Evolutionary Techniques for Some Combinatorial Optimization Problems

A thesis submitted during 2015 to the University of Hyderabad in partial fulfillment of the award of a **Ph.D. degree** in School of Computer and Information Sciences

by

Sachchida Nand Chaurasia

School of Computer and Information Sciences

University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500 046
Telangana
India

June 2015

CERTIFICATE

This is to certify that the thesis entitled **"Evolutionary Techniques for Some Combinatorial Optimization Problems"** submitted by **Sachchida Nand Chaurasia** bearing **Reg. No. 10MCPC13** 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 which is a plagiarism free thesis.

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
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500 046, India

Dean
School of Computer and Information Sciences
University of Hyderabad
Hyderabad – 500 046, India

DECLARATION

I, **Sachchida Nand Chaurasia**, hereby declare that this thesis entitled **“Evolutionary Techniques for Some Combinatorial Optimization Problems”** submitted by me under the guidance and supervision of **Dr. Alok Singh** is a bonafide research work which is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga/INFLIBNET.

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

Date :

Name: Sachchida Nand Chaurasia

Signature of the Student:

Reg. No.: 10MCPC13

Signature of the Supervisor(s):

Abstract

Combinatorial optimization problems occur in almost every discipline including those with utmost practical importance. A combinatorial optimization problem that is discrete and finite in nature deals with finding an optimal solution from its large but finite set of feasible solutions. Most of the combinatorial problems are $\mathcal{NP}$-hard in nature, i.e., the execution time of any known exact algorithm for solving such problems increases exponentially with the size of problem instance and even the optimality of a proposed solution can not be verified in polynomial time. Therefore, for most of these problems, applicability of exact methods is limited to small instances only, and for solving even moderately large instances, one has to resort to other methods such as heuristics and metaheuristics which do not guarantee optimality, but 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 through two metaheuristic techniques, viz. genetic algorithm (GA) and evolutionary algorithm with guided mutation (EA/G). GA is a well known evolutionary technique, whereas EA/G is a relatively new evolutionary technique that can be considered as a cross between genetic algorithm and estimation of distribution algorithm (EDA). EA/G has the features of both GA and EDA.

Six $\mathcal{NP}$-hard problems with diverse characteristics have been addressed in this thesis. These six problems are as follows: set packing problem, minimum weight dominating set problem, dominating tree problem, order acceptance and scheduling problem, single machine total stepwise tardiness problem with release dates and registration area planning problem. These problems are not only challenging from theoretical point of view, but also have many real world applications in diverse areas such as networks, data mining, transportation and logistics, production scheduling etc. We have developed hybrid EA/G based approaches for the first four problems above. For the last three problems, we have developed hybrid

GA based approaches. For the fifth problem, viz. single machine total stepwise tardiness problem with release dates, an artificial bee colony algorithm based hybrid approach is also presented. In addition, we have modified problem-specific heuristics and local search procedures for some of the aforementioned problems to enhance their performance. Our approaches have been compared against the state-of-the-art approaches. Computational results demonstrate the effectiveness of our proposed approaches. Our approaches can be easily extended to solve several related combinatorial optimization problems. Ideas presented in this thesis can be used to develop metaheuristic approaches for a wide range of combinatorial optimization problems.

Acknowledgements

There are so many people to thank for guiding and helping me in many ways during this journey and it is inevitable to name some of them. First and foremost, I express my deepest gratitude towards 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 under his guidance. This thesis would not have been possible without his support and constant encouragement throughout the research. I am obliged to him for being generous with his time and keeping faith in me despite my lazy nature.

I am indebted to Dr. Shyam Sunder, who stands by me always and had helped me a lot throughout my research work. I found him always available, whenever I required him.

I am thankful to the Dean of the School Prof. Arun Kumar Pujari for providing all the necessary facilities.

I am thankful to my doctoral committee members Prof. S. Bapi Raju, Prof. Rajiv Wanker and Dr. Anupama Potluri. From time to time they reviewed my research work and gave their feedback, which enhanced the quality of my work. I am equally grateful to Dr. S. Durga Bhavani, Dr. S.K. Udgata, Dr. Vineet Padmanabhan Nair and other faculty members of the school for their constant encouragement. My sincere thanks to a number of researchers in my field who furnished me with indispensable research data.

I am thankful for the unstinting support that I received from the research infrastructure and the effervescent ambiance of the university. Due credit to university for building a research oriented computer science school, a library rich in a wide range of research books & articles, and most importantly a healthy campus atmosphere. These pillars of education helped me during my studies in the University as a master student and as a research scholar.

Not every day we get an opportunity to thank our parents, holding some drops of tears in my eyes, my heartfelt thanks to my Mother, Mrs. Devki Devi, who brought me to this world, to my Father, Mr. Bisnu Deo Chaurasia who is always very protective of me and to my all other family members for their affection, care and encouragement. I owe this thesis to them.

This journey would not have been joyful without these wonderful friends, Dr. Abhay Kumar, Dr. Satpal Singh, Shankar Kumar, Ranjeet Kapoor, Ms. Aditi Patel, Ms. Kalpna Negi and many more. I thank my fellow labmates for stimulating discussion, for the sleepless night we were working together and for all the fun we had together in last five years.

A major part of my research work was supported by two fellowships, viz. Junior Research Fellowship funded by Department of Science and Technology, Government of India and Basic Scientific Research Fellowship funded by University Grant Commission. I am grateful to these organizations for their financial support.

At the last but not the least, a humble thanks to my fellow scholars, seniors, juniors, hostel mates and friends, who never made me feel like away from the home.

Sachchida Nand Chaurasia

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

Combinatorial optimization problems occur in almost every walk of life. Some of these problems also find their roots in theoretical computer science and mathematics. A combinatorial optimization problem, which is discrete and finite in nature, seeks an optimal solution from its large but finite set of feasible solutions. Such a problem is characterized by its combinatorial structure and computational aspect. Also, it is easy to explain such a problem and its results. However, these problems suffer from inherent combinatorial explosion property. This property refers to a phenomenon in which the number of possible feasible solutions for such a problem grows exponentially with the size of the problem's instance. Any problem exhibiting such a property has always perplexed researchers from both theoretical and practical points of view.

Though relevant literature suggests that combinatorial optimization is a relatively young discipline, the trail of this subject can be traced far back in history. Historically, several combinatorial optimization problems such as finding a shortest path from source to destination and tour planning were known since long and considered in isolation. Impromptu solutions were given to such problems in a particular context. However, towards the eighteenth century, several new disciplines of mathematics focusing on needs of real world were being recognized. Combinatorial optimization was one among these new disciplines. In this regard, Euler's paper on the *Seven Bridges of Königsberg* in 1735 was first scientific breakthrough paper that laid the foundation of graph theory, which in turn led to the emergence of combinatorial optimization as an independent discipline of mathematics. This discipline is in constant evolution since then. The development of mathematical linear programming by Kantorovich [3] and Dantzig [4] was a further major step in legitimizing and fortifying this discipline. Later, with the advent of continuously improving computers, the position of this discipline was further strengthened as researchers' effort to solve these computational problems got reduced manifold. Towards the late 1960's, researchers observed that while several combinatorial optimization problems such as minimum spanning tree problem can be solved optimally in polynomial time, there are many others which illude polynomial time algorithms. A problems for which polynomial time algorithms is unknown is particularly tantalizing because it looks similar to those problems that can be solved in polynomial time. Such problems pose inherent difficulties to any approach that is used to solve them. It is believed that such problems are intractable. Such complex problems can be found in diverse disciplines including those with utmost practical importance.

1.1 Combinatorial optimization

Combinatorial optimization is a subject that deals with those optimization problems whose objective is to find an optimal solution from its large, but finite set of feasible solutions. Such optimization problems are called combinatorial optimization problems and include both minimization and maximization problems. Optimality is defined with respect to some cost function which is called the objective function. Such a problem is computational in nature and consists of an infinite set of instances, where an instance of a problem refers to an input for this problem. For example, maximum clique problem seeks a complete subgraph with maximum number of nodes on a graph and any graph can be considered as an instances for this problem.

Formally, on a given instance, a combinatorial optimization problem P can be considered as a triple (S, Ω, f), where S is the set of all feasible solutions and is usually called search space or solution space, Ω is the set of constraints that have to be satisfied by every feasible solution and f is the objective function, which should be either maximized or minimized. To solve this problem, one has to find a globally optimal solution $s^* \in S$. There may exist more than one optimal solution to an instances of a problem, i.e., optimal solution need not be unique. In case of a minimization problem, one has to find a solution s^* with minimum objective function value, i.e., $f(s^*) \leq f(s)\ \forall\ s \in S$. Similarly, in case of a maximization problem, one has to find a solution s^* with maximum objective function value, i.e., $f(s^*) \geq f(s)\ \forall\ s \in S$. It is to be noted that by changing the sign of the objective function, a maximization problem can be easily transformed into a minimization problem and vice-versa. In P, the variables, which participate in building the solutions, are discrete quantities whose values are used in optimizing the objective function. For example, in case of minimum spanning tree problem, variables are edges of the graphs and the weighs associates with those edges which are part of the spanning tree contribute to the cost of that spanning tree.

The combinatorial optimization problems based on their nature can be broadly divided into three classes which are as follows:

1. *Subset selection problems:* These problems seek an optimal subset of objects from a given set of objects under certain constraints. Common examples of subset selection problems include maximum clique problem, minimum spanning tree problem, knapsack problem etc.

2. *Permutation problems:* The goal of these problems is to arrange a given set of items in an optimal order subject to some constraints. Traveling salesman problem and single machine scheduling are two common examples of permutation problems.

3. *Grouping problems:* Given a set of items, these problems seek an optimal assignment of these items into various groups without violating any constraint. Bin packing problem, graph coloring problem and clustering are well known examples of grouping problems.

However, some combinatorial optimization problems have characteristics of more than one class, i.e., these three classes are not disjoint. For example, multiple traveling salesman problem have characteristics of grouping as well as permutation problems, multiple knapsack problem involves aspects of subset selection as well as grouping. Likewise, single machine order acceptance and scheduling problem [5], where only a subset of orders need to be scheduled for processing on a single machine has aspects of subset selection and permutation problems. Similarly, fixed job scheduling problem under spread time constraints [6], where only a subset of jobs can be scheduled on fixed number of identical processors posses characteristics of all three classes.

While polynomial time algorithms are known to optimally solve some combinatorial optimization problems only, there exists many other combinatorial optimization problems which are $\mathcal{NP}$-hard, i.e., the execution time of any known exact algorithm for solving such problems grows exponentially with the size of problem instance and even the optimality of a proposed solution can not be verified in polynomial time. Here it is worthwhile to mention that a problem may be $\mathcal{NP}$-hard in general, but it may admit a polynomial time algorithm in some particular cases. For example, several $\mathcal{NP}$-hard graph problems such as maximum clique problem, max-cut problem can be solved in polynomial time for planar graphs. Hence, the use of exact methods for solving $\mathcal{NP}$-hard problems is limited to either small sized instances only or instances involving polynomially solvable cases. However, more often than not, the nature and size of the instances render an exact algorithm non-viable. This is equally true for many real world problems also. These are the situations where instead of searching for a guaranteed optimal solution, one has to resort to those methods that can find high quality solutions in a reasonable amount of time. Such methods which have been attracting massive attention from the research community since the last several decades are collectively known by the name *approximate methods* [7].

1.2 Approximate methods

Approximate methods usually provide optimal or near optimal solutions to combinatorial optimization problems, however, they do not guarantee for optimality. These algorithms usually runs in polynomial time. Since the search (solution) space of a $\mathcal{NP}$-hard problem, in general, is rugged and contains large number of locally optimal solutions some of whom are of quality similar to globally optimal solution, therefore, these algorithms may return solutions that are close to optimal solution in terms of objective function value, but may be quite far from the optimal solution in the search space. Approximate methods can be broadly classified into following four categories:

- *Heuristic* refers to an intuitive method which exploits the structure of the problem under consideration to solve it. Usually, a heuristic quickly finds a pretty good solution. However, there is no guarantee that solution returned by a heuristic can not be arbitrarily bad.

- *Metaheuristic* is a general algorithmic framework which can be adapted without much modifications to solve different optimization problems. A metaheuristic can be perceived as a general purpose heuristic which directs underlying problem-specific heuristics towards promising regions of the search space [7]. Most metaheuristics require the problem to be represented in a suitable form prior to their application. Some prominent examples of metaheuristics are genetic algorithms [8, 9], tabu search [10, 11], ant colony optimization [7, 12], artificial bee colony algorithm [13] etc.

- *Approximation algorithm* is a heuristic that is guaranteed to return a solution which is within a certain factor of optimal solution.

- The fourth category of approximate methods include those methods which are obtained by terminating an exact method prematurely. For example, a mixed integer linear programming solver for a problem can be forced to terminate after some specified interval of time and output the best solution found in that time interval in case it is able to find any solution.

However, theoretical results are available for many combinatorial optimization problems, which forbid the existence of good polynomial time approximation algorithms for them. Even terminating an exact method prematurely may not help in many cases as it may produce either a very poor solution or no solution at all. Under these circumstances, the only choices left are heuristics and metaheuristics. To obtain a good approximate solution in most of these cases, a hybrid approach is used where a metaheuristic is combined with a problem-specific heuristic. This thesis is focussed primarily on two metaheuristic techniques, viz. genetic algorithms and estimation of distribution algorithms which belong to the broad class of evolutionary algorithms.

1.3 Evolutionary algorithms

Evolutionary algorithms (EAs) are a broad class of stochastic optimization algorithms that work on the mechanisms inspired by biological evolution. The mechanisms that include selection and genetic operators such as recombination and mutation are stochastic in nature. All EAs follow an iterative process. Each iteration of an EA is called a generation. All EAs maintain a population of candidate solutions with associated fitness values which evolve over generations based on a "survival of fittest strategy". The inception of EAs took place during 1960s with the development of three algorithms exploiting the basic principles of natural evolution independently. In USA, Fogel and Houck [14] developed evolutionary programming, while Holland [8] developed genetic algorithm. Meanwhile, in Germany, Rechenberg and Schwefel [15] developed evolutionary strategies. Since then, EAs have been attracting lots of attention world-wide because of their ability to find high quality solutions for difficult and complex optimization problems and a number of EAs variants have been developed. Even now, the new EAs are being developed continually. These various EA variants differ in the manner solutions are represented inside the algorithm, new solutions are produced and selection is carried out.

A general framework for an evolutionary algorithm is given in Algorithm 1.1 [16]. EAs start with a population of solutions which are usually generated randomly. An iterative process takes over afterwards. During each iteration or generation, some population members are selected to be parents. Genetic operators are applied on these parents to produce new offspring. Population for the next generation is selected from the current population members and newly produced offspring. This process is repeated till the termination condition is satisfied.

Algorithm 1.1: A general framework of an evolutionary algorithm

```
1: Initialize the population with randonly generated solutions;
2: Evaluate the fitness of each solution in the population;
3: while Termination condition not satisfied do
4:     Perform parents selection;
5:     Apply genetic operators to generate new offspring;
6:     Evaluate newly generated solutions;
7:     Select population for the next generation;
8: end while
```

Table 1.1: Terms commonly used in genetic algorithms

Term	**Meaning**
Phenotype	A solution to the problem under consideration
Genotype or Chromosome	A representation of a solution in a form suitable for applying genetic algorithm
Gene or Locus	A position in a chromosome
Alleles or Alphabet	A set of possible values for a gene
Population	A set of chromosomes participating in evolution
Generation	A single pass from the present population to the next one
Fitness	A measure of quality of a chromosome on the actual problem
Evaluation	The conversion of a genotype into its corresponding phenotype and the calculation of its fitness
Phenotype Space	The space consisting of all possible solutions to the problem under consideration
Representation Space or Genotype Space	The space containing all possible genotypes of the problem under consideration

1.4 Genetic algorithm

Genetic algorithm (GA) is the most widely used metaheuristic technique among the evolutionary algorithms. GA was proposed by Holland [8] in 1960s. Further developments in GA were carried out by by Holland and his research group during 1960s and 1970s. Holland developed GA with the intention of simulating and studying evolutionary adaptation as it occurs in nature. Only later GA found use in solving various kinds of optimization problems. The main reason behind the success of GA is its robustness and its ability to explore several possible regions of the search space simultaneously. GA supports implicit as well as explicit parallelism. The implicit parallelism is inherent in GA, a fact proved by Holland in his famous schema theorem. Explicit parallelism in GA can be exploited by manipulating and evaluating different individuals of the population in parallel. The ability of the genetic algorithm to mimic evolution and thereby enabling solutions to adapt according to their environments makes it highly useful for solving complex optimization problems.

As GA makes use of a number of biological terms, it is necessary to define these terms in the context of GA before any further discussion. Table 1.1 defines the commonly used terms in GA.

GA starts with a population of candidate solutions (also called chromosomes) to a particular optimization problem under consideration. Usually, each candidate solution is generated randomly. However, problem-specific heuristics can also be used to generate potential candidate solutions in order to get a better start. The size of population may depend upon the problem. The fitness of each individual (solution) in the population is computed with the help of a fitness function which assigns a score to each individual in the population that indicates how much better a solution is in comparison to others. Once the fitness of each individual in the population is computed, then the evolution starts through an iterative process.

