514	514	514.0	514	514.0
5.1	253	253	253.0	253	—	—	253	253.0	253	253	—	253	253	253.0	253	253.0
5.2	302	302	303.5	302	—	—	302	302.0	302	302	—	302	302	302.0	302	302.0
5.3	226	228	228.0	226	—	—	226	226.0	226	226	—	226	226	226.0	226	226.0
5.4	242	242	242.1	242	—	—	242	242.0	242	242	—	242	242	242.0	242	242.0
5.5	211	211	211.0	211	—	—	211	211.0	211	211	—	211	211	211.0	211	211.0
5.6	213	213	213.0	213	—	—	213	213.0	213	213	—	213	213	213.0	213	213.0
5.7	293	293	293.0	293	—	—	293	293.0	293	293	—	293	293	293.0	293	293.0
5.8	288	288	288.8	288	—	—	288	288.0	288	288	—	288	288	288.0	288	288.0
5.9	279	279	279.0	279	—	—	279	279.0	279	279	—	279	279	279.0	279	279.0
5.10	265	265	265.0	265	—	—	265	265.0	265	265	—	265	265	265.0	265	265.0
6.1	138	138	138.0	138	—	—	138	138.0	138	138	—	138	138	138.0	138	138.0
6.2	146	146	146.2	146	—	—	146	146.0	146	146	—	146	146	146.0	146	146.0
6.3	145	145	145.0	145	—	—	145	145.0	145	145	—	145	145	145.0	145	145.0
6.4	131	131	131.0	131	—	—	131	131.0	131	131	—	131	131	131.0	131	131.0
6.5	161	161	161.3	161	—	—	161	161.0	161	161	—	161	161	161.0	161	161.0
A.1	253	253	253.2	253	—	—	253	253.0	253	253	—	253	253	253.0	253	253.0
A.2	252	252	252.0	252	—	—	252	252.0	252	252	—	252	252	252.0	252	252.0
A.3	232	232	232.5	232	—	—	232	232.8	232	232	—	232	232	232.0	232	232.0
A.4	234	234	234.0	234	—	—	234	234.0	234	234	—	234	234	234.0	234	234.0
A.5	236	236	236.0	236	—	—	236	236.0	236	236	—	236	236	236.0	236	236.0
B.1	69	69	69.0	69	—	—	69	69.0	69	69	—	69	69	69.0	69	69.0

Continued on next page

Table 4.2 – continued from previous page

Instance	**OPT/BKS**	**BeCh**		**IGA**	**BJT**		**RFKZ**		**CFT**	**BJ**	**CNS**	**Meta-RaPS**	**YKI**		**ABC_SCP**	
		Best	Avg	Best	Best	Avg	Best	Avg	Best	Best	Best	Best	Best	Avg	Best	Avg
B.2	76	76	76.0	76	—	—	76	76.0	76	76	—	76	76	76.0	76	76.0
B.3	80	80	80.0	80	—	—	80	80.0	80	80	—	80	80	80.0	80	80.0
B.4	79	79	79.0	79	—	—	79	79.0	79	79	—	79	79	79.0	79	79.0
B.5	72	72	72.0	72	—	—	72	72.0	72	72	—	72	72	72.0	72	72.0
C.1	227	227	227.2	227	—	—	227	227.0	227	227	—	227	227	227.0	227	227.0
C.2	219	219	220.0	219	—	—	219	219.0	219	219	—	219	219	219.0	219	219.0
C.3	243	243	246.4	243	—	—	243	243.0	243	243	—	243	243	243.0	243	243.1
C.4	219	219	219.1	219	—	—	219	219.0	219	219	—	219	219	219.0	219	219.0
C.5	215	215	215.1	215	—	—	215	215.0	215	215	—	215	215	215.0	215	215.0
D.1	60	60	60.0	60	—	—	60	60.0	60	60	—	60	60	60.0	60	60.0
D.2	66	66	66.0	66	—	—	66	66.0	66	66	—	66	66	66.0	66	66.0
D.3	72	72	72.2	72	—	—	72	72.0	72	72	—	72	72	72.0	72	72.0
D.4	62	62	62.0	63	—	—	62	62.0	62	62	—	62	62	62.0	62	62.0
D.5	61	61	61.0	61	—	—	61	61.0	61	61	—	61	61	61.0	61	61.0
NRE.1	29	29	29.0	29	29	29.0	29	29.0	29	—	—	29	29	29.0	29	29.0
NRE.2	30[a]	30	30.6	30	30	30.0	30	30.0	30	—	—	30	30	30.0	30	30.0
NRE.3	27	27	27.7	27	27	27.0	27	27.0	27	—	—	27	27	27.0	27	27.0
NRE.4	28	28	28.0	28	28	28.0	28	28.0	28	—	—	28	28	28.0	28	28.0
NRE.5	28	28	28.0	28	28	28.0	28	28.0	28	—	—	28	28	28.0	28	28.0
NRF.1	14	14	14.0	14	14	14.0	14	14.0	14	—	—	14	14	14.0	14	14.0
NRF.2	15	15	15.0	15	15	15.0	15	15.0	15	—	—	15	15	15.0	15	15.0
NRF.3	14	14	14.0	14	14	14.0	14	14.0	14	—	—	14	14	14.0	14	14.0
NRF.4	14	14	14.0	14	14	14.0	14	14.0	14	—	—	14	14	14.0	14	14.0
NRF.5	13	13	13.7	13	13	13.3	13	13.5	13	—	—	13	13	13.0	13	13.7
NRG.1	176[a]	176	177.7	176	176	176.6	176	176.0	176	—	176	176	176	176.0	176	176.0
NRG.2	154[a]	155	156.3	155	155	155.3	154	155.1	154	—	155	154	154	154.0	155	155.0
NRG.3	166[a]	166	167.9	166	166	167.6	166	167.3	166	—	167	166	166	166.0	166	166.0
NRG.4	168[a]	168	170.3	168	168	170.7	168	168.9	168	—	170	168	168	168.0	168	168.0
NRG.5	168[a]	168	169.4	168	168	168.4	168	168.1	168	—	169	168	168	168.0	168	168.0
NRH.1	63[a]	64	64.0	63	64	64.0	64	64.0	63	—	64	63	63	63.0	63	63.9
NRH.2	63[a]	64	64.0	66	63	63.7	63	63.9	63	—	64	63	63	63.3	63	63.9
NRH.3	59[a]	59	59.1	59	59	59.4	59	59.4	59	—	60	59	59	59.9	59	59.0

Continued on next page

Table 4.2 – continued from previous page

Instance	**OPT/BKS**	**BeCh**		**IGA**	**BJT**		**RFKZ**		**CFT**	**BJ**	**CNS**	**Meta-RaPS**	**YKI**		**ABC_SCP**	
		Best	Avg	Best	Best	Avg	Best	Avg	Best	Best	Best	Best	Best	Avg	Best	Avg
NRH.4	58[a]	58	58.9	59	58	58.1	58	58.7	58	—	59	58	58	58.0	58	58.0
NRH.5	55[a]	55	55.1	55	55	55.0	55	55.0	55	—	55	55	55	55.4	55	55.0

[a]Best known solution values

Table 4.3: Comparison of Different Methods in terms of Number of Instances Solved to Optimal/Best Known Solution Values

CFT	BeCh	IGA	PROGRESS	Meta-RaPS	ANTS+LB	YKI	RFKZ	ABC_SCP
65/65	61/65	61/65	20/65	65/65	65/65	65/65	64/65	64/65

find optimal/best known solution values for all 65 instances. YKI finds optimal/best known solutions in all 10 trials in 62 instances.

As far as comparison with other population-based metaheuristic approaches is concerned, ABC_SCP is clearly better than BeCh and IGA in terms of solution quality. BeCh and IGA are able to find optimal/best known solution values for 61 instances. There are only 27 instances on which BeCh finds optimal/best known solution in all 10 trials. Though RFKZ and ABC_SCP find the optimal/best known solution values for 64 out of 65 instances, there are only 55 instances on which RFKZ finds optimal/best known solution values in all 10 trials against 60 such instances of ABC_SCP. Only, ANTS+LB is better than ABC_SCP as it finds optimal solution in all 10 trials. However, the strength of ANTS+LB comes from the use of $r\text{-}flip$ local search of Yagiura *et al.* [139]. Without $r\text{-}flip$ local search of [139], it was not able to find optimal solutions even in 30% trials [136]. If we look into the earlier version of RFKZ [146], we find that the major strength of RFKZ comes from using Lagrangian relaxation of SCP to guide ACO. Moreover, from Figure 5 of [1], it is quite clear that the local search also contributes a lot to the success of their method. Unlike these swarm intelligence methods which depend heavily on other techniques for their strength, performance of ABC_SCP is affected only slightly by the local search as shown later in this section.

Table 4.4 reports the average time till best (ATTB) and average execution time (AET) of ABC_SCP on all 65 instances. ATTB of our method is much smaller than AET on most of the instances, therefore, we may reduce the iterations of our algorithm without affecting much the number of instances solved to their optimal/best known solution values.

To get an idea about the effect of local search on solution quality and execution time, we have also implemented another version of ABC_SCP without local search. This version of ABC_SCP will be referred to as ABC_SCP-NLS. Table 4.5 reports the results of ABC_SCP-NLS. Even ABC_SCP-NLS was able to find optimal/best known solution

Table 4.4: Average Time Till Best (ATTB) and Average Execution Time (AET) in Seconds of ABC_SCP on Each of the 65 Instances

Instance	ATTB	AET	Instance	ATTB	AET	Instance	ATTB	AET
4.1	0.84	5.03	6.3	0.09	10.26	D.5	1.11	67.42
4.2	0.05	4.82	6.4	0.04	9.10	NRE.1	1.64	89.05
4.3	0.06	4.87	6.5	0.11	13.11	NRE.2	3.47	98.92
4.4	0.12	5.40	A.1	0.51	11.85	NRE.3	7.46	104.09
4.5	0.30	5.57	A.2	0.71	11.14	NRE.4	3.15	93.18
4.6	0.08	5.07	A.3	2.65	10.73	NRE.5	1.74	97.71
4.7	0.13	4.97	A.4	0.29	11.61	NRF.1	3.45	330.41
4.8	0.25	5.40	A.5	0.31	10.72	NRF.2	3.32	282.54
4.9	0.34	4.30	B.1	0.41	39.41	NRF.3	3.96	308.46
4.10	0.08	5.55	B.2	0.28	37.63	NRF.4	6.00	325.11
5.1	0.15	6.99	B.3	0.21	34.80	NRF.5	66.47	324.23
5.2	0.73	6.06	B.4	0.32	41.69	NRG.1	13.62	97.54
5.3	3.65	7.41	B.5	0.18	35.94	NRG.2	10.23	96.89
5.4	0.04	6.16	C.1	0.56	17.18	NRG.3	50.81	94.41
5.5	0.04	6.56	C.2	0.51	17.27	NRG.4	26.27	92.51
5.6	0.05	6.67	C.3	6.21	18.77	NRG.5	5.01	92.80
5.7	0.10	7.29	C.4	0.38	17.54	NRH.1	52.62	532.36
5.8	0.07	6.12	C.5	0.28	17.63	NRH.2	41.18	533.91
5.9	0.27	6.12	D.1	0.80	78.57	NRH.3	91.13	557.13
5.10	0.09	6.13	D.2	1.99	74.27	NRH.4	171.79	565.78
6.1	0.15	11.60	D.3	2.11	74.56	NRH.5	13.97	539.07
6.2	0.05	10.69	D.4	0.95	66.14			

values for 63 instances. There are 57 instances on which ABC_SC-NLS finds optimal/best known solution values in all 10 trials. Therefore, ABC_SCP is only slightly better than ABC_SCP-NLS. However, if we compare AET of two methods, the we can see that in many cases ABC_SCP-NLS takes roughly the same or more time than ABC_SCP. This seems to be counterintuitive. Actually, we found that ABC_SCP generates more number of random solutions because of bees becoming scouts than ABC_SCP-NLS. As generating a random solution consumes less time than generating a solution by adding and deleting columns followed by the repair operator, ABC_SCP-NLS takes roughly the same or more time than ABC_SCP. The reason for more bees becoming scout in case of ABC_SCP is that the local search reduces the diversity of the population which leads to increased collisions. However, ABC_SCP converges faster than ABC_SCP-NLS on majority of instances as indicated by their ATTB values. As the

Table 4.5: Performance of ABC_SCP-NLS

Instance	**Best**	**Avg**	**ATTB**	**AET**	**Instance**	**Best**	**Avg**	**ATTB**	**AET**
4.1	429	429.0	0.53	4.02	B.4	79	79.0	0.78	41.44
4.2	512	512.0	0.12	4.00	B.5	72	72.0	0.47	36.10
4.3	516	516.0	0.59	4.03	C.1	227	227.0	2.29	15.73
4.4	494	494.0	0.38	4.03	C.2	219	219.0	1.25	16.27
4.5	512	512.0	0.13	4.05	C.3	243	243.4	8.04	16.57
4.6	560	560.0	0.32	4.03	C.4	219	219.0	0.80	16.20
4.7	430	430.0	0.12	4.00	C.5	215	215.0	0.75	16.40
4.8	492	492.0	0.17	3.96	D.1	60	60.0	1.78	82.06
4.9	641	641.0	0.46	3.93	D.2	66	66.0	2.72	73.65
4.10	514	514.0	0.12	4.02	D.3	72	72.0	5.09	73.97
5.1	253	253.0	0.40	5.55	D.4	62	62.0	1.59	67.40
5.2	302	302.0	0.67	5.23	D.5	61	61.0	1.78	68.47
5.3	226	226.0	0.56	6.08	NRE.1	29	29.0	1.91	85.66
5.4	242	242.0	0.20	5.87	NRE.2	30	30.0	6.13	94.41
5.5	211	211.0	0.26	6.28	NRE.3	27	27.0	6.38	101.26
5.6	213	213.0	0.13	5.55	NRE.4	28	28.0	3.70	91.20
5.7	293	293.0	0.36	6.06	NRE.5	28	28.0	2.05	97.55
5.8	288	288.0	0.22	6.19	NRF.1	14	14.0	3.88	352.4
5.9	279	279.9	0.45	6.16	NRF.2	15	15.0	3.54	289.53
5.10	265	265.0	0.59	6.05	NRF.3	14	14.0	5.35	328.14
6.1	138	138.0	1.22	12.64	NRF.4	14	14.0	4.97	318.2
6.2	146	146.0	0.31	10.22	NRF.5	13	13.3	129.31	349.98
6.3	145	145.0	0.28	11.86	NRG.1	176	176.0	18.18	95.34
6.4	131	131.0	0.30	12.24	NRG.2	155	155.0	19.02	96.31
6.5	161	161.0	0.24	10.81	NRG.3	166	166.5	46.28	87.69
A.1	253	253.0	4.30	9.89	NRG.4	168	169.1	33.09	88.76
A.2	252	252.0	1.11	9.84	NRG.5	168	168.0	10.13	90.57
A.3	232	232.0	1.86	10.31	NRH.1	64	64.0	38.24	531.43
A.4	234	234.0	0.57	9.80	NRH.2	63	63.5	164.55	520.47
A.5	236	236.0	3.13	9.41	NRH.3	59	59.0	57.80	561.13
B.1	69	69.0	0.70	40.37	NRH.4	58	58.0	153.56	559.54
B.2	76	76.0	0.78	40.84	NRH.5	55	55.0	20.17	526.51
B.3	80	80.0	0.65	36.24					

execution times of ABC_SCP and ABC_SCP-NLS are comparable and ABC_SCP gives slightly better results, we decided to use it as the primary approach of our chapter instead of ABC_SCP-NLS.

The usual practice for comparing the times of different approaches for SCP is to report the expected time taken by each method to reach the best solution for the first time on DECstation 5000/240 [123, 135, 140]. Caprara *et al.* [123] calculated the time in the following way: suppose the best solution is first found in the trial number X, then the time taken to reach the best solution for the first time is the sum of AET $\times(X-1)$ and the time taken to reach the best solution in the trial X. Though this is not the fairest way to compare the times, we have followed this method because of the past precedences. To get the expected time of ABC_SCP and ABC_SCP-NLS on DECstation 5000/240 we managed to execute ABC_SCP and ABC_SCP-NLS on a small set of instances on a Pentium 4, 1.7 GHz system which is the same system as used for executing Meta-RaPS [140]. We chose two instances from each class, one with smallest execution time and the other with largest execution time, for execution on Pentium 4, 1.7 GHz system and found that our system is 2.6 times faster on an average than Pentium 4, 1.7 GHz system. Once we got an estimate of ABC_SCP and ABC_SCP-NLS times on Pentium 4, 1.7 GHz, we have converted them to times on DECstation 5000/240 using the data provided in Table 8 of Lan *et al.* [140]. There are 3 instances in ABC_SCP and 4 instances in ABC_SCP-NLS on which best solution is found in second or subsequent trials. For ABC_SCP, these instances are NRF.5, NRH.1, NRH.2 where it finds the best solution in respectively 5^{th}, 7^{th} and 9^{th} trials. For ABC_SCP-NLS, these instances are 5.3, NRG.3, NRG.4, and NRH.2 where it finds the best solution in respectively 2^{nd}, 3^{rd}, 6^{th} and 2^{nd} trials. Table 4.6 reports the time on DECstation 5000/240 of ABC_SCP, ABC_SCP-NLS along with CFT, Meta-RaPS and BeCh for all 65 instances. CFT is clearly the fastest among all the techniques. ABC_SCP is faster than Meta-RaPS and BeCh on 32 and 31 instances respectively, whereas it is slower on the remaining instances. Here it is to be noted that this method of computing time disregards solution quality. If a method finds the same inferior solution in all trials, then its time may be less than an otherwise faster method that finds a better solution in second or subsequent trials. This is exactly the case when we compare our ABC_SCP with BeCh. Though we have not reported the times of IGA and YKI, they are faster than our approaches.

Table 4.6: Expected Average Time/Average Time TILL Best in Seconds of Different Methods on DECstation 5000/240

Instance	ABC_SCP	ABC_SCP-NLS	Meta-RaPS	BeCh	CFT
4.1	149.59	94.38	93.15	294.80	2.3
4.2	8.90	21.37	16.44	9.00	1.1
4.3	10.68	105.07	19.86	16.40	2.1
4.4	21.37	67.67	26.71	142.00	9.8
4.5	53.42	23.15	61.64	44.10	2.1
4.6	14.25	56.99	6.85	16.10	19.3
4.7	23.15	21.37	2.74	138.60	2.7
4.8	44.52	30.27	99.99	818.70	22.2
4.9	60.55	81.92	237.66	136.10	1.8
4.10	14.25	21.37	5.48	13.50	1.8
5.1	26.71	71.23	106.16	42.10	3.3
5.2	130.00	119.31	40.41	1,332.60	2.3
5.3	650.00	99.72	78.08	11.00	2.1
5.4	7.12	35.62	21.92	10.10	1.9
5.5	7.12	46.30	22.60	14.90	1.2
5.6	8.90	23.15	9.59	29.90	0.9
5.7	17.81	64.11	70.54	194.90	15.0
5.8	12.47	39.18	5.48	3,733.30	1.6
5.9	48.08	1,177.18	2.74	13.50	2.6
5.10	16.03	105.07	2.05	19.20	1.3
6.1	26.71	217.26	17.12	46.10	22.6
6.2	8.90	55.20	1.37	210.50	17.8
6.3	16.03	49.86	1.37	11.80	2.3
6.4	7.12	53.42	23.29	4.80	1.8
6.5	19.59	42.74	69.86	12.10	2.2
A.1	90.82	765.74	426.01	222.40	82.0
A.2	126.44	197.67	19.18	327.90	116.2
A.3	471.91	331.23	1,160.22	127.00	249.9
A.4	51.64	101.51	2.74	45.50	4.7
A.5	55.20	557.39	641.75	23.70	80.0
B.1	73.01	124.66	9.59	20.00	4.0
B.2	49.86	138.90	36.30	11.60	6.1
B.3	37.40	115.75	42.46	709.70	18.0
B.4	56.99	138.90	154.10	29.90	6.3
B.5	32.05	83.70	0.00	5.30	3.3
C.1	99.72	407.80	29.45	187.90	74.0
C.2	90.82	222.60	882.84	40.70	64.2
C.3	1,105.90	1,431.76	1,797.18	541.30	70.2

Continued on next page

Table 4.6 – continued from previous page

Instance	ABC_SCP	ABC_SCP-NLS	Meta-RaPS	BeCh	CFT
C.4	67.67	142.46	1,663.62	144.60	61.6
C.5	49.86	133.56	122.60	80.60	60.3
D.1	142.50	316.98	214.37	13.80	23.1
D.2	354.38	484.38	930.78	198.60	22.0
D.3	375.75	906.43	89.72	785.30	22.6
D.4	169.18	283.15	13.70	73.50	8.3
D.5	197.67	316.98	19.86	79.80	10.3
NRE.1	292.05	340.13	50.00	38.20	26.0
NRE.2	617.94	1,091.63	3,162.18	14,647.70	408.0
NRE.3	1,328.48	1,136.15	407.52	28,360.20	94.2
NRE.4	560.95	658.90	2,714.94	539.90	26.3
NRE.5	309.86	365.06	55.48	35.00	36.6
NRF.1	614.38	690.95	293.82	76.40	33.2
NRF.2	591.23	630.40	260.26	78.10	31.2
NRF.3	705.20	952.73	126.02	266.80	248.5
NRF.4	1,068.48	885.06	372.59	209.70	31.0
NRF.5	242,792.49	23,027.52	2,278.66	13,192.60	201.1
NRG.1	2,425.45	3,237.49	20,476.46	30,200.00	147.0
NRG.2	1,821.76	3,387.08	15,228.07	360.50	783.4
NRG.3	9,048.25	39,508.83	1,476.64	7,841.60	978.0
NRG.4	4,678.16	84,924.57	13,301.44	25,304.70	378.5
NRG.5	892.18	1,803.95	3,258.07	549.30	237.2
NRH.1	578,186.58	6,809.78	268,280.80	1,682.10	1,451.1
NRH.2	767,962.88	121,988.36	16,331.44	530.30	887.0
NRH.3	16,228.43	10,293.02	53,641.37	1,803.50	1,560.3
NRH.4	30,592.36	27,345.96	93,028.60	27,241.8	237.6
NRH.5	2,487.78	3,591.87	384.91	449.60	155.4

As far as comparison of times of ABC_SCP with RFKZ and ANTS+LB is concerned, we can compare them directly. RFKZ was executed on a 2.0 GHz Pentium 4 system with 1 GB RAM, whereas ANTS+LB was executed on a 2.4 GHz Xeon Processor based system with 2 GB RAM. As RFKZ and ANTS+LB are executed on different systems, it is not possible to exactly compare the speed of these algorithms with ABC_SCP, however, a rough comparison can always be made. If we compare ATTB and AET of ABC_SCP with those of RFKZ reported in [1] after compensating for the difference in processing speeds, RFKZ is clearly faster than ABC_SCP in terms of both ATTB and AET. Regarding the time comparison with ANTS+LB of [136], ANTS+LB was allowed during each trial to execute for 100s. Therefore, AET of ANTS+LB on each instance is 100s. Even after compensating for processing speeds, ABC_SCP is faster than ANTS+LB on majority of instances in terms of AET. No ATTB values were reported in [136], therefore, comparison of ATTB values of ABC_SCP with ANTS+LB are not possible.

4.4 Conclusions

In this chapter, we have presented a hybrid artificial bee colony algorithm (ABC_SCP) for the non-unicost set covering problem. ABC_SCP has been compared with the best population-based methods and the overall best methods. Its results are comparable with all these methods in terms of solution quality though the overall best methods are much faster. As far as comparison with other population-based methods is concerned, except for one method, it has outperformed all the other population-based methods in terms of solution quality. However, it is slower than some population-based methods.

Chapter 5

Dominating Tree Problem

5.1 Introduction

The recent years have seen a rapid growth in the area of wireless sensor networks (WSNs) that have attracted attention of many researchers. Dominating tree problem (DTP) is one among several new problems in WSNs that have wide applicability. DTP is defined as follows: let $G = (V, E)$ be an undirected, connected graph representing WSNs, where V denotes the set of vertices and E denotes the set of edges. Given a non-negative weight function $w : E \rightarrow \Re^+$ associated with the edges of G, DTP seeks on G a tree DT with minimum total edge weight such that for each vertex $v \in V$, v is either in DT or adjacent to a vertex in DT. Vertices in DT are called dominating vertices, whereas vertices not in DT are called non-dominating vertices.

A solution to DTP provides a virtual backbone for routing. Actually, when we compute a dominating tree, routing information need to be kept only on the dominating nodes. Non-dominating nodes are anyway one hop away from dominating nodes. Therefore, a message sent from one node to another can always be routed by forwarding the message to the nearest dominating node of the sender and then using the dominating tree to route the message to the nearest dominating node of the receiver and finally from this node to the receiver. Non-dominating nodes only need to remember the nearest dominating node. Advantage of this scheme is that routing information need to be stored only in dominating nodes which are few in numbers in comparison to total nodes [147]. Therefore, the size of the routing table is reduced significantly. Moreover, there is no need to recalculate these tables as long as topological changes in the network do not affect any of the dominating nodes [147]. In literature, connected dominating set concept have been used in [148, 149, 150, 151, 152] for building a routing backbone in wireless networks with minimum energy consumption. But these papers only consider the nodes and not the edges to minimize energy consumption. In most of these algorithms, a weight is associated with each node and not with each edge. Actually, energy consumed by each edge directly affects the energy consumption of routing. Therefore, to minimize energy consumption of routing, we have to consider the energy consumed by each edge. With this intent Shin *et al.* [153] introduced DTP. They proved inapproximability result and presented an approximation algorithm to solve DTP. As the time complexity of this approximation algorithm is quasi-polynomial ($|V|^{O(lg|V|)}$), they also proposed a polynomial time heuristic for DTP. This heuristic will be referred to as Shin_DT. Tree cover problem is a related problem that has been studied in literature [154, 155, 156], but this problem is different from DTP. A tree in the tree cover problem is defined as an edge dominating set, whereas a tree in DTP is defined as a node dominating set.

In this chapter, we have presented a problem-specific heuristic, an ABC algorithm and an ACO algorithm to solve DTP. We have compared our approaches with those in [153]. Computational results demonstrate the effectiveness of our approaches.

The rest of this chapter is organized as follows: Section 5.2 describes a heuristic approach to DTP. Section 5.3 describes an ABC approach for DTP, whereas Section 5.4 describes an ACO approach. Computational results are reported in Section 5.5. Finally, Section 5.6 contains some concluding remarks.

5.2 Heuristic for DTP

Like Shin *et al.* [153], we also propose a problem-specific heuristic called Heu_DT for DTP. It is clear that DTP seeks a minimum weight dominating tree DT which does not include all vertices of G. This characteristic of DTP is kept in mind while designing Heu_DT. Heu_DT follows a greedy approach which is based on the concept of Kruskal's algorithm [111] and the shortest paths between all pairs of vertices in G.

Heu_DT consists of two phases: an initialization phase and an iterative phase. The initialization phase begins by computing the shortest paths between all pairs of vertices in $G = (V, E)$. All edges in E are sorted into non-decreasing order according to their weights. Each vertex is assigned an unmarked label. The set DT is initialized to ϕ. Then the iterative phase begins by adding a minimum weight edge e_{ij} to DT and continues till all vertices in G are labeled marked. Vertices i and j along with their all adjacent unmarked vertices are now labeled marked. After that, at each iteration, a minimum weight unselected edge e_{ij}, whose at least one endpoint is unmarked, is selected. All unmarked vertices among i, j and their adjacent vertices are labeled marked. After that a check is done to determine whether e_{ij} is adjacent to an edge in DT or not. If it is not, then a shortest path, say ST, connecting DT and vertices $\{i, j\}$ is determined to establish a connection between DT and the edge e_{ij}. All edges in ST along with the edge e_{ij} are added to DT. Each unmarked vertex in ST along with their unmarked adjacent vertices are labeled marked now and the next iteration begins. It is to be noted that while selecting a shortest path, some or all vertices which are part of this path, may be marked already, but we consider them in constructing a DT. Heu_DT stops when all vertices in G are labeled marked.

Since there is a possibility that there can be more than one shortest path which have same cost. So a tie breaking rule is used here. According to this rule, among same cost shortest paths, we select a shortest path that has maximum number of unmarked vertices. If still, there is more than one shortest path having same number of unmarked vertices, then we go one step deeper for using another tie breaking rule, i.e., select a shortest path that contains maximum number of vertices from all shortest paths having the same number of unmarked vertices. If still, there is more than one candidate shortest path, then ties are broken arbitrarily.