In each iteration or generation, a parent set (also called mating pool) is formed by selecting (with repetition allowed) some members of the current population with the help of a fitness-based selection method. In this selection method, fitter individuals are more likely to be picked. The reason behind such a strategy is that fitter individuals in the parent set are more likely to produce offspring having even higher fitness. Next, the individuals in the parent set are permitted to participate in the reproduction process where new individuals (offspring) are produced through genetic operators such as crossover and mutation. In crossover or recombination, usually the genes of two individuals (also called parents) are recombined to generate one or two new offspring. So, some genes of each offspring will be from one parent and the remaining will be from the other parent. In mutation, an exact copy of a parent solution is generated, then some random changes, relevant to the problem under consideration, are made in the gene values of the offspring chromosome. Both crossover and mutation operators are used with a fixed probability called crossover rate and mutation rate respectively. While the crossover rate controls what fraction of old generation will move to the next generation through crossover, the mutation rate controls how many genes of a chromosome will be mutated. Since, both operators are applied with some probability, therefore, some individuals of old generation will pass to the next generation unaltered. This reproduction results in the creation of new population – called new generation of the population. This new generation is now evaluated. The whole iterative process is repeated until the termination criteria is met. The termination criteria may consist of a fixed amount of CPU time or the number of solutions generated or the total number of generations or the number of consecutive generations without improvement in quality of the best solution.

The convergence analysis of a stochastic optimization method can only provide information about the average or expected behavior. GA is also a stochastic optimization method as it utilizes a combination of at least three stochastic operators (selection, crossover and mutation). Also many different variants of each stochastic operator have been proposed since its inception. As a result, it is hard to investigate general convergence behavior of various GAs in such circumstances, as different operators affects the convergence in its own way. However, Holland [8] was the first one who tried to provide the theoretical foundation of GA through his schema theorem. This theorem addressed the fundamental question about what information is contained in the population of strings (chromosomes) to guide the GA towards better and better solutions. The general idea behind this theorem is that a string (chromosome) in the population can be seen as a sample of the quality of many different subspaces or hyperplanes of the search space. A schema is a string template in which a subset of positions have fixed alleles, while alleles at other positions can vary over an allowable set (alphabet). For example, in case of binary representation of a string, the schema *111*, where positions 2, 3 and 4 are fixed while other remaining positions are associated with '*' or don't care symbol, matches with a string (chromosome) if the string matches the schema at its fixed positions. The chromosomes 01110, 11110, 11111, 01111 match the schema described above. The order of a schema is the number of fixed positions, and the defining length of a schema is the distance between its two farthest fixed string positions. For the above schema, the order is 3 and the defining length is 2. Since a string (chromosome) of length l matches 2^l schemas, therefore, evaluation of the fitness of a chromosome implicitly provides a sample of the fitness of all the schemas to which this chromosome belongs. The schema theorem suggests that short, low-order schemas whose average fitness remains above the mean will receive exponentially increasing number of trials in successive generations. Goldberg [9] termed short, low order, highly fit schemas as building blocks and proposed the building block hypothesis. This hypothesis states that the GA seeks near optimal performance through juxtaposition of building blocks.

It is to be noted that the genetic algorithm described in this section will be referred to as simple genetic algorithm to distinguish it from the genetic algorithm discussed in next two sections for permutation and grouping problems.

1.4.1 Representation of individuals

The representation of candidate solutions in the GA plays a significant role in its success. A chromosome or genotype should represent a phenotype in most natural way so that the notion of distance in phenotype space should be preserved in representation space. A representation scheme should be able to represent all phenotypes and should have minimum redundancy. A representation scheme is said to have redundancy when for a phenotype more than one genotype is possible. Presence of redundancy makes the size of the representation space larger than the phenotype space. As GA works in representation space, it has to search a larger space. This can affect the performance of GA adversely [17]. Deciding a suitable representation for a problem is a difficult task that comes with practice and experience only.

Though there exists many schemes to represent a solution in a simple genetic algorithm, binary representation, integer representation and real-valued representations are most commonly used. Here, we will discuss the first two representation schemes as these two schemes find use in subset selection problems.

1.4.1.1 Binary representation

This is the oldest and simplest way to represent a candidate solution. Chromosome is made up of binary digits 0 and 1. The length of the chromosome is decided by the parameters involved and the precision required. Chromosome length is fixed in the simplest case. Binary representation is natural for subset selection problems where each position corresponds to a particular item and the value of 1 at a position indicates the presence of the corresponding item in the subset, whereas the 0 indicates the absence. This representation leads to fixed length chromosomes for subset selection problems. However, binary representation is not suitable in some cases, e.g., using binary strings to represent integers or real numbers is not natural and one can get better results by utilizing the integer or real valued representations directly.

1.4.1.2 Integer representation

This representation is preferable where the genes can take a value from a set of values. For example, integer representation is natural for a problem where the objective function involves a set of variables all of whom can take integer values. The set of values that a gene can take can be finite or infinite. This representation is also in use for subset selection problems. To use this representation, items are numbered from 1 to n, where n is the total number of items and a subset is represented directly be the set of items it contains. Clearly, this representation leads to variable length chromosomes for subset selection problems.

The set of permissible values for genes can be ordinal or cardinal [16]. In case of ordinal values, a predecessor and successor relationship can be established among these values, whereas in case of cardinal values no such relationship exists among these values. Our first example above of a problem where the objective function involves a set of variables all of whom can take integer values involves ordinal values. On the other hand, genes in case of subset selection problems are inherently cardinal.

1.4.2 Recombination

It is based on the idea that offspring produced by mating two different individuals with desirable features have desirable features of both the parents. However, outcomes can be worse also, but some better offspring will definitely be produced when recombination is applied many times. While designing recombination operator, one must take into account the nature of problem under consideration and the solution representation scheme. Therefore, recombination operators are proposed according to the solution encoding strategies. We have used recombination and crossover interchangeably throughout the thesis. However, recombination term can be used even when more than two parents participate in reproduction process, whereas in case of crossover conventionally it is assumed that exactly two parents participate.

1.4.2.1 1-point crossover

This was the original crossover operator developed by Holland [8] for use in genetic algorithms. Applicability of this crossover is not limited to any particular solution representation scheme, but it is mostly used with binary and integer representations. In this crossover, a single crossover site is chosen randomly in the interval [0, n-1], where n is length of string, and the substring after that crossover site are swapped between the two parents to generate two offspring. Figure 1.1 illustrates this crossover. This crossover suffers from the problem of positional bias. Two genes which are at the opposite ends of a chromosome will always be separated irrespective of the quality of the schema they are forming.

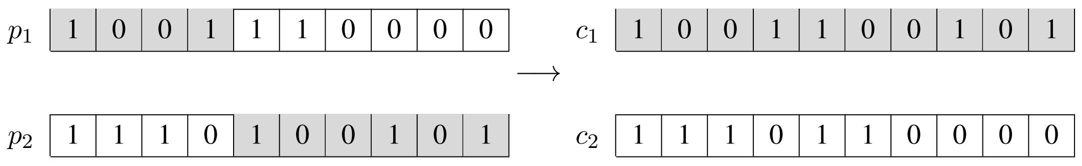


Figure 1.1: 1-point crossover

1.4.2.2 k-point crossover

k-point crossover is a generalization of 1-point crossover where k crossover sites are chosen randomly in the interval [0, n-1]. These k crossover sites divide the child chromosomes into $k+1$ segments. Two children are generated by copying alternative segments from the parents. This crossover also has positional bias. Usually, $k = 2$ and in that case this crossover is called 2-point crossover. Figure 1.2 shows k-point crossover for $k = 2$ with the help of an example.

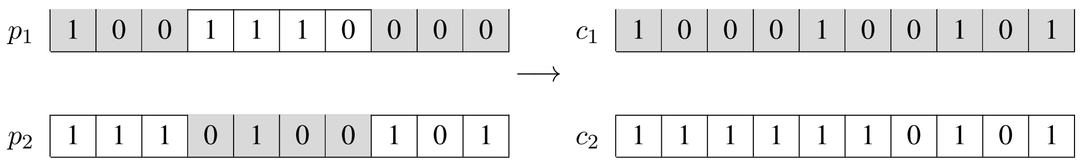


Figure 1.2: k-point crossover

1.4.2.3 Uniform crossover

In uniform crossover [18], each location is considered separately. For each location, a random number is generated uniformly in the interval [0, 1]. If this number is less than a predefined value p (usually 0.5), then the corresponding gene for the first child is taken from first parent, otherwise from second parent. Similarly, the second child is generated by interchanging the role of two parents. Figure 1.3 illustrates uniform crossover for $p = 0.5$. Performance of uniform crossover via-à-vis traditional 1- or 2-point crossover has been analysed from both theoretical and experimental point of view by [19] and more recently by [20].

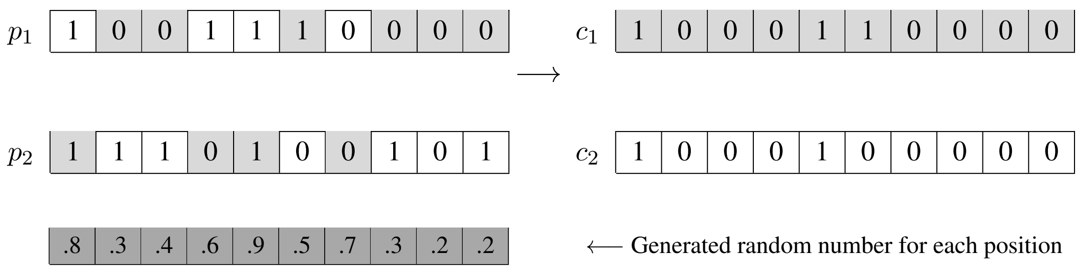


Figure 1.3: Uniform crossover

1.4.3 Mutation

The aim of mutation is to maintain sufficient diversity in the population. It also aids in exploring new regions of the search space. A single parent is selected for mutation and a new offspring is generated by making some random change in it. Usually, mutation does not have any positional bias. For different types of representation schemes, different mutation operators have been proposed.

1.4.3.1 Bitwise mutation

Bitwise mutation is applied on binary representation of genotypes. Each bit is considered separately and is flipped with small uniform probability. Figure 1.4 illustrates bitwise mutation operator.

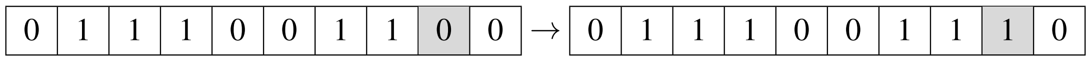


Figure 1.4: Bitwise mutation for binary representation

1.4.3.2 Random reset mutation

Random reset mutation is an extension of bitwise mutation and is used for integer encoding. For each position, a new value is chosen uniformly at random from all permissible values for that position. This operator is recommended, where permissible values are cardinal. Figure 1.5 illustrates this mutation where each gene can take one of the alleles 1, 2, 3 and 4.

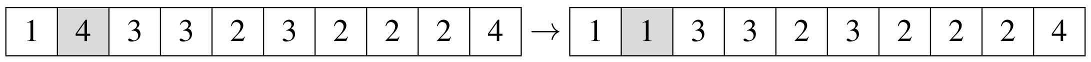


Figure 1.5: Random reset mutation for integer representation

1.4.3.3 Creep mutation

This mutation is designed for use with ordinal values in integer representations. In this mutation the value of a gene is more likely to be mutated to nearby values. For example, in Figure 1.5 if the permissible values were ordinal then the probability of 2 mutated to 1 or 3 will be more than 2 mutated to 4.

1.4.4 Population models

Population model determines the composition of next generation in terms of individuals' origin. There are mainly two population models, viz. generational and steady-state [21]. The genetic algorithm that uses the former model is called generational genetic algorithm and that uses the latter is called steady-state genetic algorithm.

1.4.4.1 Generational model

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

1.4.4.2 Steady-state model

In steady-state model, only a small fraction of population is replaced by the same number of newly generated offspring every generation. Usually, only a single offspring is produced during each generation that replaces a less fit member of the population. In this strategy, best solutions are always retained in the population, and generated offspring is available for selection and reproduction immediately. These two factors aid in finding better solutions faster. Moreover, in this strategy, multiple copies of the same solution are also forbidden in the population, i.e., the generated child is included into the population by replacing a less fit member only when it is different from all the existing population members. This prevents premature convergence, which is a common phenomenon in generational model. Actually, in generational model, multiple copies of the same solution may exist in the population. Usually, such solutions are highly fit solutions, and as a result, within few generations they start dominating the entire population, thereby making crossover totally ineffective and mutation the only way to improve the solution quality. Therefore, improvements, if any, in the solution quality are very slow. Such a situation is known as premature convergence.

1.4.5 Selection methods

Different selection methods basically differ in terms of selection pressure and degree of randomness employed to select the parents. Three types of selection methods are normally used.

1.4.5.1 Fitness proportionate selection

Fitness proportionate selection method was developed by Holland [8]. It is used to choose potential individual from the population to add into the parent set or mating pool. The probability of selection of an individual is proportional to the absolute fitness of that individual compared to rest of the individuals in the population. The probability p_i of selection of an individual i is calculated as

$$p_i = \frac{f_i}{\sum\limits_{j=1}^{N} f_j}$$

where f_i is the fitness of the individual i and N is the population size. This probability governs how many copies of an individual can go to the mating pool. To map these probabilities to exact numbers so that their sum is equal to the size of the mating pool, different sampling methods are used. Roulette wheel algorithm [9] and stochastic universal sampling [22] are most commonly used sampling methods. In both of these methods, different individuals of the population are mapped to non-overlapping sub-intervals of [0, 1] according to their selection probabilities. For example, if there are three individuals in the population with selection probabilities 0.3, 0.5 and 0.2 ,then these three individuals can be mapped to [0, 0.3], (0.3, 0.8] and (0.8, 1.0]. In roulette wheel method, to choose an individual for the mating pool, a random number is generated in [0, 1] and the individual under whose sub-interval the generated random number falls is included into the mating pool. This process is repeated till the desired number of individuals say N_{PS} are added into the mating pool. In case of stochastic universal sampling, a random number is generated in [0, 1] and N_{PS} equally spaced values are generated in [0, 1] considering this random number as one of these N_{PS} values. An individual is copied to the mating pool as many times as there are values among these N_{PS} values falling under its sub-interval.

However, fitness proportionate selection has following performance related issues:

- When genetic algorithm begins, the gap between solutions fitness among population members is very high. Due to this fitness gap, the highly fit individuals and their descendent can take over the entire population very quickly. This leads to premature convergence.
- Almost all solutions come closure to each other, in terms of fitness, after some generations. This makes selection probabilities of almost all solutions roughly the same, thereby rendering fitness proportionate selection ineffective.

To overcome these shortcomings, various scaling methods are used [23].

1.4.5.2 Ranking selection

Ranking selection method [22] was proposed to overcome the shortcoming of fitness proportionate selection. Ranks are assigned to solutions according to their relative fitness rather than absolute fitness. A mapping function is used to map ranks to selection probabilities so that better solutions may get more chance. The selection pressure can be changed by changing the mapping function or some control parameters in it, e.g. exponential mapping has higher selection pressure in comparison to linear mapping. The mapping function gets affected by the population model. In steady-state model, only one solution is replaced, so effect on ranks of the remaining solutions will be very less compared to generational model where entire population is replaced.

1.4.5.3 Tournament selection

Tournament selection method does not use the knowledge of the entire population. In other words, we can say that it does not use global information about the solutions in the population. Binary tournament selection is the simplest and most commonly used form of tournament selection method where exactly two individuals are selected uniformly at random from current population and the better one between these two is added into mating pool with a higher probability p, otherwise the worse one is added. In terms of selection pressure, binary tournament selection is similar to ranking selection but it is computationally more efficient [24].

1.4.6 Survivor selection

Survivor selection methods decide which individuals will survive after each generation. A number of survivor selection schemes have been suggested and the most commonly used are the following:

- Age-based replacement in which each individual exists in the population for some fixed number of generations.
- Fitness-based replacement in which either some best individuals (elites) in the population are always preserved or some worst members are replaced. Moreover, schemes described in previous subsection for parents selection can be adapted for survivor selection also.

1.5 Genetic algorithm for permutation problems

For permutation problems, the order in which the objects are arranged matters rather than the values or labels of objects. These problems can be divided into two categories. In the first category of problems, ordering among objects have importance, whereas in the second category of problems, adjacency between objects have importance. Job shop scheduling problem is an example of first category of problems, whereas traveling salesman problem is an example of latter. Usually, the first category of permutation problems are referred to as order-based problems, whereas the latter category of problems are referred to as adjacency based problems [16]. Permutation representation is natural and most suitable representation for both categories of permutation problems. In this representation, each position has an unique allele and each allele must be present in the chromosome. And also, genetic operators such as mutation and crossover should be designed in such a way that they should preserve the permutation property. None of the genetic operator discussed in the previous section in combination with any of the representation schemes discussed so far are capable of preserving the permutation property. Hence, permutation representation requires its own genetic operators and a number of such dedicated operators have been proposed in the literature. Though there exists other representations also for permutation problems (e.g. random key encoding [25]), here we will discuss permutation representation and associated genetic operators only. The permutation representation used throughout this thesis assumes that a value of i at the j^{th} position in the permutation indicates that object i is at j^{th} position in the permutation.

1.5.1 Recombination

Several crossover operators have been proposed keeping in mind the characteristics of permutation problems. Some of the most commonly used ones are described below.

1.5.1.1 Partially matched crossover (PMX)

Goldberg and Lingle [26] had proposed this crossover for traveling salesmen problem. This crossover operator is suitable for adjacency-based problems and is widely in use. Two crossover sites are selected uniformly at random, and then the alleles in the two parents at the same position in between these two crossover sites are matched. Then this crossover operator swaps the positions of the matched alleles in each of the two parents separately to produce two offspring. The example depicted in Figure 1.6 has two crossover sites and the positions of 3 & 3, 1 & 10, 2 & 4 are swapped in the both the parents. One of the shortcoming of PMX crossover is that it fails to preserve the adjacency information that is common in both the parents.

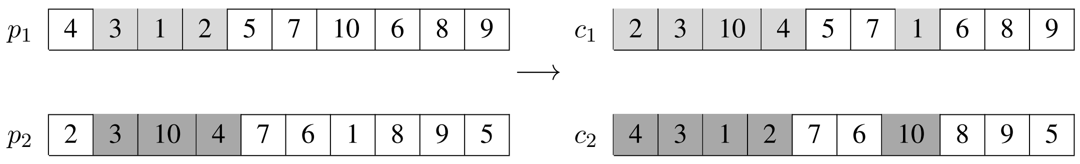


Figure 1.6: PMX Crossover

1.5.1.2 Order crossover

The order crossover operator suites circular permutation problems and it was proposed by Davis [21]. Two crossover sites are chosen uniformly at random. Substring between these two sites from first parent goes to the exactly same positions in the first offspring. Now, starting from the second crossover site in the second parent, the rest of the alleles are copied in the first offspring in the same order as they are in the second parent treating chromosome as circular. Similarly, the second offspring is generated by reversing the roles of two parents. Figure 1.7 illustrates the order crossover.

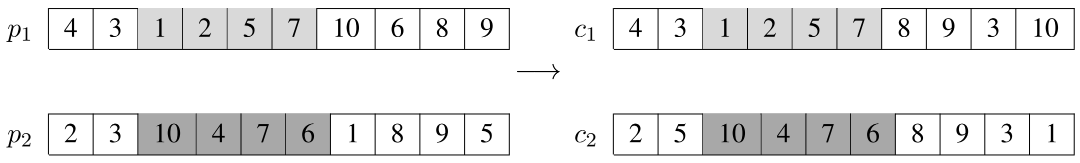

p_1 4 3 1 2 5 7 10 6 8 9
p_2 2 3 10 4 7 6 1 8 9 5
$\longrightarrow$
c_1 4 3 1 2 5 7 8 9 3 10
c_2 2 5 10 4 7 6 8 9 3 1

Figure 1.7: Order crossover

1.5.1.3 Cycle crossover

The cycle crossover was proposed by Oliver *et. al.* [27] with the intention of preserving as much as possible, the absolute position of alleles. This is achieved by dividing the alleles into cycles according to the composition of the parents. Here cycle is defined as a subset of alleles in the string. In each cycle, each allele occurs in pairs when the two parents are aligned. The method for generating an offspring is as follows:

1. The first unused position a of the first parent is chosen.
2. The allele at position a of the first parent is inserted into the child at the same position.
3. The allele in the second parent at the position a is noted and its position in the first parent is traced. Now a is reset to this location.
4. Repeat steps (2) and (3) until the allele which was selected first in this cycle is encountered again. When this happens, current cycle ends.
5. If any unused position is left, interchange the two parents and goto step (1), otherwise stop

Another offspring can be generated by reversing the role of two parents. Figure 1.8 illustrates the cycle crossover where different cycles are shown in different shades.

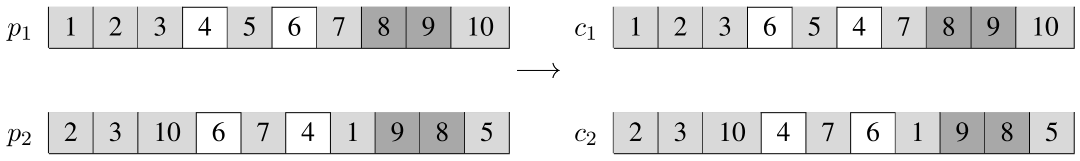


Figure 1.8: Cycle crossover

1.5.1.4 Uniform order based crossover

Uniform order based (UOB) crossover [21] was designed for order-based permutation problems. UOB crossover tries to copy alleles at each position of the first parent with probability p (Usually, $p = 0.5$) to the first child at exactly the same position. The remaining vacant positions are filled from the unused alleles in exactly the same order as they are in the second parent. Similarly, second child can be produced by exchanging the roles of two parents. Figure 1.9 illustrates UOB crossover with $p = 0.5$.

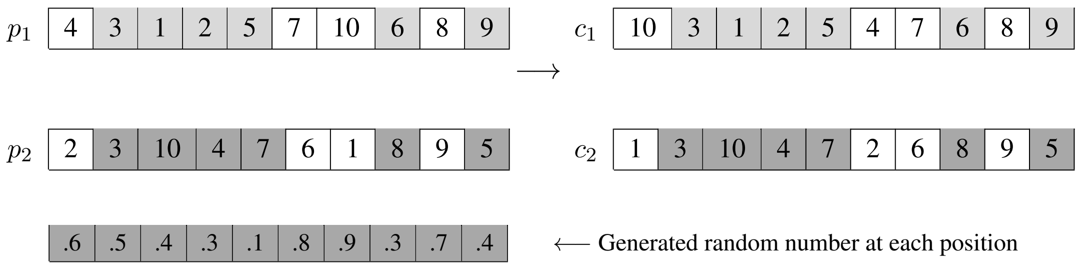


Figure 1.9: Uniform order based crossover

1.5.2 Mutation

Unlike simple genetic algorithm, where each gene is considered independently for mutation, in permutation representation, genes can not be considered independently. To overcome this problem, a number of suitable mutation operators have been proposed. The most common mutation operators for order-based problems are swap mutation, insert mutation and scramble mutation. For adjacency-based problems, inversion mutation is most suitable.

1.5.2.1 Swap mutation

Randomly, two positions are selected and the alleles at these two positions are swapped. In Figure 1.10, swap mutation is depicted, where alleles 3 and 9 are swapped.

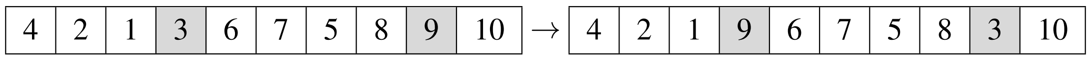


Figure 1.10: Swap mutation

1.5.2.2 Insert mutation

Randomly two different positions are selected and allele at second position is inserted immediately after the first position by shifting all the alleles affected one place towards the end. This is shown in Figure 1.11 where allele 9 is inserted immediately after the allele 3.

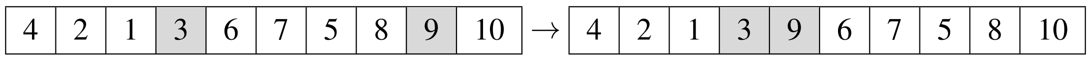


Figure 1.11: Insert mutation

1.5.2.3 Scramble mutation

Entire chromosome or a part of it is selected and alleles inside the selected portion are scrambled to produce a new chromosome. In Figure 1.12, the substring 3, 6, 7, 5 are scrambled.

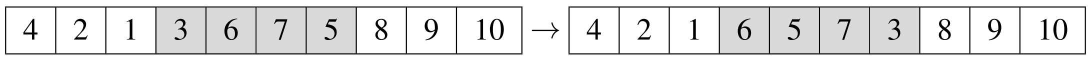


Figure 1.12: Scramble mutation

1.5.2.4 Inversion mutation

A permutation problem in which the adjacency of objects matters rather than relative ordering of objects, inversion mutation is most suitable. In this mutation, a substring is selected randomly and then the selected substring is inverted. Figure 1.13 illustrates the inversion mutation.

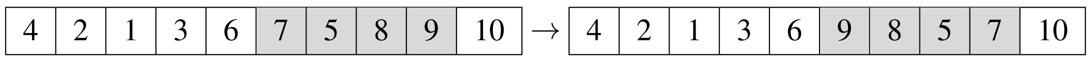


Figure 1.13: Inversion mutation

1.6 Grouping genetic algorithm (GGA)

Similar to permutation problems, grouping problems also require special attention as far as chromosome representation and genetic operators are concerned. Mühlenbin [28] observed that using the simple genetic algorithm for a given solution with g groups will have $g!$ distinct chromosomes to represent it. The degree of redundancy, using simple genetic algorithm, grows exponentially with number of groups, and hence, the simple genetic algorithm is not recommended for solving grouping problems. Also, a number of genetic algorithms have been proposed for solving grouping problems which represented chromosome as a permutation of objects and employed appropriate heuristics to decode a permutation into a valid solution for their respective grouping problems. However, permutation representation also suffer from redundancy when used for encoding a solution to a grouping problem [29]. Moreover, genetic operators used in simple or permutation-coded genetic algorithms are context insensitive and too much disruptive when applied to a grouping problem. Actually, building blocks in these genetic algorithms are not at all based on information about groups which is vital for solving any grouping problem. Falkenauer [29] introduced grouping genetic algorithm (GGA) to overcome these shortcomings. GGA employs altogether different solution representation scheme and genetic operators which are designed keeping in mind the special structure of grouping problems.

1.6.1 Representations in grouping genetic algorithm (GGA)

In the representation suggested by Falkenauer [30], chromosome consists of two parts. First part is called the standard object part and the second part is called the group part. Standard object part keeps the information about the group of each object. Group part simply lists the groups present in the standard object part on the basis of one gene per group. For example, consider the following chromosome for a grouping problem instance with ten objects:

ABACCDBABD:ABCD

The first part of the chromosome identifies each object with a group, i.e., this part shows that object 1 belongs to group A, object 2 belongs to group B, object 3 belongs to group A and so on. On the other hand, second part lists all the groups present in the chromosome, viz. A, B, C and D. There are many grouping problems such as bin packing problem where the number of groups may vary from one solution to another, and as a result, length of chromosome for such problems varies. Chromosome is of fixed length for those grouping problems, where the number of groups are fixed in advance, e.g., multiprocessor scheduling problem with minimum makespan objective. Usually but not always, genetic operators designed for this scheme utilize group part only. This scheme however suffers from redundancy, because group labels can be interchanged without affecting the solution represented, e.g. CACBBDACAD: CABD also represent the same solution as ABACCDBABD:ABCD, where one group consists of objects 1, 3 & 8, another group consists of objects 2, 7 & 9, yet another group consists of objects 4 & 5 and yet another group is composed of objects 6 & 10. In this scheme, there is an ordering among groups and results of genetic operators depend on this ordering. To reduce the effect of ordering, Falkenauer [30] suggested an inversion operator.

Another possible representation for grouping problems is to encode each solution as a set of groups [31], i.e., there is no ordering among groups. Depending on the problem, each group is represented as a sequence or a set of objects. For example, solution corresponding to the chromosome above can be represented as { {1, 3, 8}, {2, 7, 9}, {4, 5}, {6, 10} } in this scheme. This scheme does not label groups, and hence does not suffer from the problem of redundancy like the scheme described in the previous paragraph. Another advantage of this scheme is that the result of genetic operators depends only on composition of groups, and not on relative ordering among groups as there exists none. Hence, there is no need of inversion operator with this scheme

1.6.2 Recombination

To create an offspring, crossover operator proposed by Falkenauer [30] inserts the groups between the two randomly chosen crossover sites of the first parent at the first crossover site of the second parent. This makes the solution infeasible as some objects now belong to two groups. To restore the feasibility, such objects are deleted from the groups originally belonging to the second parent. Depending on the problem, next step may consist of adapting the resulting solution according to the hard constraints of the problem and cost function to optimize. In many grouping problems, this step eliminates those groups that got altered by the insertion. The objects belonging to these eliminated groups are reinserted into the solution via some problem specific heuristic. Second offspring can be produced in an analogous manner by reversing the role of two parents.

Crossover operator for the second representation is based on iteratively copying groups from one of the two parents to the offspring [31] and can be considered as analogue of uniform crossover for grouping problems. During each iteration, one of the two parent is selected (either alternately or with equal/unequal probability) and the group to be copied from selected parent is usually chosen based on some suitable problem specific criterion, though a random selection can also be made. Once a group has been copied to the offspring, objects corresponding to this group is deleted from both the parents and any group which becomes empty during the process is eliminated. Depending on the problem, all the affected groups can also be eliminated. This process is repeated either till one of the two parents becomes empty or till a specified termination condition is met. If some objects are left out in the offspring, then a problem specific heuristic is used to assign these objects to the child solution

1.6.3 Mutation

Mutation operators for grouping problems are usually designed based on one of the following strategies [30]:

- Eliminate some groups at random from the solution under consideration and then use some problem specific heuristic to reassign objects of these eliminated groups to the solution.

- Create a new group by moving objects from existing groups to it. The objects for movement are chosen either at random or based on some policy. Remove any group that becomes empty during this process from the solution.

- Randomly delete some objects from existing groups and then use some problem specific heuristic for reassigning these deleted objects into the solution.

1.7 Estimation of distribution algorithm

There exists a class of evolutionary algorithms that works by constructing and sampling an explicit probabilistic model that provides an estimation about the distribution of promising solutions in the search space [32]. Such algorithms are commonly known as probabilistic modelling based genetic algorithms (PMBGA) or estimation of distribution algorithms (EDAs). Estimation of distribution algorithm (EDA) [33, 34, 35, 36] is a relatively new variant of EAs, which has no counter part in nature. Instead of using genetic operators such as crossover and mutation, offspring in an EDA are produced by sampling an explicit probabilistic model that estimates the distribution of promising solutions. The underlying principle of EDA is to start with a population of solutions, then at every generation, a probabilistic model that tries to estimate the distribution of the selected solutions (sub-population) of the current population is constructed. After this, new offspring are generated by sampling this model. Finally, newly generated offspring replace current population either partly or wholly depending on the implementation, thereby creating the population for next generation. This process is carried out repeatedly until some termination criteria is satisfied. In some other EDA variants, instead of building the probabilistic model afresh during each generation, probabilistic model is based on promising solutions found since the beginning of the algorithm, and hence, this model is updated during each generation according to the composition of some selected solutions. Theoretical analysis regarding the convergence behaviour of some classes of EDAs can be found in [37, 38, 39].

There are two important decisions to make while designing an EDA.

- What probability model to use for approximating the distribution of promising solutions in the search space?
- How to sample the probability model to generate new offspring?

There are many ways to model the distribution of promising solutions depending upon the nature of the problem under consideration. Similarly, there are many ways to sample the probability model.

Mainly, three types of probability models are used in EDAs, viz. univariate, bivariate and multivariate models. These three models are described below.

Univariate model is one of the simplest model to approximate the distribution of promising solution in the search space. In univariate model, it is assumed that the variables are independent of each other. In other words, interaction among the variables is ignored even if there is any. Probability of any variable does not depend on any other variable, i.e., there is an unconditional probability associated with each variable. Hence, in univariate model, the probability of a candidate solution composed of n variables $y_1, y_2, \ldots, y_n$ is the product of probabilities of individual variables, i.e., $p(y_1,\ y_2,\ldots y_n)$ = $p(y_1).p(y_2)\ldots p(y_n)$, where $p(y_i)$ is is the probability of the variable y_i.

Univariate marginal distribution algorithm (UMDA) [35] is a variant of univariate EDAs that works on binary strings of candidate solutions and uses a probability vector to represent the distribution of promising solutions. The probability vector is defined as $p\ =\ (p_1,\ p_2,\ldots,\ p_n)$, where p_i is the probability of a 1 at the position i of the binary string. During each generation, probability for each position is calculated as follows:

$$p_i \leftarrow \frac{Number\ of\ solutions\ in\ the\ sub-population\ having\ 1\ at\ the\ i^{th}\ position}{Total\ number\ of\ solutions\ in\ the\ sub-population} \tag{1.1}$$

Another variant of univariate EDAs is population-based incremental learning (PBIL) [33] algorithm which also works on the binary strings. Like UMDA, it also uses probability vector to represent the distribution. However, here probability vector models the distribution of promising solutions since the beginning of the algorithm. PBIL starts by initializing all the values in the probability vector to 0.5. With such an initialization of probability vector, initially all solutions are equally likely to be generated upon sampling this probability vector. Then during each iteration, some new solutions are generated by sampling the probability vector. A sub population containing some best solutions among these newly generated solutions is formed and is used to update each element of the probability vector in the following way:

$$v_i \leftarrow \frac{Number\ of\ solutions\ in\ the\ sub-population\ having\ 1\ at\ the\ i^{th}\ position}{Total\ number\ of\ solutions\ in\ the\ sub-population} \tag{1.2}$$

$$p_i = (1-\lambda) * p_i + (\lambda * v_i) \tag{1.3}$$

where $\lambda \in (0, 1]$ is the learning rate which is a parameter to be specified by the user.

Bivariate model is used for problems where pair wise interactions between problem variables exist. This model takes into account these pairwise interactions. Several bivariate EDAs have been proposed. The mutual-information-maximizing input clustering (MIMIC) [40] is a variant in which a chain distribution is used to model the interaction between problem variables. Further, Baluja and Davies [41] proposed combining optimizers with mutual information trees (COMIT) method which was based on the use of dependency trees to model the distribution of promising candidate solutions. This method was found to be better than MIMIC. Another algorithm called bivariate marginal distribution algorithm (BMDA) [42] is based on the model in which several mutually independent dependency trees are formed.

There are many problems where there is a need to express the dependencies among the several problem variables. *Multivariate model* is used for such problems. The extended compact genetic algorithm (ECGA) [43] divides problem variables into different linkage groups and each group is considered as a single variable. The estimation of Bayesian network algorithm and learning factorized distribution algorithm [44] are based on the use of Bayesian network. The other algorithms which work on the assumption of multivariate interactions among variables are the hierarchical Bayesian optimization algorithm [45] and Markovianity based optimization algorithm (MOV) [46]. An extensive description of these approaches can be found in [32].