Hereafter, a pruning procedure is applied to check whether dominating vertices with degree one in DT can be pruned without violating the feasibility of DT. If a vertex with degree one in DT can be pruned, then the edge incident to this dominating vertex can also be deleted from DT, thereby reducing the total edge weight of DT. In pruning procedure, we check all dominating vertices in DT one-by-one. If the degree of a dominating vertex, say v, in DT is one , then it is checked whether all non-dominating vertices, which are adjacent to v, are also adjacent to other dominating vertices in DT. If it is so, then the edge incident to v is deleted from DT. Vertex v, which is a dominating vertex, now becomes a non-dominating vertex. This is quite different from the pruning procedure used in Shin_DT [153], where first a spanning tree is constructed, and

Algorithm 7: A Heuristic Approach to DTP

```
   input  : A connected graph $G = (V, E)$
   output: A dominating tree $DT \subseteq E$
1  Initialize $DT \leftarrow \emptyset$;
2  for ($each\ vertex\ i\ in\ V$) do
3      $Mark[i] \leftarrow 0$;
4  Compute shortest paths between all pairs of vertices in $G$;
5  Sort all edges in $E$ into non-decreasing order according to their weights;
6  while ($all\ vertices\ in\ V\ are\ not\ marked$) do
7      Select an unselected edge $e_{ij}$ of minimum cost, whose at least one endpoint is
       unmarked;
8      if ($Mark[i] \neq 0$) then $Mark[i] \leftarrow 1$;
9      for ($each\ vertex\ v\ adjacent\ to\ i$) do
10         if ($Mark[v] \neq 0$) then $Mark[v] \leftarrow 1$;
11     for ($each\ vertex\ v\ adjacent\ to\ j$) do
12         if ($Mark[v] \neq 0$) then $Mark[v] \leftarrow 1$;
13     if ($DT = \emptyset$) then $DT \leftarrow DT \cup \{e_{ij}\}$;
14     else if ($e_{ij}\ is\ not\ adjacent\ to\ an\ edge\ in\ DT$) then
15         Find a shortest path $ST$ connecting $DT$ and $\{i, j\}$;        // Use tie
           breaking rule if needed
16         $DT \leftarrow DT \cup ST \cup \{e_{ij}\}$;
17         for ($each\ vertex\ v\ in\ ST$) do
18             if ($Mark[v] \neq 0$) then $Mark[v] \leftarrow 1$;
19         for ($each\ vertex\ v\ in\ ST$) do
20             for ($each\ vertex\ k\ adjacent\ to\ v$) do
21                 if ($Mark[k] \neq 0$) then $Mark[k] \leftarrow 1$;
22 Apply pruning procedure on $DT$;
23 Return $DT$;
```

then all leaf vertices are pruned without any further checks. The pseudo-code of Heu_DT is given in Algorithm 7.

It is to be noted that the idea of marking vertices is based on common sense and avoids as far as possible placing both a vertex and its neighbors in DT. A similar marking procedure is used in [147] for finding connected dominating set. The motivation behind considering a shortest path between a vertex in DT and a vertex of the selected edge is also based on common-sense as doing so will increase the cost of DT by minimum amount at each iteration. The first tie breaking rule considers maximum number of

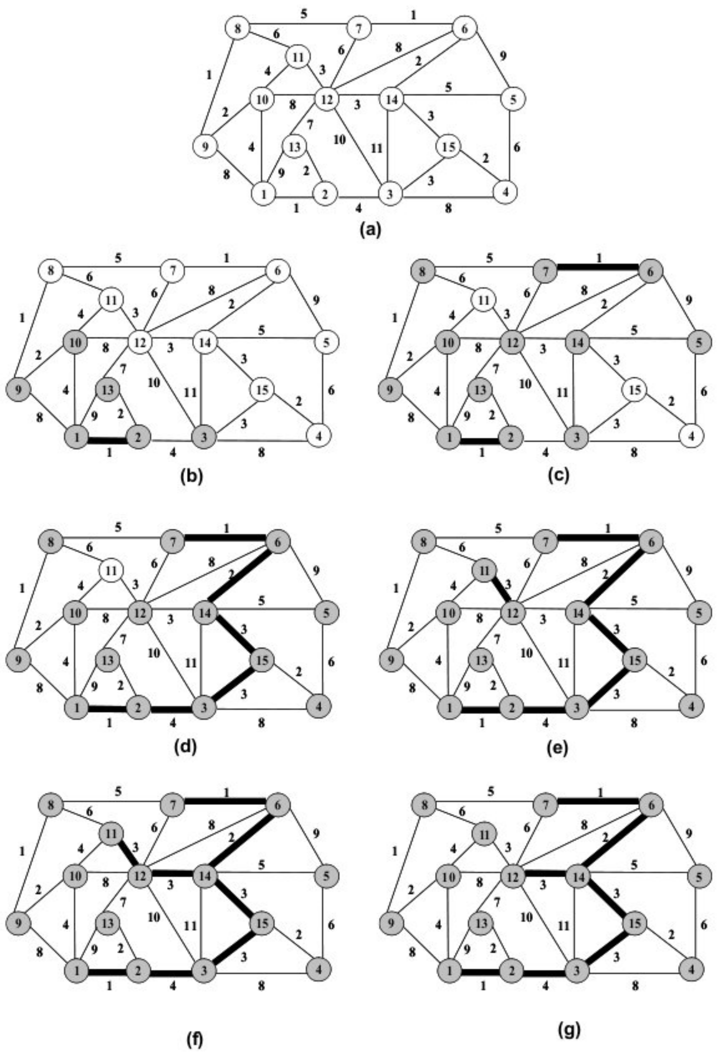


Figure 5.1: Execution of Heu_DT

unmarked vertices in the path to break the tie. This helps in maximizing the number of non-dominating vertices as far as possible by making not only unmarked vertices on the path marked, but also all their adjacent unmarked vertices. The second tie breaking rule also helps in maximizing the number of non-dominating vertices. Actually, these tie breaking rules are based on the fact that in the neighborhood of unmarked vertices, in general, we can find more unmarked vertices than in the neighborhood of marked vertices.

The pseudo-code of Heu_DT clearly shows that the running time of Heu_DT is mainly dominated by computing the shortest paths between all pairs of vertices in G.

Figure 5.1 illustrates Heu_DT with the help of an example: Figures 5.1 (b)-5.1 (f) represent the various stages in execution of Heu_DT. Figure 5.1 (a) shows an undirected, connected and weighted graph $G = (V, E)$, where $|V| = 15$ and $|E| = 27$. Initially, each vertex in V is unmarked (such vertices are shown in white in these figures). Initially, DT is empty. As all vertices are unmarked at this juncture, therefore, a minimum cost edge $(1, 2)$ is added to DT. Vertices 1, 2 and all of their adjacent vertices are marked now (such vertices are shown in grey in these figures). This is shown in Figure 5.1 (b). Then, in next iteration, an unselected minimum cost edge $(6, 7)$ (neither of its endpoints are marked) is selected. Vertices 6, 7 and all of their adjacent vertices are marked next. This is shown in Figure 5.1 (c). As edge $(6, 7)$ is not connected to DT, therefore a shortest path connecting these two components is determined. There are two candidate shortest paths with cost 12, i.e., first path with edges $(2, 3)$, $(3, 15)$, $(15, 14)$, $(14, 6)$ and another path with edges $(1, 10)$, $(10, 9)$, $(9, 8)$, $(8, 7)$. As the first path has more number of unmarked vertices, therefore, as per first tie breaking rule, it is selected. Let this selected path be called ST. All edges in ST and the edge $(6, 7)$ itself are added to DT. Next, vertices 3, 15 and 14, and their adjacent vertices are marked. This is shown in Figure 5.1 (d). In next two iterations of Heu_DT, we reach the situations shown in Figure 5.1 (e) and Figure 5.1 (f). At this juncture, all vertices are marked and Heu_DT stops. Hereafter, the pruning procedure is applied. It can be seen that only dominating vertex 11 with degree one in DT has two non-dominating vertices 8 and 10 as its neighbors. Vertices 8 and 10 both are adjacent to other dominating vertices. Therefore, the edge $(11, 12)$, incident to the dominating vertex 11, can be pruned. The resulting final DT with cost 17 is shown in Figure 5.1 (g).

5.3 ABC Algorithm for DTP

As a second approach for DTP, we present an ABC algorithm. This algorithm is referred to as ABC_DT in this chapter. The important features of ABC_DT are described below:

5.3.1 Initial Solution Generation

Each initial solution is generated by an iterative process. Initially, S, U and DT are the three empty sets. A vertex v_1 is selected randomly and added to S. All vertices, which are adjacent to v_1, are added to U. At each step, an edge is selected by one of the two procedures in order to maintain a balance between quality and diversity of the solution. With probability q_0, an edge with minimum weight, connecting a vertex v_1 in S to a vertex v_2 in U, is selected. Otherwise, an edge, connecting a vertex v_1 in S to a vertex v_2 in U, is selected randomly from all candidate edges using the roulette wheel selection method where the probability of the selecting an edge is inversely proportional to its weight. After that vertex v_2 is deleted from U and added to S. Vertices, which are adjacent to v_2 and neither a member of U nor S, are added to U. This selected edge is added to DT. This whole process is repeated again and again until the sum of cardinalities of S and U becomes equal to the total number of vertices in G. At this juncture, we have a dominating tree DT.

Now, we check whether dominating vertices with degree one in DT can be pruned without disturbing the feasibility of the solution. If a vertex with degree one in DT can be pruned, then the edge incident to this dominating vertex can also be deleted from DT, resulting in further reduction of the total edge weight of DT. In pruning procedure, we check all dominating vertices in DT one-by-one. If the degree of a dominating vertex, say v_p, in DT is one , then it is checked whether all non-dominating vertices, which are adjacent to v_p, are also adjacent to other dominating vertices in DT. If it is so, then the edge incident to v_p, is deleted from DT. Vertex v_p, which is a dominating vertex, now becomes a non-dominating vertex. This pruning procedure is repeated again and again until no dominating vertex with degree one can be pruned.

Thereafter, a minimum spanning tree is constructed on the subgraph of G induced by the set of dominating vertices of the solution with the help of Prim's algorithm [114]. This way total weight of DT can be reduced further. The reason behind this is that even after pruning, the total weight of DT may not be minimum due to the selection of incorrect edges while constructing DT. On a given a set of dominating vertices, many dominating trees can be constructed in G. Obviously, a dominating tree obtained using Prim's algorithm will be of minimum cost among all such dominating trees.

Each solution is uniquely associated with an employed bee and the fitness of each solution is computed.

5.3.2 Probability of Selecting a Food Source

In our ABC approach, an onlooker selects a food source with the help of binary tournament selection method where the better food source is selected with probability p_{bt}.

5.3.3 Determination of a Food Source in the Neighborhood of a Food Source

In order to determine a neighboring food source, we follow two procedures which are mutually exclusive. Initially, a copy s' of a solution (food source) s is created. Then, with probability q_1, the first procedure (referred to as PDE) is called for deleting an edge randomly from s', otherwise the second procedure (referred to as $PANDV$) is called for adding a non-dominating vertex randomly to s'. The reason behind choosing two procedures, which are mutually exclusive, is that it is experimentally observed that better solutions are obtained this way than using only one of the two procedures. The two procedures are described below:

PDE procedure: an edge e_{ij}, whose at least one endpoint has at least one non-dominating adjacent vertex, is randomly deleted from s'. This delete operation causes the partitioning of s' into two components, say L_1 and L_2, and makes s' infeasible. To make s' feasible again, a shortest path (excluding e_{ij}) between L_1 and L_2 in G is determined and L_1 and L_2 are reconnected through this path.

It is to be noted that if this procedure is not able to find even a single edge (different from deleted edge e_{ij}) connecting these two components, then deleted edge e_{ij} is added back to s' and the whole procedure is applied again by randomly selecting another edge. This continues until we find an edge different from the deleted edge.

$PANDV$ procedure: a non-dominating vertex v, whose degree is greater than one in G, is randomly selected from the set of current non-dominating vertices. A shortest path is determined between v and the set of dominating vertices of s'. All edges on this path are added to s'.

Hereafter, a pruning procedure is applied on s'. This pruning procedure is similar to the one used in initial solution generation except for one small difference. This procedure does not consider those vertices which lie on the newly added shortest path (This is done to avoid loosing immediately the newly added vertices in pruning). Then a minimum spanning tree is constructed on the subgraph of G induced by the set of dominating vertices of the solution with the help of Prim's algorithm. Then pruning procedure is applied again for all dominating vertices (including those excluded previously). After this pruning, again a minimum spanning tree is constructed on the subgraph of G induced by the set of dominating vertices of the solution with the help of Prim's algorithm. We tried further pruning, but experimental observations suggested that it is not needed.

5.3.4 Other Features

If a solution (food source) does not improve for a predetermined number of iterations, called *limit*, then this solution is replaced with a newly generated solution. This new solution is generated in the same way as an initial solution.

5.4 ACO for DTP

We also present another swarm-based approach, i.e., ACO approach for DTP. This algorithm is referred to as ACO_DT in this chapter. In ACO_DT, pheromone is laid on the vertices of the graph. This is due to the fact that the choice of vertices plays a more important role than the choice of edges in construction of good solutions . Once we have a dominating tree containing certain vertices, we can always find a dominating tree of minimum cost comprising only these vertices by running Prim's algorithm on the subgraph of G induced by these vertices. Moreover, laying pheromones on edges, which are more in numbers than vertices, will slow down the algorithm substantially. In ACO_DT, we have incorporated iteration best pheromone trail update strategy.

5.4.1 Solution Construction

In order to construct a solution by an ant, a slightly modified version of problem-specific heuristic used to generate an initial solution for ABC_DT (as described in Section 5.3.1) is used in ACO_DT. The only difference lies in the process of selecting an edge connecting a vertex in S to an unselected vertex in U. At each step, an ant k selects an edge e_{ij}, connecting a vertex i in S to an unselected vertex j in U from all candidate edges, probabilistically with the help of pheromones on vertices and heuristic information. The probability $p^k_{e_{ij}}$ of selecting an edge e_{ij} by an ant k is determined as follows:

$$p^k_{e_{ij}} = \frac{[\tau_j]^\alpha [\eta_{e_{ij}}]^\beta}{\sum\limits_{l \in N^k_i} [\tau_l]^\alpha [\eta_{e_{il}}]^\beta} \tag{5.1}$$

where τ_j is the pheromone value on the vertex j and $\eta_{e_{ij}}$ is the heuristic term which is equal to $\frac{1}{w_{ij}}$. Here w_{ij} is the weight of the edge e_{ij}. α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information in the solution construction process. N^k_i is the set of unselected vertices which are adjacent to the vertex i. This probabilistic decision rule is influenced by pheromone values on vertices and heuristic information both and helps in finding good solutions in the search space.

Once a solution is constructed, a pruning procedure is called for all dominating vertices in the solution, and then a minimum spanning tree is constructed on the set of dominating vertices of the solution (as explained in Section 5.3.1).

5.4.2 Pheromone Update

Pheromone update rule plays a vital role in finding good solutions. It is used to increase the pheromone values on solution components, i.e., vertices that have been present in high quality solutions and helps in directing ants in future iterations towards better solutions. In our approach, pheromone is updated once at the end of each iteration, say t, of ACO algorithm when all ants have constructed their dominating trees. The ant that constructed the best solution of the iteration updates the pheromone trails on the vertices of the graph in the following way:

$$\tau_i(t+1) = \rho\, \tau_i(t) + \Delta\tau^{ib}_i \tag{5.2}$$

where ρ is the persistence rate and $\Delta\tau^{ib}_i$ is the amount of reinforcement on vertex i due to the iteration best solution S^{ib}. $\Delta\tau^{ib}_i$ is computed using following expression:

$$\Delta\tau^{ib}_i = \begin{cases} P, & \text{if } i \text{ is present in } S^{ib}; \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

where P is a parameter to be determined empirically.

As described above, pheromone evaporation, which can be seen as an exploratory mechanism, takes place on all vertices of the graph at a fixed rate (1-ρ), then the dominating vertices of the iteration best solution S^{ib} receive reinforcement.

We have explicitly imposed a lower bound τ_{min} on the pheromone concentrations. However, no upper bound was imposed.

5.5 Computational Results

Our approaches for solving DTP have been implemented in C and executed on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. For ABC_DT, we have used a population of 250 bees, out of which 100 are employed bees and remaining 150 are onlooker bees, i.e., $n_e = 100$ and $n_o = 150$. We have used $limit = 50$, $p_{bt} = 0.85$, $q_0 = 0.80$ and $q_1 = 0.4$. For ACO_DT, we have used a population of 20 ants, i.e., $n_a = 20$. We have used $\alpha = 1, \beta = 1, \rho = 0.97$, $P = 0.1$, $\tau_{min} = 0.005$. All pheromone values are initialized to 13. Such a pheromone initialization leads to a wider exploration of the search space during initial iterations. Note that the maximum pheromone value that can be sustained with our value of $\rho = 0.97$ and $P = 0.1$ is about 3.333. We have allowed ABC_DT to execute for 5000 generations, whereas 2500 generations are allowed for ACO_DT. All these parameter values are chosen empirically. These parameter values give good results though they may not be optimal for all test instances. ABC_DT and ACO_DT have been executed on each test instance 20 independent times.

As the sizes of test instances used in Shin *et al.* [153] were too small for any real scenario, therefore, we have generated a set of 18 test instances of bigger size. Like Shin *et al.* [153], we also consider a disk graph $G = (V, E)$ where each disk represents the transmission range of each node. The weight of each edge e_{ij} in E is defined as $w(i, j) = C_j \times d_{ij}^2$ where d_{ij} is the Euclidean distance between two nodes i and j, and C_j is a random constant. We have taken C_j to be 1. We assume that $|V|$ nodes are randomly deployed in a $500m \times 500m$ area and the transmission range of each node is $100m$. For each value of $|V| \in \{50, 100, 200, 300, 400, 500\}$, three different test instances are generated leading to a total of 18 test instances. In addition, for the sake of comparison, we have implemented the problem-specific heuristic (Shin_DT) of Shin *et al.* [153] and an algorithm to compute the minimum spanning tree without leaf edges (MST-L). Note that a minimum spanning tree of G after removing its leaf edges is also

Table 5.1: Results of MST-L, Shin_DT, and Heu_DT for DTP

Instance	**MST-L**		**Shin_DT**		**Heu_DT**	
	Value	NDV	Value	NDV	Value	NDV
50_1	1860.67	38	1819.91	35	1430.82	23
50_2	1780.66	37	1795.89	35	1619.90	29
50_3	1860.12	39	1863.01	35	1648.55	27
100_1	2491.23	76	2290.28	61	1856.98	28
100_2	2515.82	78	2265.68	62	1591.36	25
100_3	2670.84	77	2488.15	59	1763.91	30
200_1	3652.20	154	3093.01	115	2009.59	33
200_2	3597.99	150	3437.79	125	2002.20	34
200_3	3592.74	152	3132.56	112	1588.98	27
300_1	4445.38	231	3653.64	165	1934.97	34
300_2	4498.58	233	4136.57	183	1898.96	33
300_3	4673.49	239	3990.55	170	1793.97	33
400_1	5110.49	311	4524.29	228	2237.95	35
400_2	5225.01	310	4744.41	248	2094.81	38
400_3	5227.94	314	4394.95	218	1957.95	30
500_1	5761.72	390	4534.93	257	1930.14	29
500_2	5953.15	398	5251.35	309	1897.32	36
500_3	5840.50	390	4944.21	269	1954.76	30

a dominating tree of G. Shin *et al.* [153] compared the performance of Shin_DT with MST-L. Next three subsections present our results. In the first subsection, we report the performance of Heu_DT along with Shin_DT and MST-L. In the second subsection, we report the performance of ABC_DT and ACO_DT. Finally, in the last subsection we present the overall picture.

5.5.1 Comparison of our Heu_DT with Shin_DT and MST-L

We have compared the performance of Heu_DT with Shin_DT and MST-L in terms of solution quality (value) and the number of dominating vertices (NDV). The number of dominating vertices also plays a crucial role since the performance of any routing protocols based on virtual backbone depends on the the number of dominating vertices. Table 5.1 reports for each test instance the results obtained by MST-L, Shin_DT and Heu_DT respectively. Table 5.1 clearly shows the superiority of Heu_DT over MST-L and Shin_DT in terms of solution quality as well as the number of dominating vertices on all test instances. On both parameters Heu_DT performs much better than the other two heuristics on all test instances. We have not reported running times of MST-L, Shin_DT and Heu_DT as all are less than a second. However, it is to be noted that

Table 5.2: Results of ABC_DT and ACO_DT for DTP

Instance	ABC_DT						ACO_DT					
	Value	Avg	SD	ANDV	TTB	TET	Value	Avg	SD	ANDV	TTB	TET
50_1	1204.41	1204.41	0.00	19.00	1.38	25.57	1204.41	1204.41	0.00	19.00	0.34	2.41
50_2	1340.44	1340.44	0.00	21.00	0.36	21.46	1340.44	1340.44	0.00	21.00	0.49	4.18
50_3	1316.39	1316.39	0.00	19.00	0.29	22.99	1316.39	1316.39	0.00	19.00	0.29	2.50
100_1	1217.47	1218.15	0.69	18.45	9.26	28.64	1217.47	1217.47	0.00	19.00	5.43	12.71
100_2	1128.40	1128.42	0.09	17.90	6.85	27.58	1152.85	1152.85	0.00	17.00	4.45	10.86
100_3	1252.99	1253.14	0.23	19.70	8.39	28.39	1253.49	1253.49	0.00	19.00	4.51	8.96
200_1	1206.79	1209.52	2.69	18.25	42.60	84.10	1206.79	1207.61	3.58	18.05	73.45	81.13
200_2	1216.41	1219.74	2.15	18.90	45.22	87.78	1216.23	1217.73	2.61	17.65	65.04	78.72
200_3	1253.02	1258.06	3.42	22.15	48.38	90.44	1247.25	1248.94	2.99	20.90	87.60	97.93
300_1	1229.97	1237.47	2.89	21.75	87.40	145.17	1228.24	1243.70	9.71	22.85	333.65	352.89
300_2	1182.52	1200.79	7.82	19.60	89.26	162.59	1176.45	1193.95	10.51	21.10	246.52	260.30
300_3	1257.21	1271.20	6.74	20.50	66.05	145.75	1261.18	1276.75	9.27	24.60	234.55	251.91
400_1	1223.61	1241.75	7.88	21.90	113.82	263.13	1220.62	1237.45	9.50	26.05	558.00	600.74
400_2	1220.54	1235.29	6.97	22.45	114.71	249.39	1209.69	1246.14	21.41	24.40	548.39	591.44
400_3	1266.41	1276.80	4.59	22.30	123.56	216.95	1254.10	1270.34	9.42	25.85	503.12	530.58
500_1	1233.14	1241.60	4.56	21.40	193.83	379.72	1219.66	1240.05	9.17	26.50	1079.34	1163.20
500_2	1245.59	1258.33	5.40	22.35	176.37	364.04	1273.86	1295.51	13.39	28.65	960.24	1031.81
500_3	1249.17	1278.67	11.96	21.60	190.57	338.25	1232.71	1259.08	20.03	24.35	871.53	917.73

Heu_DT is slightly slower than MST-L and Shin_DT. The running time of Heu_DT is mainly dominated by precomputing shortest paths between all pairs of vertices in G. But its better performance in terms of solution quality and the number of dominating vertices for each test instance compensates for its running time. MST-L is the fastest among all three heuristics, but it performs the worst.

5.5.2 Comparison between ABC_DT and ACO_DT Approaches

Table 5.2 reports for each test instance the best solution (value), average solution quality (Avg), standard deviation (SD) of solutions, average number of dominating vertices (ANDV), average time till best (TTB) and average total execution time (TET) obtained through ABC_DT and ACO_DT. This table shows that ACO_DT is better than ABC_DT in finding best solutions on 14 test instances out of 18 test instances. In terms of average solution quality, again ACO_DT is better than ABC_DT on 12 test instances out of 18 instances. However, it should be noted that the best as well as the average solution quality of each test instance obtained by ABC_DT are very close to that of ACO_DT. It is clear from the Table 5.2 that ABC_DT is much faster than ACO_DT especially on larger test instances. In terms of average number of dominating vertices, ABC_DT is better than ACO_DT on 13 test instances out of 18 test instances.

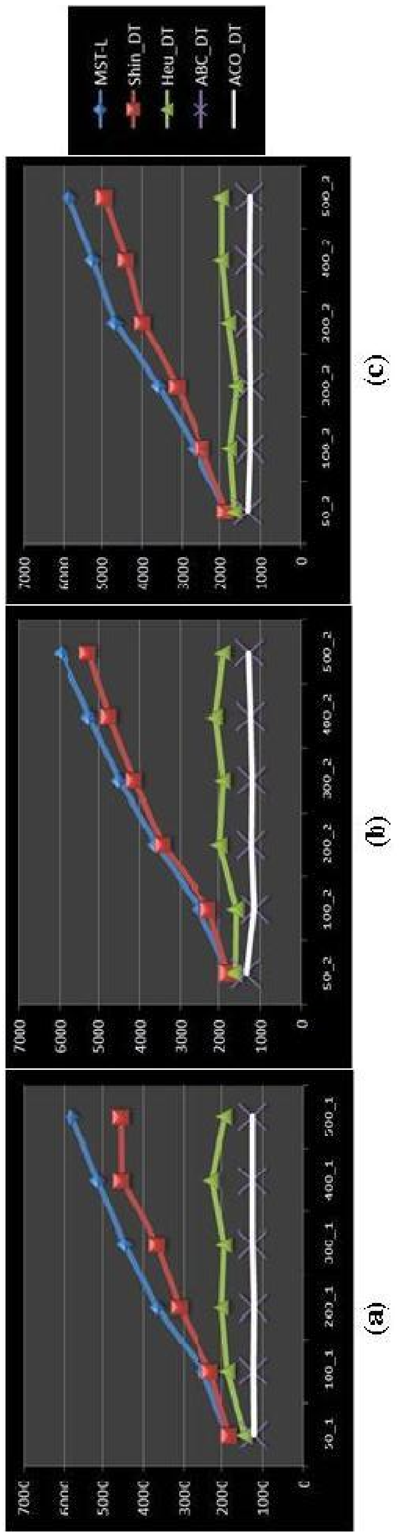


Figure 5.2: Comparison of average total cost of dominating tree obtained through various approaches

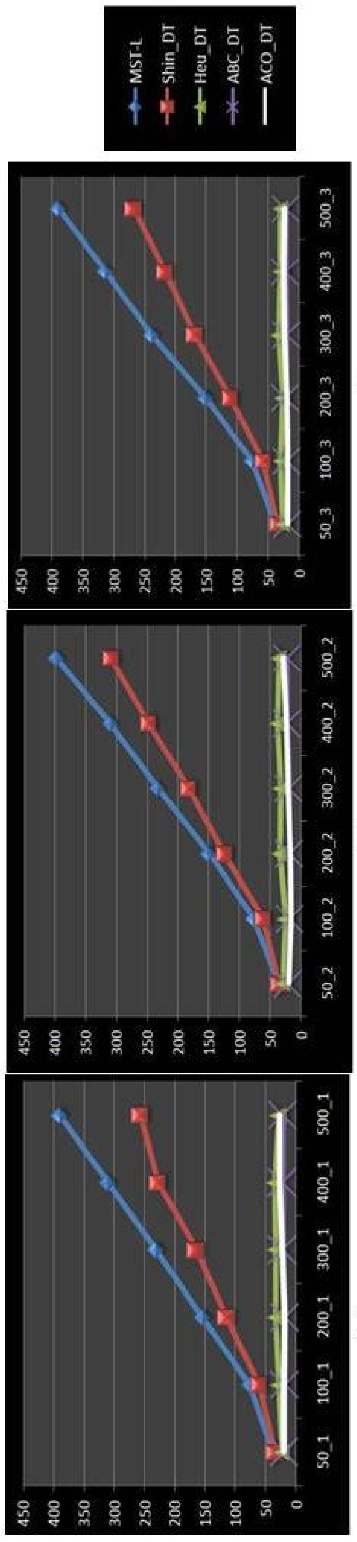


Figure 5.3: Comparison of average number of dominating vertices obtained through various approaches

5.5.3 The Overall Picture

This subsection presents the overall picture with respect to all the approaches considered in this chapter viz. MST-L, Shin_DT, Heu_DT, ABC_DT and ACO_DT for DTP. Figures 5.2 (a), 5.2 (b), 5.2 (c) graphically compare the average solution quality of all the approaches. As Heu_DT, Shin_DT and MST-L have been executed only once on each instance, therefore, their average solution quality is same as the solution obtained. In these figures, the y-axis represents the solution quality and the x-axis represents various instances. These figures show that the relative performance of Shin_DT and MST-L deteriorates sharply with the increase in instance size in comparison to Heu_DT, ABC_DT and ACO_DT. On larger instances, solution obtained by these heuristics are 2-3 times higher than Heu_DT and 4-5 times higher than ABC_DT and ACO_DT. Therefore, applicability of Shin_DT and MST-L is limited to only small instances. ABC_DT and ACO_DT have obtained solutions of best quality, but with a large execution time. If a solution to DTP is desired in a very short time, then Heu_DT is a good option as the solution, obtained by it, is always within a factor of two with respect to the corresponding solution obtained through ABC_DT and ACO_DT. Figures 5.3 (a), 5.3 (b), 5.3 (c) graphically compare the average number of dominating vertices in the solution obtained by all the approaches. Conclusions similar to Figures 5.2 (a), 5.2 (b), 5.2 (c) can be drawn here also.

It is to be noted that for our approaches, the number of dominating vertices in the solution and the solution quality do not vary significantly with instance size. This is also expected theoretically as all instances consist of vertices randomly distributed in a $500m \times 500m$ area. Therefore, with increase in instance size, the average degree of vertices also increases, and as a result, the number of dominating vertices in the solution and the solution quality do not vary significantly with the instance size.

5.6 Conclusions

In this chapter, we have proposed three approaches viz. Heu_DT, ABC_DT and ACO_DT for DTP. Heu_DT is a problem-specific heuristic and produces much better results in comparison with other problem-specific heuristics for DTP. With the intent of improving the solution quality even further, we have also presented two swarm intelligence based metaheuristic techniques viz. ABC_DT and ACO_DT which are respectively based on artificial bee colony algorithm and ant colony optimization algorithm. Performance of these metaheuristic techniques are comparable in terms of solution quality. Though ACO_DT produces slightly better results, it is several times slower than ABC_DT on large instances.

Chapter 6

Two Bounded-Degree Spanning Tree Problems

6.1 Introduction

Given a connected graph $G = (V, E)$, a vertex v of G is called a branch vertex if its degree is greater than two. Finding a spanning tree T of G with the minimum number of branch vertices (MBV) and finding a spanning tree T of G with the minimum degree sum of branch vertices (MDS) are among the several variants of the spanning tree problems which find applications in communication networks. This chapter is concerned with MBV and MDS, both of which have roots in optical networks.

In an optical network, Wavelength Division Multiplexing (WDM) technology allows the transmission of multiple light beams of different wavelength concurrently through the same optical fibre cable. Multicasting in these networks is implemented using a light splitting switch that replicates the input optical signal on multiple paths. A *lightpath* is an optical path (data channel) that connects two nodes in the network and is created by the allocation of the same wavelength throughout the path [157]. A *light-tree* [158], an extension of the light path, is capable of optical multicasting, i.e., it transmits a signal from a single source node to a set of destination nodes in an optical WDM network. Multicasting is required for efficient implementations of many services such as worldwide web browsing, video conferencing and video-on-demand services. Zhang *et al.* [159] showed that multicasting can be supported at the WDM layer by letting WDM switches make copies of data packets in the optical domain via light splitting. Thus, *light-tree* enables all optical communications from a source node to a set of destination nodes. Switches with splitting ability are placed on the branch nodes. Each node with splitting ability is capable to transmit a number of copies of the optical signal equal to its neighbors, while other nodes only transmit a copy of it. However, a WDM optical network can have only a limited number of these sophisticated switches due to cost considerations. Therefore, the problem reduces to finding a spanning tree of the optical network with the minimum number of branch nodes, where switches can be placed so that all possible multicast connections can be ensured in the optical network.

Gargano *et al.* [160] were the first to study the computational complexity of MBV. Let $s(G)$ be the smallest number of branch vertices in any spanning tree of G, then finding a spanning tree T of G with $s(G) = 0$ is same as finding a *Hamiltonian* path (a well known $\mathcal{NP}$-Complete problem) of G. A spanning tree T of G with $s(G) \leq 1$ is called a spanning spider. It is $\mathcal{NP}$-Complete to decide whether a graph G admits a spanning spider. More generally, the problem of deciding whether a given graph satisfies $s(G) \leq k$ is $\mathcal{NP}$-Complete problem, where k is a fixed non-negative integer. Therefore, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial time algorithms for such type of problems. Therefore, any exact method is not practical even for moderately large instances. However, Gargano *et al.* [160] proposed a polynomial time algorithm for MBV on a class of graphs satisfying certain density conditions. But in real scenarios, it seldom happens that the network satisfies these density conditions. Therefore, such an algorithm cannot be used in real situations and heuristics are the only available options.

Cerulli *et al.* [161] proposed three heuristics for MBV. They also introduced and studied MDS in the same paper. Actually, many optical devices are capable of only duplicating an optical signal. Therefore, in order to replicate the optical signal on each edge, the number of devices to be placed on a branch node depends on the number of edges incident to it, i.e., one has to place $C[v] - 2$ different devices on a branch node v, where $C[v]$ is the degree of v in the spanning tree. Therefore, in order to minimize the number of devices, one has to minimize the degree sum of branch vertices. This was the motivation behind introducing MDS by Cerulli *et al.* [161]. Let $q(G)$ be the minimum degree sum of branch vertices of an optimal solution of MDS. Cerulli *et al.* [161] proved that if $\mathcal{P} \neq \mathcal{NP}$, then there is no polynomial time algorithm to check whether $q(G) \leq k$, where k is a fixed positive number. They also showed that MBV and MDS are *related*, but not the *same*. The optimal solution value $q(G)$ of MDS cannot always be directly derived from the optimal solution value $s(G)$ of MBV. They also proposed three heuristics for MDS.

In this chapter, we propose a problem-specific heuristic and a hybrid ant colony optimization approach to solve MBV and also a heuristic and a hybrid ant colony optimization approach to solve MDS. A special feature of our ant colony optimization approaches is the use of two pheromones (one for vertices and another for edges). We have compared our approaches with those presented in [161]. Computational results demonstrate the effectiveness of our approaches.

The remainder of this chapter is organized as follows: Section 6.2 describes our problem-specific heuristic approach for MBV, whereas Section 6.3 describes our ant colony optimization (ACO) approach for MBV. Sections 6.4 and 6.5 respectively describe our problem-specific heuristic and ACO approach for MDS. Computational results are reported in Section 6.6. Finally, Section 6.7 contains some concluding remarks.

6.2 Heuristic for MBV

Our proposed heuristic for MBV (referred to as Heu_MBV) consists of two phases. It begins by selecting a vertex, say v_1, randomly and then it proceeds as follows:

Phase 1: It tries to repeatedly add an edge of G that can connect the last selected vertex to one of its adjacent unselected vertices of minimum degree into the partially constructed tree. If there are more than one such edges, then ties are broken arbitrarily. If there exists no such edge and if local degree of v_1 in the partially constructed tree is still one, then again it tries to find an edge using v_1 in place of last selected vertex. Note that this is a one time measure, because once this measure is used the degree of v_1 will become two. This procedure is repeated again and again till it fails to find an edge. Upon failure we enter phase 2.

Phase 2: If the set of unselected vertices is nonempty, then a branch vertex needs to be created from among the selected non-branch vertices. Among all the selected non-branch vertices, a vertex having maximum number of adjacent unselected vertices (ties are broken arbitrarily) is chosen to become a branch vertex. All edges connecting this newly created branch vertex to its set S of unselected adjacent vertices are added into the partially constructed tree. All vertices in S are now sorted according to non-decreasing order of their degrees in G, and then, *phase* 1 is applied (without going into *phase* 2 upon failure) by setting each vertex in S as the last selected vertex one-by-one. Once *phase* 1 has been applied to all vertices in S, then *phase* 2 restarts. This procedure is repeated again and again till the set of unselected vertices becomes empty.

Here, it is to be noted that by a selected vertex, we mean, a vertex one of whose incident edge has been included in the partially constructed tree, and by an unselected vertex, we mean, a vertex none of whose incident edges have been included in the partially constructed tree. Also note that if the spanning tree T of G is constructed without using phase 2, then this spanning tree T of G has no branch vertex and is actually a $Hamiltonian$ path of G. In *phase* 1, instead of selecting a vertex randomly, we always select a vertex of minimum degree. This is based on the fact that such a selection will direct the search process to a spanning tree with longer diameter which is expected to have lesser number of branch vertices. Same is the motivation behind ordering the vertices in S according to non-decreasing order of their degrees in *phase* 2. Similarly, in *phase* 2, a selected non-branch vertex with maximum number of adjacent unselected vertices is always chosen to become a branch vertex. Such a choice decreases the number of unselected vertices by maximum amount and gives *phase* 1 the maximum number of starting vertices to explore from. Both of these factors help in reducing the number of restarts of *phase* 2, thereby reducing the number of branch vertices. The idea of connecting all unselected vertices adjacent to the newly created branch vertex is based on commonsense. MBV heuristics of [161] also used this idea.

Algorithm 8 gives the pseudo-code of aforementioned heuristic for MBV (Heu_MBV), where for each vertex i, $C[i]$ represents the local degree of i in the tree. $C[i]$ should not be confused with the degree $deg[i]$ of i in G.

Figures 6.1 and 6.2 explain the proposed heuristic for MBV (Heu_MBV) with the help of an example. The input graph containing 20 vertices and 28 edges is shown in Figure 6.1 and the spanning tree obtained through MBV heuristic is given in Figure 6.2(a), where branch vertices are shown in light shade and edges are named according to the order in which they are added to the spanning tree, i.e., edge e_i is the i^{th} edge added to the tree. Heu_MBV begins by selecting a vertex randomly. Let us assume that it begins by selecting a vertex 15. Vertices 3, 4, 5, 7 are adjacent to the vertex 15 in the graph. Out of these adjacent vertices, vertex 7 has minimum degree 2 so edge

Algorithm 8: Heuristic for MBV

```
   input  : A connected graph $G = (V, E)$
   output: A spanning tree $T$
   begin
1      Initialize $T \leftarrow \emptyset$;
2      for (each vertex $i \in V$) do
3          $C[i] \leftarrow 0$;
4      Select a starting vertex $r \in V$ randomly;
5      $v_1 \leftarrow r$; $us \leftarrow |V| - 1$;
6      Find an unselected vertex $v_2$ of minimum degree adjacent to the vertex $v_1$;
7      if ($v_2$ exists) then
8          $T \leftarrow T \cup \{v_1, v_2\}$; $C[v_1]$++; $C[v_2]$++; $v_1 \leftarrow v_2$, $us$−−;
9          goto step 6;
10     else
11         Find an unselected vertex $v_2$ of minimum degree adjacent to the vertex $r$;
12         if ($v_2$ exists) then
13             $T \leftarrow T \cup \{r, v_2\}$; $C[r]$++; $C[v_2]$++; $r \leftarrow v_2$; $us$−−;
14             goto step 11;
15     while ($us \neq 0$) do
16         Select a vertex $v_3$ as a branch vertex among all selected vertices having
           maximum unselected vertices;
17         $S \leftarrow \emptyset$;
18         for (each unselected vertex $i$ adjacent to $v_3$) do
19             $S \leftarrow S \cup i$;
20         for (each vertex $i \in S$) do
21             $T \leftarrow T \cup \{v_3, i\}$; $C[v_3]$++; $C[i]$++; $us$−−;
22         sort all vertices $\in S$ into non_decreasing order based on their degrees in $G$;
23         for (each vertex $i \in S$) do
24             $v_1 \leftarrow i$;
25             Find an unselected vertex $v_2$ of minimum degree adjacent to the vertex $v_1$;
26             if ($v_2$ exists) then
27                 $T \leftarrow T \cup \{v_1, v_2\}$; $C[v_1]$++; $C[v_2]$++; $v_1 \leftarrow v_2$; $us$−−;
28                 goto step 25;
```

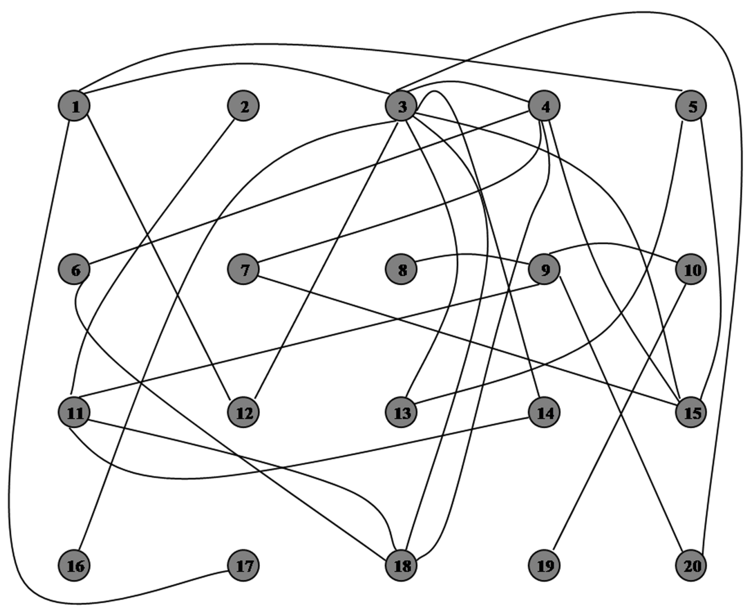


Figure 6.1: Input graph

(15, 7) is the first edge that is added to the tree (this is shown by e_1 in the Figure 6.2(a)). Now, vertex 7 is the vertex that is selected last so we have to look into its adjacent unselected vertices. Vertex 4 is only such vertex, and therefore, the edge (7, 4) is added to the tree (this is shown by e_2 in the Figure 6.2(a)). Continuing in this way, edges (4, 6), (6, 18), (18, 11), (11, 2) are added to the tree. Now, vertex 2 does not have any unselected adjacent vertices, so Heu_MBV restarts from the first selected vertex, i.e., vertex 15. Now vertex 5 is the lowest degree unselected adjacent vertex, so edge (15, 5) is added to the tree. Then edges (5, 13), (13, 3), (3, 16) are added into the tree. At this stage, we cannot proceed further using *phase* 1 of Heu_MBV, so we enter *phase* 2.

Among all the selected vertices, vertex 3 has the maximum number of unselected adjacent vertices, so vertex 3 is made a branch vertex and edges (3, 1), (3, 12), (3, 14), (3, 20) are added to the tree. Newly selected vertices 1, 12, 14 and 20 (set S) are sorted according to non-decreasing order of their degrees in G leading to order 12, 14, 20, 1.

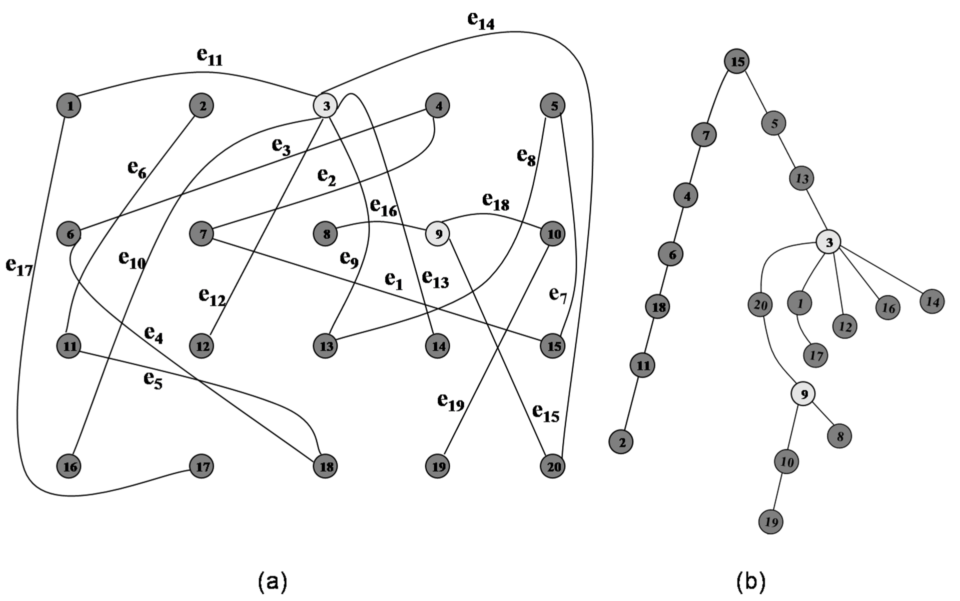


Figure 6.2: Output of MBV heuristic

Now, *phase* 1 of Heu_MBV is restarted by setting vertex 12 as the last selected vertex, but it cannot go further because there is no unselected vertex adjacent to the vertex 12. Same happens with vertex 14. Now, vertex 20 becomes the last selected vertex and *phase* 1 adds edges (20, 9) and (9, 8) to the tree. Once again, *phase* 1 starts with vertex 1 as the last selected vertex and only one edge (1, 17) is added to the tree. At this juncture, all vertices in S have been explored using *phase* 1, and therefore, *phase* 2 restarts again. This time vertex 9 is selected as a branch vertex and the edge (9, 10) is added to the tree. Again, we switch to *phase* 1 with vertex 10 as the last selected vertex. Now, *phase* 1 adds the edge (10, 19) to the tree and *phase* 2 restarts. Since there is no unselected vertex now, *phase* 2 ends and Heu_MBV stops. So, we get a spanning tree with two branch vertices which also happens to be the optimal number of branch vertices for this particular example. For the sake of clarity, the spanning tree of Figure 6.2(a) is redrawn in Figure 6.2(b).

6.3 ACO for MBV

It can be observed from the computational results (Section 6.6) that merely giving a heuristic approach to solve MBV is not enough to find even good solutions. To overcome this limitation, we attack MBV with an ant colony optimization (ACO) algorithm which is referred to as ACO_MBV subsequently in this chapter. It is in essence a $\mathcal{MMAS}$ algorithm [99, 100]. As MBV seeks a spanning tree with the minimum number of branch vertices, therefore, while solving this problem two factors play crucial roles in finding good solutions. First one which edge should be selected and second one which vertex should be selected as a branch vertex, if there exists any, while constructing a spanning tree. Our ACO_MBV takes care of these two factors by incorporating them in the search process. For this, our ACO_MBV is empowered through the use of two different pheromone laying procedures, i.e., one for the edges and other for the vertices of the graph. The motivation behind using two pheromones is that pheromone laid on the edges helps in identifying good edges in constructing a spanning tree, whereas pheromone laid on the vertices helps in choosing promising vertices as branch vertices. We have incorporated iteration best pheromone trail update strategy as well as global best pheromone trail update strategy. We have coupled ACO_MBV with a local search to further improve the solution quality.

6.3.1 Solution Construction

In our ACO_MBV approach, we have incorporated major ideas of heuristic of Section 6.2 in solution construction. During each iteration t, each ant k constructs a solution in a manner which is similar to heuristic of Section 6.2 except for the selection of an edge connecting the last selected vertex i to an unselected adjacent vertex and the selection of a branch vertex which are done probabilistically with the help of pheromones and heuristic information. The probability $p^k_{e_{ij}}(t)$ of selecting an edge e_{ij} is determined as follows:

$$p^k_{e_{ij}}(t) = \frac{[\tau_{e_{ij}}(t)]^{\alpha_e}[\eta_{e_{ij}}]^{\beta_e}}{\sum\limits_{l \in N^k_i} [\tau_{e_{il}}(t)]^{\alpha_e}[\eta_{e_{il}}]^{\beta_e}} \tag{6.1}$$

where $\eta_{e_{ij}} = \frac{1}{deg[j]}$ is a heuristic term that is available a priori and $\tau_{e_{ij}}$ is the pheromone trail on the edge e_{ij}. α_e and β_e are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and N^k_i is the set of unselected vertices which are adjacent to the last selected vertex i.

Similarly, the probability $p^k_{v_i}(t)$ of the selection of a vertex v_i to become a branch vertex is determined as follows:

$$p^k_{v_i}(t) = \frac{[\tau_{v_i}(t)]^{\alpha_v}[\eta_{v_i}]^{\beta_v}}{\sum\limits_{s \in S^k}[\tau_{v_s}(t)]^{\alpha_v}[\eta_{v_s}]^{\beta_v}} \tag{6.2}$$

where η_{v_i} is a heuristic term that is set equal to the number of unselected vertices adjacent to v_i and τ_{v_i} is the pheromone trail on vertex v_i. α_v and β_v are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and S^k is the set of those non-branch selected vertices which are candidates to become a branch vertex.

6.3.2 Pheromone Update

Pheromone update rule is used to augment the pheromone values on components that are present in high quality solutions and helps in guiding ants in future iterations towards better solutions. Once all ants have constructed their solutions in a particular iteration, say t, only the ant, which has constructed the iteration's best solution S^{ib}, is allowed to update the pheromone trails in the following way:

$$\tau_{e_{ij}}(t+1) = \rho\, \tau_{e_{ij}}(t) + \Delta\tau^{ib}_{e_{ij}} \tag{6.3}$$

$$\Delta\tau^{ib}_{e_{ij}} = \begin{cases} P, & \text{if } e_{ij} \in S^{ib}; \\ 0, & \text{otherwise.} \end{cases} \tag{6.4}$$

$$\tau_{v_i}(t+1) = \rho\, \tau_{v_i}(t) + \Delta\tau^{ib}_{v_i} \tag{6.5}$$

$$\Delta\tau^{ib}_{v_i} = \begin{cases} P, & \text{if } v_i \text{ is a branch vertex in } S^{ib}; \\ 0, & \text{otherwise.} \end{cases} \tag{6.6}$$

where ρ is persistence rate which is taken to be same for both edges and vertices of the graph G, $\tau_{e_{ij}}(t)$ is the pheromone value of the edge $e_{i,j}$ at iteration t, $\tau_{v_i}(t)$ is the pheromone value of the vertex v_i at iteration t. $\Delta\tau^{ib}_{e_{ij}}$ and $\Delta\tau^{ib}_{v_i}$ are pheromone augmentation terms. P is a parameter to be determined empirically.

Moreover, we augment pheromone on components of the global best solution (i.e., the overall best solution since the beginning of the ACO algorithm) S^{gb} once every GB_{it} iterations in the following way:

$$\tau_{e_{ij}} = \begin{cases} \tau_{e_{ij}} + Q, & \text{if } e_{ij} \in S^{gb} \text{ and either } i \text{ or } j \text{ is a branch vertex;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.7}$$

$$\tau_{v_i} = \begin{cases} \tau_{v_i} + Q, & \text{if } v_i \text{ is a branch vertex in } S^{gb}; \\ 0, & \text{otherwise.} \end{cases} \tag{6.8}$$

According to equation (6.7), a constant amount of pheromone Q is deposited on those edges of the global best solution S^{gb} whose at least one end point is a branch vertex, and according to the equation (6.8), the same amount of pheromone Q is deposited on the branch vertices of the global best solution S^{gb}. We have also tried with the strategy of augmenting the pheromone on every edge of the global best solution, but this strategy is always outperformed by the strategy of depositing pheromone on only those edges whose at least one end point is a branch vertex.

6.3.3 Local Search

The literature on the variants of ACO algorithms to optimization problems suggests that a high quality solution is usually obtained by coupling a local search with a probabilistic solution construction by an ant. In our algorithm, we also couple a local search, which is a greedy approach, with ACO_MBV to obtain high quality solutions. This algorithm is referred to as ACO_MBV+LS. The description of ACO_MBV+LS is as follows: once each ant has constructed a solution as per the procedure described in Section 6.3.1, then a local search is applied to further improve the quality of solution. The pheromone trail is updated with this improved solution.

The purpose of this local search is to reduce the number of branch vertices as much as possible. Once an ant has constructed a solution, we have complete information about the local degree $C[i]$ for each vertex i in the spanning tree. The local search begins by sorting the branch vertices of the spanning tree according to non-decreasing order of their local degrees. Each branch vertex i with $C[i] = 3$ or $C[i] = 4$ or $C[i] = 5$ is of concern to the local search. The local search contains three separate cases to handle these three classes of branch vertices. All these cases begin by deleting an edge e_{ir} of the spanning tree, where i is a branch vertex, thereby creating two components. Then an attempt is made to find an edge that can connect these components and is different from e_{ir}, so that the degree of i can be reduced by one. Obviously, endpoints v_1 and v_2 of an edge (v_1, v_2) in G that can connect these two components must satisfy

$C[v_1] < deg[v_1]$ and $C[v_2] < deg[v_2]$. This property is used to reduce the search space in all the three cases. Details of each of these cases are given below.

Case 1: This case tries to transform each branch vertex i with $C[i] = 3$ into a non-branch vertex. It consists of two procedures. If the first procedure (called $Procedure(1_a)$) is not successful in transforming a branch vertex i into a non-branch vertex, then we move to the second procedure (called $Procedure(1_b)$). In $Procedure(1_a)$, an edge e_{ir} connecting the branch vertex i and the vertex r is deleted, thereby creating the partition of the spanning tree into two components and making the solution infeasible. Let L_1 be the set of vertices belonging to the component containing i and L_2 be the set of vertices belonging to the component containing r. The values of $C[i]$ and $C[r]$ are updated to $C[i] - 1$ and $C[r] - 1$. To make this solution feasible again, an edge which can connect these two components and is different from e_{ir} is searched in G using the following rules:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 3))$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 3))$.

2. If first rule fails and if $(C[r] = 2)$, then find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 3))$ to the vertex $r \in L_2$.

If any one of these rules is satisfied, then an edge is found, which can connect these two components, and one branch vertex, i.e., vertex i is transformed into a non-branch vertex and the next branch vertex is considered. If both of these rules fail, then the spanning tree is restored by adding the deleted edge e_{ir} into the tree. $C[i]$ and $C[r]$ get their previous values and the next edge incident to the branch vertex i is chosen for deletion and $Procedure(1_a)$ is applied again.

If $Procedure(1_a)$ is not successful even after trying all edges incident to i, i.e., vertex i is still a branch vertex, then we move to $Procedure(1_b)$. Like $Procedure(1_a)$, $Procedure(1_b)$ deletes an edge e_{ir}. Then a search for another edge in G, which can make the solution feasible, is performed using the following rules:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $(C[v_2] = 3)$.

2. If first rule fails, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 3)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] = 3))$.

If the search is successful, then the branch vertex i becomes a non-branch vertex and the next branch vertex is considered, otherwise the spanning tree is restored like $Procedure(1_a)$ and the next edge incident to i is chosen for deletion. If $Procedure(1_b)$ fails on all edges incident to i, then i cannot be transformed into a non-branch vertex and the next branch vertex is considered.

Case 2: This case tries to transform each branch vertex i with $C[i] = 4$ into a non-branch vertex. It consists of a single procedure called $Procedure(2)$. Like $Procedure(1_a)$, $Procedure(2)$ deletes an edge e_{ir}. Then a search for another edge in G, which can make the solution feasible, is performed using the following rules:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 3))$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 3\ ))$.

2. If first rule fails and if $(C[r] = 2)$, then find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 3))$ to the vertex $r \in L_2$.

If the search is successful, then the searched edge is added to the partial spanning tree of the solution, which makes $C[i] = 3$, and subsequently the procedures of Case 1 are called for vertex i, otherwise the spanning tree is restored, and the next edge of the branch vertex i is tried like $Procedure(1_a)$. If $Procedure(2)$ fails on all edges incident to i, then the degree of i cannot be reduced and the next branch vertex is considered.

Case 3: This case considers each branch vertex i with $C[i] = 5$ and tries to transform it into a non-branch vertex. It consists of a single procedure called $Procedure(3)$. $Procedure(3)$ also deletes an edge e_{ir} and then searches for another edge in G, which can make the solution feasible, using the following two rules:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 4))$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 4))$.

2. If first rule fails and if $(C[r] = 4)$, then find an edge connecting a vertex $v_1 \in L_1$ with $((C[v_1] = 1)$ *or* $(C[v_1] > 4))$ to the vertex $r \in L_2$.

If the search is successful, then the searched edge is added to the partial spanning tree of the solution, which makes $C[i] = 4$, and subsequently $Procedure(2)$ of Case 2 is called for vertex i, otherwise the spanning tree of the solution is restored and the next edge is considered. If $Procedure(3)$ fails on all edges incident to i, then the degree of i cannot be reduced and the next branch vertex is considered.

We do not consider the branch vertices with local degree greater than 5, as limited computational experiments showed no significant improvements in results. Besides, computational cost of considering the branch vertices with local degree greater than 5 is relatively high.

6.4 Heuristic for MDS

It is known that MBV and MDS are related problems, but not the same. Therefore, our proposed heuristic for MDS (referred to as Heu_MDS) is similar to heuristic for MBV (Heu_MBV) and begins by selecting a vertex randomly. *Phase* 1 of Heu_MDS is exactly same as the *phase* 1 of Heu_MBV, however, *phase* 2 differs in the manner the set S of unselected vertices adjacent to the newly created branch vertex is processed. In *phase* 2 of Heu_MBV, whenever a vertex is made a branch vertex v_b, then all edges connecting v_b to vertices in set S are added to the partially constructed spanning tree. But this policy is not used in Heu_MDS. Instead, a vertex v_u having minimum degree (ties are broken arbitrarily) is selected from among the vertices in S and the edge connecting v_b and v_u is added into the partially constructed tree. After this, *phase* 1 is applied (without going into *phase* 2 upon failure) by setting v_u as the last selected vertex. S is re-computed to account for any change introduced due to the application of *phase* 1. Now, another vertex with minimum degree is chosen from S and processed in the same manner as v_u. *Phase* 2 restarts when S becomes empty. This new policy avoids unnecessary increasing the degree of branch vertices. However, when it has to choose between increasing the degree by one of a branch vertex and creating a new branch vertex, it prefers the former. MDS heuristics of [161] also avoid as far as possible increasing the degree of branch vertices.

Figure 6.3(a) shows the spanning tree obtained through Heu_MDS on the input graph of Figure 6.1. Heu_MDS also begins by selecting a vertex randomly. To understand the difference between heuristics for MBV and MDS, let us assume that Heu_MDS

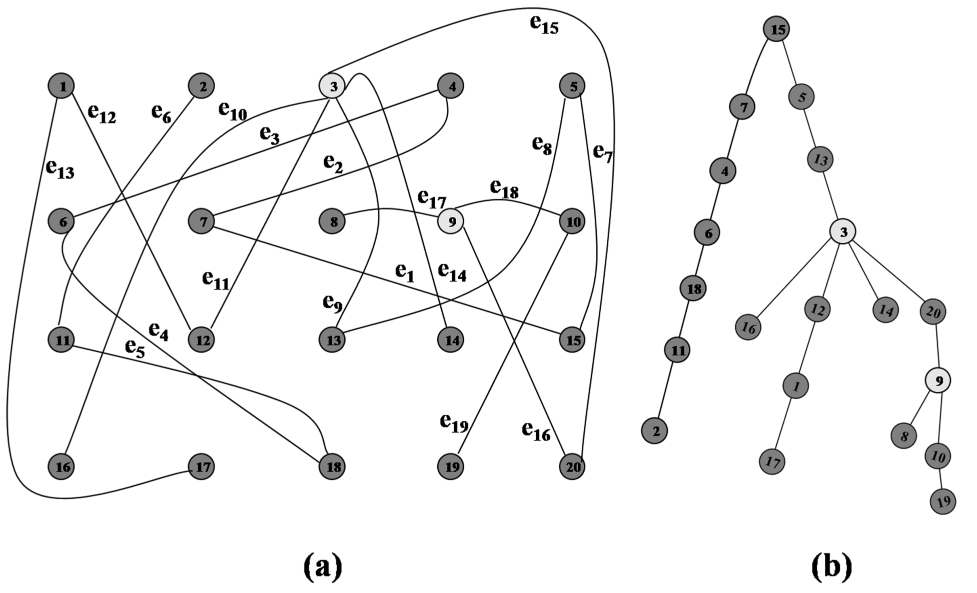


Figure 6.3: Output of MDS heuristic

also begins by selecting vertex 15. Exactly like Heu_MBV, edges (15, 7), (7, 4) (4, 6), (6, 18), (18, 11) , (11, 2), (15, 5), (5, 13), (13, 3) and (3, 16) are added to the tree. At this stage, we cannot proceed further using *phase* 1 of the heuristic, so we enter *phase* 2. Among all the selected vertices, vertex 3 has the maximum number of unselected adjacent vertices, so the vertex 3 is made a branch vertex. Beginning at this juncture, Heu_MDS proceeds in a way different from Heu_MBV. Vertices 1, 12, 14 and 20 are unselected vertices adjacent to the vertex 3. Of these vertices 12, 14 and 20 have the minimum degree 2. Breaking the tie arbitrarily and selecting a vertex 12 will add an edge (3, 12) to the tree. Now, *phase* 1 is restarted by setting vertex 12 as the last selected vertex. This leads to edges (12, 1), (1, 17) added to the tree and we return to *phase* 2. Next we break the tie between vertex 14 and vertex 20 by selecting the vertex 14 which leads to the addition of an edge (3, 14) to the tree. Now, *phase* 1 is called with vertex 14 as the last selected vertex, but there is no unselected vertex adjacent to the vertex 14, so we again return to *phase* 2. Next edge (3, 20) is added to the tree and *phase* 1 is called with vertex 20 as the last selected vertex. Now, edges (20, 9) and (9, 8) are added to the tree. Then *phase* 2 restarts again. This time vertex 9 is selected as a branch vertex and an edge (9, 10) is added to the tree. Again we call *phase* 1 with vertex 10 as the last selected vertex. Now, *phase* 1 adds an edge (10, 19) to the tree and *phase* 2 restarts. This time, since there is no unselected vertex, so the spanning tree is complete and heuristic stops. Note that Heu_MDS obtains a spanning tree with degree sum of branch vertices equal to 8, whereas the degree sum of branch vertices is 9 in the spanning tree obtained through Heu_MBV (see Figure 6.2(a)). Incidentally, in this particular example, the tree obtained through Heu_MDS is optimal. For the sake of clarity, the spanning tree of Figure 6.3(a) is redrawn in Figure 6.3(b).