1.7.1 EDA for permutation problems

The existing EDAs which were designed for integer domain problems can be used to solve permutation-based problems by modifying the sampling step. The probabilistic model is sampled by using probabilistic logic sampling algorithm [47]. Many other integer-based EDAs such as UMDA [48], MIMIC [49], dependency-trees [50] have been used to solve permutation problems. Another way to deal with permutation problems through EDAs is to use real-valued EDAs in combination with random key encoding [25].

Apart from these two ways, many permutation-based EDAs are also proposed to deal with permutation problems. Bengoetxea *et al.* [49] proposed an approach which is based on Bayesian network model. Pelikan *et al.* [50] proposed a dependency-tree model based EDA (dtEDA) to deal with permutation problems. Edge histogram based sampling algorithm (EHBSA) [51] and node histogram based sampling algorithm (NHBSA) [52] ware also proposed. A recent addition to the list of EDAs for permutation problems is a distance-based ranking model [53] that is based on the generalized Mallows model [54].

1.8 Evolutionary algorithm with guided mutation

Evolutionary algorithm with guided mutation (EA/G) is a recent addition to the family of evolutionary algorithms. It was developed by Zhang *et al.* [55]. EA/G can be considered as a cross between genetic algorithm (GA) and estimation of distribution algorithm (EDA). EA/G has the features of both GA and EDA. As discussed already, GA uses genetic operators such as crossover and mutation to generate an offspring from the selected parents. GA directly utilizes only the location information of the solutions and does not make use of global information about the search space which can be collected by keeping track of all the solutions generated since the beginning of the algorithm. On the other hand, EDA relies only on a probability model to generate an offspring. This probability model characterizes the distribution of the promising solutions in the search space. An offspring is generated by sampling this probability model. Contrary to GA, EDA does not directly utilize the location information of solutions. Here, by the location information of a solution, we mean the information that can uniquely identify a solution in the search space of all solutions.

EA/G was developed to combine the best of both worlds, i.e., EA/G utilizes the location information of the solutions like GA and the global statistical information about the search space like EDA while generating an offspring. EA/G uses a mutation operator, called guided mutation (GM), to generate offspring. Guided mutation generates a new solution considering both the location information about the parent solution as well as the global statistical information about the search space, i.e., a solution is generated partly by sampling a probability model characterizing the global statistical information and partly by copying elements from its parent.

Though, Zhang *et al.* [55] proposed EA/G approach in the context of maximum clique problem, a general framework for EA/G can be extracted from their description. This general framework is presented in Algorithm 1.2. EA/G uses a probability vector $p = (p_1, p_2 \ldots p_n)$ to model the distribution of promising solutions in the search space since the beginning of algorithm in a manner which is exactly same as PBIL [33]. This probability vector is also updated in the same manner as PBIL. However, the probability vector is initialized differently. Each element p_i of this probability vector is initialized based on the composition of initial population in the following manner.

$$p_i \leftarrow \frac{Number\ of\ solutions\ in\ the\ initial\ population\ having\ 1\ at\ the\ i^{th}\ position}{Total\ number\ of\ solutions\ in\ the\ initial\ population} \tag{1.4}$$

As already mentioned, guided mutation operator generates an offspring partly by sampling the probability vector p and partly by copying elements from its parent. Algorithm 1.3 provides the pseudo-code of guided mutation operator where β is a parameter that controls the relative contribution of probability vector p and parent solution in the creation of offspring. Zhang *et al.* [55] applied guided mutation multiple times on the best solution of the population based on the proximate optimality principle (POP) [56]. The POP assumes that good solutions have similar structure.

Here, it is to be noted that EA/G also used the concept of partitioning the search space in the context of maximum clique problem. Idea was to explore different subspaces during different

Algorithm 1.2: The general framework of EA/G

```
1: $t \leftarrow 0$;
2: Generate an initial population *Pop(t)* consisting of $N$ random solutions;
3: Initialize the probability vector $p$;
4: Select best $\frac{N}{2}$ solutions from *Pop(t)* to form a parent set *parent(t)*, and then update the probability
   vector $p$;
5: Apply the *GM* operator $\frac{N}{2}$ on the best solution of population to generate $\frac{N}{2}$ solutions. Apply a repair
   operator on the newly generated solutions if necessary. Add all newly generated solutions along with
   those in *parent(t)* to form *Pop(t+1)*. If the stopping condition is met, return the best solution found
   so far;
6: $t \leftarrow t + 1$;
7: If all the solutions are equivalent, then re-initialize *Pop(t)* and then go to step 3;
8: Go to step 4;
```

Algorithm 1.3: The pseudo-code of guided mutation

```
1: Guided mutation operator $y = GM(p,\ x,\ \beta)$;
2: **Input:**
3: $p\ =\ (p_1,\ p_2,\ \dots,\ p_n)$ where $p \in [0,\ 1]^n$;
4: $x\ =\ (x_1,\ x_2,\ \dots,\ x_n)$ where $x \in \{0,\ 1\}^n$;
5: $\beta\ \in [0,\ 1]^n$;
6: **Output:**
7: $o\ =\ (o_1,\ o_2,\ \dots,\ o_n)$;
8: **for** $i\ =\ 1$ to $n$ **do**
9:     Generate a random number $r_1$ such that $0 \leq r \leq 1$;
10:    **if** $r_1 \leq \beta$ **then**
11:        Generate a random number $r_2$ such that $0 \leq r \leq 1$;
12:        **if** $r2 \leq p_i$ **then**
13:            $o_i$ = 1;
14:        **else**
15:            $o_i$ = 0;
16:        **end if**
17:    **else**
18:        $o_i$ = $x_i$;
19:    **end if**
20: **end for**
```

search phases. The subspace was partitioned based on the size of the largest clique found so far and the search process was guided to a new subspace as soon as a clique better than the largest clique so far is found. This subspace consists of all subgraphs whose cardinality exceeds the largest clique by at least 1 and at most Δ. However, this kind of partitioning can not be applied on all problems. For example, no such partitioning can be done in case of maximum weight clique problem where cardinality of the clique with largest weight may be less than several other cliques. So, we have ignored this feature while providing the general framework for EA/G.

1.9 Scope of the thesis

This thesis is focused primarily on solving some $\mathcal{NP}$-hard combinatorial optimization problems through two evolutionary techniques, viz. genetic algorithm (GA) and evolutionary algorithm with guided mutation (EA/G). GA is a well known evolutionary algorithm, whereas EA/G is a relatively new and under explored evolutionary technique. We have considered six problems in this thesis. Out of these six problems, first three problems are subset selection problems, fourth problem has aspects of subset selection as well as permutation, fifth problem is a permutation problem, whereas the last one is a grouping problem. These problems are not only challenging from theoretical point of view, but also have many practical applications in diverse areas such as networks, data mining, transportation and logistics, production scheduling etc. We have developed hybrid evolutionary approaches combining either GA or EA/G with local search heuristics for these problems. In addition to local search heuristics, we have also made use of problem specific knowledge in the design of genetic operators wherever possible. Actually, the field of solving combinatorial optimization problems using metaheuristic techniques has been advanced to a level where no new approach for solving any combinatorial optimization problem can compete with the state-of-the-art approaches unless it makes proper use of problem specific knowledge.

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

Chapter 2 deals with the set packing problem (SSP). Given a finite set $I = \{1, \ldots, n\}$ of objects and $T_j, j \in J = \{1, \ldots, m\}$, a list of m subsets of I, a packing P is a subset of I such that $|T_j \cap P| \leq 1$, $\forall j \in J$, i.e., at most one object of T_j can be in P. Each set $T_j, j \in J = \{1, ..., m\}$ is a set of exclusive constraints between some objects of I. Each object $i \in I$ has a positive weight c_i. The objective of the SPP is to find out a packing that maximizes the total weight of the objects it contains without violating any constraints. We have developed a hybrid approach comprising an EA/G and a local search to solve the SPP. We have compared our hybrid approach with the state-of-the-art approaches on standard benchmark instances for the problem. Computational results show the effectiveness of our approach.

Chapter 3 presents a hybrid approach combining EA/G with an improvement operator for the minimum weight dominating set problem (MWDS). For any graph, a dominating set is a set of nodes such that each node of the graph either belongs to the dominating set or adjacent to a node in the dominating set. The minimum dominating set problem (MDS) seeks a dominating set on a graph G whose cardinality is least among all the possible dominating sets on G. MWDS is a generalization of MDS where nodes are assumed to have positive weights and the goal is to find a dominating set with minimum sum of node weights. In addition to the hybrid EA/G approach, an improved version of a greedy heuristic available in literature is also presented which has been used in our hybrid approach for initial solution generation and repairing an infeasible solution. We have compared the performance of our hybrid EA/G approach with the state-of-the-art approaches on standard benchmark instances comprising general graphs as well as unit disk graphs (UDG). Computational results show the superiority of our approach in terms of solution quality as well as execution time on both kind of graphs.

Chapter 4 addresses the dominating tree problem (DTP). Given an undirected, connected, edge-weighted graph, the DTP seeks a tree of minimum weight on this graph so that each node of the graph either belongs to the tree or is adjacent to a node in the tree, i.e., the set of the nodes present in the tree should form a dominating set. In this chapter, we present a heuristic and an EA/G approach to solve the DTP. The heuristic presented is an improved version of a heuristic available in the literature. On various types of disk graphs, our heuristic produced better results in comparison to the state-of-the-art heuristic approaches available in the literature. Our EA/G approach also obtained better results on most of the instances in a much shorter time in comparison to two previously proposed metaheuristic approaches which are the only metaheuristic approaches proposed so far for the DTP.

Chapter 5 is devoted to the order acceptance and scheduling (OAS) problem in a single machine environment where orders are supposed to have release dates and sequence dependent setup times are incurred in switching from one order to next in the schedule. Actually, in a make-to-order system, customers put their orders at a time for processing. Due to this an additional wait time is incurred to receive the product. But it gives system the flexibility to customize the processing of these orders according to its capability. Tight order delivery deadline requirements may force a system with limited processing capability to first decide which orders should be accepted and then where to put the accepted orders in the processing sequence. The OAS problem deals with this class of problems where acceptance and timely delivery of orders play important role in generating revenue for the system. The OAS problem has two foldsfirst is to decide which orders should be accepted and the second is sequencing of the accepted orders. The OAS problem can be considered as a mix of knapsack and scheduling problems. If there is no scheduling component then the OAS problem reduces to knapsack problem. On the other side, if all the orders are accepted then the OAS problem reduces to a single machine scheduling problem. The coupled decisions of the OAS problem makes it a complex problem, but solving this problem maximizes the net revenue gained through the production system. We have developed two hybrid metaheuristic approaches, viz. a hybrid steady-state GA (SSGA) and a hybrid EA/G for OAS problem. We have compared our approaches with two state-of-the-art approaches for OAS problem and the computational results show the superiority of the proposed approaches in terms of solution quality. However, in terms of computational times, previous two approaches are faster than SSGA and EA/G approaches on most of the instances. As far as comparison between the two proposed approaches is concerned, SSGA and EA/G obtained solutions close to each other on most of the instances. Though, SSGA is slightly better than EA/G in terms of solution quality, it is slower.

Chapter 6 presents two hybrid metaheuristic approaches, viz. a hybrid GA and a hybrid artificial bee colony (ABC) algorithm for a single machine scheduling problem where tardiness cost of a job increases stepwise with various due dates. This kind of tardiness cost occurs in several practical scenarios mostly related to transportation. Transportation services are not always available and a finished job has to wait till the next transportation service is available. Hence, the delivery time of a job finished anywhere after the departure of one service and before the departure of the next service remains unaffected at the end customer. As a result, such a job incurs the same tardiness cost irrespective of its exact completion time. Hence, the tardiness cost in this situation increases in a stepwise manner. Another situation where tardiness costs of jobs increase in a stepwise manner arises when modes of shipping (rail, road, air etc.) of jobs depend on their completion times with respect to their various due dates. The objective of the scheduling problem is to find a schedule that minimizes the sumtotal of the tardiness costs of all the jobs. Two versions of the scheduling problem are considered. In the first version, all jobs are assumed to be available for processing at the beginning, whereas in the latter version jobs have release dates. For both versions, we have employed a local search to further improve the solutions obtained through our metaheuristic approaches. To the best of our knowledge, our approaches are the first metaheuristic approaches for the latter version of the problem. For the first version, we have compared our approaches with the state-of-the-art approaches available in the literature. Computational results show the superiority of our approaches over previous approaches in terms of solution quality and running time both. For the latter version, hybrid ABC algorithm based approach outperformed the hybrid GA based approach in terms of solution quality and running time both.

The penultimate chapter of the thesis, i.e., Chapter 7 is concerned with a grouping problem, viz. the registration area planning (RAP) problem in mobile wireless networks. The last two decades have witnessed a phenomenal growth in the number of cellular wireless network users which in turn stressed the need to utilize the limited network bandwidth in an efficient manner. The network bandwidth is consumed not only by user traffic, but also by control traffic needed for ensuring the mobility of users. As we don't have any control over the volume of user traffic, all attempts to efficiently use bandwidth are based on frequency reuse and minimizing the control traffic. The RAP problem seeks a partition of the cells of the network into contiguous areas called registration areas so that the bandwidth consumed by control signals is minimized. In this chapter, we present a steady-state grouping genetic algorithm with local search to solve this problem. We have compared our approach with the state-of-the-art approaches reported in the literature. Computational results show the superiority of our approach in all the aspects, viz. quality of best solution found, average solution quality, average total execution time and standard deviation of solution values.

Chapter 8 concludes the thesis by summarizing the contributions of the thesis. In addition, some directions for future research are also suggested.

Chapter 2

Set Packing Problem

2.1 Introduction

The set packing problem (SPP) is an important combinatorial optimization problem having applications in diverse domains. Following the notational conventions used in [1, 2], SPP can be defined as follows: Given a finite set $I = \{1, \ldots, n\}$ of objects and $T_j, j \in J = \{1, \ldots, m\}$, a list of m subsets of I, a packing P is a subset of I such that $|T_j \cap P| \leq 1, \forall j \in J$, i.e., at most one object of T_j can be in P. Each set $T_j, j \in J = \{1, ..., m\}$ is a set of exclusive constraints between some objects of I. Each object $i \in I$ has a positive weight c_i. The objective of the SPP is to find out a packing that maximizes the total weight of the objects it contains without violating any constraints. Formally, the SPP can be formulated as:

$$Max \;\; z = \sum_{i \in I} c_i x_i \tag{2.1}$$

$$\sum_{i \in I} t_{i,j} x_i \leq 1, \; \forall j \in J \tag{2.2}$$

$$x_i \in \{0, 1\}, \; \forall i \in I \tag{2.3}$$

$$t_{i,j} \in \{0, 1\}, \; \forall i \in I, \; \forall j \in J \tag{2.4}$$

where x_i is a binary variable which indicates whether an object $i \in P$ $(x_i = 1)$ or not $(x_i = 0)$, c_i is the cost of the object i, and $t_{i,j}$ represents exclusive constraints with $t_{i,j} = 1$ if the object i belongs to set T_j, otherwise $t_{i,j} = 0$. There exists a second formulation also for SPP, which defines SPP as follows: Given a universal set U, and a collection S of subsets of U, find a subset S' of S with maximum cardinality such that no two subsets in S' have common elements.

This formulation is equivalent to the unweighted version of the first formulation, because if we replace each subset in the second formulation with an object with unit weight and each set of different subsets of S sharing common elements in the second formulation with T_j containing corresponding objects, we will get the SPP defined exactly as per the first formulation.

The SPP is an $\mathcal{NP}$-hard problem in the strong sense [57]. The best exact method known till date for solving the SPP is proposed by [58]. It is essentially, a branch & cut method based on polyhedral theory to obtain facets. However, such exact methods can find optimal solution for small instances only. In order to solve larger instances, several metaheuristics have been proposed. Delorme *et al.* [1] presented a greedy randomized adaptive search procedure (GRASP) metaheuristic. Delorme *et al.* [1] worked on two different types of instances, i.e., random instances and railway problem instances. Gandibleux *et al.* [2] proposed an ant colony optimization (ACO) approach and presented their results on random instances. Later, two different versions of ACO were presented in [59] and [60] and both ACO approaches were applied on railway problem instances.

Several important practical applications of the SPP have been reported in the literature. Rönnqvist [61] considered a SPP formulation for a cutting stock problem and solved it with the help of Lagrangian relaxation combined with sub-gradient optimization. Zwaneveld *et al.* [62] worked on a real railway feasibility problem as a SPP formulation and solved it exactly using reduction tests and a branch & cut method. Kim [63] considered a ship scheduling problem as a SPP and solved it using LINDO software. Mingozzi *et al.* [64] used a SPP formulation to find out the bounds for a resource constrained project scheduling problem with the help of a greedy method. Tajima and Misono [65] used a SPP formulation to solve airline seat allocation/reallocation problem. Delorme *et al.* [66] modelled the railway infrastructure saturation problem as an unicost SPP and solved it using GRASP. Rossi and Smriglio [67] worked on a ground holding problem formulated as a SPP and solved it with a branch & cut method.

Lusby *et al.* [68] formulated the problem of routing trains through junctions, as a SPP and presented a branch-and-price algorithm for it. Given the detailed track layout of a junction and a timetable containing the arrival and departure timings for a set of trains, this problem in its simplest form seeks to maximize the number of trains which can be assigned a route through this junction. The tracks are divided into sections and at a particular time only one train can pass through a section. To map this problem into the SPP, time is also divided into a number of short periods and pairs are made out of track sections and resulting time periods. The possible paths of trains through the junction can then be represented in terms of track section time period pairs and can be treated like objects in the set packing problem. The constraints arise as a result of conflicts among different paths, i.e., different paths that use the same track section time period pair can not be put together in a solution. Obviously, this is an instance of SPP. If the goal is to maximize the number of trains then unicost SPP will result. In case, we have different priority for different trains or different preferences for different paths then we will get exactly the SPP formulation described above.