6.5 ACO for MDS

Like ACO_MBV, we also attack MDS with $\mathcal{MMAS}$ algorithm [99, 100]. This algorithm is referred to as ACO_MDS in this chapter. ACO_MDS uses the same ideas as ACO_MBV, but only after suitable modifications. Therefore, in this section, we will describe only the differences and similarities between ACO_MDS and ACO_MBV. Like ACO_MBV, ACO_MDS also uses two types of pheromones and pheromones are updated using both iteration best and global best solutions. ACO_MDS is also coupled with a local search that is derived from the local search used in ACO_MBV.

6.5.1 Solution Construction

During each iteration t, each ant k constructs a solution in a manner which is similar to Heu_MDS of Section 6.4 except for the selection of an edge connecting the last selected vertex i to an unselected vertex and the selection of a branch vertex which are done probabilistically with the help of pheromones and heuristic information. The probability of selecting an edge and the probability of selecting a branch vertex are still determined using equations (6.1) and (6.2) respectively.

6.5.2 Pheromone Update

Pheromone update rules for ACO_MDS are exactly same as ACO_MBV.

6.5.3 Local Search

Local search tries to reduce the degree sum of branch vertices of the solution as much as possible. ACO_MDS with local search will be referred to as ACO_MDS+LS. Like the local search for MBV, here also branch vertices of the spanning tree are sorted according to non-decreasing order of their local degrees. Also branch vertices with

$C[i] = 3$ or $C[i] = 4$ or $C[i] = 5$ are considered one-by-one. Cases to handle each of these three classes of branch vertices are similar to respective cases of the local search for MBV, only the number of procedures and rules to select new edges differ to account for different structure of MDS. In addition, we have one more case to handle each branch vertex i with $C[i] > 5$. If a case has more than one procedure, then they are coupled in the same manner as in the case of the local search for MBV.

Case 1: This case is similar to Case 1 of the local search for MBV. Only the rules governing the selection of edges differ. This case consists of three procedures – $Procedure(1_a)$, $Procedure(1_b)$ and $Procedure(1_c)$. These procedures are applied one after the other till one of them finds a suitable edge or all of them fail to find a suitable edge. $Procedure(1_a)$ is similar to $Procedure(1_a)$ of the local search for MBV except it uses the following two rules for edge selection:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0))$.

2. If rule 1 fails and if $(C[r] = 2)$, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to the vertex $r \in L_2$.

$Procedure(1_b)$ is similar to $Procedure(1_b)$ of the local search for MBV except on following two points: first, upon failure of $Procedure(1_b)$ to find a suitable edge, here $Procedure(1_c)$ is called. Second, it uses the following rules for edge selection.

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $(C[v_2] > 2)$.

2. If first rule fails, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] > 2)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 2))$.

$Procedure(1_c)$ is similar to $Procedure(1_b)$ of the local search for MBV except the rule for edge selection is as follows.

1. If $(C[r] > 1)$, then find an edge connecting a vertex $v_2 \in L_2$ with $(C[v_2] = 1)$ to the vertex $i \in L_1$.

Case 2: This case considers each branch vertex i with $C[i] = 4$. It consists of two procedures viz. $Procedure(2_a)$ and $Procedure(2_b)$. $Procedure(2_a)$ is similar to $Procedure(2)$ of the local search for MBV except on the following points: first, upon finding a new edge, it calls procedures of Case 1 of MDS. Second, upon failure, $procedure(2_b)$ is called here. Third, following rules are used for edge selection:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0))$.

2. If rule 1 fails and if $(C[r] > 1)$, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to the vertex $r \in L_2$.

3. If rules 1 and 2 both fail and if $(C[r] > 1)$, then find an edge connecting a vertex $v_2 \in L_2$ with $(C[v_2] = 1)$ to the vertex $i \in L_1$.

$Procedure(2_b)$ is similar to $Procedure(2_a)$ except on two counts. First, upon failure, the degree of vertex i cannot be reduced. Second, it uses the following rules for edge selection:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $(C[v_2] > 3)$.

2. If rule 1 fails, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] > 3)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 3))$.

Case 3: This case considers each branch vertex i with $C[i] = 5$. It also consists of two procedures viz. $Procedure(3_a)$ and $Procedure(3_b)$. $Procedure(3_a)$ is similar to $Procedure(3)$ of MBV except on the following points: first, upon finding a new edge it calls procedures of Case 2 of MDS. Second, upon failure, $procedure(3_b)$ is called here. Third, following rules are used for edge selection:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1\ )$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0))$.

2. If rule 1 fails and if $(C[r] > 1)$, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to the vertex $r \in L_2$.

3. If rules 1 and 2 both fail and if $(C[r] > 1)$, then find an edge connecting a vertex $v_2 \in L_2$ with $(C[v_2] = 1)$ to the vertex $i \in L_1$.

$Procedure(3_b)$ is similar to $Procedure(3_a)$ except on two counts. First, upon failure, the degree of vertex i cannot be reduced. Second, it uses the following rules for edge selection:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $(C[v_2] > 4)$.

2. If rule 1 fails, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] > 4)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0)$ *or* $(C[v_2] > 4))$.

Case 4: This case has only one procedure viz. $Procedure(4)$, which considers each branch vertex i with $C[i] > 5$. Like other procedures, $Procedure(4)$ deletes an edge e_{ir} which makes the solution infeasible. To make this solution feasible again, an edge is searched in G, which can connect these two components, in the following way:

1. Find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to a vertex $v_2 \in L_2$ with $((C[v_2] = 1)$ *or* $(C[v_2] = 0))$.

2. If rule 1 fails and if $(C[r] > 1)$, then find an edge connecting a vertex $v_1 \in L_1$ with $(C[v_1] = 1)$ to the vertex $r \in L_2$.

3. If rules 1 and 2 both fail and if $(C[r] > 1)$, then find an edge connecting a vertex $v_2 \in L_2$ with $(C[v_2] = 1)$ to the vertex $i \in L_1$.

If the search is successful, then the searched edge is added to the partial spanning tree of the solution and the local degree of the branch vertex i or the branch vertex r (if r also happens to be a branch vertex) or both in the spanning tree is reduced by one, otherwise the spanning tree of the solution is restored by inserting edge e_{ir} and the search proceeds like other procedures.

6.6 Computational Results

Our approaches for MBV and MDS have been implemented in C and executed on a Linux based 3.2 GHz Pentium 4 system with 1 GB RAM. In all our computational experiments with ACO_MBV+LS and ACO_MDS+LS, we have used a colony of 25 ants, i.e., $n_a = 25$. We have set $\alpha_e = 2$, $\beta_e = 2$, $\alpha_v = 1$, $\beta_v = 2$, $\rho = 0.98$, $P = 0.1$, $Q = 0.1$ and $GB_{it} = 20$. All pheromone values are constrained in the interval [0.01, 5] and are initialized to 5 at the beginning. We have allowed ACO_MBV+LS and ACO_MDS+LS to execute for 3000 iterations. These approaches may terminate before 3000 iterations, if a solution without a branch vertex is found. All these parameter values are chosen empirically after a large number of trials. These parameter values provide good results though they may not be optimal for all instances.

As the instances that were used in [161] are unavailable, therefore, to compare our approaches with the heuristics of [161], we have generated problem instances by using the same three graph instance generators viz., *Netgen*, *Genmax* and *Random* as used in [161]. All these generators are available from DIMACS (`http://dimacs.rutgers.edu/`). *Netgen* and *Genmax* are network flow problem instance generators which generates problem instances with random edges and uniform capacity. Like [161], we have discarded the edge capacity and transformed the network from directed to undirected. For each generator, we have also taken 8 different values of $|V|$ into account, i.e., $|V| = 20, 30, 40, 50, 100, 300, 500, 1000$. For each value of $|V|$, we have taken 5 different values of density (the ratio of the number of edges to the number of vertices) into account, i.e., $d = 1.5, 2, 4, 10, 15$. However, we are not able to generate instances with $|V| = 20, d = 10; |V| = 20, d = 15; |V| = 30, d = 15$ using *Random* instance generator due to some internal restrictions in the generator program. Thus, we have 40 different combinations of V and d for each generator except *Random* for which we have 37 combinations. For each combination, a set of 5 instances were generated leading to overall 117 sets. Similar to [161], for each set we report the average of these 5 instances.

In addition to generating instances, for the purpose of comparison, we have reimplemented the best heuristics for MBV and MDS presented in [161] viz. Heuristic C.A. (combined approach of edge weighting and node coloring) for MBV and heuristic E.W. (edge weighting approach) for MDS. Moreover, in order to get exact solution or lower bound for each instance, the mathematical formulations presented in [161] for MBV and MDS are solved using Xpress-MP solver (http://www.dashoptimization.com/). Results of different approaches for MBV are reported in Tables 6.1, 6.2,6.3, whereas for MDS they are reported in Tables 6.4, 6.5, 6.6. For each instance set and each method, these tables report the average solution quality and the average execution time (AET) in seconds.

We have allowed Xpress-MP solver to execute for at most one hour time on each instance. When Xpress-MP solver cannot find a solution in one hour, we will get a lower bound on the solution. Such cases are reported in the tables with a '*' in the Value column and *dnf* (did not finish) in AET column.

In subsequent subsections, we compare the performance of different approaches for MBV and MDS.

6.6.1 Comparison of Our Approaches with C.A. Heuristic for MBV

We first compare Heu_MBV with C.A. heuristic [161]. Tables 6.1, 6.2, 6.3 clearly show the superiority of our Heu_MBV over C.A. heuristic in terms of both solution quality as well as execution time. Heu_MBV always performs as good as or better than C.A. heuristic in terms of both these parameters. Solutions obtained by Heu_MBV are better than C.A. heuristic on 113 instance sets and equal on 4 remaining sets. Heu_MBV is much faster than C.A. heuristic on larger instances.

However, comparing the results obtained by Heu_MBV with those of Xpress-MP solver, clearly show that Heu_MBV, in itself, is not so robust. Moreover, the difference in solution quality grows with increase in instances size. Therefore, to get good solutions even on larger instances, we have proposed ACO_MBV+LS as described in Section 6.3. ACO_MBV+LS is able to find 94 optimal solutions out of a total of 100 instance sets for which we know the optimal solutions. We have idea about the optimal solution values of only 100 instance sets, since Xpress-MP solver is able to find 100 optimal solutions out of 117 instance sets, therefore, results of ACO_MBV+LS are quite good.

To see the effect of local search, we also run our ACO approach without local search (referred to as ACO_MBV), while keeping all ACO parameters unchanged. ACO_MBV is able to find optimal solutions for 90 sets in comparison to 94 sets for ACO_MBV+LS. Overall, on all 117 sets, the results of ACO_MBV+LS is better than ACO_MBV on 26 sets, whereas it is worse on one set. As expected, ACO_MBV is much faster than ACO_MBV+LS.

Since we have used two types of pheromone (pheromone on edges and pheromone on vertices) which is different from the traditional way of using only one pheromone. To our knowledge, [162] is the only work where the concept of multiple pheromones was

Table 6.1: Results of MBV on Genmax Instances

Instance		**XPress MP**		**C.A.**		**Heu_MBV**		**ACO_MBV-EP**		**ACO_MBV**		**ACO_MBV+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	1.40	0.09	3.00	0.00	2.20	0.00	1.40	0.25	1.40	0.26	1.40	0.65
20	2	0.60	0.10	2.20	0.00	1.00	0.00	0.60	0.13	0.60	0.22	0.60	0.30
20	4	0.00	0.10	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	10	0.00	0.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	15	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	2.40	0.18	5.20	0.00	3.40	0.00	2.40	0.49	2.40	0.56	2.40	1.47
30	2	0.60	0.40	3.60	0.00	1.60	0.00	0.60	0.33	0.60	0.32	0.60	0.65
30	4	0.40	0.41	2.80	0.00	1.40	0.00	0.40	0.19	0.40	0.20	0.40	0.39
30	10	0.00	0.83	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	15	0.00	0.51	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	2.00	0.31	5.60	0.00	3.00	0.00	2.20	0.72	2.00	0.63	2.00	1.98
40	2	0.60	0.47	4.20	0.00	1.60	0.00	0.60	0.34	0.60	0.43	0.60	1.71
40	4	0.00	0.59	1.80	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	0.88	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	1.60	0.80	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	3.20	0.70	8.00	0.00	5.00	0.00	3.40	0.96	3.20	1.07	3.20	2.67
50	2	0.80	0.71	6.00	0.00	2.20	0.00	1.00	0.93	0.80	0.83	0.80	1.07
50	4	0.00	0.83	3.20	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	1.51	1.40	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	1.68	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	5.80	12.11	15.20	0.00	9.80	0.00	6.40	3.00	6.20	2.66	6.00	9.51
100	2	2.00	11.22	12.60	0.00	4.80	0.00	2.20	2.87	2.00	2.72	2.00	4.43
100	4	0.00	4.78	6.00	0.00	1.80	0.00	0.00	0.01	0.00	0.05	0.00	0.01
100	10	0.00	14.42	2.40	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	13.66	1.20	0.01	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	18.20*	dnf	47.80	0.00	32.80	0.00	25.00	24.91	21.20	19.08	20.40	74.15
300	2	4.80*	dnf	34.80	0.01	17.00	0.00	11.80	22.49	7.40	15.59	6.80	38.94
300	4	0.00	92.09	18.00	0.03	4.80	0.00	1.20	14.80	0.40	10.73	0.20	12.71
300	10	0.00	554.34	8.40	0.11	2.20	0.00	0.40	8.57	0.00	3.49	0.00	0.58
300	15	0.00	1345.46	5.80	0.13	1.60	0.00	0.00	0.07	0.00	0.29	0.00	0.03
500	1.5	28.80*	dnf	79.40	0.02	53.40	0.00	45.00	59.58	35.00	58.64	33.40	251.31
500	2	7.60*	dnf	58.60	0.05	26.20	0.00	21.40	58.48	11.80	47.57	10.60	99.51
500	4	0.20	1162.97	28.60	0.09	9.00	0.00	4.20	45.46	0.60	26.80	0.60	49.57
500	10	0.00	2537.12	14.40	0.24	3.20	0.00	1.20	33.30	0.00	12.91	0.00	15.02
500	15	0.00	2728.29	7.60	0.37	2.20	0.00	0.40	23.03	0.00	10.28	0.00	5.87
1000	1.5	59.00*	dnf	159.80	0.13	106.40	0.00	95.20	291.69	78.20	270.74	72.20	1289.14
1000	2	16.60*	dnf	115.20	0.18	56.80	0.00	50.20	245.81	28.40	175.24	24.60	485.64
1000	4	0.60	3410.95	60.60	0.40	16.60	0.00	12.20	145.52	1.20	102.42	1.00	203.92
1000	10	0.00*	dnf	22.20	1.02	6.60	0.00	3.40	122.06	0.00	53.21	0.00	88.37
1000	15	0.00*	dnf	15.00	1.66	4.80	0.00	2.40	113.32	0.00	55.89	0.00	57.95

Table 6.2: Results of MBV on Netgen Instances

Instance		**XPress MP**		**C.A.**		**Heu_MBV**		**ACO_MBV-EP**		**ACO_MBV**		**ACO_MBV+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	0.00	0.05	2.40	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	0.00	0.11	1.40	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	4	0.00	0.16	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	10	0.00	0.20	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	15	0.00	0.32	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	0.00	0.09	3.20	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	2	0.00	0.17	2.60	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	4	0.00	0.42	1.20	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	10	0.00	0.67	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	15	0.00	0.70	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	0.00	0.20	4.40	0.00	1.60	0.00	0.20	0.09	0.00	0.00	0.00	0.00
40	2	0.00	0.40	3.00	0.00	1.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	4	0.00	0.68	2.40	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	1.03	1.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	1.22	1.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	0.00	0.19	4.00	0.00	1.20	0.00	0.00	0.01	0.00	0.00	0.00	0.00
50	2	0.00	0.56	4.40	0.00	2.20	0.00	0.00	0.02	0.00	0.00	0.00	0.00
50	4	0.00	1.09	2.40	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	1.47	1.40	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	1.24	1.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	0.00	1.09	10.00	0.00	3.80	0.00	0.40	0.67	0.00	0.06	0.00	0.18
100	2	0.00	2.75	9.20	0.00	4.40	0.00	0.20	0.80	0.00	0.18	0.00	0.36
100	4	0.00	4.19	5.20	0.00	2.00	0.00	0.20	0.77	0.00	0.05	0.00	0.02
100	10	0.00	8.95	2.60	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	11.53	2.00	0.01	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	0.00	8.24	26.20	0.00	8.80	0.00	3.80	12.28	0.20	2.17	0.00	5.01
300	2	0.00	21.96	24.40	0.00	11.20	0.00	4.40	18.56	0.20	3.97	0.00	10.63
300	4	0.00	104.53	17.80	0.03	4.80	0.00	1.60	16.56	0.00	3.78	0.00	9.66
300	10	0.00	504.14	7.80	0.10	1.80	0.00	0.00	0.94	0.00	2.45	0.00	0.36
300	15	0.00	1675.10	7.00	0.14	1.20	0.00	0.00	0.06	0.00	0.04	0.00	0.01
500	1.5	0.00	22.10	45.80	0.03	18.40	0.00	9.00	27.51	1.00	15.25	0.40	25.07
500	2	0.00	100.59	43.40	0.06	19.20	0.00	11.20	48.86	0.60	19.24	0.20	43.24
500	4	0.00	646.94	26.40	0.11	9.40	0.00	4.60	44.12	0.20	18.90	0.00	29.48
500	10	0.00	2101.29	15.00	0.23	3.20	0.00	0.80	26.67	0.00	11.69	0.00	11.47
500	15	0.00	3376.00	12.00	0.38	2.20	0.00	0.20	10.17	0.00	9.56	0.00	2.11
1000	1.5	0.00	121.11	96.80	0.10	36.00	0.00	23.60	133.90	2.20	58.10	1.40	155.71
1000	2	0.00	934.20	85.60	0.17	34.20	0.01	27.40	167.79	1.60	86.61	0.60	213.57
1000	4	0.00	3375.60	50.40	0.39	17.40	0.01	13.00	156.24	0.40	85.30	0.00	160.54
1000	10	0.00*	dnf	30.60	1.06	5.20	0.00	2.80	110.05	0.00	56.06	0.00	86.81
1000	15	0.00*	dnf	22.80	1.57	4.40	0.00	1.60	117.65	0.00	60.81	0.00	57.13

Table 6.3: Results of MBV on Random Instances

Instance		**XPress MP**		**C.A.**		**Heu_MBV**		**ACO_MBV-EP**		**ACO_MBV**		**ACO_MBV+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	1.00	0.12	2.40	0.00	1.80	0.00	1.00	0.19	1.00	0.21	1.00	0.55
20	2	0.00	0.13	2.00	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	4	0.00	0.16	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	2.60	0.32	5.00	0.00	3.20	0.00	2.60	0.48	2.60	0.62	2.60	1.72
30	2	0.20	0.13	3.20	0.00	1.00	0.00	0.20	0.08	0.20	0.10	0.20	0.21
30	4	0.00	0.48	1.40	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	10	0.00	1.05	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	3.20	0.45	6.00	0.00	4.60	0.00	3.20	0.78	3.20	0.81	3.20	2.48
40	2	1.20	0.76	4.40	0.00	2.20	0.00	1.40	0.76	1.20	0.84	1.20	1.45
40	4	0.00	0.70	2.40	0.00	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	0.99	0.40	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	2.59	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	3.00	0.60	7.60	0.00	5.20	0.00	3.60	1.08	3.00	0.97	3.00	2.64
50	2	1.00	1.26	4.80	0.00	2.00	0.00	1.00	0.90	1.00	0.97	1.00	1.57
50	4	0.00	0.09	2.80	0.00	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	2.27	0.80	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	3.11	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	6.60	11.41	16.20	0.00	11.00	0.00	7.20	3.25	6.80	2.97	6.60	10.38
100	2	2.00	69.32	11.00	0.00	5.00	0.00	2.40	2.82	2.00	2.62	2.20	4.11
100	4	0.00	7.03	5.00	0.00	1.40	0.00	0.00	0.01	0.00	0.04	0.00	0.01
100	10	0.00	13.60	2.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	25.65	1.00	0.01	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	18.00*	dnf	49.00	0.00	31.80	0.00	25.00	22.79	21.00	21.40	20.20	89.76
300	2	5.20	3424.07	35.60	0.01	17.00	0.00	11.80	22.56	7.40	18.15	6.60	46.09
300	4	0.00	99.72	16.60	0.03	4.60	0.00	1.20	15.79	0.00	4.43	0.00	6.58
300	10	0.00	1104.37	7.40	0.08	2.80	0.00	0.60	10.63	0.00	3.17	0.00	0.71
300	15	0.00	1307.45	5.80	0.14	1.40	0.00	0.20	3.43	0.00	0.80	0.00	0.05
500	1.5	28.00*	dnf	78.20	0.02	51.60	0.00	43.20	71.81	34.60	60.86	32.60	250.44
500	2	8.60*	dnf	57.80	0.03	28.80	0.00	23.20	52.17	13.80	42.68	11.80	124.08
500	4	0.20	1549.24	28.80	0.11	7.60	0.00	4.20	45.90	0.80	30.11	0.60	50.43
500	10	0.00	3216.41	11.80	0.27	4.20	0.00	1.40	36.55	0.00	10.73	0.00	16.12
500	15	0.00	2900.89	8.40	0.42	2.60	0.00	1.00	38.12	0.00	10.55	0.00	3.78
1000	1.5	58.00*	dnf	151.20	0.11	104.40	0.00	94.60	265.30	76.00	222.84	70.20	1371.29
1000	2	15.40*	dnf	115.00	0.17	55.20	0.01	50.00	235.61	27.40	186.69	23.20	509.28
1000	4	0.20	3446.78	60.40	0.37	14.40	0.00	12.80	171.65	0.80	100.47	0.60	206.81
1000	10	0.00*	dnf	23.20	1.24	7.20	0.00	4.00	113.99	0.00	47.05	0.00	75.68
1000	15	0.00*	dnf	16.60	2.09	5.00	0.00	2.00	106.29	0.00	52.47	0.00	71.28

used. To support our approach, we have also experimented with a single pheromone version of our ACO algorithm where we have deposited pheromone only on vertices. In this situation, instead of selecting an edge e_{ij} connecting the last selected vertex i in the partially constructed tree to an unselected adjacent vertex j based on pheromone concentration on edges, an edge e_{ij} is selected exactly like *phase* 1 of Heu_MBV. The resulting approach is referred to as ACO_MBV-EP. It is clear from Tables 6.1 6.2 6.3 that the results obtained by ACO_MBV-EP are worse or equal to ACO_MBV on all instances. There are 55 instances where ACO_MBV-EP performs worse than ACO_MBV. Most of these instances are large instances. Even AETs of ACO_MBV-EP on most of the instance sets are higher than ACO_MBV. Hence, this experimentation provides the justification for using two pheromones instead of one in our ACO approach for MBV.

As far as comparison of solution quality of different ACO variants with Heu_MBV and C.W. heuristic is concerned, all ACO variants obtain better results except on few smaller instances where results are same.

6.6.2 Comparison of Our Approaches with E.W. Heuristic for MDS

We first compare Heu_MDS with E.W. heuristic [161]. Tables 6.4, 6.5, 6.6 clearly show the superiority of our Heu_MDS over E.W. heuristic in terms of solution quality as well as execution time. Solutions obtained by Heu_MDS are better than E.W. heuristic on 98 sets, worse on 18 sets and equal on 1 set. AET for Heu_MDS is as good as or better than E.W. heuristic. On larger instances Heu_MDS is much faster than E.W. heuristic.

Similar to ACO_MBV+LS, ACO_MBV and ACO_MBV-EP, here also we have three ACO variants viz. ACO_MDS+LS, ACO_MDS and ACO_MDS-EP. ACO_MDS+LS and ACO_MDS are able to find 94 and 91 optimal results respectively. There are 102 instance sets for which Xpress-MP solver is able to find optimal results, so again ACO_MDS+LS and ACO_MDS perform quite well. Overall, there are 24 instances on which ACO_MDS+LS is better than ACO_MDS. Here also single pheromone version ACO_MDS-EP performs worse both in terms of solution quality and execution time, thereby justifying once again the use of two pheromones. There are 61 instances where ACO_MDS-EP performs worse than ACO_MDS. Similar to MBV, here also all ACO variants obtain as good as or better quality solutions in comparison to Heu_MDS and E.W. heuristic.

Table 6.4: Results of MDS on Genmax Instances

Instance		**XPress MP**		**E.W.**		**Heu_MDS**		**ACO_MDS-EP**		**ACO_MDS**		**ACO_MDS+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	5.20	0.06	9.60	0.00	9.60	0.00	5.40	0.26	5.20	0.28	5.20	0.98
20	2	2.00	0.20	4.80	0.00	4.00	0.00	2.00	0.15	2.00	0.20	2.00	0.52
20	4	0.00	0.16	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	10	0.00	0.21	5.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	15	0.00	0.32	5.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	9.20	0.27	15.80	0.00	14.00	0.00	9.40	0.48	9.20	0.53	9.20	2.31
30	2	2.60	0.35	9.00	0.00	7.80	0.00	2.80	0.23	2.60	0.37	2.60	0.77
30	4	1.80	0.23	7.80	0.00	5.60	0.00	1.80	0.16	1.80	0.24	1.80	0.48
30	10	0.00	0.64	9.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	15	0.00	0.65	7.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	8.20	0.25	18.60	0.00	18.20	0.00	8.60	0.67	8.20	0.66	8.20	2.27
40	2	1.80	25.73	9.60	0.00	7.60	0.00	1.80	0.36	1.80	0.41	1.80	0.97
40	4	0.00	0.56	9.00	0.00	1.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	1.05	13.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	1.68	12.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	14.20	0.87	25.60	0.00	27.00	0.00	14.40	1.00	14.40	1.11	14.20	5.05
50	2	4.00	19.17	15.00	0.00	16.20	0.00	5.20	0.95	4.00	0.74	4.00	1.60
50	4	0.00	0.80	7.20	0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	1.77	15.00	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	1.79	14.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	28.60	27.25	55.00	0.00	54.00	0.00	30.20	2.96	29.60	2.81	28.80	15.58
100	2	10.80	738.78	29.40	0.00	34.40	0.00	13.20	2.60	11.20	2.68	11.20	8.25
100	4	0.00	3.33	21.00	0.00	10.40	0.00	0.00	0.00	0.00	0.05	0.00	0.02
100	10	0.00	9.05	22.20	0.01	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	6.60	25.80	0.03	1.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	94.40*	dnf	174.60	0.01	180.60	0.00	114.80	21.70	104.40	18.82	99.20	172.70
300	2	35.80*	dnf	103.20	0.01	107.80	0.00	59.20	19.81	41.60	19.02	40.20	83.19
300	4	0.00	23.91	58.20	0.05	24.80	0.00	7.00	9.07	0.00	5.59	0.00	13.15
300	10	0.00	110.05	76.40	0.15	14.80	0.00	0.60	5.04	0.00	2.56	0.00	0.74
300	15	0.00	276.30	88.20	0.22	9.40	0.00	0.00	0.16	0.00	0.93	0.00	0.04
500	1.5	153.80*	dnf	292.40	0.03	297.40	0.00	200.40	67.56	172.20	55.69	164.80	533.67
500	2	59.60*	dnf	169.40	0.06	185.20	0.00	104.80	52.48	72.00	42.74	66.80	210.87
500	4	0.80	910.99	88.80	0.14	58.00	0.00	19.80	34.67	1.00	20.80	1.00	45.98
500	10	0.00	1573.46	146.60	0.39	27.00	0.00	5.80	30.61	0.00	10.82	0.00	16.18
500	15	0.00	1198.72	130.00	0.65	16.60	0.00	1.20	24.89	0.00	11.58	0.00	4.00
1000	1.5	320.60*	dnf	606.40	0.15	603.20	0.01	427.40	253.48	375.00	250.94	352.80	2810.33
1000	2	128.00*	dnf	353.60	0.22	386.80	0.03	236.20	234.31	167.00	188.31	151.20	1116.61
1000	4	2.00*	dnf	183.60	0.49	118.60	0.01	54.20	160.32	3.20	97.76	2.60	273.84
1000	10	0.00	2027.56	272.20	1.78	51.40	0.01	20.00	121.72	0.00	44.99	0.00	109.59
1000	15	0.00*	dnf	297.40	2.92	39.20	0.01	12.60	102.66	0.00	51.94	0.00	90.03

Table 6.5: Results of MDS on Netgen Instances

Instance		**XPress MP**		**E.W.**		**Heu_MDS**		**ACO_MDS-EP**		**ACO_MDS**		**ACO_MDS+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	0.00	0.02	1.80	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	2	0.00	0.09	2.40	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	4	0.00	0.06	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	10	0.00	0.29	3.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	15	0.00	0.25	5.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	0.00	0.10	5.40	0.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	2	0.00	0.20	8.40	0.00	4.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	4	0.00	0.28	3.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	10	0.00	0.75	7.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	15	0.00	0.39	8.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	0.00	0.12	9.60	0.00	5.00	0.00	0.60	0.11	0.00	0.00	0.00	0.00
40	2	0.00	0.42	9.00	0.00	6.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	4	0.00	0.44	6.60	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	1.07	10.20	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	1.78	11.40	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	0.00	0.15	9.00	0.00	4.20	0.00	0.00	0.00	0.00	0.00	0.00	0.01
50	2	0.00	0.46	10.20	0.00	8.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	4	0.00	0.89	11.40	0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	2.22	12.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	1.13	11.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	0.00	0.85	15.00	0.00	13.60	0.00	0.60	0.43	0.00	0.06	0.00	0.18
100	2	0.00	1.79	22.20	0.00	15.20	0.00	0.60	0.60	0.00	0.15	0.00	0.32
100	4	0.00	3.37	19.20	0.00	13.00	0.00	0.60	0.78	0.00	0.06	0.00	0.01
100	10	0.00	9.00	27.00	0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	22.35	27.60	0.02	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	0.00	9.35	52.80	0.00	44.20	0.00	14.00	10.82	0.60	2.32	0.00	7.18
300	2	0.00	39.55	54.60	0.01	52.80	0.00	17.80	15.09	0.00	3.43	0.00	12.72
300	4	0.00	73.84	66.00	0.03	32.00	0.00	6.80	16.34	0.00	3.89	0.00	10.04
300	10	0.00	47.79	82.80	0.15	13.20	0.01	0.60	2.62	0.00	3.14	0.00	0.42
300	15	0.00	187.73	53.40	0.20	7.20	0.00	0.00	0.12	0.00	0.27	0.00	0.02
500	1.5	0.00	18.55	82.20	0.03	80.00	0.00	30.40	30.08	2.20	12.55	0.60	24.18
500	2	0.00	252.68	96.60	0.06	88.40	0.00	42.60	44.12	0.00	13.61	0.00	45.12
500	4	0.00	210.41	97.20	0.14	62.20	0.00	20.40	38.80	0.00	14.09	0.00	44.46
500	10	0.00	1539.98	131.80	0.39	24.80	0.00	3.20	28.68	0.00	10.83	0.00	14.49
500	15	0.00	882.76	129.40	0.56	14.00	0.00	0.80	12.19	0.00	9.79	0.00	2.14
1000	1.5	0.00	233.06	168.00	0.13	151.00	0.01	80.60	115.17	6.40	63.35	3.00	151.71
1000	2	0.00	2021.92	180.60	0.22	187.20	0.03	101.40	180.67	2.00	71.09	0.60	223.90
1000	4	0.00	2764.78	204.60	0.52	119.60	0.02	53.80	159.24	0.00	63.20	0.00	182.61
1000	10	0.00	905.48	274.80	1.72	40.60	0.01	13.20	92.35	0.00	57.91	0.00	96.96
1000	15	0.00	3140.60	282.00	2.82	33.60	0.01	8.60	103.33	0.00	45.86	0.00	68.12

Table 6.6: Results of MDS on Random Instances

Instance		**XPress MP**		**E.W.**		**Heu_MDS**		**ACO_MDS-EP**		**ACO_MBV**		**ACO_MDS+LS**	
V	d	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET	Value	AET
20	1.5	3.40	0.10	6.60	0.00	6.20	0.00	3.40	0.19	3.40	0.22	3.40	0.66
20	2	0.00	0.07	1.80	0.00	2.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	4	0.00	0.06	3.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	1.5	10.20	0.32	16.00	0.00	15.60	0.00	10.40	0.48	10.20	0.59	10.20	2.33
30	2	0.60	0.54	6.60	0.00	3.80	0.00	0.60	0.08	0.60	0.10	0.60	0.29
30	4	0.00	0.22	5.40	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	10	0.00	0.93	8.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	1.5	13.20	0.68	23.60	0.00	23.80	0.00	13.40	0.71	13.20	0.88	13.20	4.05
40	2	4.80	36.44	14.40	0.00	11.60	0.00	5.00	0.76	4.80	0.73	4.80	2.08
40	4	0.00	0.60	7.20	0.00	1.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	10	0.00	1.00	14.40	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	15	0.00	2.73	14.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.5	13.40	1.31	25.80	0.00	26.00	0.00	15.40	0.99	13.60	1.03	13.40	4.20
50	2	3.60	2.13	15.60	0.00	12.00	0.00	4.20	0.85	3.60	0.89	3.60	2.08
50	4	0.00	0.60	9.60	0.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	0.00	2.05	12.00	0.00	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	15	0.00	2.38	10.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	1.5	30.60	110.73	59.40	0.00	59.20	0.00	32.40	3.15	31.40	3.14	30.80	18.94
100	2	11.20	842.16	36.60	0.00	29.80	0.00	13.00	2.36	11.80	2.56	11.40	6.97
100	4	0.00	3.67	19.20	0.00	8.40	0.00	0.00	0.01	0.00	0.04	0.00	0.01
100	10	0.00	7.76	27.00	0.01	2.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100	15	0.00	13.12	34.80	0.03	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00
300	1.5	91.00*	dnf	171.20	0.01	181.80	0.00	111.80	23.52	98.80	22.32	96.20	171.48
300	2	33.20*	dnf	92.40	0.02	114.40	0.00	55.00	19.62	39.80	16.24	37.20	76.63
300	4	0.00	37.83	51.60	0.06	29.60	0.00	5.60	12.91	0.00	3.57	0.00	10.56
300	10	0.00	50.23	77.20	0.15	18.00	0.00	1.80	9.89	0.00	3.27	0.00	0.67
300	15	0.00	471.79	90.80	0.23	8.20	0.00	0.60	5.31	0.00	1.06	0.00	0.06
500	1.5	151.80*	dnf	291.60	0.03	306.00	0.00	197.60	60.72	169.20	56.56	161.60	558.23
500	2	63.60*	dnf	174.60	0.04	190.80	0.00	111.60	60.12	76.00	47.30	73.00	248.26
500	4	0.40	1057.76	88.80	0.14	49.20	0.00	20.20	36.72	1.20	23.14	0.60	52.61
500	10	0.00	1817.40	133.20	0.40	29.80	0.00	5.80	30.98	0.00	12.26	0.00	19.61
500	15	0.00	1161.14	144.80	0.61	22.80	0.00	4.00	41.96	0.00	14.38	0.00	3.79
1000	1.5	309.80*	dnf	581.40	0.15	585.60	0.01	418.80	250.74	367.20	244.13	337.80	2762.58
1000	2	118.80*	dnf	334.80	0.22	370.60	0.03	233.00	245.86	158.20	173.53	142.40	1096.66
1000	4	0.40*	dnf	186.00	0.49	110.60	0.02	53.60	137.00	1.20	85.42	0.60	232.04
1000	10	0.00	2550.20	257.40	1.80	52.80	0.01	18.80	121.37	0.00	56.52	0.00	95.26
1000	15	0.00	2226.30	296.00	3.19	38.60	0.01	11.00	107.05	0.00	46.12	0.00	69.85

6.7 Conclusions

In this chapter, we have developed heuristic approaches for two spanning tree problems viz. MBV and MDS having relevance in optical network design. For each of these two problems, our first approach is a problem-specific heuristic, whereas the second approach is based on ACO. Our problem-specific heuristic approaches outperform the previously proposed best heuristic approaches for these problems. As the results obtained by these heuristics are inferior to exact approaches and this difference in solution quality grows with instance size, we have proposed the hybrid ACO approaches. A significant feature of our ACO approaches is the use of two two types of pheromones. Computational results show the effectiveness of ACO approaches.

Chapter 7

Early/Tardy Scheduling Problem

7.1 Introduction

This chapter addresses a single machine scheduling problem with earliness and tardiness costs and no machine idle time. Using the same notational conventions as used in [163, 164], the problem can be formulated as follows: Given n independent jobs J_1, J_2,...,J_n, each of which has to be sequentially processed without preemption on a single machine. The machine and the jobs are supposed to be continuously available. Each job J_j needs a processing time p_j on this machine and, ideally, should be finished exactly on its due date d_j. Earliness E_j and tardiness T_j of a job J_j in a schedule can be defined as $E_j = max(0, d_j - C_j)$ and $T_j = max(C_j - d_j, 0)$ respectively, where C_j is the time at which job J_j finishes in this schedule. The objective of early/tardy scheduling problem (ETSP) considered in this chapter is to find a schedule that minimizes

$$\sum_{j=1}^{n}(h_j E_j + w_j T_j) \tag{7.1}$$

where h_j and w_j are the earliness and tardiness penalties of job J_j.

ETSP is an $\mathcal{NP}$-Hard problem, as it is a generalization of weighted tardiness scheduling problem which is $\mathcal{NP}$-Hard [165].

Scheduling models, which penalize both earliness and tardiness, are based on the philosophy of just-in-time production, i.e., producing something just at the instant it is needed. This is indeed desirable in many real life production environments. For example, producing goods early invariably incurs the holding cost. Additional problems occur with perishable goods which may spoil by the time they are needed. Similarly, producing goods late may lead to loss of sales, loss of goodwill and increased shipping cost due to rush shipping [163].

No unforced machine idle time is allowed in the scheduling model discussed here, i.e., a machine can be idle only when no jobs are available for processing. This suits those production environments where either machine has to be operated continuously to meet the demands or operating and startup costs of the machine exceed the cost of producing some jobs early. Some specific examples, which can be found in [166] and [167], show the importance of these kind of scheduling models in real life scenarios.

For ETSP, many exact and heuristic approaches have been proposed. Among exact approaches, Abdul-Rajaq and Potts [168] presented a branch-and-bound method in which lower bounding procedure was based on subgradient optimization and dynamic programming state-space relaxation method, while Li [169] and Liaw [170] used Lagrangian relaxation and multiplier adjustment methods for lower bounding procedure in their branch-and-bound methods. Valente and Alves [171] further improved the lower bounds proposed by Li [169] and Liaw [170]. Tanaka [172] proposed an exact method based on successive sublimation dynamic programming. This method begins with a very basic relaxation of the original problem and then successively solves relaxations with more and more detailed information through dynamic programming.

As far as heuristics approaches for ETSP are concerned, Ow and Morton [173] proposed several early/tardy dispatch rules and a filtered beam search procedure for ETSP. Though filtered beam search procedure was better than early/tardy dispatch rules, but found to be too slow for large instances with more than 100 jobs. Li [169] proposed a heuristic procedure based on neighborhood search that is better than filtered beam search procedure of [173] in terms of both solution quality and running time. Valente and Alves [174] presented some more dispatch rules and greedy heuristics. They also used improvement procedures to further reduce the cost of the schedules obtained by the heuristics.

Valente *et al.* [163] proposed twelve variants of a hybrid generational genetic algorithm. All twelve genetic algorithms (GAs) were elitists, i.e., from the current generation, these GAs copy the best 10% of the population unaltered to the next generation. All these GAs employs random-key encoding [175] and parameterized unifom crossover [176]. No mutation operator was used by any of these genetic algorithms, and therefore, some solutions are generated randomly every generation for the sake of maintaining diversity in the population. The basic version of genetic algorithm with only above mentioned features was named GA. Genetic algorithms, which employ as local search a single pass or up to eight passes (no further passes are performed if no improvement is achieved during a pass) of an adjacent interchange procedure, were named GA-SAI and GA-MAI respectively. During each pass, the adjacent interchange procedure starts with the first job in the schedule and interchanges two adjacent jobs whenever doing so reduces the cost of the schedule. For each of these three versions, another variant is created that further reduces the cost of the best solution obtained by the genetic algorithm through successive application of a non-adjacent pairwise interchange procedure till no further reduction in the cost of the solution is possible. This resulted in three more versions GA-MNAI, GA-SAI-MNAI and GA-MAI-MNAI. During each pass, starting from the first job in the schedule, the non-adjacent pairwise interchange procedure tries to interchange the concerned job with all the other jobs. The interchange that leads to a schedule of minimum cost is carried out, and then the next job in the schedule is considered. Initial population in all these six versions consist of randomly generated solutions. For each of these six versions, one more variant is created, where initial population is seeded with one solution obtained through the EXP-ET dispatching procedure [173] and one solution obtained through the NSearch neighborhood search algorithm [169]. Remaining members of initial population were generated randomly. This yielded the six more versions viz. GA-INI, GA-SAI-INI, GA-MAI-INI, GA-MNAI-INI, GA-SAI-MNAI-INI, GA-MAI-MNAI-INI. These 12 genetic algorithms were compared against the EXP-ET dispatching procedure and the NSearch neighborhood search algorithm. These two methods were chosen for comparison because of the fact that the NSearch was the best heuristic approach known at that time and the EXP-ET dispatching procedure was the best among all the dispatching procedures. All the genetic algorithm variants outperformed the EXP-ET dispatching procedure in terms of solution quality by a large margin. All variants except GA, GA-INI and GA-SAI, obtained better quality solution in comparison to the NSearch algorithm. However, all the genetic algorithm variants were slower than the NSearch algorithm and the EXP-ET dispatching procedure. The EXP-ET dispatching procedure was even faster than NSearch algorithm.

Singh [164] proposed a hybrid permutation-coded steady-state genetic algorithm for ETSP. This genetic algorithm uses uniform order-based crossover and multiple swap mutations. Like [163], it also employs the multiple passes of adjacent interchange procedure as local search. This genetic algorithm was named SS-GA. Like MNAI variants of [163], another version of SS-GA was also considered where the best solution obtained by the SS-GA was improved through successive applications of non-adjacent pairwise interchange procedure. This latter version was called SS-GA-MNAI. Both these versions outperformed the 14 approaches (12 genetic algorithms, the EXP-ET dispatching procedure and the NSearch neighborhood search algorithm) considered in [163].

In this chapter, we have proposed an artificial bee colony algorithm (ABC) to solve ETSP. To our knowledge, only [177, 178] describe ABC algorithms for permutation problems. [177] describes an application of ABC algorithm for solving the lot-streaming flowshop scheduling problem, whereas [178] applies ABC algorithm for minimizing the total flowtime in permutation flow shops. Therefore, the domain of permutation problems are under-investigated as far as ABC algorithm is concerned. This has motivated us to develop an approach based on ABC algorithm for ETSP which is a permutation problem. Similar to [163, 164], a variant of the basic approach is also presented, where the best solution obtained through ABC approach is improved further by successive passes of non-adjacent pairwise interchange procedure. We have compared our approaches with those presented in [163, 164] which are the best heuristic approaches known so far for the problem. Computational results show the effectiveness of our approaches.

The remainder of this chapter is organized as follows: Section 7.2 describes our approaches for ETSP. Computational results are reported in Section 7.3, whereas Section 7.4 contains some concluding remarks.

7.2 ABC Algorithm for ETSP

The main features of our ABC algorithm for ETSP are described below:

7.2.1 Solution Encoding

Since, ETSP is inherently a linear permutation problem, therefore, we have encoded each solution by a linear permutation of jobs that denotes the ordering of execution of the jobs in that solution.

7.2.2 Initialization

Each initial solution, which is essentially a random permutation of n jobs, is generated by following an iterative process. Initially, let U be the set containing all n jobs and let S be an empty schedule. A job J_i is selected uniformly at random from U. This job is added to S at the first position and deleted from U. Next, iteratively a job J_j is selected from U using the roulette wheel selection method, where the probability of selecting a job is inversely proportional to its earliness or tardiness cost as the case may be with respect to the jobs already in S. The selected job J_j is deleted from U and added to S. It is to be noted that if during the beginning of an iteration a job is found to have zero cost, then that job is selected immediately and the roulette wheel selection method is not used in that iteration. This whole process is repeated again and again until U becomes empty.

Each initial solution is uniquely associated with an employed bee and the fitness of each solution is computed.

7.2.3 Probability of Selecting a Food Source

We have used the binary tournament selection method where the candidate with better fitness is selected with probability p_{bt}.

7.2.4 Determination of a Food Source in the Neighborhood of a Food Source

In order to find a high quality solution, the problem structure should be exploited effectively so that the search process leads towards the optimal. As the problem-structure of ETSP is based on the permutation of n jobs, we apply two operators, i.e., multi-point insert operator and 3-point swap operator in a mutually exclusive way for determining a neighboring solution in the search space of $n!$, where n is the number of jobs. Insert operator is applied with probability p_q, otherwise swap operator is applied.

7.2.4.1 Multi-Point Insert Operator

Like traditional ABC algorithms, this method for determining a neighboring solution also utilizes components from another solution. It is based on a simple observation that in case of a scheduling problem like ETSP, if a job is present at a particular position in one good schedule (solution), then the likelihood that this job is present exactly at the same position or around the same position in many other good schedules is high. To generate a solution s' in the neighborhood of a solution s, another solution t is randomly selected from employed bee population. If t is different from s, then we have used the following procedure. Initially, s' is an empty schedule, i.e., all positions in s' are marked empty. nbr different positions in t are randomly chosen and jobs in these positions are inserted into the corresponding positions in s'. The remaining $n - nbr$ empty positions in s' are filled with the jobs not yet inserted. These not yet inserted jobs are sorted in ascending order according to the positions they occupy in s. Then beginning at the first empty position, these jobs are inserted one-by-one in their sort order at the empty positions in s'. nbr is a parameter to be determined empirically. Its value should be small in comparison to total number of jobs n, otherwise the generated solution s' may be far off from s in the search space. We have taken nbr to be equal to $\frac{n}{10}$.

If t is same as s, then there is no point in using the multi-point insert operator and the 3-point swap operator, which is described next, is applied on s in an attempt to diversify the population.

7.2.4.2 3-Point Swap Operator

A swap operation is like a perturbation strategy which not only explores the search space, but also prevents the solution, as far as possible, from trapping in a local optima. We have used 3-point swap (3PS) which was introduced in [179]. To generate a solution s' in the neighborhood of a solution s through 3PS, the following procedure is used. Initially, s' is a copy of s. Then three positions i, j and k are selected randomly in s', then two swap operations are applied among the jobs at these three selected positions, i.e., the job at position i is swapped with the job at position j, and then the new job at position i is swapped with the job at position k. The 3PS leads to a schedule that is at a distance (number of swaps required to convert a schedule to another) of 2 from

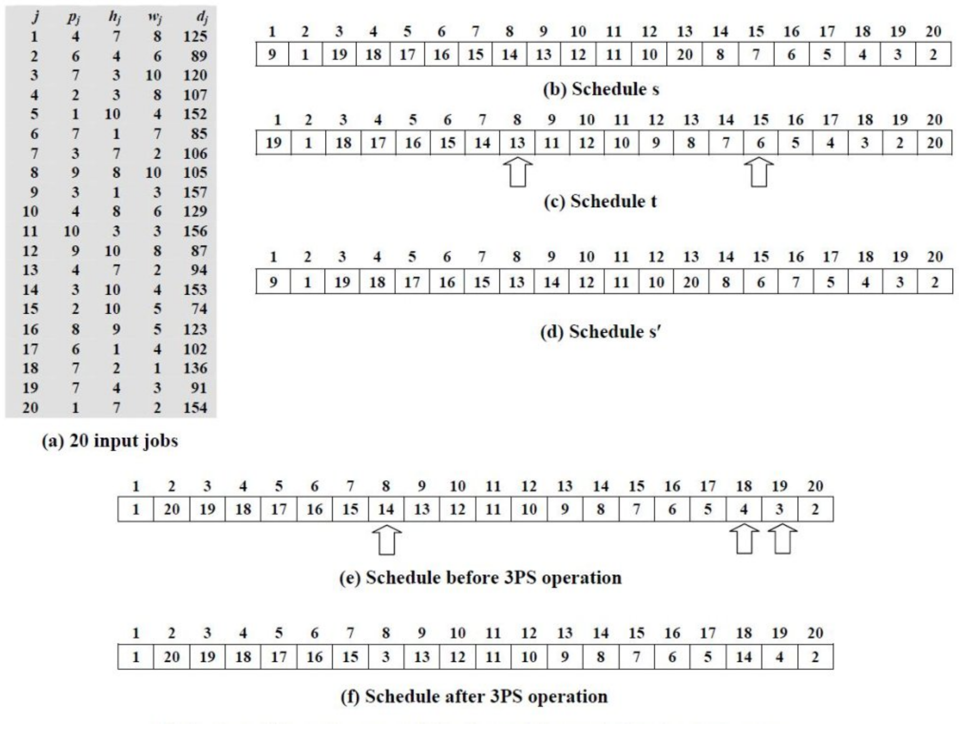
Algorithm 9: Method for Determining a Neighboring Food Source

```
input  : A food source $s$
output: A neighboring food source $s'$
Select another food source $t$ randomly from the employed bee population;
if (($u01 < p_q$) $\wedge$ ($t$ is different from $s$)) then
    Generate $s'$ from $s$ and $t$ using the multi-point insert operator;
else
    Create a copy $s'$ of $s$;
    Apply 3-point swap operator on $s'$;
return $s'$;
```

the original schedule. This may accelerate the search towards global optimum [179] in comparison to using only a single swap.

The pseudo-code of the aforementioned procedure for determining a neighboring food source is given in Algorithm 9, where $u01$ is a uniform variate in [0, 1].

Figure 7.1 explains our multi-point insert and 3-point swap operators with the help of an example. Figure 7.1(a) shows 20 input jobs along with their processing times, due dates, and earliness and tardiness penalties. Figures 7.1(b) and 7.1(c) show the schedules s and t as defined in Section 7.2.4.1. $nbr = 2$ and hence two positions, say positions 8 and 15, are chosen randomly. Positions 8 and 15 in the schedule t respectively contain jobs J_{13} and J_6. These jobs are inserted into the exactly same positions in the schedule s' as shown in Figure 7.1(d). The remaining 18 jobs are sorted in ascending order according to the position they occupy in s. Therefore, the order is J_9, J_1, J_{19}, J_{18}, J_{17}, J_{16}, J_{15}, J_{14}, J_{12}, J_{11}, J_{10}, J_{20}, J_8, J_7, J_5, J_4, J_3, J_2. Beginning at the first empty position in the schedule s', one-by-one these jobs are inserted in this order to the empty positions in s'. This leads to a schedule shown in Figure 7.1(d). Figures 7.1(e) and 7.1(f) respectively show the schedules before and after 3PS. In this example, $i = 8$, $j = 18$ and $k = 19$. The job at position 8 (J_{14}) is swapped with the job at position 18 (J_4), and then the new job at position 8 (J_4) is swapped with the job at position 19 (J_3) to get the schedule shown in Figure 7.1(f).



j	p_j	h_j	w_j	d_j
1	4	7	8	125
2	6	4	6	89
3	7	3	10	120
4	2	3	8	107
5	1	10	4	152
6	7	1	7	85
7	3	7	2	106
8	9	8	10	105
9	3	1	3	157
10	4	8	6	129
11	10	3	3	156
12	9	10	8	87
13	4	7	2	94
14	3	10	4	153
15	2	10	5	74
16	8	9	5	123
17	6	1	4	102
18	7	2	1	136
19	7	4	3	91
20	1	7	2	154

(a) 20 input jobs

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	1	19	18	17	16	15	14	13	12	11	10	20	8	7	6	5	4	3	2

(b) Schedule s

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
19	1	18	17	16	15	14	13	11	12	10	9	8	7	6	5	4	3	2	20

(c) Schedule t

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	1	19	18	17	16	15	13	14	12	11	10	20	8	6	7	5	4	3	2

(d) Schedule s′

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2

(e) Schedule before 3PS operation

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	20	19	18	17	16	15	3	13	12	11	10	9	8	7	6	5	14	4	2

(f) Schedule after 3PS operation

Figure 7.1: An illustrative example for determining a neighboring food source

7.2.5 Local Search

To further improve the solution quality, we have used multiple passes of 3-swap procedure [180] as local search. A pass of 3-swap procedure starts with the first job in the schedule and considers in succession all $n - 2$ triples of consecutive jobs. All possible permutations of the jobs in a triple are considered, and then the best permutation is selected as the new ordering of the three consecutive jobs in that triple. The parameter *npass* controls the number of passes used. However, if one complete pass fails to improve the solution quality, then that means that the solution cannot be improved further through 3-swap procedure, therefore, in this situation local search terminates before *npass* passes.

It is to be noted that in order to keep the running time manageable, we have applied the local search only on those solutions where the difference between its fitness and the fitness of the best solution found so far is less than *range*% of the fitness of the best solution found so far.

7.2.6 Other Features

If a food source (solution) does not improve for a predetermined number of iterations *limit*, then the employed bee associated with that food source abandons it and becomes a scout. Instead of generating a new food source randomly from scratch (as described in Section 7.2.2) for making this new scout employed again, we have generated a random solution in the neighborhood of the just abandoned solution via 3-point swap operator (as described in Section 7.2.4.2). The reason behind using such an strategy is that the efficiency of the search increases as the random solution generated from scratch is expected to have much worse fitness than the solution obtained from an existing solution through 3-point swap operator. Besides, generating a food source from scratch requires more effort in comparison to generating it through 3-point swap operator. For an scout bee, Pan *et al.* [177] always generates a solution in the neighborhood of the best solution in the population. We have also tried this strategy, but we got better results with our strategy.

7.2.7 ABC-MNAI Approach

Similar to MNAI variants of [163] and [164], we have also implemented another variant of our ABC approach, where the best schedule obtained through it is improved further by repeatedly applying passes of non-adjacent pairwise interchange procedure till a complete pass fails to improve the schedule. We call this variant ABC-MNAI.

7.3 Computational Results

This section reports the computational results of ABC and ABC-MNAI approaches for ETSP and compare them with other 16 approaches described in [163, 164]. Our approaches have been implemented in C and executed on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. In all our computational experiments with ABC and ABC-MNAI, we have used a population of 100 bees consisting of 50 employed and 50 onlooker bees, i.e., $n_e = 50$ and $n_o = 50$. We have taken $limit = 50$, $p_{bt} = 0.8$, $p_q = 0.4$, $range = 10$, $nbr = \frac{n}{10}$, and $npass = 2$. On each instance with n jobs, we have allowed our approaches to execute for 1000 iterations if $n \leq 250$, otherwise for 1500 iterations. We have chosen these parameters empirically after a large number of trials. Though these parameter values provide good results, they may not be optimal for all instances.

Table 7.1: Comparisons of ABC with 16 Approaches in terms of Number of Instances on which ABC Performs Better, Same or Worse

	ABC																							
	n=15			n=50			n=75			n=100			n=250			n=500			n=750			n=1000		
	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W
EXP-ET	100	0	0	100	0	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
Nsearch	38	62	0	97	3	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA	53	47	0	100	0	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MNAI	28	72	0	88	12	0	97	3	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-INI	34	66	0	95	5	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MNAI-INI	29	71	0	92	8	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI	1	99	0	75	25	0	99	1	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-MNAI	1	99	0	63	37	0	92	8	0	99	1	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-INI	3	97	0	69	31	0	97	3	0	97	2	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-MNAI-INI	3	97	0	64	36	0	90	9	1	95	4	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI	0	100	0	39	61	0	79	20	1	97	3	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-MNAI	0	100	0	37	63	0	70	29	1	92	7	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-INI	0	100	0	33	67	0	79	20	1	81	17	2	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-MNAI-INI	0	100	0	32	68	0	72	27	1	74	24	2	-	-	-	-	-	-	-	-	-	-	-	-
SS-GA	0	100	0	16	84	0	31	64	5	57	38	5	85	6	9	88	2	10	89	1	10	89	0	11
SS-GA-MNAI	0	100	0	16	84	0	29	66	5	55	40	5	77	6	17	59	3	38	34	1	65	28	1	71
Overall	0	100	0	7	93	0	19	75	6	44	51	5	77	6	17	59	3	38	34	1	65	28	1	71

Our approaches are tested on the same instances as used in Valente *et al.* [163] and Singh [164]. These instances were generated by Valente *et al.* [163] in a manner similar to those used in [168, 169, 170]. Based on the number of jobs, these instances are categorized into four groups viz. instances with 15, 50, 75 and 100 jobs. Each group consists of 100 instances. Each job J_j in these instances consists of an integer processing time p_j, earliness penalty h_j and tardiness penalty w_j drawn uniformly at random from the interval [1, 10]. An integer due date d_j for each job J_j is drawn uniformly at random from the interval $[T(1 - LF - RDD/2), T(1 - LF + RDD/2)]$, where T is the sum of the processing times of all jobs, LF is the lateness factor and RDD is the range of due dates. LF is set to 0.2 and 0.4, whereas RDD is set to 0.2, 0.4, 0.6, 0.8 and 1.0. Ten instances were generated for each of the 10 combinations of these two parameters. Therefore, each group consists of 100 instances. In order to test our approaches on large instances, we have generated 4 additional groups of instances with 250, 500, 750 and 1000 jobs using the same procedure as described above.