In this chapter, we present a hybrid approach comprising evolutionary algorithm with guided mutation (EA/G) and a local search to solve the SPP. We have compared our hybrid EA/G approach with the state-of-the-art metaheuristic approaches. Computational results demonstrate the effectiveness of our approach.

The remainder of this chapter is organized as follows: Section 2.2 describes our hybrid EA/G approach for the SPP. Computational results are reported in Section 2.3, whereas Section 2.4 contains some concluding remarks.

2.2 EA/G for the SPP

Our proposed hybrid EA/G approach for the SPP is inspired by the approach of Zhang *et al.* [55] for the maximum clique problem. Before starting EA/G for the SPP, we precompute a *conflict set* cs_i for each object $i \in I$. Here, by the conflict set cs_i for object i, we mean the set of those objects which can not be packed with object i. If the object i has been included in a packing, then all objects belonging to cs_i cannot be present in this packing. The pseudo-code of our EA/G approach for the SPP is given in Algorithm 2.1.

The main components of our EA/G approach for the SPP are described below:

2.2.1 Solution encoding

Subset encoding has been used to represent a solution, i.e., each solution is represented directly by the set of objects it contains.

2.2.2 Initial solution

In order to have sufficient diversity in the initial population, each initial solution is generated randomly by using an iterative procedure which is as follows: Initially, let U be a set containing all the objects and let S be an empty packing. At each iteration, an object from U is selected

Algorithm 2.1: The pseudo-code of our EA/G approach for the SPP

```
1: At generation $t \leftarrow 0$, an initial population Pop(t) consisting of $N$ solutions are generated randomly ($N$ should be even);
2: Initialize the probability vector $p$ by using Algorithm 2.2;
3: Select best $\frac{N}{2}$ solutions from Pop(t) to form a parent set parent(t), and then update the probability vector $p$ by using Algorithm 2.3;
4: Apply the GM operator $\frac{N}{2}$ times on the best solution $s_b$ in the parent(t) in order to generate $\frac{N}{2}$ new solutions. A repair operator is applied to each generated solution, if necessary, and then the local search is applied to each generated solution. Add all $\frac{N}{2}$ newly generated solutions alongwith those in parent(t) to form Pop(t+1). If the stopping condition is met, return the packing with the maximum weight found so far;
5: $t \leftarrow t + 1$;
6: If all the solutions are equivalent, then apply the perturbation procedure to reinitialize Pop(t), and then go to step 2 (line number 2);
7: Go to step 3 (line number 3);
```

randomly using the roulette wheel selection method, where the probability of selection of an object is proportional to the ratio of the cost of that object to the number of unselected objects conflicting with that object. If at any iteration, it is found that an object belonging to U does not have conflict with any other object in U, then it is immediately selected without the use of roulette wheel selection. The selected object is added to S. All unselected objects conflicting with this selected object and the selected object itself are deleted from U. This iterative process is repeated until U becomes empty.

2.2.3 Initialization and update of the probability vector

Similar to [55], to model the distribution of promising solutions in the search space, an univariate marginal distribution (UMD) model is used. In this model, a probability vector $p = \{p_1, p_2, \ldots, p_n\} \in [0, 1]^n$ is used to characterize the distribution of promising solutions in the search space, where p_i is the probability of the i^{th} object of the set I to be present in a packing. The *guided mutation (GM)* operator uses this probability vector to generate offspring at each generation t. The probability vector is initialized using N initial solutions. However, in addition to using the information about the number of initial solutions containing object i, our method also make use of the ratio of the weight of object i to the cardinality of its conflict set while initializing the probability for object i. It is to be noted that Zhang *et al.* [55] only utilized information about initial solutions for initializing the probability vector. The pseudo-code for initializing a probability vector p for the SPP is given in Algorithm 2.2.

Algorithm 2.2: The pseudo-code for initializing a probability vector p

```
1: Compute $val_i \leftarrow \frac{c_i}{|cs_i|}, \forall i \in I$;
2: Compute $y_i \leftarrow$ number of initial solutions containing object $i$, $\forall i \in I$;
3: Compute $p_i \leftarrow \frac{(y_i+val_i)}{(N+val_i)}, \forall i \in I$;
```

At each generation t, a parent set *parent(t)* is formed by selecting best $\frac{N}{2}$ solutions of the current population *pop(t)*. The *parent(t)* is used for updating the probability vector p in generation t. The pseudo-code for updating the probability vector is given in Algorithm 2.3, where $\lambda \in (0, 1]$ is the learning rate. The contribution of solutions in *parent(t)* to the updated probability vector p depends on the value of λ, i.e., higher the value of λ, more is the contribution of solutions in *parent(t)*.

Algorithm 2.3: The pseudo-code for updating a probability vector p in generation t

```
1: Compute $z_i \leftarrow$ number of solutions in parent(t) containing object $i$, $\forall i \in I$;
2: Compute $p_i \leftarrow (1-\lambda)p_i + \lambda \frac{z_i}{\frac{N}{2}}, \forall i \in I$;
```

2.2.4 Guided mutation (GM) operator

The *GM* operator is applied on the best solution s_b in the current population to generate new solutions. The pseudo-code of generating a new solution through *GM* operator is given in Algorithm 2.4, where $u01$ is a uniform variate in [0, 1] and $\beta \in [0, 1]$ is a tunable parameter. $y = (y_1, y_2, \ldots, y_{yc})$ is a new packing whose objects are either sampled from the probability vector p or directly copied from the best solution s_b in *parent(t)*. The variable yc is number of objects in y.

2.2.5 Repair and improve operators

Each solution, say s, generated through the *GM* operator is checked for feasibility. If s is infeasible, then a repair operator is applied to make s feasible. Next, an improvement operator is applied on s to enhance its fitness. These two operators are described below:

Repair operator: This operator iteratively deletes objects from s till it becomes feasible. To implement this operator, a counter variable cnt_i is associated with each object i in s. The purpose of cnt_i is to keep track of the number of objects in cs_i (i.e. conflict set of object i) that are also present in s. During each iteration, an object, say j, is deleted from s randomly with the

Algorithm 2.4: The pseudo-code of generating a solution through *GM* operator

```
1:  $yc \leftarrow 0$;
2:  $y \leftarrow \emptyset$;
3:  for each object $i \in I$ in some random order do
4:      $r_1 \leftarrow u01$;
5:      if $(r_1 < \beta)$ then
6:          $r_2 \leftarrow u01$;
7:          if $(r_2 < p_i)$ then
8:              $y_{yc} \leftarrow i$;
9:              $yc \leftarrow yc + 1$;
10:         end if
11:     else
12:         if $(i \in s_b)$ then
13:             $y_{yc} \leftarrow i$;
14:             $yc \leftarrow yc + 1$;
15:         end if
16:     end if
17: end for
18: return $y$;
```

help of roulette wheel selection method, where the probability of selection of j is proportional to the ratio of cnt_j to the cost of the object j, i.e., c_j. Note that no object, say i, whose associated counter cnt_i is zero will participate in this operation. The counter variables are updated and another iteration begins. This process is repeated till all the counter variables corresponding to the objects present in s become zero.

Improvement operator: Improvement operator iteratively tries to add unassigned objects to s without violating the feasibility constraints. During each iteration, first, all candidate objects which can be added to s without violating the feasibility are determined, and put in a set, say U. After that, a counter variable cu_i, which is maintained for each object $i \in U$, is set equal to the number of objects in U which also belong to cs_i. If there are objects in U with cu_i equal to zero then all such objects are added to s and deleted from U. If U is not empty, then an object, say j, is selected randomly from U for adding it to s with the help of roulette wheel selection method, where the probability of selection of j from U is proportional to the ratio of the cost of the object j, i.e., c_j to cu_j. The object j is added to s and deleted from U and next iteration begins. This process is repeated until U becomes empty.

2.2.6 Local search

To further improve the quality of each solution obtained after an application of EA/G for the SPP, a local search is applied on each solution. The local search uses first-improvement strategy based 1-1 exchange (an object in a packing is tried to be replaced with an object not in the packing to improve the cost of the packing) and 1-2 exchange (an object in a packing is tried to be replaced with two objects not in the packing to improve the cost of the packing) one after the other. However, 1-1 exchange local search is not applied on unicost instances as $c_i = c_j, \forall i, j \in I$. In order to maintain a balance between the quality of the solutions and execution time, we have allowed at most Num_{Ex1-1} successful 1-1 exchanges and at most Num_{Ex1-2} successful 1-2 exchanges in the local search where Num_{Ex1-1} and Num_{Ex1-2} are the parameters to be determined empirically. Local search can terminate prematurely also if no further beneficial exchange exists. The ACO approach of [2] also uses the first improvement 1-1 exchange on non-unicost instances, but in a non-iterative manner. On the other hand, GRASP approach of [1] iteratively uses first improvement 0-1 exchange (an object not in the packing is tried for insertion into the packing), 1-1 exchange, 2-1 exchange (two objects in a packing is tried to be replaced with an object not in the packing to improve the cost of the packing) and 1-2 exchange till it is not possible to improve the solution any further.

2.2.7 Perturbation method to diversify the current population

When all solutions in the current population *pop(t)* are same, then it can mean that our approach has been trapped in a local optima. To escape from this situation, a perturbation method is applied on each solution. The aim of such an strategy is to find some new solutions and consequently provide diversity in the current population *pop(t)*.

In this method, initially, all the objects which are not in the solution, say s, are added to a set, say U. Iteratively an object i is selected randomly from U with the help of roulette wheel selection method, where the probability of selection of i is proportional to the ratio of the cost of the object i, i.e., c_i to the cardinality of cs_i. After selecting i from U, all those objects of U which also belong to cs_i are deleted from U. The selected object i is added to a second set, say V. This procedure is continued until U becomes empty or at most $f \times |s|$ number of objects are added to V, where $0 < f \leq 1$ is a parameter to be determined empirically. Hereafter, each object j in s is checked one-by-one for conflict against all objects in V. If a conflict is detected, then the conflicting object j is deleted immediately from s. Once each object in

s is checked, all objects in V are added to s. Next, the improvement operator (as described in Section 2.2.5) is applied to s for exploring the possibility of further enhancements. The pseudo-code of perturbation method is given in Algorithm 2.5.

Algorithm 2.5: The pseudo-code of perturbation method

```
1:  $U \leftarrow I - s$;
2:  $k \leftarrow 1$;
3:  while ($U \neq \emptyset$ $or$ $k < (f \times |s|)$) do
4:      $i \leftarrow$ random select an order from $U$ using RWS;
5:      $U \leftarrow U \backslash \{U \cap cs_i\}$;
6:      $V \leftarrow V \cup \{i\}$;
7:      $k \leftarrow k + 1$;
8:  end while
9:  $s \leftarrow s \cup V$;
10: return $s$
```

2.3 Computational results

Our hybrid EA/G approach for the SPP has been implemented in C and executed on a Core 2 Duo system with 2 GB RAM running under Fedora 12 at 3.0GHz. `gcc 4.4.4-10` compiler with `O3` flag has been used to compile the C program for our approach. In all computational experiments with our approach, we have set $N = 30$, $\beta = 0.9$, $\lambda = 0.5$, $f = 0.7$, $Num_{Ex1-1} = 4$ and $Num_{Ex1-2} = 6$. 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. We have allowed our approach to execute for 200 iterations. Like GRASP [1] and ACO [2] approaches, EA/G approach is also executed 16 independent times on each instance.

There are two kinds of test instances for set packing problem – random instances and railway problem instances [1]. Only random instances are publicly available. Railway problem instances are not publicly available as these instances contain the confidential data pertaining to French railways (SNCF). Therefore, we report the results of hybrid EA/G approach on random instances only.

Tables 2.1 and 2.2 report the characteristics of instances, and results obtained by CPLEX 6.0 solver (0/1 solution), GRASP approach [1] (GRASP), ACO approach [2] (ACO) and our hybrid EA/G approach (EA/G). Table 2.1 presents instances with up to 200 variables, while Table 2.2 presents instances with 500 and 1000 variables. In these two tables, for each instance, columns *Var*, *Cnst* and *Density* indicate the number of objects, the number of constraints and the percentage of non-null elements in the constraint matrix respectively. Column *Max_One* contains the size of largest set T_j over all $j \in \{1, \ldots, m\}$, and column *Weight* indicates the interval in which the costs c_i are distributed for each instance. Note that the instances corresponding to the interval [1-1] are instances of the unicost SPP. As reported in [2], CPLEX 6.0 solver can find the optimal values (column Opt) in reasonable time (column TET) for instances with up to 200 variables which is shown in Table 2.1. However, CPLEX 6.0 solver is not able to solve all larger instances. In such cases, the best known value (column BKV in Table 2.2) is reported. When CPLEX 6.0 solver is not able to find optimal solution, the best solution value obtained is marked with an asterix ('*'). For GRASP, ACO and EA/G approaches, we report the best solution (column Best) obtained, average solution quality (column Avrg) and average execution time in seconds (column ATET) over 16 runs in Table 2.1 and Table 2.2. Results of CPLEX 6.0 solver, GRASP and ACO approaches are taken from [2]. To present an overall picture, in Figure 2.1, we have plotted the ratios of the best solution found by each of the three methods (EA/G, GRASP and ACO) to the optimal/ BKV for each instance of Tables 2.1 and 2.2. For the sake of clarity, only the name of alternating instances are shown in this figure.

Tables 2.1 and 2.2 and Figure 2.1 clearly show the superiority of hybrid EA/G approach over GRASP and ACO approaches in terms of solution quality. Out of a total of 56 instances, the best solution obtained by EA/G is better than GRASP on 12 instances and same as GRASP on remaining instances. In terms of average solution quality, EA/G is better than GRASP on 23 instances, same as GRASP on 15 instances and worse than GRASP on 18 instances. The best solution obtained by EA/G is better than ACO on 7 instances, same as ACO on 48 instances and worse than ACO on 1 instance. In terms of average solution quality, EA/G is better than ACO on 35 instances, same as ACO on 13 instances and worse than ACO on 8 instances. There are 6 instances on which the best solution found by EA/G is better than those of GRASP and ACO both. Similarly, there are 20 instances on which average solution quality of EA/G is better than GRASP and ACO both. EA/G improved the BKVs reported in [2] on 3 instances. Figure 2.1 also shows that whenever our approach has found a solution inferior to BKV, it is only slightly inferior when compared with other two approaches in a similar situation.

Table 2.1: Comparison of EA/G with GRASP [1] and ACO [2] on instances with upto 200 variables

Instance	**Characteristics**					**0/1 Solution**		**GRASP**			**ACO**			**EA/G**		
	Var	Cnst	Density	Max_One	Weight	Opt	TET	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET
pb100rnd01	100	500	2.0%	2	[1-20]	372	2.92	372	372.00	1.97	372	372.00	3.33	372	372.00	0.38
pb100rnd02	100	500	2.0%	2	[1-1]	34	0.60	34	34.00	1.31	34	34.00	2.00	34	34.00	0.22
pb100rnd03	100	500	3.0%	4	[1-20]	203	7.81	203	203.00	1.14	203	203.00	2.00	203	203.00	0.37
pb100rnd04	100	500	3.0%	4	[1-1]	16	52.86	16	**16.00**	1.29	16	15.56	0.67	16	15.69	0.18
pb100rnd05	100	100	2.0%	2	[1-20]	639	0.01	639	639.00	0.80	639	639.00	1.67	639	639.00	0.35
pb100rnd06	100	100	2.0%	2	[1-1]	64	0.01	64	64.00	0.69	64	64.00	1.00	64	64.00	0.14
pb100rnd07	100	100	2.9%	4	[1-20]	503	0.00	503	503.00	1.00	503	503.00	1.00	503	503.00	0.38
pb100rnd08	100	100	3.1%	4	[1-1]	39	0.02	39	38.75	0.57	39	38.68	0.67	39	**38.81**	0.21
pb100rnd09	100	300	2.0%	2	[1-20]	463	0.49	463	463.00	1.26	463	463.00	1.67	463	463.00	0.38
pb100rnd10	100	300	2.0%	2	[1-1]	40	1.13	40	**40.00**	1.28	40	39.62	1.00	40	39.88	0.24
pb100rnd11	100	300	3.1%	4	[1-20]	306	0.48	306	306.00	0.68	306	306.00	1.67	306	306.00	0.40
pb100rnd12	100	300	3.0%	4	[1-1]	23	6.80	23	23.00	1.13	23	22.93	0.33	23	23.00	0.21
pb200rnd01	200	1000	1.5%	4	[1-20]	416	8760.73	416	415.18	7.32	416	415.25	27.33	416	**416.00**	1.66
pb200rnd02	200	1000	1.5%	4	[1-1]	32	156109.36	32	32.00	7.35	32	31.56	14.67	32	32.00	0.76
pb200rnd03	200	1000	1.0%	2	[1-20]	731	5403.23	726	722.81	10.81	729	725.12	44.33	731	**727.00**	1.85
pb200rnd04	200	1000	1.0%	2	[1-1]	64	63970.91	63	**63.00**	9.12	64	62.93	24.33	63	62.75	0.92
pb200rnd05	200	1000	2.5%	8	[1-20]	184	1211.37	184	184.00	4.62	184	182.56	16.00	184	184.00	1.07
pb200rnd06	200	1000	2.5%	8	[1-1]	14	8068.20	14	13.37	3.48	14	12.87	4.00	14	**13.50**	0.55
pb200rnd07	200	200	1.5%	4	[1-20]	1004	0.02	1002	1001.12	4.20	1004	1003.50	6.33	1004	**1003.94**	1.61
pb200rnd08	200	200	1.5%	4	[1-1]	83	0.04	83	**82.87**	2.71	83	82.75	2.67	83	82.81	0.80
pb200rnd09	200	200	1.0%	2	[1-20]	1324	0.01	1324	1324.00	3.75	1324	1324.00	7.33	1324	1324.00	1.31
pb200rnd10	200	200	1.0%	2	[1-1]	118	0.02	118	118.00	3.64	118	118.00	4.00	118	118.00	0.76
pb200rnd11	200	200	2.5%	8	[1-20]	545	0.33	545	544.75	2.36	545	**545.00**	4.33	545	545.00	1.65
pb200rnd12	200	200	2.6%	8	[1-1]	43	1.70	43	43.00	1.01	43	43.00	1.33	43	43.00	0.80
pb200rnd13	200	600	1.5%	4	[1-20]	571	830.39	571	566.43	6.01	571	568.50	20.33	571	**570.75**	1.74
pb200rnd14	200	600	1.5%	4	[1-1]	45	10066.91	45	**45.00**	3.92	45	44.43	8.67	45	44.62	0.75
pb200rnd15	200	600	1.0%	2	[1-20]	926	12.20	926	926.00	4.22	926	926.00	27.00	926	926.00	1.72
pb200rnd16	200	600	1.0%	2	[1-1]	79	14372.85	79	78.31	6.80	79	**78.37**	15.33	79	78.12	0.98
pb200rnd17	200	600	2.5%	8	[1-20]	255	741.52	255	251.31	3.61	255	253.25	11.00	255	**254.19**	1.22
pb200rnd18	200	600	2.6%	8	[1-1]	19	19285.06	19	18.06	2.35	19	18.12	3.00	19	**18.19**	0.65