Tables 7.1 and 7.2 compare the results of our ABC approach with each of the 16 approaches presented in [163, 164]. In Table 7.1, comparison is done on the basis of number of instances out of 100 in each of the 8 groups where ABC performs better (B), equal (E) or worse (W) in terms of solution quality. Exact solution values of 14 approaches reported in [163] were obtained through personal communication. These

Table 7.2: Average Percentage Deviation and Average Computation Times for 16 Approaches and ABC

	n=15		n=50		n=75		n=100		n=250		n=500		n=750		n=1000	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT
EXP-ET	18.59	0.0	15.20	0.0	12.99	0.0	14.68	0.0	-	-	-	-	-	-	-	-
Nsearch	1.15	0.0	2.66	1.8	2.32	9.9	2.52	25.5	-	-	-	-	-	-	-	-
GA	1.91	1.7	6.47	15.2	7.31	39.4	9.64	75.4	-	-	-	-	-	-	-	-
GA-MNAI	0.69	1.7	1.29	15.5	1.01	40.7	0.87	78.8	-	-	-	-	-	-	-	-
GA-INI	0.84	2.0	1.92	16.7	2.40	44.0	2.63	91.8	-	-	-	-	-	-	-	-
GA-MNAI-INI	0.64	2.0	1.15	16.9	1.26	44.9	1.16	94.1	-	-	-	-	-	-	-	-
GA-SAI	0.00	2.3	1.37	21.9	2.24	63.4	3.13	114.5	-	-	-	-	-	-	-	-
GA-SAI-MNAI	0.00	2.3	0.55	22.1	0.90	64.4	1.02	117.1	-	-	-	-	-	-	-	-
GA-SAI-INI	0.02	2.8	0.67	26.7	1.21	74.5	1.30	148.1	-	-	-	-	-	-	-	-
GA-SAI-MNAI-INI	0.02	2.8	0.48	26.9	0.72	75.1	0.71	149.9	-	-	-	-	-	-	-	-
GA-MAI	0.00	3.6	0.26	46.6	0.68	130.0	0.98	267.7	-	-	-	-	-	-	-	-
GA-MAI-MNAI	0.00	3.6	0.21	46.7	0.43	130.6	0.53	269.6	-	-	-	-	-	-	-	-
GA-MAI-INI	0.00	3.9	0.26	50.8	0.58	130.5	0.64	260.2	-	-	-	-	-	-	-	-
GA-MAI-MNAI-INI	0.00	3.9	0.25	50.9	0.53	130.9	0.46	261.5	-	-	-	-	-	-	-	-
SS-GA	0.00	0.4	0.04	0.9	0.09	1.2	0.16	1.8	0.05	6.7	0.09	26.6	0.17	64.7	0.27	117.6
SS-GA-MNAI	0.00	0.4	0.04	0.9	0.08	1.2	0.09	1.8	0.02	7.0	0.01	28.8	0.00	66.2	0.00	121.0
ABC	0.00	0.1	0.00	0.3	0.01	0.7	0.01	1.2	0.01	7.0	0.01	31.8	0.03	56.5	0.04	90.8

14 approaches were executed only on instances with up to 100 jobs. [164] also reports the results of SS-GA and SS-GA-MNAI on instances with up to 100 jobs only. As the source codes for these two approaches were available, we have executed these two approaches along with ABC and ABC-MNAI on 4 new groups of large instances. In addition, SS-GA and SS-GA-MNAI have been re-executed on older instances with 15, 50, 75 and 100 jobs so that their execution times can be compared exactly with those of ABC and ABC-MNAI. The last row in Table 7.1 compare the solution obtained through ABC approach with the best solution among all the 16 approaches for instances up to 100 jobs and with the best solution among SS-GA and SS-GA-MNAI for the remaining instances. Table 7.2 compares ABC approach with other approaches in terms of average percentage deviation (APD) and average computation time (ACT) in seconds on each of the eight groups of instances. On a particular instance, let A be the solution obtained by an approach and let B be the best known or optimal solution (if known), then the percentage deviation of this approach on this instance is defined as $\frac{(A-B)}{B} \times 100$. Reported APD values are average of percentage deviations over 100 instances of each group. Only on instances with 15 jobs, we know the optimum solutions [163]. For the remaining groups, we have taken the best solution among all the approaches applied to instances of that group as the best known solution and computed APD values with respect to it. As ABC approach improved the best known solution values on instances

Table 7.3: Comparisons of ABC-MNAI with 16 Approaches in terms of Number of Instances on which ABC-MNAI Performs Better, Same or Worse

	ABC-MNAI																							
	n=15			n=50			n=75			n=100			n=250			n=500			n=750			n=1000		
	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W
EXP-ET	100	0	0	100	0	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
Nsearch	38	62	0	97	3	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA	53	47	0	100	0	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MNAI	28	72	0	88	12	0	97	3	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-INI	34	66	0	95	5	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MNAI-INI	29	71	0	92	8	0	100	0	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI	1	99	0	75	25	0	99	1	0	100	0	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-MNAI	1	99	0	63	37	0	92	8	0	99	1	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-INI	3	97	0	69	31	0	97	3	0	97	2	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-SAI-MNAI-INI	3	97	0	64	36	0	90	9	1	95	4	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI	0	100	0	39	61	0	80	20	0	97	3	0	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-MNAI	0	100	0	37	63	0	71	29	0	92	7	1	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-INI	0	100	0	33	67	0	79	21	0	81	17	2	-	-	-	-	-	-	-	-	-	-	-	-
GA-MAI-MNAI-INI	0	100	0	32	68	0	72	28	0	74	24	2	-	-	-	-	-	-	-	-	-	-	-	-
SS-GA	0	100	0	16	84	0	31	65	4	57	38	5	87	7	6	94	2	4	98	0	2	97	0	3
SS-GA-MNAI	0	100	0	16	84	0	29	67	4	55	40	5	84	9	7	81	3	16	84	0	16	83	0	17
Overall	0	100	0	7	93	0	19	76	5	44	51	5	84	9	7	81	3	16	84	0	16	83	0	17

with 50, 75 and 100 jobs, therefore, APD values reported for these groups of instances differ from those in [163, 164].

As can be seen from the Tables 7.1 and 7.2, ABC is clearly better than all the other approaches on instances of Valente *et al.* [163] in terms of solution quality. As shown by the last row of table 7.1, for these instances even the 16 approaches taken together cannot compete with ABC approach. Our approach improves the best known solution values of 7, 19 and 44 instances respectively belonging to groups with 50, 75 and 100 jobs. APD values for our ABC approach are also least among all the approaches for these instances. As far as the performance of ABC on large instances is concerned, SS-GA-MNAI performs better than ABC on instances of size 750 and 1000, whereas ABC performs better on instances with 250 and 500 jobs. For instances of size 750 and 1000, respective APD values are 0.03 and 0.04 for ABC indicating that solutions obtained through ABC approach are only slightly inferior to the best solutions for these instances. ABC anyway is better than SS-GA on all 4 groups of large instances. Therefore, the superior performance of SS-GA-MNAI on large instances can be attributed to repeated application of passes of non-adjacent pairwise interchange procedure. This fact has motivated us to consider ABC-MNAI.

Our ABC approach along with SS-GA and SS-GA-MNAI have been executed on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. As shown in Table 7.2,

Table 7.4: Average Percentage Deviation and Average Computation Times for 16 Approaches and ABC-MNAI

	n=15		n=50		n=75		n=100		n=250		n=500		n=750		n=1000	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT
EXP-ET	18.59	0.0	15.20	0.0	12.99	0.0	14.69	0.0	-	-	-	-	-	-	-	-
Nsearch	1.15	0.0	2.66	1.8	2.32	9.9	2.53	25.5	-	-	-	-	-	-	-	-
GA	1.91	1.7	6.47	15.2	7.31	39.4	9.64	75.4	-	-	-	-	-	-	-	-
GA-MNAI	0.69	1.7	1.29	15.5	1.01	40.7	0.88	78.8	-	-	-	-	-	-	-	-
GA-INI	0.84	2.0	1.92	16.7	2.40	44.0	2.63	91.8	-	-	-	-	-	-	-	-
GA-MNAI-INI	0.64	2.0	1.15	16.9	1.26	44.9	1.16	94.1	-	-	-	-	-	-	-	-
GA-SAI	0.00	2.3	1.37	21.9	2.24	63.4	3.14	114.5	-	-	-	-	-	-	-	-
GA-SAI-MNAI	0.00	2.3	0.55	22.1	0.90	64.4	1.02	117.1	-	-	-	-	-	-	-	-
GA-SAI-INI	0.02	2.8	0.67	26.7	1.21	74.5	1.31	148.1	-	-	-	-	-	-	-	-
GA-SAI-MNAI-INI	0.02	2.8	0.48	26.9	0.72	75.1	0.71	149.9	-	-	-	-	-	-	-	-
GA-MAI	0.00	3.6	0.26	46.6	0.68	130.0	0.99	267.7	-	-	-	-	-	-	-	-
GA-MAI-MNAI	0.00	3.6	0.21	46.7	0.43	130.6	0.53	269.6	-	-	-	-	-	-	-	-
GA-MAI-INI	0.00	3.9	0.26	50.8	0.58	130.5	0.65	260.2	-	-	-	-	-	-	-	-
GA-MAI-MNAI-INI	0.00	3.9	0.25	50.9	0.53	130.9	0.47	261.5	-	-	-	-	-	-	-	-
SS-GA	0.00	0.4	0.04	0.9	0.09	1.2	0.16	1.8	0.05	6.7	0.10	26.6	0.18	64.7	0.28	117.6
SS-GA-MNAI	0.00	0.4	0.04	0.9	0.08	1.2	0.09	1.8	0.02	7.0	0.02	28.8	0.01	66.2	0.01	121.0
ABC-MNAI	0.00	0.1	0.00	0.3	0.01	0.7	0.01	1.2	0.00	7.1	0.00	32.5	0.00	59.4	0.00	98.6

ABC is clearly faster than SS-GA and SS-GA-MNAI except for instances of size 250 and 500 where it is slightly slower. Approaches of Valente et. al. [163] were executed on a Pentium II, 350 MHz system. As these approaches were executed on a different system, therefore it is not possible to exactly compare the execution times of these approaches with ABC. However, we can safely say that our ABC approach is faster than the 12 GA variants of [163] on instances with 50, 75 and 100 jobs.

Tables 7.3 and 7.4 compare our ABC-MNAI approach with each of the 16 approaches presented in [163, 164]. Tables 7.3 and 7.4 are respectively similar in content to Tables 7.1 and 7.2 except for the fact that best solution values obtained through ABC-MNAI are used in APD computations wherever these values are better than all other 16 approaches. These tables clearly show the superiority of ABC-MNAI in terms of solution quality over all other approaches on all groups of instances. In particular, ABC-MNAI is better than SS-GA-MNAI on all 4 groups of large instances in terms of solution quality. As shown by the last row of Table 7.3, for each of the 8 groups of instances, ABC-MNAI performs even better than all other approaches applied to that group taken together. ABC-MNAI also has the smallest APD values for all groups of instances. These two facts taken together show the robustness of ABC-MNAI. Regarding comparison of execution times, conclusions on the lines similar to ABC can be drawn here also.

As far as comparison between ABC-MNAI and ABC is concerned, ABC-MNAI improves 2, 1, 49, 90, 97, 98 solutions obtained through ABC on each group of 100 instances with 75, 100, 250, 500, 750, 1000 jobs respectively. As ABC-MNAI is nothing, but ABC followed by multiple passes of MNAI procedure. Execution time of ABC-MNAI is always slightly more than ABC.

7.4 Conclusions

In this chapter, we have developed two artificial bee colony based hybrid approaches viz. ABC and ABC-MNAI. We have tested our approaches against 16 previously proposed approaches [163, 164] not only on existing set of instances, but also on large instances. ABC approach has performed better than all the 16 methods on existing set of instances, but it has performed worse on two largest group of instances with 750 and 1000 jobs in comparison to SS-GA-MNAI approach of [164]. This motivated the development of ABC-MNAI which is same as ABC except for the fact that the best solution obtained through ABC approach is improved further through successive application of passes of non-adjacent pairwise interchange procedure. ABC-MNAI has outperformed all other approaches in terms of solution quality. It also requires less execution time in comparison to other best performing approaches on most group of instances. We have shown that for all 8 groups of instances, even if we compare the results of ABC-MNAI approach against the best results among all the approaches applied to that group, our results are still better.

Chapter 8

Blockmodel Problem

8.1 Introduction

Social network analysis has emerged as an indispensable technique for getting insight into the structure of complex social networks. It finds the relationships among interacting entities which in turn provides many interesting facts necessary for understanding the structures or patterns of interacting entities in social networks. In order to analyze the relational aspects of the structures of a social network, social network analysis methods often visualize social relationships in terms of graphs consisting of vertices and edges. It is always possible to represent a social network as a graph, where vertices are the individual entities and edges show the interdependency or relationship between vertices. The interdependency among vertices varies from one social network to another and can represent common interest, dislike, knowledge etc. Usually, a social network graph is represented by an adjacency matrix whose individual entries show the presence or absence of the relationship between entities. In order to detect structures or patterns in the graph, the adjacency matrix is provided as an input to a social network analysis tool.

Pertaining to social network analysis, a blockmodel [181] is a representation of structural relationships among entities. The objective of this model is to find groups of entities that exhibit a high degree of interconnectedness. In the literature of social network analysis, cliques and blocks are the two ways that have been commonly used to group the entities or vertices based on their structural relationships. A clique is a cluster (group) of vertices in which every pair of vertices is connected to each other and vertices are allowed to be members of more than one clique, i.e., cliques may overlap. On the other hand a block is a group of vertices which are highly interconnected among themselves and vertices are not allowed to be members of more than one block, i.e., blocks are disjoint [182].

Blockmodel is primarily based on the concept of structural equivalence and structural similarity. Two entities are said to be structurally equivalent, if they have exactly same pattern of interaction with all entities. Two entities are said to be structurally similar, if they have nearly the same pattern of interaction with all entities. Such structurally equivalent or similar entities can be grouped into a block. The goal of blockmodeling is to partition a social network graph into blocks, i.e., groups of structurally equivalent or similar entities. However, there may exist multiple ways of partitioning a graph into blocks, and therefore, some criterion is required to guide the partitioning process. Jessop [182] proposed one such criterion. According to Jessop's criteria called maximum concentration, blockmodeling should find small number of large, dense blocks, i.e., blocks of size as large as possible and for which the intra-block edge density is close to 1. The motivation for this criterion is to maximize the number of edges within the blocks and consequently minimize the inter-block interactions.

Apart from several practical applications of blockmodeling in social network analysis [183, 184, 185, 186], recent research has been witnessed to the potential applications of blockmodeling to analyze multi-attribute measures of performance in airlines [187], airports [188], and universities [182]. Besides, blockmodeling also provides an attractive alternative for well-established technique of data envelopment analysis [189].

Jessop is the most potent contributor to the field of blockmodeling [3, 182, 187, 188, 190, 191]. Proll [2] developed an integer linear programming (ILP) formulation of blockmodel problem (BMP) after linearizing the quadratic programming formulation of [182]. Proll's ILP is able to find feasible solutions for each of the seven problems tested and on four of the problems the results are superior to those of Jessop [190]. Later, Jessop *et al.* [3] presented two ILP formulations and a heuristic approach to BMP. These two ILP formulations are known as the vertex formulation and the clique formulation. Both ILP formulations are improved versions of Proll's ILP formulation [2]. The heuristic approach first decomposes the input graph into subgraphs with non-overlapping vertices and finds the optimal clique for each of these subgraphs. These cliques are combined to form a solution for the input graph. This solution is enhanced further by shifting vertices from the smallest clique to the biggest clique until no such shifts are possible.

As far as metaheuristic techniques are concerned, only Tabitha *et al.* [4] proposed a generational grouping genetic algorithm (GGGA) for BMP. Actually, BMP is a grouping problem, i.e., a problem whose objective is to find an optimal assignment of entities according to a given fitness function into different groups subject to some constraints. Grouping genetic algorithms [192] are specially designed to handle grouping problems as traditional genetic algorithm suffers from the problem of redundancy, context insensitivity and schema disruption while handling grouping problems.

In this chapter, we have proposed two metaheuristic techniques for BMP. The first one is a steady-state grouping genetic algorithm (SSGGA), whereas the latter one is an ABC algorithm. Hereafter, our ABC algorithm will be referred to as GABC. To our knowledge, this is the first time an ABC algorithm is applied to a problem with variable number of groups. Previously, we have extended ABC algorithm to a problem with fixed number of groups [87]. Both SSGGA and GABC are population-based approaches. However, underlying principles for both the approaches are quite different. SSGGA is an evolutionary algorithm based on survival of the fittest [8], whereas GABC is a swarm intelligence technique based on intelligent foraging behavior of honey bee swarm [13]. We have compared our approaches against those proposed in [2, 3, 4]. Computational results show the effectiveness of our approaches.

Note that it is obvious from computational results reported in [4] that GGGA for BMP is not so good in finding high quality solutions. We strongly felt that even another version of grouping genetic algorithm can easily beat GGGA [4] in terms of solution quality. Therefore, we have proposed a steady-state grouping genetic algorithm (SSGGA) which is entirely different from the one proposed in [4]. In fact, SSGGA for BMP is not directly within the scope of our thesis, but it is needed to test the capability of GABC in finding high quality solutions and its robustness. Comparing GABC solely with GGGA would not have served this purpose.

The remainder of this chapter is organized as follows: Section 8.2 provides the mathematical formulation for BMP. SSGGA is described in Section 8.3. Section 8.4 describes our GABC approach. Computational results are reported in Section 8.5. Finally, Section 8.6 presents some concluding remarks.

8.2 Problem Definition

Following the notational conventions used in [182] and [2], blockmodel problem (BMP) can be formulated as follows:

$$max\ \ HHI = \sum_{k=1}^{b} (\sum_{i=1}^{n} \lambda_{ik})^2 \tag{8.1}$$

subject to

$$\sum_{k=1}^{b} \lambda_{ik} = 1\ \ \forall i \in \{1, 2, ..., n\} \tag{8.2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \lambda_{ik} \lambda_{jk} - \beta (\sum_{i=1}^{n} \lambda_{ik})^2 \geq 0 \forall k \tag{8.3}$$

$$\lambda_{ik} \in \{0, 1\} \tag{8.4}$$

where n is the number of vertices, b is the maximum number of blocks, λ_{ik} and x_{ij} are binary variables. If vertex i belongs to block k, then $\lambda_{ik} = 1$, otherwise $\lambda_{ik} = 0$. If there is an edge between vertices i and j, then $x_{ij} = 1$, otherwise $x_{ij} = 0$. β is a parameter which represents the minimum acceptable block density. Typically, a value of β is chosen to be close to 1.

The objective of BMP is to maximize the sum of the squares of block sizes (see equation (8.1)). In economic parlance, this objective function is called Herfindahl-Hirshman Index (HHI) [193, 194], and is a popular measure of industrial concentration. In case of BMP, HHI is a measure of concentration of vertices into blocks, and it favors the solutions with small number of large blocks. Constraint (8.2) restricts that each vertex is a member of one block only. Constraint (8.3) can be rewritten as follows:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \lambda_{ik} \lambda_{jk} / (\sum_{i=1}^{n} \lambda_{ik})^2 \geq \beta$$

where $\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \lambda_{ik} \lambda_{jk}$ is the number of intra-block edges in block k. Therefore, constraint (8.3) ensures that the edge density of each block should be greater than or equal to a given threshold β. If $\beta = 1$, then all vertices in a block are directly connected to each other, i.e., a block in this case is precisely a clique. Therefore, one can solve BMP in this case by iteratively solving the maximum clique problem and removing the vertices forming the clique before beginning the next iteration. However, It is well-known that finding a maximum clique in a graph is an $\mathcal{NP}$-Hard combinatorial optimization problem [195]. Therefore, BMP is also an $\mathcal{NP}$-Hard problem. In order to compare our approaches with those proposed in [2, 3, 4], we have also used $\beta = 1$ in this chapter.

8.3 Steady-State Grouping Genetic Algorithm

We present a steady-state grouping genetic algorithm (SSGGA) for BMP that uses steady-state population replacement method [196]. In this method, genetic algorithm iteratively selects two parents, performs crossover and mutation to generate an offspring that replaces a worse member of the population. This differs from the generational method, where the whole parent population is replaced with an equal number of newly generated offsprings every generation. The steady-state population replacement method generally finds better solutions faster in comparison to the generational method. This is due to retaining the best solutions in the population permanently and the instant availability of the generated offspring for selection and reproduction. The steady-state population replacement method has one more advantage. It can easily avoid multiple copies of the same individuals in the population. In the generational method, multiple copies of highly fit individuals may exist in the population. Within few generations, these highly fit individuals can dominate the whole population. The crossover becomes totally ineffective in such a situation, and the only possible way to improve solution quality is through the mutation. When this happens, improvement, if any, in solution quality is quite slow. This situation is known as the premature convergence. In the steady-state method, we can easily prevent this situation by simply comparing each newly generated offspring with present population members and discarding the newly generated offspring in case it is identical to any member.

Subsequent subsections describe other salient features of SSGGA.

8.3.1 Encoding

We have represented a chromosome as a set of blocks. Therefore, no ordering exists among blocks. With such a representation there is no redundancy. The GGGA [4] uses the same chromosome representation as proposed by Falkenauer [192], which divides the chromosome into two parts viz. a group part and an entity part. The first part simply enumerates the number of groups, i.e., blocks present in the chromosome, whereas the entity part associates each entity with a group present in the chromosome. This representation suffers from the problem of redundancy because we can always rename the groups in the group part and can change accordingly the corresponding association in the entity part. However, as this representation is used with a grouping genetic algorithm, so if two chromosomes representing the same solution are ever crossed then they produce a child that also represent the same solution.

The result of crossover operator used in GGGA depends on the ordering among blocks. To reduce the effect of ordering, Falkenauer [192] describes an inversion operator, but this operator is not used in [4]. On the other hand, our genetic operators are designed in such a way that their results do not depend on the ordering among blocks. Therefore, inversion operator is not at all required in our approach.

8.3.2 Initial Population Generation

Each solution of the initial population consists of a set of blocks. In order to maintain diversity, each solution of the initial population is generated randomly by the following procedure: initially, all vertices belong to a set, say U. Then, blocks are constructed one-by-one and added to the solution. In order to construct a block, say B, first a vertex, say v, is deleted randomly from U and added to B. Then, iteratively a vertex $\in U$ that is adjacent to all vertices currently in B is deleted randomly from U and added to B. This process is repeated till no vertex $\in U$ can be added to B. If U is not yet empty, then a new block is constructed from U in an analogous manner. We keep on adding new blocks to the solution until U becomes empty.

Uniqueness of each generated individual (solution) is checked against the individuals of the population generated so far and if it is unique, it is included in the initial population, otherwise it is discarded.

8.3.3 Fitness

Fitness of each solution is taken to be equal to its HHI value, i.e., fitness function is same as the objective function.

8.3.4 Selection

In order to select two different chromosomes as parents for crossover, the binary tournament selection method is used consecutively two times. In this method, two chromosomes are picked uniformly at random from the current population and with probability p_b, the chromosome with better fitness is selected, otherwise worse one is selected (with probability $1 - p_b$). In case of mutation also, the binary tournament selection method is used to select a chromosome for mutation.

8.3.5 Crossover

Our crossover operator is inspired by the crossover operator proposed in [197]. Suppose that p_1 and p_2 are two parents selected for crossover, and nb_1 and nb_2 are the number of blocks in p_1 and p_2 respectively. Let $nb = min(nb_1, nb_2) - sub$, where sub is a parameter to be determined empirically. The crossover operator consists of a maximum of nb iterations. During each iteration, one of the two parents is selected uniformly at random and a largest block (ties are broken arbitrarily) from the selected parent is copied in the child. All vertices belonging to the block just copied are deleted from both the parents.

It is to be noted that while selecting a largest block from the selected parent, if it is found that the largest block is a singleton block, then iterative procedure for the crossover is stopped immediately without copying this singleton block to the child solution. Here, a singleton block means a block containing only one vertex. Moreover, iterative procedure for the crossover is also stopped prematurely, if parents become empty.

There is a high possibility that some vertices may be left unassigned in the child solution produced by the crossover. If this happens, then a repair procedure, which is described next, is used to assign these vertices to blocks.

8.3.6 Repair Procedure

Repair procedure consists of two stages. The first stage considers each unassigned vertex one-by-one and tries to assign it to the first block where it can be assigned without violating the feasibility. The blocks are checked according to non-increasing order of their sizes. The reason behind this policy lies in the objective function of the problem which increases by a larger amount, if we add a vertex to a larger block in comparison to adding the same vertex to a smaller block. To facilitate efficient implementation of this policy, blocks are kept continuously sorted according to non-increasing order.

If some vertices are left unassigned even after the completion of the first stage, then the second stage is used. This stage is responsible for iteratively constructing new blocks from unassigned vertices in a manner analogous to Section 8.3.2 and adding these to the child solution till no unassigned vertices are left. However, instead of selecting an unassigned vertex randomly from candidate unassigned vertices, we select an unassigned vertex of maximum degree (ties are broken arbitrarily).

8.3.7 Mutation

Our mutation operator deletes at most nb_m non-singleton blocks randomly along with all singleton blocks from the solution selected for mutation. All vertices belonging to these deleted blocks is added to a list of unassigned vertices. These unassigned vertices are again added to the solution by using the two stage procedure described in the previous section.

Like [197], here also a child is generated either by crossover or by mutation, but never by both, i.e., crossover and mutation are used mutually exclusively. Actually, our crossover operator greedily builds the child by copying the best blocks from two parents. If we use the mutation after the crossover, then the mutation can destroy some of these blocks, thereby reducing the benefits of using a greedy strategy in crossover. The parameter p_c governs the decision to use crossover or mutation in each generation. Crossover is used with probability p_c, otherwise mutation is used.

8.3.8 Replacement Policy

In our replacement policy, uniqueness of the newly generated child solution is tested against the existing solutions of the population. If it is found to be different from all population members, then it replaces the worst solution of the population, irrespective of its own fitness, otherwise it is discarded.

8.4 Grouping-based Artificial Bee Colony Algorithm

Our ABC algorithm is designed in such a way that it tries to preserve grouping information as far as possible. To our knowledge this is the first application of ABC algorithm for a problem with variable number of groups. The main components of grouping-based artificial bee colony algorithm (GABC) are described below:

8.4.1 Encoding

Each solution is encoded as a set of blocks, which is similar to the encoding scheme of SSGGA explained in Section 8.3.1. Like SSGGA, the results obtained by GABC do not depend on the ordering among the blocks.

8.4.2 Initialization

Each initial solution is generated in a way exactly similar to the initial solutions of SSGGA as explained in Section 8.3.2. Each initial solution is uniquely associated with an employed bee and the fitness of each solution is computed.

8.4.3 Probability of Selecting a Food Source

In GABC, an onlooker bee chooses a food source with the help of binary tournament selection method where the candidate with better fitness is selected with probability p_{bt}. Also note that SSGGA also uses the binary tournament selection method to select the parents for crossover and mutation.

8.4.4 Determination of a Food Source in the Neighborhood of a Food Source

In order to search a high quality solution, the problem-structure should be exploited effectively so that it leads the search process towards the optimal. As the problem-structure of BMP is grouping-based, therefore, in order to search a neighboring food source, our GABC for BMP is designed in such a way that it can exploit the grouping aspects of BMP as far as possible. In order to determine a food source in the neighborhood of a food source s, we follow two procedures, viz. *Block_Copy* and *Perturbation* in a mutually exclusive way. With probability p_{nbr}, *Block_Copy* is used, otherwise *Perturbation* is used. Both procedures begin by creating a copy s_c of solution s.
Block_Copy: In order to determine a neighboring food source, a solution s_1, which is different from s, is selected randomly. A block bl of maximum size and different from the blocks currently in s_c is selected from s_1. If there are more than one such blocks, then ties are broken arbitrarily. However, if this procedure fails to find even one block in solution s_1 different from the blocks in s_c, then solutions s_c and s_1 are identical and a "collision" occurs. If a collision takes place while determining a new neighboring food

source for an employed bee, then in order to maintain diversity of the population, this employed bee is converted into a scout bee by discarding its associated food source. This scout bee is again made employed by associating it with a new randomly generated solution. However, if a collision takes place in case of an onlooker bee, then we discard this procedure and use *Perturbation* procedure instead.

If no collision occurs, then all blocks of s_c, which contain at least one vertex belonging to *bl* are identified. All such identified blocks along with all singleton blocks are deleted from s_c. All vertices associated with these deleted blocks is added to the list of unassigned vertices. Then, the block *bl* of s_1 is added to s_c. All vertices in *bl* are deleted from the list of unassigned vertices as these vertices are now assigned to s_c. The repair procedure, described in Section 8.3.6, is used to reassign the remaining unassigned vertices to s_c,

Perturbation: This method is similar to mutation operator of SSGGA as described in Section 8.3.7. Only difference is that here parameter nb_p controls the number of non-singleton blocks deleted instead of nb_m.

8.4.5 Other Features

Regarding the scout bees, same rules as used in Chapters 2, 3 and 4 are used here.

8.5 Computational Results

We have implemented our approaches in C and executed these approaches on a Linux based 3.0 GHz Core 2 Duo system with 2 GB RAM. In all computational experiments with SSGGA, we have taken a population of 200 solutions. We have set $p_b = 0.7$, $p_c = 0.8$, $sub = 2$ and $nb_m = 3$. While in case of GABC, we have used a population of 40 bees, i.e., $n_e = 10$ and $n_o = 30$. We have set $limit = 100$, $p_{bt} = 0.8$, $p_{nbr} = 0.6$ and $nb_p = 4$. All parameter values for both the approaches are chosen empirically after a large number of trials. These parameter values provide good results though they may not be optimal for all instances. Both the approaches have been executed 10 independent times on each instance like GGGA [4]. In subsequent subsections, we provide a brief description of problem instances used, and the performance comparison of our GABC and SSGGA with the three best approaches – two ILP approaches [2, 3] and GGGA [4]. The comparison with GGGA is specially important because it is a metaheuristic technique like our approaches. As Tabitha *et al.* [4] compared GGGA with aforementioned ILP approaches, we have also compared our approaches with these approaches.

8.5.1 Problem Instances

In order to compare our approaches with other approaches, we have used a set of 13 problem instances presented in [3, 182, 188, 190, 191]. These problem instances are taken from diverse domains. Problem 1 arises from the study of competitiveness in the English Soccer Premier League [191]. Problem 2 was used by [182] for studying the pattern of elective choices by MBA students. Problem 3, used in [190], is related to the attachment of a group of dwellings to a city. Problems 4-7 arise from the airport performance assessment discussed in [188]. Problems 8 and 9 are related to the performance evaluation of MBA programs [3]. Problem 10 is about partitioning design problems of an Indian village [3]. Problem 11, used in [3], is related to the architectural and urban design pattern language . Problem 12 is a social network analysis problem concerned with the relationship among the board of directors of companies [3]. Last problem, i.e., problem 13 is about the performance of 500 universities worldwide in the year 2005 [3].

8.5.2 Comparisons of our GABC and SSGGA Approaches with ILP Approaches and GGGA

We compare our GABC and SSGGA approaches with two ILP approaches [2, 3] and GGGA [4]. We have reported our results under two different termination conditions. First of all, since GGGA [4] uses a population of 100 solutions and was executed for 20 generations on each instance. Therefore, including initial population, GGGA generates 2100 solutions which is unusually low for an evolutionary algorithm. However, to allow a fair comparison with GGGA, we have also generated roughly the same number of solutions in our approaches for BMP. We have executed GABC for 50 iterations leading to generation of 2010 solutions (10 solutions as initial population, 2000 solutions from 50 iterations). In a similar way for SSGGA, we have used 1900 generations resulting into 2100 solutions (200 solutions as initial population and 1900 solution from 1900 generations).

Table 8.1: Comparisons of GABC and SSGGA with ILP [2][3] and GGGA [4]

					ILP [2]		ILP [3]			GGGA					GABC					SSGGA				
No.	Problem Name	V	E	D	HHI	TTE	HHI	NB	TTE	HHI	NB	AHHI	ANB	ATET	HHI	NB	AHHI	ANB	ATET	HHI	NB	AHHI	ANB	ATET
1	Soccer	20	95	50	78	6.79	78	-	0	72	10	65.0	7.6	0.12	78	6	78.0	6.0	0.01	78	6	78.0	6.0	0.01
2	MBA	30	117	27	136	7.16	136	-	0	136	10	136.0	10.0	0.20	136	10	136.0	10.0	0.01	136	10	136.0	10.0	0.01
3	Dwellings	33	162	31	125	135.34	125	-	0	137	10	136.0	10.0	0.24	125	10	125.0	10.0	0.01	125	10	125.0	10.0	0.02
4	Airport1	40	282	36	280	396.69	280	-	1	280	8	279.6	8.0	0.20	280	8	280.0	8.0	0.01	280	8	280.0	8.0	0.01
5	Airport2	40	408	52	390	35.75	398	-	22	388	6	374.8	6.5	0.18	398	7	398.0	7.0	0.02	398	7	398.0	7.0	0.01
6	Airport3	47	393	36	363	900	369	-	3	367	9	359.2	9.1	0.25	369	9	369.0	9.0	0.01	369	9	369.0	9.0	0.01
7	Airport4	47	569	54	519	152.87	527	-	231	519	6	484.2	6.3	0.21	527	7	527.0	7.0	0.02	527	7	527.0	7.0	0.01
8	FTMBA1	100	2332	24	-	-	1084	15	317	1058	16	1027.6	15.6	0.97	1084	15	1079.20	15.4	0.02	1084	15	1083.0	15.0	0.02
9	FTMBA2	100	3496	35	-	-	1444	11	175	1718	8	1647.0	8.1	0.54	1742	8	1742.0	8.0	0.03	1742	8	1742.0	8.0	0.03
10	Indian Village	141	2806	14	-	-	593	40	0	519	45	505.4	45.8	7.07	577	41	564.2	41.4	0.02	591	40	588.0	41.0	0.05
11	Pattern Language	253	3678	6	-	-	977	-	0	773	96	766.4	96.8	121.29	949	79	928.2	81.9	0.03	969	80	963.8	79.6	0.09
12	FTSE350	309	1076	1	-	-	825	136	5	787	145	778.2	146.0	220.63	819	139	813.8	142.1	0.07	823	140	823.0	140.9	0.25
13	WUAR	500	39418	16	-	-	-	-	-	19078	37	18454.0	38.4	26.45	20344	34	20226.6	34.6	0.08	20342	34	20316.4	34.8	0.08

Table 8.1 reports the results of our approaches under the aforementioned termination conditions along with ILP approaches [2, 3] and GGGA [4]. The descriptions of various columns of this table are as follows: Columns 1 and 2 respectively list the number and the name of each instance. Columns 3-5 provide information about each instance, i.e., the number of vertices (V), the number of edges (E), and the density (D). Columns 6-7 present the solution value (HHI) and the total execution time (TET) on each instance obtained by ILP of [2]. Columns 8-10 present the solution value (HHI), the number of blocks (NB) and the total execution time (TET) of each instance obtained by ILP of [3]. Each set of columns 11-15, 16-20 and 21-25 presents the best solution value (HHI), the number of blocks in the best solution (NB), the average solution value (AHHI), the average number of blocks (ANB), and the average total execution time (ATET) on each instance obtained by GGGA [4], GABC and SSGGA approaches respectively. Data for ILP approaches and GGGA are taken from their respective papers. ILP approach of [2] was executed only on first seven problem instances and it was able to find the feasible solutions for all seven problems out of which for first 4 problems it was able to find proven optimal solutions. ILP approach of [3] was executed on all 13 instances and it was able to find a feasible solution for first 12 instances only. It was able to find optimal solutions for all instances except problems 8, 9 and 13. For these two ILP approaches, unavailable information is indicated by '-' in the corresponding column.

Table 8.1 clearly shows that our GABC and SSGGA are much better than GGGA in terms of solution quality and computational time. Considering all 13 instances, comparing with GGGA, our both approaches are better on 10 problem instances, equal on 2 instance and worse on 1 problem instance in terms of HHI and AHHI. Only on problem instance 3 (Dwellings) our approaches have performed worse. On this problem instance, our approaches have obtained a HHI of 125, whereas Tabitha *et al.* [4] reported a HHI value of 137. However, our personal communication with Proll (author of [2]) confirmed that 125 is indeed the optimal value. Proll further confirmed that the HHI value of problem instance 11 (Pattern Language) is 977 instead of the reported value 797 [3] which is possibly a misprint. Though Tabitha *et al.* [4] used a 2.40 GHz Intel Core 2 Duo for executing the GGGA which is different from the system used for executing GABC and SSGGA. Still, we can safely say that even after compensating for processing speeds, both of our approaches are many times faster than GGGA for BMP.

Table 8.2: Comparisons of GABC and SSGGA with ILP [2][3] and GGGA [4]

					ILP [2]		ILP [3]			GGGA					GABC					SSGGA				
No.	Problem Name	V	E	D	HHI	TTE	HHI	NB	TTE	HHI	NB	AHHI	ANB	ATET	HHI	NB	AHHI	ANB	ATET	HHI	NB	AHHI	ANB	ATET
1	Soccer	20	95	50	78	6.79	78	-	0	72	10	65.0	7.6	0.12	78	6	78.0	6.0	0.30	78	6	78.0	6.0	0.10
2	MBA	30	117	27	136	7.16	136	-	0	136	10	136.0	10.0	0.20	136	10	136.0	10.0	0.31	136	10	136.0	10.0	0.12
3	Dwellings	33	162	31	125	135.34	125	-	0	137	10	136.0	10.0	0.24	125	10	125.0	10.0	0.28	125	10	125.0	10.0	0.17
4	Airport1	40	282	36	280	396.69	280	-	1	280	8	279.6	8.0	0.20	280	8	280.0	8.0	0.40	280	8	280.0	8.0	0.15
5	Airport2	40	408	52	390	35.75	398	-	22	388	6	374.8	6.5	0.18	398	7	398.0	7.0	0.88	398	7	398.0	7.0	0.13
6	Airport3	47	393	36	363	900	369	-	3	367	9	359.2	9.1	0.25	369	9	369.0	9.0	0.39	369	9	369.0	9.0	0.15
7	Airport4	47	569	54	519	152.87	527	-	231	519	6	484.2	6.3	0.21	527	7	527.0	7.0	1.14	527	7	527.0	7.0	0.14
8	FTMBA1	100	2332	24	-	-	1084	15	317	1058	16	1027.6	15.6	0.97	1084	15	1084.0	15.0	0.75	1084	15	1084.0	15.0	0.31
9	FTMBA2	100	3496	35	-	-	1444	11	175	1718	8	1647.0	8.1	0.54	1742	8	1742.0	8.0	1.86	1742	8	1742.0	8.0	0.27
10	Indian Village	141	2806	14	-	-	593	40	0	519	45	505.4	45.8	7.07	591	40	590.8	40.5	1.32	591	40	589.2	40.8	1.21
11	Pattern Language	253	3678	6	-	-	977	-	0	773	96	766.4	96.8	121.29	977	75	974.4	75.4	2.26	977	75	973.8	77.3	3.26
12	FTSE350	309	1076	1	-	-	825	136	5	787	145	778.2	146.0	220.63	825	138	824.6	138.4	3.73	825	140	823.2	140.8	4.99
13	WUAR	500	39418	16	-	-	-	-	-	19078	37	18454.0	38.4	26.45	20344	34	20344.0	34.0	4.68	20344	34	20341.6	34.3	1.13

For example, the maximum ATET of GGGA is 220.63 seconds on FTSE350 instance, whereas the maximum ATET of GABC is 0.07 seconds and the maximum ATET of SSGGA is 0.25 seconds on the same instance. In comparison to ILP approach of [2], our approaches obtained better solutions on 3 instances and same solutions on 4 instances out of a total of 7 instances on which ILP approach of [2] was executed. In comparison to ILP approach of [3], the results of our approaches are better on 2 instances, worse on 3 instances, and same on remaining instances. These ILP approaches were executed on a 2.8 GHz processor which is different from the system used to execute GABC and SSGGA, therefore, no exact comparison of running times is possible. However, our approaches are relatively faster than ILP approach of [2] on all instances and ILP approach of [3] on majority of instances.

To explore the full potential of our approaches, we have executed them under another set of termination conditions. SSGGA is allowed to execute till the best solution fails to improve over 20000 iterations, whereas GABC is allowed to execute till the best solution fails to improve over 2000 generations. Results of our approaches obtained under these termination conditions are reported in Table 8.2 along with those of ILP approaches [2, 3] and GGGA [4]. This table has the same format as Table 8.1. These results clearly demonstrate the superiority of our approaches. With new termination conditions both GABC and SSGGA have improved the results of 3 instances viz. problems 10, 11 and 12. There is no change in statistics from Table 8.1 as far as comparison of solution quality of our approaches with GGGA and ILP of [2] is concerned. In comparison to ILP approach of [3], our approaches have obtained better solutions on 2 instances, worse solution on 1 instance and same solution on remaining instances. The execution time of our approaches increases with new termination conditions. However, even with increased computation times our approaches are faster than all the other approaches on majority of instances.

As far as comparison between SSGGA and GABC is concerned, average solution quality of GABC is slightly better than SSGGA though both have obtained the same best solutions. However, GABC is a bit slower than SSGGA on majority of instances.

8.6 Conclusions

In this chapter, we have presented two metaheuristic approaches viz. a steady-state grouping genetic algorithm (SSGGA) and a grouping-based artificial bee colony (GABC) algorithm for the block model problem. We have compared our approaches against the best approaches reported in the literature. Computational results clearly demonstrate the superiority of our approaches. Our approaches have obtained better quality solutions in shorter time. The superiority of our approaches over GGGA [4] is specially important as all these approaches are metaheuristic approaches. As BMP is an $\mathcal{NP}$-Hard problem, ILP approaches may not be able to find even feasible solutions on larger instances. This is exactly the case with ILP approach of [3] on problem 13 (WUAR). Our approaches provide an attractive alternative in these situations.

Chapter 9

Conclusions and Directions for Future Research

Combinatorial optimization problems have been witnessing a tremendous amount of research due to their practical and theoretical importance. In particular, a number of metaheuristics based on swarm intelligence has been proposed in the last two decades. While some metaheuristics have been proposed originally for optimization in continuous domain and later they have been extended for combinatorial optimization problems, the reverse is true for others. ABC algorithm belongs to the former class of metaheuristics, whereas ACO algorithm belongs to the latter. In fact, ABC algorithm is still under-explored in the domain of combinatorial optimization problems. Through our work, we have explored the potential of ABC algorithm in solving some $\mathcal{NP}$-Hard combinatorial optimization problems. Our research work revolved around designing ABC algorithms for six different $\mathcal{NP}$-Hard combinatorial optimization problems and making them robust and competitive with respect to the other state-of-the-art metaheuristic approaches as far as possible. We have also developed variants of ACO algorithms for three $\mathcal{NP}$-Hard combinatorial optimization problems. These are the major contributions of this thesis. In addition, we have developed new problem-specific heuristics and local search procedures for some problems and extended the already existing local search procedures for some others. A genetic algorithm is also developed for blockmodel problem.

In the following, we describe the contributions made by various chapters along with possible directions for future research.

In chapter 2, we have proposed an ABC algorithm based approach for minimum routing cost spanning tree problem (MRCST) and compared it with the best heuristic approaches reported in the literature. On the benchmark instances considered, our approach has outperformed other approaches in terms of solution quality. Except for two previously proposed approaches, our approach is faster than other approaches.

The method that we have used in this chapter for determining the neighboring solutions extends the method used in [85]. Like [85], it is based on the concept that if a solution component is present in one good solution, then it is highly likely that the same component is present in many good solutions. However, unlike [85], which selects a solution component randomly for insertion from all candidate components, it uses a greedy criterion and selects a solution component which leads to a solution of least cost from all candidate components. Similar greedy methods can be developed in case of other subset selection problems, where fitness can be evaluated quickly. ABC approach developed in this chapter can be easily extended for optimal communication spanning tree problem (OCST) which is a generalization of MRCST.

Chapter 3 described an ABC algorithm based approach for quadratic minimum spanning tree problem (Q-MST) which is similar to the ABC approach for MRCST. We have compared our approach with the best approaches known so far. Our approach obtains better quality solutions than other approaches. Though one among proposed approaches in the literature is faster than our approach, this approach performs worst in terms of solution quality among all the approaches.

Ideas presented in Chapters 2 and 3 can be easily adapted for developing ABC approaches to other $\mathcal{NP}$-Hard constrained spanning tree problems such as degree-constrained minimum spanning tree problem (DCMST), bounded-diameter minimum spanning tree problem (BDMST) etc.

Chapter 4 addressed set covering problem (SCP) through a hybrid approach comprising an ABC algorithm and a local search. We have extended an already existing greedy local search for SCP and used it within ABC algorithm to improve the neighboring solutions in both employed and onlooker bee phases. Our hybrid approach is competitive with other best approaches in terms of solution quality though it is slower than most. Except for one population-based approach, our approach is better than all population-based approaches in terms of solution quality.

For determining a neighboring solution, we first add some columns from another randomly chosen solution and then delete some columns. This is followed by procedures for repairing the solution and removing the redundant columns. This is computationally much more expensive in comparison to the methods used in previous two chapters. Actually, the nature of SCP is such that good neighboring solutions cannot be generated simply by deleting a column and inserting in its place another column. Columns interact with each other in a more complex way in SCP than edges in case of first two problems. The hybrid approach developed in this chapter can be modified easily for target coverage problems in wireless sensor networks and set packing problem.

Chapter 5 describes three approaches for dominating tree problem (DTP). First a problem-specific heuristic has been proposed which produces much better results in comparison with existing problem-specific heuristics for this problem. With the intent of improving the solution quality even further, two swarm intelligence techniques viz. an ABC algorithm and an ACO algorithm have been proposed. These two techniques obtain much better results at the expense of increased computation time. Performance of these two techniques are comparable to each other in terms of solution quality. Though ACO algorithm produces slightly better results, it is several times slower than ABC algorithm on large instances for this problem. Solutions obtained through our approaches also contain fewer number of dominating vertices which is a secondary quality criterion and is desirable for many applications.

In ABC algorithm, determining a neighboring solution for DTP is based on two procedures, i.e., PDE procedure and PANDV procedure which are applied in a mutually exclusive way. Note that it is quite different from the methods of determining neighboring solutions for MRCST, Q-MST and SCP. In order to generate a solution in the neighborhood of a given solution, the concept of utilizing solution components from another solution, which is applied in case of MRCST, Q-MST and SCP, is not applied for DTP. The reason behind this is that a dominating tree consists of a subset of vertices of the graph rather than the complete set of vertices of the graph, and sets of dominating vertices of two different dominating trees may be quite different. In such cases, there may not be even a single candidate edge in another dominating tree different from the deleted edge of a given dominating tree, and therefore, we cannot use a method similar to MRCST and Q-MST for DTP. Similar methods for determining a neighboring solution can be designed for other related $\mathcal{NP}$-Hard problems.

Chapter 6 is concerned with finding two special spanning trees on a connected graph – one with minimum number of branch vertices (MBV) and the other with minimum degree sum of branch vertices (MDS), where a branch vertex is defined as a vertex with degree greater than two. For each problem, we have presented two approaches viz. a problem-specific heuristic and a hybrid ACO algorithm. Our problem-specific heuristic approaches have outperformed the existing problem-specific heuristic approaches for these problems. As the results obtained by all these problem-specific heuristics are inferior in comparison to exact approaches and the gap in solution quality increases with increase in instance size, we have developed the hybrid ACO approaches. Computational results show the effectiveness of our hybrid ACO approaches.

Our ACO approaches use the same framework as used in the problem-specific heuristics. The only difference is that edges and branch vertices are chosen using pheromones. A novel feature of our hybrid ACO approaches is the use of two types of pheromones. The first type of pheromone is associated with edges of the graph, whereas the second type of pheromone is associated with vertices of the graph. Actually, two factors play a crucial role while constructing a solution to these problems. First one which edge should be selected and second one which vertex should be selected as a branch vertex, if there exists any, while constructing a spanning tree. The pheromone values associated with edges help in identifying good edges in constructing a spanning tree, whereas pheromone values associated with vertices help in choosing promising vertices as branch vertices. To our knowledge, [162] is the only other work, where the concept of multiple pheromones was used for a single objective problem. Cordone *et al.* [162] described a problem of partitioning the vertices of a graph into p trees, where p different pheromone values are associated with each vertex. Each of these p pheromone values associated with a vertex indicates the desirability of that vertex to a particular tree. This is quite different from the way multiple pheromones are used in our approaches. Approaches similar to our ACO approaches can be designed for other problems also, where more than one factor is responsible for constructing a good solution. In particular, such approaches can be beneficial for other graph problems also, where the choice of vertices and choice of edges are equally important in construction of good solutions.

Chapter 7 presented a hybrid ABC algorithm for a single machine early/tardy scheduling problem where no unforced machine idle time is allowed. A local search is used inside ABC algorithm to improve the schedules obtained through it. A variant of this basic hybrid ABC approach, referred to as ABC-MNAI, is also presented in this chapter, where the best solution obtained by hybrid ABC approach is improved further by using an exhaustive local search. Performance of these two approaches are evaluated by comparing them with 16 previously proposed approaches on existing set of benchmark instances as well as on a new set of large instances. Though the basic ABC approach has performed better than all the 16 approaches on existing set of instances, it has performed worse on two largest groups of instances containing 750 and 1000 jobs in comparison to one approach. ABC-MNAI, on the other hand, is superior to all other approaches in terms of solution quality on both existing as well as on new groups of instances. In comparison to other best performing approaches, ABC-MNAI also requires less execution time on most group of instances. Our approaches can be easily extended for other versions of early/tardy scheduling problem.

For this permutation problem, we have generated the neighboring solutions in two ways, i.e., through multi-point insert operator and through 3-point swap operator. While multi-point insert operator is responsible for exploitation, 3-point swap operator, which can be seen as a perturbation strategy, is responsible for exploration. These operators are used in a mutually exclusive manner. The multi-point insert operator utilizes solution components from another solution to generate a solution in the neighborhood of a given solution. This operator is based on the fact that if a job is present at a particular position in one good solution, then it is highly likely that the same job is present exactly at the same position or around the same position in many good solutions. The previously proposed methods [177, 178] for generating a neighboring solution for permutation problems have not utilized components from another solution and relied on perturbing a solution via insert and swap operators to generate its neighboring solutions. Our approach provides an attractive alternative to these methods. Different variations of multi-point insert operators can be designed depending on the nature of permutation problem under investigation, i.e., whether it is an absolute permutation problem or a relative permutation problem or a mixture of both.

Chapter 8 described two approaches viz. a steady-state grouping genetic algorithm (SSGGA) and a grouping based artificial bee colony (GABC) algorithm for blockmodel problem. The objective of this problem is to find small number of large blocks containing structural similarities or equivalences among entities in a given graph representing a complex network. We have compared our approaches against the existing best approaches on 13 standard benchmark instances for this problem. Computational results have shown the superiority of our approaches in terms of solution quality and execution time both. Though both GABC and SSGGA have obtained the same best solutions, GABC is better in terms of average solution quality at the expense of slightly more computation times.

The method that we have used for determining the neighboring solutions in GABC is designed by keeping in mind the grouping structure of BMP. This method tries to preserve grouping information as far as possible. To our knowledge, GABC is the first approach based on ABC algorithm for a grouping problem with variable number of groups. Therefore, the success of GABC in solving blockmodel problem demonstrates the applicability of ABC algorithm in a new domain. Similar ABC approaches can be designed for other grouping problems also.

In all, 8 problems have been considered in this thesis. Chapters 2 to 6 deal with subset selection problems, chapter 7 addresses a permutation problem and chapter 8 focuses on a grouping problem. We have developed ABC algorithm based approaches for 5 subset selection problems, 1 permutation problem and 1 grouping problem. In essence, based on the work reported in this thesis, we can say that a properly designed ABC algorithm based approach can compete with any other state-of-the-art metaheuristic approaches for any category of $\mathcal{NP}$-Hard combinatorial optimization problems. While designing ABC algorithm for a problem, particular attention should be given to the method for determining neighboring solutions as it plays a crucial role in finding high quality solutions.

It is well-known that ACO is one among most successful metaheuristic approaches. Our work on ACO approaches for three different $\mathcal{NP}$-Hard combinatorial optimization problems – DTP, MBV and MDS – support this basis, as ACO approaches are able to find high quality solutions for these problems. Moreover, our ACO approaches are also the first metaheuristic approaches for these problems. Therefore, these approaches along with ABC algorithm for DTP will serve as the baseline for any future metaheuristic approaches for these problems. All our ACO approaches are incorporated into frameworks provided by problem-specific constructive heuristics (see Section 5.3.1 for DTP, and Section 6.2 for MBV and Section 6.4 MDS). This again stresses the importance of incorporating ACO approaches into the framework provided by good constructive heuristic. In the absence of a good constructive heuristic, probabilistic decision rule and pheromone update rule alone cannot generate good solutions.

Succinctly, it would be apt to say that despite the work presented in this thesis, there might be many unexplored territories, particularly in ABC algorithm, that need to be unearthed and studied further. These cases, in turn, will provide us with innumerable opportunities to further improve ABC algorithm and other techniques.

References

[1] Z.G. Ren, Z.R. Feng, L.J. Ke, and Z.J. Zhang. **New ideas for applying ant colony optimization to the set covering problem**. *Computers & Industrial Engineering*, **58**:774–784, 2010. (vi, 54, 57, 59, 60, 61, 62, 67, 72)

[2] L. Proll. **ILP approach to the blockmodel problem**. *European Journal of Operational Research*, **177**:840–850, 2007. (viii, 134, 135, 136, 137, 142, 143, 144, 145, 146, 147)

[3] A. Jessop, L. Proll, and B.M. Smith. **Optimal cliques: applications and solutions**, 2007. Technical Report, University of Leeds, U.K. Available online at `http://www.comp.leeds.ac.uk/research/pubs/reports/2007/2007_03.pdf`. (viii, 134, 135, 137, 142, 143, 144, 145, 146, 147, 148)

[4] T. James, E. Brown, and C.T. Ragsdale. **Grouping genetic algorithm for the blockmodel problem**. *IEEE Transactions on Evolutionary Computation*, **14**:103–111, 2010. (viii, 135, 137, 138, 142, 143, 144, 145, 146, 147, 148)

[5] L.V. Kantorovich. **Mathematical methods of organizing and planning production**. *Management Science*, **6**:366–422, 1939. (2)

[6] L.V. Kantorovich. **George B. Dantzig: Operations Research Icon**. *Operations Research*, **53**:892–898, 2005. (2)

[7] M. Dorigo and T. Stützle. *Ant colony optimization.* MIT Press, Cambridge, MA, 2004. (5, 6, 19)

[8] J.H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975. (6, 135)

[9] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989. (6, 14)

[10] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. **Optimization by simulated annealing**. *Science*, **220**:671–680, 1983. (6)

[11] V. Černý. **Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm**. *Journal of Optimization Theory and Applications*, **45**:41–51, 1985. (6)

[12] 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. (6, 19, 20)

[13] D. Karaboga. **An idea based on honey bee swarm for numerical optimization. Technical Report TR06**, 2005. Computer Engineering Department, Erciyes University, Turkey. (6, 9, 10, 14, 135)

[14] F. Glover. **Tabu search - part 1**. *ORSA Journal on Computing*, **1**:190–206, 1989. (6)

[15] F. Glover. **Tabu search - part 2**. *ORSA Journal on Computing*, **2**:4–32, 1990. (6)

[16] G. Beni and J. Wang. **Swarm intelligence in cellular robotic systems, proceed**, 1989. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy. (6)

[17] E. Bonabeau, M. Dorigo, and G.Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press, New York, 1999. (6)

[18] T. Seeley and P. Visscer. **Group decision making in nest-site selection by honey bees**. *Apidologie*, **35**:101–116, 2006. (8)

[19] K.V. Frisch. *The dancing bees: an account of the life and senses of honey bee.* Harcourt, Brace, 1953. (8)

[20] K.V. Frisch and M. Lindauer. **The "language" and orientation of the honey bee**. *Annual Review of Entomology*, **1**:45–58, 1956. (8)

[21] T. Seeley. *Honeybee ecology: a study of adaptation in social life.* Princeton University Press, Princeton, 1985. (8)

[22] R. Menzel, R.J. De Marco, and U. Greggers. **Spatial memory, navigation and dance behaviour in Apis mellifera**. *Journal of Comparative Physiology A*, **192**:889–903, 2006. (8)

[23] K. Hamdan. **How do bees make honey**, 2008. Bee Research Unit, National Center for Agriculture Research and Technology Transfer, bee. (NCARTT). (9)

[24] D.G. Mackean. **The honey bee (Apis mellifera)**, 2008. Resources for Biology Education. (9)

[25] D. Karaboga and B. Akay. **A survey: algorithms simulating bee swarm intelligence**. *Artificial Intelligence Review*, **31**:61–85, 2009. (9, 16)

[26] S.H. Jung. **Queen-bee evolution for genetic algorithms**. *Electronics Letters*, **39**:575–576, 2003. (9)

[27] L.D. Qin, Q.Y. Jiang, Z.Y. Zou, and Y.J. Cao. **A queen-bee evolution based on genetic algorithm for economic power dispatch**. In *39th International Universities Power Engineering Conference*, **1**, pages 453–456. IEEE, 2004. (9)

[28] M.F. Azeem and A.M. Saad. **Modified queen bee evolution based genetic algorithm for tuning of scaling factors of fuzzy knowledge base controller**. In *Proceedings of the IEEE INDICON*, pages 299–303. IEEE, 2004. (9)

[29] A. Karci. **Imitation of bee reproduction as a crossover operator in genetic algorithms**. In *PRICAI 2004, Lecture Notes in Artificial Intelligence*, **3157**, pages 1015–1016. Springer, 2004. (9)

[30] C. Xu, Q. Zhang, J. Li, and X. Zhao. **A bee swarm genetic algorithm for the optimization of DNA encoding**. In *3rd International Conference on Innovative Computing, Information and Control*, pages 35–35. IEEE, 2008. (9)

[31] X. Lu and Y. Zhou. **A genetic algorithm based on multi-bee population evolutionary for numerical optimization**. In *7th World Congress on Intelligent Control and Automation*, pages 1294–1298. IEEE, 2008. (9)

[32] T. Sato and M. Hagiwara. **Bee system: finding solution by a concentrated search**. In *International Conference on Computational Cybernetics and Simulation*, **4**, pages 3954–3959. IEEE, 1997. (9)

[33] N. Gordon, I.A. Wagner, and A.M. Bruckstein. **Discrete bee dance algorithm for pattern formation on a grid**. In *International Conference on Intelligent Agent Technology*, pages 545–549. IEEE, 2003. (9)

[34] H.F. Wedde, M. Farooq, and Y. Zhang. **BeeHive: an efficient fault-tolerant routing algorithm inspired by honey bee behavior**. In *ANTS 2004, Lecture Notes in Computer Science*, **3172**, pages 83–94. Springer, 2004. (9)

[35] P. Navrat and M. Kovacik. **Web search engine as a bee hive**. In *International Conference on Web Intelligence*, pages 694–701. IEEE, 2006. (9)

[36] G. Olague and C. Puente. **The honeybee search algorithm for three-dimensional reconstruction**. In *EvoWorkshops 2006, Lecture Notes in Computer Science*, **3907**, pages 427–437. Springer, 2006. (9)

[37] S. Nakrani and C. Tovey. **On honey bees and dynamic server allocation in internet hosting centers**. *Adaptive Behavior*, **12**:223–240, 2004. (9)

[38] S. Nakrani and C. Tovey. **From honeybees to internet servers: biomimicry for distributed management of internet hosting centers**. *Bioinspiration & Biomimetics*, **2**:182–197, 2007. (9)

[39] S. Sadik, A. Ali, H.F. Ahmad, and H. Suguri. **Honey bee teamwork architecture in multi-agent systems**. In *CSCWD 2006, Lecture Notes in Computer Science*, **4402**, pages 428–437. Springer, 2007. (9)

[40] A. Gupta and N. Koul. **Swan: a swarm intelligence based framework for network management of IP networks**. In *International Conference on Computational Intelligence and Multimedia Applications*, **1**, pages 114–118. IEEE, 2007. (9)

[41] Y. Yonezawa and T. Kikuchi. **Ecological algorithm for optimal ordering used by collective honey bee behavior**. In *Proceedings of the Seventh International Symposium on Micro Machine and Human Science*, pages 249–256. IEEE, 1996. (9)

[42] K. Passino. **Systems biology of group decision making**. In *14th Mediterranean Conference on Control and Automation*, pages 1–1. IEEE, 2006. (9)