Table 2.2: Comparison of EA/G with GRASP [1] and ACO [2] on instances with 500 and 1000 variables

Instance	**Characteristics**					**0/1 solution**	**GRASP**			**ACO**			**EA/G**		
	Var	Cnst	Density	Max_One	Weight	BKV	Best	Avrg	ATET	Best	Avrg	ATET	Best	Avrg	ATET
pb500rnd01	500	2500	1.2%	10	[1-20]	323*	323	319.38	32.08	323	319.87	154.67	323	**321.31**	7.03
pb500rnd02	500	2500	1.2%	10	[1-1]	24*	24	**23.69**	25.62	24	23.06	26.67	**25**	23.56	5.16
pb500rnd03	500	2500	0.7%	5	[1-20]	776*	772	767.63	70.33	776	**772.75**	244.00	776	771.38	10.61
pb500rnd04	500	2500	0.7%	5	[1-1]	61*	61	60.13	57.30	61	60.06	69.00	**62**	**60.38**	6.06
pb500rnd05	500	2500	2.2%	20	[1-20]	122	122	121.50	15.48	122	120.62	71.00	122	**121.56**	3.92
pb500rnd06	500	2500	2.2%	20	[1-1]	8*	8	**8.00**	12.08	8	7.87	9.67	8	7.88	2.27
pb500rnd07	500	500	1.2%	10	[1-20]	1141	1141	**1141.00**	13.43	1141	**1141.00**	60.00	1141	1139.94	9.64
pb500rnd08	500	500	1.2%	10	[1-1]	89	89	88.25	15.80	89	88.12	21.67	89	**88.94**	4.75
pb500rnd09	500	500	0.7%	5	[1-20]	2236	2235	**2235.00**	23.44	2236	2234.43	84.33	2236	2232.81	12.49
pb500rnd10	500	500	0.7%	5	[1-1]	179	179	178.06	18.20	179	178.31	44.67	179	**178.56**	6.12
pb500rnd11	500	500	2.3%	20	[1-20]	424	423	419.31	19.25	424	418.25	37.33	424	**421.56**	7.92
pb500rnd12	500	500	2.2%	20	[1-1]	33*	33	**33.00**	11.91	33	32.62	8.00	33	32.88	4.00
pb500rnd13	500	1500	1.2%	10	[1-20]	474*	474	**470.00**	32.88	474	468.00	105.00	474	468.56	8.90
pb500rnd14	500	1500	1.2%	10	[1-1]	37*	37	**36.94**	20.77	37	36.50	25.66	**38**	36.88	4.39
pb500rnd15	500	1500	0.7%	5	[1-20]	1196*	1196	1186.94	59.36	1196	**1190.93**	161.67	1196	1185.62	10.68
pb500rnd16	500	1500	0.7%	5	[1-1]	88*	88	86.63	36.31	88	86.12	60.67	88	**87.00**	5.12
pb500rnd17	500	1500	2.2%	20	[1-20]	192*	192	**191.75**	18.38	192	188.31	57.00	192	191.38	5.55
pb500rnd18	500	1500	2.2%	20	[1-1]	13*	13	**13.00**	12.03	13	12.43	9.00	13	12.81	2.85
pb1000rnd100	1000	5000	2.60%	50	[1-20]	67	67	65.50	53.50	67	64.00	117.67	67	**67.00**	10.64
pb1000rnd200	1000	5000	2.59%	50	[1-1]	4	4	3.15	39.30	4	**3.81**	15.00	4	3.44	3.58
pb1000rnd300	1000	5000	0.60%	10	[1-20]	661*	649	639.50	221.20	661	640.50	700.67	661	**642.81**	32.92
pb1000rnd400	1000	5000	0.60%	10	[1-1]	48*	48	**46.83**	149.70	47	45.06	108.33	48	46.50	18.12
pb1000rnd500	1000	1000	2.60%	50	[1-20]	222*	222	**217.98**	64.80	222	219.62	85.67	222	217.88	23.11
pb1000rnd600	1000	1000	2.65%	50	[1-1]	15*	14	13.68	41.40	15	13.37	8.00	15	**14.06**	12.03
pb1000rnd700	1000	1000	0.58%	10	[1-20]	2260	2222	2214.10	119.70	2248	2 239.56	296.67	2253	**2242.06**	44.48
pb1000rnd800	1000	1000	0.60%	10	[1-1]	175*	172	170.81	82.60	173	170.50	94.00	174	**172.62**	21.21

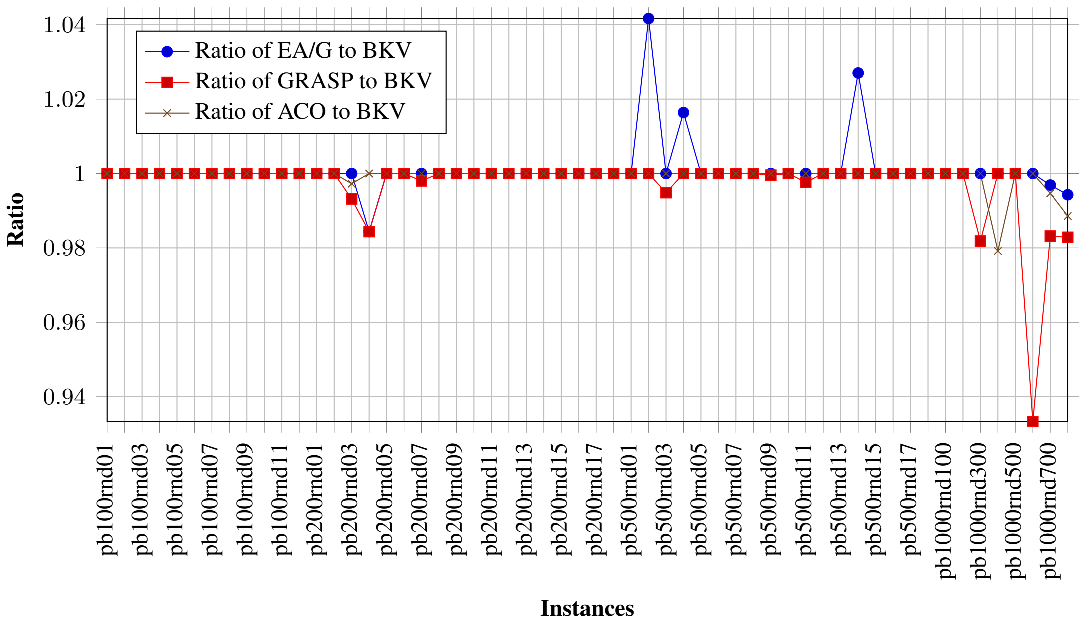


Figure 2.1: Comparison of EA/G, GRASP and ACO in terms of the ratios of the best solution found by each of them to the best known value (BKV)

Table 2.3: Comparison of EA/G with GRASP [1] on instances with 2000 variables

Instance	**Characteristics**					**0/1 solution**	**GRASP**	**EA/G**		
	Var	Cnst	Density	Max_One	Weight	BKV	Best	Best	Avrg	ATET
pb2000rnd0100	2000	10000	2.54%	100	[1-20]	40	Yes	40	40.00	31.66
pb2000rnd0200	2000	10000	2.55%	100	[1-1]	2	Yes	2	2.00	10.38
pb2000rnd0300	2000	10000	0.55%	20	[1-20]	478*	No	478	463.75	129.86
pb2000rnd0400	2000	10000	0.55%	20	[1-1]	32*	Yes	32	30.25	81.58
pb2000rnd0500	2000	2000	2.55%	100	[1-20]	140*	Yes	140	135.50	61.03
pb2000rnd0600	2000	2000	2.56%	100	[1-1]	9*	Yes	9	8.50	34.23
pb2000rnd0700	2000	2000	0.56%	20	[1-20]	1784*	No	**1794**	1765.94	160.14
pb2000rnd0800	2000	2000	0.56%	20	[1-1]	131*	No	**132**	130.38	73.84

Note that the system that is used to execute GRASP and ACO approaches [2] is Pentium III at 800 MHz which is different from the system used to execute the EA/G. Therefore, execution times cannot be compared exactly. However, a rough comparison can always be made. Even after compensating for processing speeds, we can safely say that EA/G is faster than ACO on majority of instances. GRASP seems to be a bit faster than EA/G, but it performed the worse among the three.

Table 2.4: Influence of parameter settings on solution quality

		pb500rnd14			**pb500rnd02**			**pb500rnd09**			**pb200rnd11**		
Parameter	Value	Best	Avg.	Time	Best	Avg.	Time	Best	Avg.	Time	Best	Avg.	Time
	20	37	36.56	2.89	24	23.55	2.41	2236	2229.25	9.21	545	544.50	1.15
N	**30**	**38**	**36.88**	**4.39**	**25**	**23.56**	**5.16**	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	40	37	36.94	5.53	25	23.50	5.03	2236	2230.69	14.71	545	544.75	2.18
	0.45	37	36.75	3.96	24	23.38	3.74	2236	2231.00	12.54	545	544.62	1.62
λ	0.50	**38**	**36.88**	**4.39**	**25**	**23.56**	**5.16**	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	0.55	37	36.88	4.15	24	23.31	3.65	2236	2229.06	11.66	545	544.88	1.65
	0.85	38	36.75	3.98	25	23.19	3.67	2236	2229.62	11.95	545	544.88	1.65
β	0.90	**38**	**36.88**	**4.39**	**25**	**23.56**	**5.16**	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	0.95	37	36.81	4.25	24	23.69	3.81	2236	2230.06	12.46	545	544.75	1.62
	3	-	-	-	-	-	-	2236	2230.19	11.99	545	544.88	1.62
Num_{Ex1-1}	4	-	-	-	-	-	-	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	5	-	-	-	-	-	-	2236	2231.88	12.24	545	544.75	1.66
	5	37	36.81	4.07	24	23.44	3.72	2236	2229.06	12.17	545	544.75	1.62
Num_{Ex1-2}	6	**38**	**36.88**	**4.39**	**25**	**23.56**	**5.16**	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	7	37	36.81	4.02	24	23.38	3.72	2236	2229.50	12.23	545	545.00	1.64
	0.65	38	36.81	4.23	25	23.44	3.66	2236	2230.94	12.22	545	545.00	1.62
f	0.70	**38**	**36.88**	**4.39**	**25**	**23.56**	**5.16**	**2236**	**2232.81**	**12.49**	**545**	**545.00**	**1.65**
	0.75	38	36.88	4.28	24	23.44	3.75	2236	2230.94	12.26	545	544.88	1.63

There are 8 more random instances with 2000 variables that are not used in [2] to test ACO. However, these instances are used in [1] to test GRASP. Table 2.3 presents the results of CPLEX 6.0 solver, GRASP and EA/G on these instances. Data for CPLEX 6.0 solver and GRASP is taken from [1]. As [1] reports the results of GRASP only graphically in terms of % of BKVs obtained through CPLEX 6.0 (Figure 5 of [1]), the column corresponding to GRASP reports "Yes" for a particular instance if GRASP is able to find the BKV reported in [1] for that instance, otherwise it reports "No". Out of these 8 instances, GRASP is able to find the BKVs for only 5 instances. EA/G, on the other hand, finds as good as or better than the BKVs reported in [1] on all instances. On 2 instances, EA/G even improved the BKVs reported in [1]. Execution time of GRASP is not reported in [1].

The superior performance of EA/G over ACO and GRASP approaches can be attributed to the combined use of global information about the search space in the form of a probability vector and the best solution found so far in generating an offspring. ACO uses only global information about the search space in the form of pheromone values, whereas GRASP only uses information about few best solutions to guide the search.

In order to investigate the influence of parameter settings on the solution quality, we have taken four instances comprising two unicost instances viz. `pb500rnd14` & `pb500rnd02` and two non-unicost instances viz. `pb500rnd09` & `pb200rnd11`. We have varied all the parameters one by one, keeping all other parameters unchanged. In doing so all other parameters were set to their values reported at the beginning of this section. The results are reported in Table 2.4. The 1-1 exchange is not applied on unicost instances and therefore corresponding columns for two unicost instances have '-' in the table. From this table, it is clear that for all parameters, the values of the parameters reported at the beginning of the computational results section give best results. The average solution quality for all four instances vary with parameter values. The best solution quality of two non-unicost instances remains the same irrespective of parameter values.

2.4 Conclusions

In this chapter, we have presented a hybrid evolutionary algorithm with guided mutation for the set packing problem. In comparison with two state-of-the-art approaches, viz. GRASP approach [1] and ACO approach [2] for the set packing problem, our approach has shown promising results. Our approach has outperformed these approaches in terms of both best as well as average solution quality. Our approach is faster than ACO approach on majority of instances. Though, GRASP is a bit faster than our EA/G approach, it performed the worst.

Chapter 3

Minimum Weight Dominating Set

3.1 Introduction

For any graph, a dominating set is a set of nodes such that each node of the graph either belongs to the dominating set or adjacent to a node in the dominating set. A minimum dominating set (MDS) is a dominating set with minimum cardinality. Use of dominating sets is widespread in wireless networks for clustering and forming a virtual backbone for routing and several efficient centralized and distributed algorithms have been proposed for this purpose in the literature, e.g., see [69, 70, 71, 72, 73]. However, all these approaches work under the assumption that all nodes are same in all respects. These approaches may not yield good results in heterogeneous environments where the capabilities of the nodes differ. For example, different nodes may have different residual energy, allotted bandwidth or transmission range. Under these circumstances, instead of minimizing the cardinality of the dominating set, it is more appropriate to minimize the weighted sum of nodes present in the dominating set. This problem of minimizing the weighted sum of nodes present in the dominating set is called minimum weight dominating set problem. Both minimum dominating set problem (MDS) and minimum weight dominating set problem (MWDS) are $\mathcal{NP}$-hard [57].

Many approaches have been proposed in the literature for MWDS problem. But most of them are for unit disk graphs (UDGs) which are used to model wireless networks. In UDG, nodes lie on a Euclidean plane and an edge exists between pair of nodes, iff Euclidean distance between them does not exceed a fixed threshold. A characteristic feature of UDG is that the number of independent neighbors of a node is polynomially bounded. To form a backbone for ad hoc wireless networks, Wang *et al.* [74] proposed new centralized and distributed approximation algorithms, where each node has some cost and a minimum weight dominating set is created. These approaches assume that node weights are smooth, i.e., ratio of weights of adjacent nodes does not exceed a fixed constant. Dai *et al.* [75] proposed $5 + \epsilon$ -approximation algorithm to form a minimum weight dominating set for unit disk graphs. This goal is achieved in two steps. In the first step, plane is partitioned into $k\mu \times k\mu$ blocks, where $\mu = \frac{\sqrt{2}}{2}$ and k is some large integer constant, and in the second step, each block is partitioned into k^2 squares. And finally results of both the steps are combined to get minimum weight dominating set. Another $4+\epsilon$-approximation scheme [76] is based on polynomial-time dynamic programming algorithm for min-weight chromatic disk cover, where weights of the nodes in the graph are not smooth. Last two approaches do not assume node weights to be smooth. Zhu *et al.* [77] proposed another polynomial time approximation scheme to form minimum weight dominating set for UDG, where the weights of the nodes in the graph are smooth.