[43] R.L.Z. Gutierrez and M. Huhns. **Multiagent-based fault tolerance management for robustness**. In *Robust intelligent systems*, pages 23–41. Springer, 2008. (9)

[44] H.A. Abbass. **MBO: marriage in honey bees optimization a haplometrosis polygynous swarming approach**. In *Proceedings of the 2001 Congress on Evolutionary Computation*, **1**, pages 207–214. IEEE, 2001. (9)

[45] J. Teo and H.A. Abbass. **An annealing approach to the mating-flight trajectories in the marriage in honey bees optimization algorithm**, 2001. Technical Report CS04/01, School of Computer Science, University of New South Wales at ADFA. (9)

[46] K. Benatchba, L. Admane, and M. Koudil. **Using bees to solve a data-mining problem expressed as a max-sat one**. In *IWINAC 2005, Lecture Notes in Computer Science*, **3562**, pages 212–220. Springer, 2005. (9)

[47] M. Koudil, K.Benatchba, A. Tarabet, and E.B. Sahraoui. **Using artificial bees to solve partitioning and scheduling problems in codesign**. *Applied Mathematics and Computation*, **186**:1710–1722, 2007. (9)

[48] P. Curkovic and B. Jerbic. **Honey-bees optimization algorithm applied to path planning problem**. *International Journal of Simulation Modelling*, **6**:154–165, 2007. (9)

[49] H.S. Chang. **Converging marriage in honey-bees optimization and application to stochastic dynamic programming**. *Journal of Global Optimization*, **35**:423–441, 2006. (9)

[50] A. Afshar, O.B. Haddad, and M.A. Marino. **Converging marriage in honey-bees optimization and application to stochastic dynamic programming**. *Journal of the Franklin Institute*, **344**:452–462, 2007. (9)

[51] B. Amiri and M. Fathian. **Integration of self organizing feature maps and honey bee mating optimization algorithm for market segmentation**. *Journal of Theoretical and Applied Information Technology*, **3**:70–86, 2007. (9)

[52] C. Yang, J. Chen, and X. Tu. **Algorithm of fast marriage in honey bees optimization and convergence analysis**. In *International Conference on Automation and Logistics*, pages 1794–1799. IEEE, 2007. (9)

[53] Y. Marinakis, M. Marinaki, and G. Dounias. **Honey bees mating optimization algorithm for the vehicle routing problem**. In *SCI*, **129**, pages 139–148. Springer, 2008. (9)

[54] T. Niknam, J. Olamaie, and R. Khorshidi. **A hybrid algorithm based on HBMO and fuzzy set for multi-objective distribution feeder reconfiguration**. *World Applied Sciences Journal*, **4**:308–315, 2008. (9)

[55] D. Ashlock and J. Oftelie. **Simulation of floral specialization in bees**. In *Proceedings of the 2004 Congress on Evolutionary Computation*, **2**, pages 1859–1864. IEEE, 2004. (9)

[56] T. Niknam, J. Olamaie, and R. Khorshidi. **Pheromone communication in a robot swarm: necrophoric bee behaviour and its replication**. *Robotica*, **23**:731–742, 2005. (9)

[57] G.M. Bianco. **Getting inspired from bees to perform large scale visual precise navigation**. In *International conference on Intelligent Robots and Systems*, pages 619–624. IEEE, 2004. (9)

[58] N. Lemmens, S.D. Jong, K. Tuyls, and A. Nowe. **A bee algorithm for multi-agent systems: recruitment and navigation combined**. In *Proceedings of ALAG 2007, an AAMAS 2007 workshop*, Honolulu, Hawai, 2007. (9)

[59] A. Walker, J. Hallam, and D. Willshaw. **Bee-havior in a mobile robot: the construction of a self-organized cognitive map and its use in robot navigation within a complex, natural environment**. In *International conference on Neural Networks*, **3**, pages 1451–1456. IEEE, 1993. (9)

[60] D.J.T. Sumpter and D.S. Broomhead. **Formalising the link between worker and society in honey bee colonies**. In *MABS 1998, Lecture Notes in Artificial Intelligence*, **1534**, pages 95–110. Springer, 1998. (9)

[61] P. Lučić and D. Teodorović. **Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence**. In *Preprints of the Tristan IV Triennial Symposium on Transportation Analysis*, pages 441–445, Sao Miguel, Azores Islands, Portugal, 2001. (9)

[62] G.Z. Marković, D.B. Teodorović, and V.S. Aćimović-Raspopović. **Routing and wavelength assignment in all-optical networks based on the bee colony optimization**. *AI Communications – Network Analysis in Natural Sciences and Engineering*, **20**:273–285, 2007. (9)

[63] V. Vassiliadis and G. Dounias. **Nature inspired intelligence for the constrained portfolio optimization problem**. In *SETN 2008, Lecture Notes in Artificial Intelligence*, **5138**, pages 431–436. Springer, 2008. (9)

[64] S. Banarjee, G.S. Dangayach, S.K. Mukherjee, and P.K. Mohanti. **Modelling process and supply chain scheduling using hybrid metaheuristics**. In *SCI*, **128**, pages 277–300. Springer, 2008. (9)

[65] L.-P. Wong, M.Y.H. Low, and C.S. Chong. **A bee colony optimization algorithm for traveling salesman problem**. In *Second Asia International Conference on Modeling & Simulation*, pages 818–823. IEEE, 2008. (9)

[66] V. Tereshko. **Reaction-diffusion model of a honeybee colony's foraging behaviour**. In *PPSN VI 2000, Lecture Notes in Computer Science*, **1917**, pages 807–816. Springer, 2000. (9)

[67] S. Ghosh and I.W. Marshall. **Simple model of collective decision making during nectar source selection by honey bees**. In *Workshop on Memory and*

Learning Mechanisms in Autonomous Robots, Canterbury, United Kingdom, 2005. (9)

[68] R.L. Walker. **Honeybee search strategies: adaptive exploration of an information ecosystem**. In *Proceedings of the 2004 Congress on Evolutionary Computation*, **1**, pages 1209–1216. IEEE, 2004. (9)

[69] H.R. Wedde and M. Farooq. **The wisdom of the hive applied to mobile ad-hoc networks**. In *IEEE Swarm Intelligence Symposium*, pages 341–348. IEEE, 2005. (9)

[70] X.-S. Yang. **Engineering optimizations via nature-inspired virtual bee algorithms**. In *IWINAC 2005, Lecture Notes in Computer Science*, **3562**, pages 317–323. Springer, 2005. (9)

[71] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. **The bees algorithm**, 2005. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK. (9)

[72] H.A.A. Bahamish, R. Abdullah, and R.A. Salam. **Protein conformational search using bees algorithm**. In *Second Asia International Conference on Modeling & Simulation*, pages 911–916. IEEE, 2008. (9)

[73] K. Guney and M. Onay. **Bees algorithm for design of dual-beam linear antenna arrays with digital attenuators and digital phase shifters**. *International Journal of RF and Microwave Computer–Aided Engineering*, **18**:337–347, 2008. (9)

[74] J.Y. Lee and A.H. Darwish. **Multi-objective environmental/economic dispatch using the bees algorithm with weighted sum**. In *Proceedings of the EU-Korea Conference on Science and Technology, Springer Proceedings in Physics*, **124**, pages 267–274. Springer, 2008. (9)

[75] H. Drias, S. Sadeg, and S. Yahi. **Cooperative bees swarm for solving the maximum weighted satisfiability problem**. In *IWANN 2005, Lecture Notes in Computer Science*, **3512**, pages 417–448. Springer, 2005. (9)

[76] C.S. Chong, A.I. Sivakumar, M.Y.H. Low, and K.L. Gay. **A bee colony optimization algorithm to job shop scheduling**. In *Proceedings of the 2006 Winter Simulation Conference*, pages 1954–1961. IEEE, 2006. (9)

[77] N. Quijano and K.M. Passino. **Honey bee social foraging algorithms for resource allocation, part I: algorithm and theory**. In *Proceedings of the American Control Conference*, pages 3389–3394. IEEE, 2007. (9)

[78] X. Lu and Y. Zhou. **A novel global convergence algorithm: bee collecting pollen algorithm**. In *ICIC 2008, Lecture Notes in Artificial Intelligence*, **5227**, pages 518–525. Springer, 2008. (9)

[79] S.Y. Ko, I. Gupta, and Y. Jo. **A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing**. *ACM Transactions on Autonomous and Adaptive Systems*, **3**:1–34, 2008. (9)

[80] 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. (10)

[81] 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. (10, 55)

[82] 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. (10)

[83] D. Karaboga and B. Basturk. **On the performance of artificial bee colony (ABC) algorithm**. *Applied Soft Computing*, **8**:687–697, 2008. (10)

[84] D. Karaboga and B. Akay. **A modified Artificial Bee Colony (ABC) algorithm for constrained optimization problems**. *Applied Soft Computing*, **11**:3021–3031, 2011. (10)

[85] A. Singh. **An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem**. *Applied Soft Computing*, **9**:625–631, 2009. (10, 13, 22, 31, 32, 33, 55, 56, 150)

[86] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, and T.J. Chua. **A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem**. *Information Sciences*, **181**:2455–2468, 2011. (10, 14)

[87] S. Sundar and A. Singh. **A swarm intelligence approach to the quadratic multiple knapsack problem**. In *ICONIP 2010, Part I, Lecture Notes in Computer Science*, **6443**, pages 626–633. Springer, 2010. (10, 14, 135)

[88] W. Gao and S. Liu. **Improved artificial bee colony algorithm for global optimization**. *Information Processing Letters*, **111**:871–882, 2011. (10)

[89] W.-F. Gao and S.-Y Liu. **A modified artificial bee colony algorithm**. *Computers and Operations Research*, **39**:687–697, 2012. (10)

[90] B. Akay and D. Karaboga. **A modified artificial bee colony algorithm for real-parameter optimization**. *Information Sciences*, 2010. In Press, DOI: 10.1016/j.ins.2010.07.015. (10)

[91] D.E. Goldberg and K. Deb. **A comparative analysis of selection schemes used in genetic algorithms**. In *FOGA*, pages 69–93. Morgan Kaufmann, 1990. (14)

[92] B. Hölldobler and E.O. Wilson. *The Ants*. Harvard University Press, 1990. (16)

[93] S. Goss, S. Aron, J.-L. Deneubourg, S. Aron, and J.-M. Pasteels. **Self-organized shortcuts in the Argentine ant**. *Naturwissenschaften*, **76**:579–581, 1989. (18)

[94] J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. **The self-organizing exploratory pattern of the Argentine ant**. *Journal of Insect Behavior*, **3**:159–168, 1990. (18)

[95] M. Dorigo. **Optimization, Learning and Natural Algorithms [In Italian]**, 1992. PhD thesis, Departimento di Elettronica, Politecnico di Milano, Milan. (19, 20)

[96] A. Colorni, M. Dorigo, and V. Maniezzo. **Distributed optimization by ant colonies**. In *Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, 1992. (19, 20)

[97] M. Dorigo and L. M. Gambardella. **Ant colonies for the traveling salesman problem**. *BioSystems*, **43**:73–81, 1997. (20)

[98] M. Dorigo and L. M. Gambardella. **A cooperative learning approach to the traveling salesman problem**. *IEEE Transactions on Evolutionary Computation*, **1**:53–66, 1997. (20)

[99] T. Stützle and H. H. Hoos. **Improving the ant system: A detailed report on the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System**, 1996. Technical report AIDA-96-12, FG Intellektik, FB Informatic, TU Darmstadt, Germany. (21, 98, 105)

[100] T. Stützle and H. H. Hoos. **$\mathcal{MAX}$-$\mathcal{MIN}$ ant system**. *Future Generation Computer Systems*, **16**:889–914, 2000. (21, 98, 105)

[101] D. S. Johnson, J. K. Lenstra, and A. H. G. Roonoy Kan. **The complexity of network design problem**. *Networks*, **8**:279–285, 1978. (27)

[102] B.A. Julstrom. **The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem**. In *Proceedings of the Genetic and Evolutionary Computation Conference*, **1**, pages 585–590. ACM, 2005. (27, 29, 30, 33, 35, 37, 40)

[103] B.Y. Wu and K.M. Chao. *Spanning trees and optimization problems.* CRC Press, New York, 2004. (28)

[104] M. Ricardo R. Campos. **A fast algorithm for computing the minimum routing cost spanning tree**. *Computer Networks*, **52**:3229–3247, 2008. (28)

[105] R. Wong. **Worst case analysis of network design problem heuristics**. *SIAM Journal of Algebraic Discrete Mathematics*, **1**:51–63, 1980. (28)

[106] B.Y. Wu, G. Lancia, V. Bafna, K.M. Chao, R. Ravi, and C.Y. Tang. **A polynomial time approximation schemes for minimum routing cost spanning trees**. *SIAM Journal on Computing*, **29**:761–768, 1999. (28)

[107] V. Grout. **Principles of cost minimization in wireless networks**. *Journal of Heuristics*, **11**:115–133, 2005. (28)

[108] M. Fischetti, G. Lancia, and P. Serafini. **Exact algorithms for minimum routing cost trees**. *Networks*, **39**:161–173, 2002. (28)

[109] S. Picciotto. **How to Encode a Tree**, 1999. PhD Thesis, University of California, San Diego. (29)

[110] G.R. Raidl and B.A. Julstrom. **Edge sets: an effective evolutionary coding of spanning trees**. *IEEE Transactions on Evolutionary Computation*, **7**:225–239, 2003. (29, 30, 42)

[111] J.B. Kruskal. **On the shortest path spanning subtree of a graph and the traveling salesman problem**. *Proceedings of the American Mathematical society*, **7**:48–50, 1956. (29, 75)

[112] B. A. Julstrom. **A genetic algorithm and two hill climbers for the minimum routing cost spanning tree problem**. In *Proceedings of the International Conference on Artificial Intelligence*, **3**, pages 934–940. CSREA Press, 2002. (29)

[113] A. Singh. **A new heuristic for the minimum routing cost spanning tree problem**. In *International Conference on Information Technology*, pages 9–13. IEEE, 1999. (29, 30, 35, 37, 40)

[114] R.C. Prim. **Shortest connection networks and some generalizations**. *Bell Systems Technical Journal*, **36**:1389–1401, 1957. (30, 43, 80)

[115] A.V. Aho and J.D. Ullman J.E. Hopcroft. *Data structures and algorithms.* Addison-Wesley, New York, 1983. (33)

[116] G. Zhou and M. Gen. **An effective genetic algorithm approach to the quadratic minimum spanning tree problem**. *Computers and Operations Research*, **25**:229–237, 1998. (42, 46, 47, 52)

[117] A. Assad and W. Xu. **The quadratic minimum spanning tree problem**. *Naval Research Logistics*, **39**:399–417, 1992. (42)

[118] W. Xu. **On the quadratic minimum spanning tree problem**. In *Proceedings of Japan-China International Workshops on Information Systems*, pages 141–148, 1995. (42)

[119] H. Prüfer. **Neuer beweis eines satzes über permutationen**. *Archiv für Mathematik and Physik*, **27**:742–744, 1918. (42)

[120] S.-M. Soak, D.W. Corne, and B.-H. Ahn. **The edge-window-decoder representation for tree-based problems**. *IEEE Transactions on Evolutionary Computation*, **10**:124–144, 2006. (42, 46, 49, 52)

[121] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman & Company, San Francisco, 1979. (54)

[122] B.M. Smith. **Impacs–a bus crew scheduling system using integer programming**. *Mathematical Programming*, **42**:181–187, 1988. (54)

[123] A. Caprara, M. Fischetti, and P. Toth. **A heuristic method for the set covering problem**. *Operations Research*, **47**:730–743, 1999. (54, 62, 70)

[124] C. Toregas, R. Swain, and L. Bergman C. Revelle. **The location of emergency service facilities**. *Operations Research*, **19**:1363–1373, 1971. (54)

[125] E. Boros, P.L. Hammer, T. Ibaraki, and A. Kogan. **Logical analysis of numerical data**. *Mathematical Programming*, **79**:163–190, 1997. (54)

[126] F.J. Vasko, F.E. Wolf, and K.L. Stott. **A set covering approach to metallurgical grade assignment**. *European Journal of Operational Research*, **38**:27–34, 1989. (54)

[127] B.A. Foster and D.M. Ryan. **An integer programming approach to the vehicle scheduling problem**. *Operational Research*, **27**:367–384, 1976. (54)

[128] M.L. Fisher and M.B. Rosenwein. **An interactive optimization system for bulk-cargo ship scheduling**. *Naval Research Logistics*, **36**:27–42, 1989. (54)

[129] M.L. Fisher and P. Kedia. **Optimal solution of set covering/partitioning problems using dual heuristics**. *Management Science*, **36**:674–688, 1990. (54)

[130] J.E. Beasley and K. JØrnsten. **Enhancing an algorithm for set covering problems**. *European Journal of Operational Research*, **58**:293–300, 1992. (54, 62)

[131] J.E. Beasley. **An algorithm for set covering problem**. *European Journal of Operational Research*, **31**:85–93, 1987. (54)

[132] J.E. Beasley. **A Lagrangian heuristic for set covering problems**. *Naval Research Logistics*, **37**:151–164, 1990. (54)

[133] E. Balas and M. Carrera. **A dynamic subgradient-based branch-and-bound procedure for set covering**. *Operations Research*, **44**:875–890, 1996. (54)

[134] J.E. Beasley and P.C. Chu. **A genetic algorithm for the set covering problem**. *European Journal of Operational Research*, **94**:392–404, 1996. (54, 55, 57, 62)

[135] U. Aickelin. **An indirect genetic algorithm for set covering problems**. *Journal of the Operational Research Society*, **53**:1118–1126, 2002. (54, 62, 70)

[136] L. Lessing, I. Dumitrescu, and T. Stützle. **A comparison between ACO algorithms for the set covering problem**. In *ANTS 2004, Lecture Notes in Computer Science*, **3172**, pages 1–12. Springer, 2004. (54, 63, 67, 72, 73)

[137] M.J. Brusco, L.W. Jacobs, and G.M. Thompson. **A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems**. *Annals of Operations Research*, **86**:611–627, 1999. (54, 57, 62)

[138] S. Ceria, P. Nobili, and A. Sassano. **A Lagrangian-based heuristic for large-scale set covering problems**. *Mathematical Programming*, **81**:215–228, 1998. (54, 62)

[139] M. Yagiura, M. Kishida, and T. Ibaraki. **A 3-flip neighborhood local search for the set covering problem**. *European Journal of Operational Research*, **172**:472–499, 2006. (54, 62, 67)

[140] G. Lan, G.W. Depuy, and G.E. Whitehouse. **An effective and simple heuristic for the set covering problem**. *European Journal of Operational Research*, **176**:1387–1403, 2007. (54, 62, 70)

[141] M. Haouari and J.S. Chaouachi. **A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem**. *Journal of the Operational Research Society*, **53**:792–799, 2002. (54, 63)

[142] M. Caserata. **Tabu search-based metaheuristic algorithm for large-scale set covering problems**. *Metaheuristics: Operations Research/Computer Science Interfaces Series*, **39**:43–63, 2007. (54)

[143] S. Ceria, P. Nobili, and A. Sassano. *Set covering problem. annotated bibliographies in combinatorial optimization.* Wiley, New York, 1998. (54)

[144] A. Caprara, P. Toth, and M. Fischetti. **Algorithms for the set covering problem**. *Annals of Operations Research*, **98**:353–371, 2000. (54)

[145] S. Umetani and M. Yagiura. **Relaxation heuristics for the set covering problem**. *Journal of the Operations Research Society of Japan*, **50**:350–375, 2007. (54)

[146] Z. Ren, Z. Feng, L. Ke, and H. Chang. **A fast and efficient ant colony optimization approach for the set covering problem**. In *Proceedings of the 2008 Congress on Evolutionary Computation*, pages 1839–1844. IEEE, 2008. (67)

[147] J. Wu and H. Li. **On calculating connected dominating set for efficient routing in ad hoc wireless networks**. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, pages 7–14. ACM, 1999. (74, 75, 77)

[148] M.T. Thai, F. Wang, D. Liu, S. Zhu, and D.-Z. Du. **Connected dominating sets in wireless networks with different transmission ranges**. *IEEE Transactions on Mobile Computing*, **6**:721–730, 2007. (75)

[149] P.J. Wan, K.M. Alzoubi, and O. Frieder. **Distributed construction on connected dominating set in wireless ad hoc networks**. In *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, **3**, pages 1597–1604. IEEE, 2002. (75)

[150] S. Guha and S. Khuller. **Approximation algorithms for connected dominating sets**. *Algorithmica*, **20**:374–387, 1998. (75)

[151] M. Park, C. Wang, J. Willson, M.T. Thai, W. Wu, and A. Farago. **A dominating and absorbent set in wireless ad-hoc networks with different transmission range**. In *Proceedings of the 8th International Symposium on Mobile Ad Hoc Networking and Computing*, pages 22–31. ACM, 2007. (75)

[152] M.T. Thai, R. Tiwari, and D.-Z. Du. **On construction of virtual backbone in wireless ad hoc networks with unidirectional links**. *IEEE Transactions on Mobile Computing*, **7**:1–12, 2008. (75)

[153] I. Shin, Y. Shen, and M.T. Thai. **On approximation of dominating tree in wireless sensor networks**. *Optimization Letters*, **4**:393–403, 2010. (75, 76, 84, 85)

[154] E.M. Arkin, M.M. Halldórssom, and R. Hassin. **Approximating the tree and tour covers of a graph**. *Information Processing Letters*, **47**:275–282, 1993. (75)

[155] T. Fujito. **On approximability of the independent/connected edge dominating set problems**. *Information Processing Letters*, **79**:261–266, 2001. (75)

[156] T. Fujito. **How to trim an MST: a 2-approximation algorithm for minimum cost tree cover**. In *ICALP 2006, Part I, Lecture Notes in Computer Science*, **4051**, pages 431–442. Springer, 2006. (75)

[157] I. Chlamtac, A. Ganz, and G. Karmi. **Lightpath communications: an approach to high bandwidth optical WANs**. *IEEE Transactions on Communications*, **40**:1171–1182, 1992. (91)

[158] L.H. Sahasrabuddhe and B. Mukherjee. **Light-trees: optical multicasting for improved performance in wavelength-routed networks**. *IEEE Communications Magazine*, **37**:67–73, 1999. (91)

[159] X. Zhang, J.Y. Wei, and C. Qiao. **Constrained multicast routing in WDM networks with sparse light splitting**. *Journal of Lightwave Technology*, **18**:1917–1927, 2000. (91)

[160] L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. **Spanning trees with bounded number of branch vertices**. In *29th International Colloquium on Automata, Languages and Programming*, pages 355–365. Springer, 2002. (92)

[161] R. Cerulli, M. Gentili, and A. Iossa. **Bounded-degree spanning tree problems: models and new algorithms**. *Computational Optimization and Applications*, **42**:353–370, 2009. (92, 93, 94, 103, 109, 110, 114)

[162] R. Cordone, F. Cordone, and F. Maffioli. **Coloured ant system and local search to design local telecommunication networks**. In *EvoWorkshop 2001, Lecture Notes in Computer Science*, **2037**, pages 60–69. Springer, 2001. (110, 152)

[163] J.M.S. Valente, J.F. Gonçalves, and R.A.F.S. Alves. **A hybrid genetic algorithm for the early/tardy scheduling problem**. *Asia-Pacific Journal of Operational Research*, **23**:393–405, 2006. (119, 120, 122, 127, 128, 129, 130, 131, 132)

[164] A. Singh. **A hybrid permutation-coded evolutionary algorithm for the early/tardy scheduling problem**. *Asia-Pacific Journal of Operational Research*, **27**:713–725, 2010. (119, 122, 127, 128, 129, 130, 131, 132)

[165] J.K. Lenstra, A.H.G. Rinooy Kan, and P. Brucker. **Complexity of machine scheduling problems**. *Annals of Discrete Mathematics*, **1**:343–362, 1977. (119)

[166] K. Korman. **A Pressing Matter**. Video (1994) 46-50. (120)

[167] K. Landis. **Group technology and cellular manufacturing in the Westvaco Los Angeles VH Department**, 1993. Project Report in IOM 581, School of Business, University of Southern California. (120)

[168] T.S. Abdul-Razaq and C.N. Potts. **Dynamic programming state-space relaxation for single-machine scheduling**. *Journal of the Operational Research Society*, **39**:141–142, 1988. (120, 128)

[169] G. Li. **Single machine earliness and tardiness scheduling**. *European Journal of Operational Research*, **96**:546–558, 1977. (120, 121, 128)

[170] C.F. Liaw. **A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem**. *Computers & Operations Research*, **26**:679–693, 1999. (120, 128)

[171] J.M.S. Valente and R.A.F.S. Alves. **Improved lower bounds for the early/tardy scheduling problem with no idle time**. *Journal of the Operational Research Society*, **56**:604–612, 2005. (120)

[172] S. Tanaka. **An exact algorithm for single-machine scheduling without idle time**. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference*, pages 614–617, 2007. (120)

[173] P.S. Ow and T.E. Morton. **The single machine early-tardy problem**. *Management Science*, **35**:177–191, 1989. (120, 121)

[174] J.M.S. Valente and R.A.F.S. Alves. **Improved heuristics for the early/tardy scheduling problem with no idle time**. *Computers & Operations Research*, **32**:557–569, 2005. (120)

[175] J.C. Bean. **Genetic algorithms and random keys for sequencing and optimization**. *ORSA Journal on Computing*, **6**:154–160, 1994. (120)

[176] W.M. Spears and K.A. DeJong. **On the virtues of parameterized uniform crossover operator**. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 230–236, 1991. (120)

[177] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, and T.J. Chua. **A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem**. *Information Sciences*, **181**:2455–2468, 2011. (122, 127, 153)

[178] M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, and H.-L. Chen Angela. **A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops**. *Information Sciences*, **181**:3459–3475, 2011. (122, 153)

[179] K.-H. Liang, X. Yao, C. Newton, and D. Hoffman. **A new evolutionary approach to cutting stock problems with and without contiguity**. *Computers & Operations Research*, **29**:1641–1659, 2002. (124, 125)

[180] J.M.S. Valente and R.A.F.S. Alves. **Heuristics for the single machine scheduling problem with quadratic earliness and tardiness penalties**. *Computers & Operations Research*, **35**:3696–3713, 2008. (126)

[181] H.C. White, S.A. Boorman, and R.L. Breiger. **Social structure from multiple networks. I. Blockmodels of roles and positions**. *American Journal of Sociology*, **81**:730–780, 1976. (133)

[182] A. Jessop. **Blockmodels with maximum concentration**. *European Journal of Operational Research*, **148**:53–64, 2003. (134, 136, 143)

[183] P. Arabie. **Cluster representations of group overlap**. *Journal of Mathematical Sociology*, **5**:113–128, 1977. (134)

[184] J.M. Light and N.C. Mullins. *A primer on blockmodelling procedure. In: P.W. Holland & S. Leinhardt (Eds.), Perspectives on Social Network Research.* Academic Press, New York, 1979. (134)

[185] S. Wasserman and K. Faust. *Social network analysis: methods and applications.* Cambridge University Press, Cambridge, 1997. (134)

[186] J. Scott. **Social network analysis: a handbook**, 2000. Sage publications. (134)

[187] A. Jessop. **Multiple attribute probabilistic assessment of the performance of some airlines**. In *Lecture Notes in Economics and Mathematical Systems*, pages 417–426. Springer, 2001. (134)

[188] A. Jessop. **A multiattribute assessment of airport performance**, 1999. presented at 25th European Working Group Financial Modelling. (134, 143)

[189] A. Charnes, W.W. Cooper, A.Y. Lewin, and L.M. Seiford. *Data envelopment analysis: theory, methodology and applications.* Kluwer Academic Publishers, Dordrecht, 1995. (134)

[190] A. Jessop. **Exploring structure: a blockmodel approach**. *Civil Engineering and Environmental Systems*, **19**:263–284, 2002. (134, 143)

[191] A. Jessop. **A measure of competitiveness in leagues: a network approach**. *Journal of the Operational Research Society*, **57**:1425–1434, 2006. (134, 143)

[192] E. Falkenauer. *Genetic algorithms and grouping problems.* John Wiley & Sons, Chicester, 1998. (135, 137, 138)

[193] O.C. Herfindahl. **Concentration in the U.S. steel industry**, 1950. Ph.D. Thesis, Columbia University. (136)

[194] A.O. Hirschman. **The paternity of an index**. *The American Economic Review*, **54**:761–762, 1964. (136)

[195] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity.* Prentice-Hall, 1982. (137)

[196] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991. (137)

[197] A. Singh and A.K. Gupta. **Two heuristics for the one-dimensional bin-packing problem**. *OR Spectrum*, **29**:765–781, 2007. (139, 140)

List of Publications

[1] S. Sundar and A. Singh. **A swarm intelligence approach to the quadratic minimum spanning tree problem**. *Information Sciences*, **180**:3182–3191, 2010.

[2] A. Singh and S. Sundar. **An artificial bee colony algorithm for the minimum routing cost spanning tree problem**. *Soft Computing*, **15**:2489–2499, 2011.

[3] S. Sundar and A. Singh. **A hybrid heuristic for the set covering problem**. *Operational Research*, In Press, DOI: 10.1007/s12351-010-0086-y.

[4] S. Sundar and A. Singh. **A swarm intelligence approach to the early/tardy scheduling problem**. *Swarm and Evolutionary Computation*, In Press, DOI: 10.1016/j.swevo.2011.12.002.

[5] S. Sundar, A. Singh and A. Rossi. **New heuristics for two bounded-degree spanning tree problems**. *Information Sciences*, In Press, DOI: 10.1016/j.ins.2012.01.037.

[6] S. Sundar and A. Singh. **New heuristics for the dominating tree problem**. Communicated to *Swarm Intelligence*.

[7] S. Sundar and A. Singh. **Two metaheuristic approaches for the block-model problem**. Communicated to *Information Sciences*.