However, use of MWDS is not limited to clustering and routing in wireless networks. In fact, MWDS has applications in several diverse domains. It finds application in gateway placement in a wireless mesh network. A polynomial time near-optimal algorithm is proposed in [78] to recursively compute minimum weight dominating set in each iteration so as to minimize the number of gateways and also satisfy the QoS (quality of service) requirements. During installation of wavelength division multiplexing optical-networks, MWDS is used to minimize the number of full wavelength converters required in order to cut down the installation cost and to subdue the technological limitations [79]. MWDS was also used in determining the nodes in an ad hoc network where intrusion detection software for squandering detection needs to be installed [80]. Shen *et al.* [81] used the concept of dominating sets in information retrieval for the task of multi-document summarization. In this approach, first a graph is constructed where nodes represent the sentences from various documents and an edge exists between two sentences if they are similar. Then a minimum dominating set is computed over this graph to get a summarization of all the documents. The application of MWDS is extended to the task of a query selection so as to harvest the data records from hidden web database efficiently [82]. For this purpose, each web database is modeled as an undirected attribute-value graph (AVG) and task of optimal query selection for the web database is shown equivalent to solving MWDS in corresponding AVG. In none of the applications cited in this paragraph, we can assume the underlying graph to be UDG. However, all the approximation schemes available in the literature work for UDG only, therefore, these schemes can not be used for aforementioned applications. In fact, underlying graphs in these applications vary depending on the situation, and as a result, only those approaches which can be applied on any general graph can be used. In the absence of approximation schemes for general graphs, only options left are heuristics and metaheuristics.

To the best of our knowledge, only two metaheuristic approaches have been proposed in the literature for MWDS. Jovanovic *et al.* [83] developed an ant colony optimization (ACO) algorithm which was inspired by the ant-colony optimization approach of Shyu *et al.* [84] for minimum weight vertex cover problem and tested it on general graphs. This ACO approach will be referred to as Raka-ACO subsequently. Potluri and Singh [85] proposed four greedy heuristics, a hybrid genetic algorithm and two versions of hybrid ant-colony optimization algorithm for MWDS. The solutions obtained through genetic algorithm and ACO algorithms are improved further though a minimization heuristic. The genetic algorithm uses steady-state population replacement method, selects parents using binary tournament selection and employs fitness-based crossover and simple bit flip mutation to produce offspring. Hereafter, this hybrid genetic algorithm will be referred to as HGA. Except for pheromone initialization, the two ACO versions are same in all other respect. Starting with an empty set S, these two ACO versions construct a dominating set by performing a random walk on the graph and adding the nodes visited to S until S becomes a dominating set. The next node to be visited is selected based on pheromone concentration on the nodes. The first version of ACO initializes all pheromone trails to same value without any bias, whereas the second version uses a pre-processing step to initialize the pheromone trails according to the quality and compositions of the solutions generated in the preprocessing step. The first version will be referred to as ACO-LS and second version as ACO-PP-LS subsequently. All the approaches presented in [85] have been tested on same general graphs instances as used in [83]. In addition, some UDG instances have also been used. HGA, ACO-LS and ACO-PP-LS approaches obtained much better results in comparison to Raka-ACO. As far as relative performance of HGA, ACO-LS and ACO-PP-LS approaches is concerned, HGA obtained better quality solutions with increase in average node degree and graph size, but at the expense of much larger execution times. There is not much difference in solution quality of ACO-LS and ACO-PP-LS, but ACO-PP-LS is faster than ACO-LS.

Similar to the approach of previous chapter, in this chapter also we present a hybrid approach comprising evolutionary algorithm with guided mutation (EA/G) and an improvement operator for MWDS. In addition, we present the modified version of a greedy heuristic proposed in [85] and use it inside our hybrid EA/G approach for initial solution generation and repairing an infeasible solution. We have compared our hybrid EA/G approach with the approaches presented in [83, 85] on the same benchmark instances as used in [83, 85]. In comparison to these approaches, our hybrid approach obtained better quality solutions in shorter time.

The remainder of this chapter is organized as follows: Section 3.2 formally defines the minimum weight dominating set problem and introduces the notational conventions used in this chapter. Section 3.3 describes the proposed modifications in one of the heuristics of [85]. Our hybrid EA/G approach is presented in Section 3.4. Section 3.5 reports the computational results and provides a comparative analysis of our hybrid EA/G approach vis-à-vis existing approaches for MWDS. Finally, Section 3.6 outlines some concluding remarks.

3.2 Problem formulation

In an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Two nodes u and v are called neighbor of each other or adjacent to each other, iff, there exists an edge between them, i.e., $(u, v) \in E$. A dominating set $D \subseteq V$ is a set of nodes such that each node $v \in V$ either belongs to D or is neighbor of at least one node in D. Any node belonging to D is called a dominating node or dominator. Any node not in D is called dominatee or non-dominating node. The minimum dominating set problem (MDS) seeks a dominating set on G whose cardinality is least among all the possible dominating sets on G. In minimum weight dominating set problem (MWDS), a weight is assigned to each node of V through a weight function $w : V \rightarrow \Re^+$ and the goal is to find a dominating set with minimum sum of node weights. More formally, MWDS seeks $\underset{D \in DS}{\arg\min} \sum_{v \in D} w(v)$, where DS is the set of all dominating sets of G. Both MDS and MWDS have been proven to be $\mathcal{NP}$-hard [57]. In fact, MWDS is a generalization of MDS as it reduces to MDS when $w(v) = 1 \; \forall v \in V$. Important notational conventions used in this chapter are given in Table 3.1. Additional notational conventions will be introduced as and when required.

Table 3.1: Notational convention

Notation	Definition
$w(v)$	Weight of node $v \in V$.
$ON(v) \subseteq V$	$\{u : u \in V$ and $(u, v) \in E\}$ is called open neighborhood of node $v \in V$
$CN(v) \subseteq V$	$ON(v) \cup \{v\}$ is called closed neighborhood of node $v \in V$.
$dc(v)$	$\lvert ON(v)\rvert$, i.e., Total number of neighboring nodes of node $v \in V$ or degree of v in G

3.3 Heuristic

In this section, we present an improved version of a heuristic (viz. heuristic 2) which has been presented in [85]. Actually, four different heuristics have been presented in [85]. Reason for choosing heuristic 2 is that among these four heuristics, it performed the best. In all these heuristics, initially, all nodes are assumed to have color WHITE. Starting with an empty set D, these heuristics build a dominating set by iteratively adding a node to D till it satisfies the property of a dominating set. A node that is added to D is re-colored BLACK and all its neighbors are re-colored GREY. Obviously, with this coloring scheme, a heuristic stops when no WHITE node exists. The manner in which the nodes are selected for inclusion into the partially constructed dominating set leads to different heuristics. We have made two modifications in heuristic 2 of [85]. Suppose D is the partially constructed dominating set, than $S = V - D$ is the set from which the next node is added to the dominating set D. Also suppose $c : V \rightarrow \{0, 1\}$ is a function such that $c(v) = 1$, iff color of node v is WHITE at present, 0 otherwise. Our first modification utilizes closed neighborhood instead of open neighborhood while choosing the next node to be added to the dominating set. In heuristic 2 of [85], to determine the next node to be added to the dominating set, sum of weights of WHITE nodes belonging to the open neighborhood of each node $u \in S$ (Equation (3.1)) is first calculated and then next node is selected using Equation (3.2), whereas in our heuristic, sum of weights of WHITE nodes belonging to the closed neighborhood of each node $u \in S$ is first calculated (Equation (3.3)) and then next node to be added is determined using Equation (3.4). The motivation for using closed neighborhood instead of open neighborhood lies in the fact that when a node to be added is itself WHITE, its weight should be added to the sum of weights of its neighboring WHITE nodes because with respect to this node also property of dominating set will be satisfied once it is included in the partially constructed dominating set along with satisfying the property of dominating set with respect to the neighboring WHITE nodes.

$$W(u) \leftarrow \sum_{v \in ON(u)} w(v) \times c(v) \tag{3.1}$$

$$v \leftarrow \arg\max_{u \in S} \frac{W(u)}{w(u)} \tag{3.2}$$

$$\overline{W}(u) \leftarrow \sum_{v \in CN(u)} w(v) \times c(v) \tag{3.3}$$

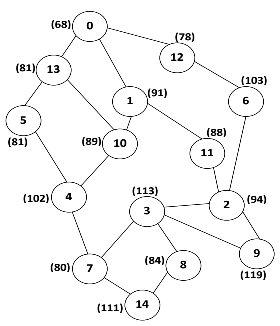
$$v \leftarrow \arg\max_{u \in S} \frac{\overline{W}(u)}{w(u)} \tag{3.4}$$

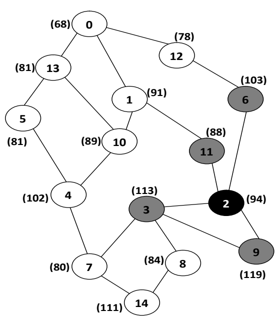
Figures 3.1(a) to 3.1(g) illustrate our first modification with the help of an example. Figure 3.1(a) shows an undirected connected weighted graph with 15 nodes where the number inside a circle indicates the ID of the node represented by that circle, and the number outside the circle indicates the weight of the corresponding node. All nodes are colored WHITE in this figure to show the initial state.

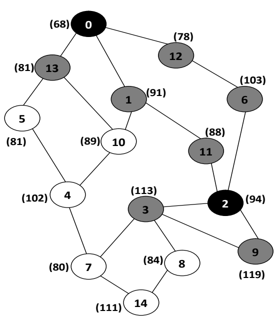
Both the heuristics select node 2 during first iteration for inclusion into the partially constructed dominating set. Node 2 is colored BLACK and all the neighbors of node 2, viz. nodes 9, 3, 11 and 6 are colored GREY. This situation is shown in Figure 3.1(b). Both the heuristics continue to select identical nodes for three more iterations leading to the further inclusion of node 0, 7 and 13 in that order to the partially constructed dominating set. Figures 3.1(c) to 3.1(e) depict the partially constructed dominating set at the end of each of these three iterations. The partially constructed dominating set depicted in Figure 3.1(e) has weight 323. Now, the only WHITE node is node 8. Our heuristic selects node 8 as it has the maximum ratio of 1 as per Equation (3.4) in the next iteration and produces a dominating set with nodes 2, 0, 7, 13 and 8, and total weight 407, which is shown in Figure 3.1(f).

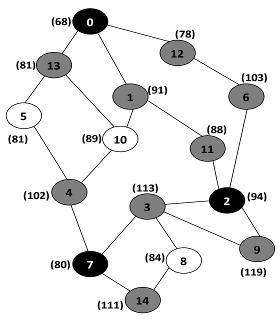
Heuristic 2 of [85], on the other hand, returns GREY node 14 to cover WHITE node 8, and produces a dominating set with nodes 2, 0, 7, 13 and 14, and total weight 434, which is shown in Figure 3.1(g). As the open neighborhood is used here, so node 8 ironically has the minimum ratio of 0 and node 14 has the maximum ratio of $\frac{84}{111}$ as per Equation (3.2), and, that is why the node 14 is selected. From the dominating sets depicted in Figures 3.1(f) and 3.1(g), we can say that our heuristic can produce better results than heuristic of [85]. It is to be noted that even if we swap the weights of node 8 and 14, our heuristic will select the appropriate node according to the changed scenario. In the changed scenario node 8 will have the weight of 111 and node 14 will have the weight of 84. Under this situation node 14 has the maximum ratio of $\frac{111}{84}$ and node 8 again has the ratio 1 as per Equation (3.4), so our heuristic will correctly select node 14. In this case, however, heuristic 2 of [85] will also select node 14.

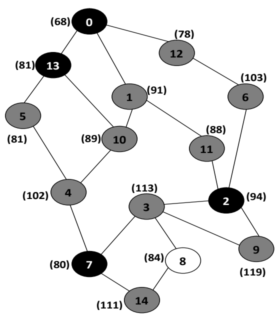
Our second modification is a tie-breaking rule. Actually, when there are more than one node satisfying Equation (3.4) then the next node to be added is selected using this tie-breaking rule. Suppose S' is the set of nodes satisfying Equation (3.4), then our heuristic computes the number of WHITE nodes belonging to the closed neighborhood of each node $u \in S'$ (Equation (3.5)), and then the next node to be added is determined using Equation (3.6). In case,

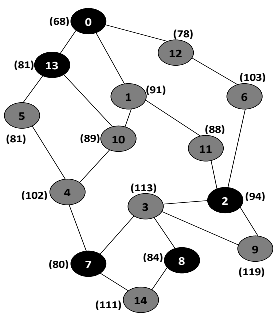

(a) Input graph. Initially, all nodes are colored WHITE and $weight = 0$


(b) Node 2 is selected by both heuristics and $weight = 94$


(c) Node 0 is selected by both heuristics and $weight = 162$


(d) Node 7 is selected by both heuristics and $weight = 242$


(e) Node 13 is selected by both heuristics and $weight = 323$


(f) Node 8 is selected with our heuristic and $weight = 407$

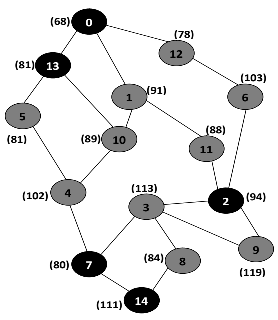

(g) Node 14 is selected with heuristic of [85] and $weight = 434$

Figure 3.1: Example of H_DT and M_DT

there are more than one node satisfying even Equation (3.6), then the next node to be added is selected arbitrarily from among these nodes. The motivation behind this tie-breaking rule lies in experimental observation that using this rule helps in getting better results on more instances.

$$wd(u) \leftarrow \sum_{v \in CN(u)} c(v) \tag{3.5}$$

$$v \leftarrow \arg\max_{u \in S'} \frac{wd(u)}{w(u)} \tag{3.6}$$

Hereafter, original heuristic 2 of [85] will be referred to as Heu_A and our improved version as Heu_I.

3.4 Hybrid EA/G approach for MWDS

We have developed a hybrid approach to MWDS combining EA/G algorithm with an improvement operator. The solutions generated through EA/G algorithm are passed through the improvement operator in a bid to improve them further. Hereafter, we will refer to our hybrid EA/G approach with improvement operator as EA/G-IR (EA/G with improvement operator). Before starting our EA/G-IR approach, we precompute the set of neighbor nodes for each node $v \in V$. The salient features of our EA/G-IR approach for MWDS are described in subsequent subsections.

3.4.1 Solution encoding

Subset encoding has been used to represent a solution, i.e., each solution is represented as a subset of nodes that it contains.

3.4.2 Initial solution

The first member of initial population is always constructed through our heuristic (see Section 3.3). Remaining members of the initial population are also constructed in the same manner as in our heuristic except for the fact that during each iteration Equation (3.4) (and Equation (3.6) if needed) is used for selecting the next node for inclusion in the partially constructed dominating set with probability π_i, otherwise the next node is selected randomly. Each member of the initial population, irrespective of how it got constructed, is passed through an improvement operator in a bid to improve it further. Pseudo-code of construction of an initial solution except the first solution is presented in Algorithm 3.1.

Algorithm 3.1: The pseudo-code for generation of each initial solution except first solution

```
    ▷ Initially all nodes in $V$ are colored WHITE
 1: $I \leftarrow V$;
 2: $W_n \leftarrow V$;
 3: $WD \leftarrow \Phi$;
 4: while $W_n \neq \emptyset$ do
 5:     Generate a random number $r$ such that $0 \leq r \leq 1$;
 6:     if $r < \pi_i$ then
 7:         $v \leftarrow \arg\max_{u \in I} \frac{\overline{W}(u)}{w(u)}$;        ▷ Equation (3.6)
 8:     else
 9:         $v \leftarrow random(W_n)$;
10:     end if
11:     if $v$ is WHITE then
12:         $W_n \leftarrow W_n \backslash \{v\}$;
13:     end if
14:     $wd(v) \leftarrow \{u : u \in ON(v) \cap W_n\}$;
15:     $W_n \leftarrow W_n \backslash wd(v)$;
16:     $WD \leftarrow WD \cup \{v\}$;
17:     Make $v$ BLACK
18:     Make all vertices $\in wd(v)$ GREY;
19:     $I \leftarrow I \backslash \{v\}$;
20: end while
21: return $WD$;
```

3.4.3 Initialization and update of the probability vector

Like the previous chapter, we have also used an univariate marginal distribution (UMD) model to maintain the distribution of promising solutions in the search space. In this model, a probability vector $p = \{p_1, p_2, \ldots, p_{|V|}\} \in [0,1]^{|V|}$ is used to characterize the distribution of promising solutions in the search space, where $|V|$ is the cardinality of V, i.e., number of nodes in the graph. Each $p_i \in p$ gives the probability of the node i of V to be present in a dominating set. Our *guided mutation operator* (described in Section 3.4.4) uses this probability vector to generate offspring at each generation g. This probability vector is initialised using the N_c number of initial solutions. The pseudo-code for initializing a probability vector p is given in Algorithm 3.2.

At each generation g, a parent set *parent(g)* is formed by selecting best $\frac{N_c}{4}$ solutions from current population *pop(g)*. Once *parent(g)* is formed, it is used for updating the probability vector p. The pseudo-code for updating the probability vector is given in Algorithm 3.3, where

Algorithm 3.2: The pseudo-code for initializing a probability vector p

```
1: Compute $n_v \leftarrow$ number of initial solutions containing node $v$, $\forall v \in V$;
2: Compute $p_v \leftarrow \frac{n_v}{N_c}$, $\forall v \in V$;
```

$\lambda \in (0, 1]$ is the learning rate and it governs the contribution of solutions in *parent(g)* to the updated probability vector p, i.e., higher the value of λ, more is the contribution of solutions in *parent(g)*.

Algorithm 3.3: The pseudo-code for updating a probability vector p in generation g

```
1: Compute $n_v \leftarrow$ number of solutions in parent(g) containing node $v$, $\forall v \in V$;
2: Compute $p_v \leftarrow (1 - \lambda)p_v + \lambda \frac{n_v}{\frac{N_c}{4}}$, $\forall v \in V$;
```

3.4.4 Guided mutation (GM) operator

The *GM* operator is applied on the best solution b_s in the current population to generate new solutions. The pseudo-code of generating a new solution through the *GM* operator is given in Algorithm 3.4, where β is an adjustable parameter in $[0, 1]$ and D is a new solution constructed through the *GM* operator whose nodes are either sampled from probability vector p or directly copied from best solution b_s in *parent(g)*. Parameter β controls the relative contribution of probability vector p and best solution b_s in the generation of a new solution. The larger the value of β is, more nodes of D are sampled from probability vector p, on the other side smaller the value of β is, more nodes are directly copied from b_s. There is no guarantee of feasibility of solution D generated through the *GM* operator, i.e., D may not be a dominating set. Therefore, every solution D generated through the *GM* operator is checked for feasibility and if D is found to be infeasible, then it is passed through the repair operator which transforms D into a feasible solution. This repair operator is described in the next section.

3.4.5 Repair operator

Repair operator is applied only on an infeasible solution. Our repair operator is quite different from the repair heuristic of [85] and computationally much more efficient. In the repair heuristic of [85], initially, S is the set of $V \setminus D$ nodes, where V is the set of nodes and D is the infeasible solution to be repaired. The repair heuristic then iteratively selects a node from S as per

Algorithm 3.4: The pseudo-code of generating a solution through the *GM* operator

```
1: $D \leftarrow \Phi$;
2: for node $v \in V$ do
3:     Generate a random number $r_1$ such that $0 \leq r_1 \leq 1$;
4:     if $r_1 < \beta$ then
5:         Generate a random number $r_2$ such that $0 \leq r_2 \leq 1$;
6:         if $r_2 < p_v$ then
7:             $D \leftarrow D \cup \{v\}$;
8:         end if
9:     else
10:        if $v \in b_s$ then
11:            $D \leftarrow D \cup \{v\}$;
12:        end if
13:    end if
14: end for
15: return $D$;
```

Algorithm 3.5: The pseudo-code of repair operator

```
    ▷ $D$ is an infeasible solution on which repair operator is applied
1: $W_n \leftarrow \{v : v \in V$ and $v$ is WHITE$\}$;
2: while $W_n \neq \emptyset$ do
3:     $v_r \leftarrow random(W_n)$;
4:     $S \leftarrow CN(v_r)$;
5:     $v \leftarrow \arg\max_{u \in S} \frac{\overline{W(u)}}{w(u)}$;                ▷ Equation (3.6)
6:     $wd(v) \leftarrow \{u : u \in CN(v) \cap W_n\}$;
7:     make $v$ BLACK;
8:     $D \leftarrow D \cup \{v\}$;
9:     make all vertices $\in wd(v) \backslash \{v\}$ GREY;
10:    $W_n \leftarrow W_n \backslash wd(v)$;
11: end while
12: return $D$;
```

Equation (3.2) and add it to D after deleting it from S. This process is repeated till D becomes a dominating set. Actually, set S contains those GREY nodes also which have no WHITE neighboring node. Such GREY nodes can never be selected for inclusion in D, but in every iteration of repair heuristic used in [85], corresponding ratios needed for Equation (3.2) are calculated even for these GREY nodes. From empirical observations, it is found that there are a large number of such GREY nodes in S which unnecessarily increases the computational burden of the repair heuristic used in [85]. To overcome this shortcoming, in our repair operator, we begin by computing the set of WHITE nodes W_n, i.e., W_n contain those nodes which are not the neighbor of any node in the infeasible solution D. Then in every iteration, we initialize set S to the closed neighborhood of a randomly chosen node in W_n. With this initialization of S in every iteration, the cardinality of set S is always much smaller, specially for graphs with large number of nodes. In every iteration, after initializing S, our repair operator selects a node from set S using Equation (3.4) (and Equation (3.6) if needed) and include it into the infeasible solution D. In addition, after each iteration in our repair operator, we delete all those nodes from W_n which are no longer WHITE. This further helps in speeding-up the repair operator. These iterations continue till no WHITE node remains, i.e., till D becomes feasible. In the repair heuristic of [85], with probability p_h (which was set to 0.7), the infeasible solution is repaired as mentioned already, otherwise it is repaired randomly, i.e., some infeasible solutions are repaired by selecting the node from S in a totally random manner. This is done to maintain the diversity of the population as the other approach was purely greedy. Our repair operator is a proper mix of randomness and greediness and as such all solutions are repaired as described above and no infeasible solutions need to be repaired in a purely random manner just for the sake of diversity. The pseudo-code of repair operator is presented in Algorithm 3.5.

3.4.6 Improvement operator

All feasible solutions are passed through improvement operator in a bid to further improve the solution. Improvement operator, removes redundant nodes from a dominating set D. A node $v \in D$ is redundant if $CN(v) \subseteq (\cup_{u \in D \setminus \{v\}} CN(u))$, i.e., all nodes other than v in $CN(v)$ are either in D or the neighbor of a node in $D \setminus \{v\}$ and v is the neighbor of a node in D. If node v is redundant then it can be removed from D without affecting the dominating set property of D. Our improvement operator begins by computing the set of redundant nodes and then an iterative process takes over, where during each iteration, the redundant node having the highest ratio of its weight to its degree is deleted from the dominating set and set of redundant nodes is recomputed. Iterative process stops when the set of redundant nodes becomes empty. Clearly, our improvement operator follows a greedy strategy.

The pseudo-code of improvement operator is given in Algorithm 3.6 where D is the input dominating set and R_s is the set of redundant nodes in D. This is different from *Minimization heuristic* of [85] where during each iteration the redundant node to be deleted is either chosen based on highest ratio of the weight of the node to its degree or chosen randomly from the set of redundant nodes. In every iteration, first strategy is used with probability p_r (which

Algorithm 3.6: The pseudo-code of improvement operator

```
1: $R_s \leftarrow \{v : v \in D$ and $CN(v) \subseteq (\cup_{u \in D \setminus \{v\}} CN(u))$;
2: while ($R_s \neq \emptyset$) do
3:     $v \leftarrow \arg\max_{u \in R_s} \frac{w(u)}{dc(u)}$;
4:     $D \leftarrow D \setminus \{v\}$
5:     $R_s \leftarrow \{v : v \in D$ and $CN(v) \subseteq (\cup_{u \in D \setminus \{v\}} CN(u))$;
6: end while
7: return $D$;
```

was set to 0.6), otherwise the random strategy is used. Our improvement operator follows the greedy strategy always. We have experimented with random strategy also, but greedy approach produced better solutions in comparison to a combination of greedy and random strategies of [85].

3.4.7 Other features

Zhang *et al.* [55] put the best $\frac{N_c}{2}$ solutions of *pop(g)* into the *parent(g)* and create $\frac{N_c}{2}$ new solutions in every generation. The population for the next generation is formed using $\frac{N_c}{2}$ solutions in *parent(g)* and $\frac{N_c}{2}$ newly created solutions. On the other hand, in our approach *parent(g)* is formed using best $\frac{N_c}{4}$ solutions of *pop(g)* and $\frac{N_c}{4}$ new solutions are produced in every generation. The best $\frac{3N_c}{4}$ solutions of *pop(g)* along with $\frac{N_c}{4}$ newly created solutions constitute *pop(g+1)* in our approach. We have also tried the same strategy as [55], but our strategy gave better results.

If in a generation, we found all solutions to be identical then we re-initialize the population in the same manner as described in Section 3.4.2.

The pseudo-code of our EA/G-IR approach for MWDS problem is given in Algorithm 3.7.

3.5 Computational results

In this section, we present the computational results of our EA/G-IR approach for MWDS and compare them with the state-of-the-art approaches. In the literature, three sets of instances were used to evaluate the performance of different algorithms, viz. Type I, Type II and unit disk graphs (UDG). Type I and Type II instances were generated by Jovanovic *et al.* [83]. Type I and Type II instances consist of undirected connected weighted graphs. For Type I instances,

Algorithm 3.7: EA/G-IR Approach for MWDS

```
1: At generation $g \leftarrow 0$, an initial population pop(g) consisting of $N_c$ solutions ($N_c$ should be divisible by 4) are generated in a manner described in Section 3.4.2;
2: Initialize the probability vector $p$ for all nodes using Algorithm 3.2;
3: Select best $\frac{N_c}{4}$ solutions from pop(g) to form a parent set parent(g), and then update the probability vector $p$ using Algorithm 3.3;
4: Apply the GM operator $\frac{N_c}{4}$ times on the best solution $b_s$ in parent(g) in order to generate $\frac{N_c}{4}$ new solutions. A repair operator is applied to each generated solution, if necessary, and then an improvement operator is applied to each generated solution to improve the solution fitness. Add all $\frac{N_c}{4}$ newly generated solutions along with best $\frac{3N_c}{4}$ parent(g) solutions to form pop(g+1). If the stopping condition is met, return the dominating set with the minimum weight found so far ;
5: $g \leftarrow g + 1$;
6: If all solutions are identical, then reinitialize pop(g), and go to step 2;
7: Go to step 3 ;
```

the weights of the nodes are randomly distributed in the closed interval [20, 70]. For Type II instances, weights are randomly distributed in the closed interval [1, $dc(v)^2$], where $dc(v)^2$ is square of the degree of the node v. For both Type I and Type II instances, the number of nodes varies from 50 to 1000 and the number of edges varies from 50 to 20000. 10 instances were generated for each combination of the number of nodes ($|V|$) and the number of edges ($|E|$). The number of edges is also varied keeping the number of nodes same so as to observe the variation in the solution with the number of edges. Total 530 instances (53 $\times$ 10) were generated for each of Type I and Type II datasets leading to a grand total of 1060 instances in these two datasets.

UDG instances were generated by Potluri and Singh [85] using the topology generator of [86]. The nodes are distributed randomly in an area of 1000 $\times$ 1000 units. The number of nodes in these instances belong to $\{50, 100, 250, 500, 750, 1000\}$. Transmission range of all nodes is fixed to either 150 or 200 units. Similar to Type I and Type II instances, 10 instances were generated for each combination of number of nodes and the transmission range. This leads to a total of 120 UDG instances.

Our EA/G-IR approach for MWDS has been implemented in C and executed on an Intel core i5-2400 processor based system with 4 GB RAM running under Fedora 16 at 3.10 GHz. `gcc 4.6.3-2` compiler with `O3` flag has been used to compile the C program for our approach. In all computational experiments with our approach, we have used $N_c = 100$, $\beta = 0.70$, $\lambda = 0.60$ and $\pi_i = 0.55$. 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. We have allowed our approach to execute for 800 iterations on each instance. In each iteration $\frac{N_c}{4}$, i.e., 25 new solutions are generated. So total 800 $\times$ 25 = 20000 solutions are generated for each instance which is same as generated by metaheuristic approaches proposed in [85]. ACO of [83] generates 100000 solutions. To get an idea about the role of the improvement operator in finding high quality solutions, we have also implemented another version of our approach where no improvement operator is used. This version will be referred to as EA/G subsequently.

We have compared EA/G-IR and EA/G with four different approaches, viz. HGA, ACO-LS & ACO-PP-LS approaches proposed in [85] and Raka-ACO approach proposed in [83].

Like [83] and [85], we have analysed our results according to the instance groups. By an instance group, we mean the 10 instances with the same number of nodes and edges. An instance group is characterized by an ordered pair ($|V|$, $|E|$). For this analysis, EA/G-IR is executed only once on each instance like the metaheuristic approaches of [83] and [85]. Tables 3.2 to 3.5 report the results of Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G on Type I and Type II instances, whereas Table 3.6 does the same for UDG instances except for the fact an instance group here corresponds to 10 instances with the same number of nodes and transmission range. Data for Raka-ACO is taken from [83], where only Type I and Type II instances were used, and therefore, results of Raka-ACO on UDG instances are not reported. Moreover, standard deviations of solution values and average execution times were also not reported in [83] for Raka-ACO and that is why these two quantities for Raka-ACO are not reported. As the C programs for HGA, ACO-LS and ACO-PP-LS were available, so we have re-executed these programs on our system, so that execution time of these approaches can be compared directly with our approaches. This re-execution only changed the execution times from those reported in [85]. For each combination of number of nodes ($|V|$) and number of edges ($|E|$) (transmission range in case of UDG instances), these tables (Tables 3.2 to 3.6) report the average solution quality (Mean) and standard deviation of solution values (SD) over 10 instances. All approaches have high standard deviation as these approaches were executed once on each of the 10 instance and the value of the solution varies from one instance to the other. For the two heuristics Heu_A and Heu_I, we have only reported the average solution quality. In all these tables, a result in bold for EA/G-IR indicate that it is better than all the existing approaches in the literature. '*' indicates that EA/G-IR is worse than the best among HGA, ACO-LS and ACO-PP-LS. A result in bold italic indicates that it is the best result among Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS and ACO-PP-LS. '†' indicates that result of EA/G is equal to the best result among HGA, ACO-LS and ACO-PP-LS and '‡' indicates that the result of EA/G is better than the best result among HGA, ACO-LS and ACO-PP-LS. The results of Heu_I which are worse than Heu_A are marked with "- -" and which are better than Heu_A are marked with "++". Average time taken by different approaches is shown in Tables 3.7 to 3.11. For better analysis of results, instances are divided into two classes, viz. small & medium size and large size. Instances with 50 to 250 nodes are classified as small & medium size instances and instances with 300 to 1000 nodes are classified as large size instances.

After comprehensive analysis of results returned by various approaches, following observations can be made:

1. From Tables 3.2 to 3.5, it can be observed that EA/G-IR outperformed Raka-ACO in terms of solution quality on all Type I and Type II instance groups by a huge margin. As the graph size increases the differences in solution quality also increases. As expected, EA/G-IR obtained much better solutions in comparison to Heu_A and Heu_I also on all instance groups including groups of UDG instances.

2. For small & medium size Type I instances, EA/G-IR performed worse than HGA, ACO-LS and ACO-PP-LS on two instance groups ((50, 50) and (50, 750)), worse than best among HGA, ACO-LS and ACO-PP-LS for three instance groups ((50, 100), (100, 100) and (150, 150)), but equal to ACO-PP-LS for instance group (50, 100), better than HGA and ACO-PP-LS for instance group (100, 100) and better than HGA for instance group (150, 150). Out of 32 instance groups, average solution quality of EA/G-IR is worse than best among HGA, ACO-LS and ACO-PP-LS on 5 instances, same as best among HGA, ACO-LS and ACO-PP-LS on 4 instances and better than best among HGA, ACO-LS and ACO-PP-LS on remaining 23 instance groups.

3. For small and medium size Type II instances, EA/G-IR performed worse than the best among HGA, ACO-LS and ACO-PP-LS for instance group (150, 1000), but better than ACO-LS and ACO-PP-LS. Out of 32 instance groups, EA/G-IR is worse than best among HGA, ACO-LS and ACO-PP-LS on 1 group, equal to best among HGA, ACO-LS and ACO-PP-LS on 11 groups and better than best among HGA, ACO-LS and ACO-PP-LS on 20 groups.

4. EA/G-IR outperformed HGA, ACO-LS and ACO-PP-LS on all large size Type I and Type II instance groups except for two large Type I instances group (300, 300) and (300,

Table 3.2: Results of Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G for small & medium size Type I instances.

|V|	|E|	**Heu_A**	**Heu_I**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**		**EA/G-IR**		**EA/G**	
					Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
50	50	578.30	578.30	539.80	***531.30***	26.86	***531.30***	26.86	532.60	26.26	532.90*	26.65	531.80	26.69
50	100	410.50	410.50	391.90	***371.20***	22.63	***371.20***	22.63	371.50	22.60	371.50*	22.57	371.30	22.59
50	250	197.20	197.20	195.30	***175.70***	8.38	176	8.41	***175.70***	8.38	175.70	8.387	175.70†	8.38
50	500	105.50	105.50	112.80	***94.90***	6.30	***94.90***	6.30	95.20	6.58	94.90	6.30	94.90†	6.30
50	750	72.90	72.90	69.00	***63.10***	6.15	***63.10***	6.15	63.20	6.12	63.30*	6.18	63.30	6.18
50	1000	48.10	48.10	44.70	***41.50***	1.78	***41.50***	1.78	***41.50***	1.78	41.50	1.78	41.50†	1.78
100	100	1168.00	1168.00	1087.20	1081.00	52.40	1066.90	47.85	***1065.40***	47.85	1065.50*	44.35	1065.40†	45.10
100	250	676.70	676.70	698.70	***626.20***	36.17	627.20	30.70	627.40	30.84	**620.00**	33.95	620.80	34.41
100	500	397.30	397.30	442.80	***358.30***	19.98	362.50	19.23	363.20	18.47	**355.90**	16.22	357.00‡	17.33
100	750	294.90	294.90	313.70	***261.20***	13.85	263.50	15.12	265.00	12.80	**256.70**	8.91	257.90‡	11.54
100	1000	235.10	235.10	247.80	***205.60***	8.10	209.20	8.61	208.80	9.24	**203.60**	7.07	204.80‡	6.94
100	2000	122.30	122.30	125.90	108.20	3.12	***108.10***	3.07	108.40	3.47	108.10	2.92	107.90‡	3.21
150	150	1727.60	1724.40^{++}	1630.10	1607.00	64.49	***1582.80***	57.90	1585.20	59.21	1587.40*	59.18	1594.60	58.49
150	250	1340.50	1340.50	1317.70	1238.60	48.56	***1237.20***	45.48	1238.30	44.26	**1224.50**	46.14	1240.30	54.61
150	500	830.40	830.40	899.90	***763.00***	43.02	767.70	45.69	768.60	42.79	**755.30**	41.34	762.30‡	42.91
150	750	611.40	611.40	674.40	***558.50***	27.83	565.00	31.16	562.80	33.39	**550.80**	28.12	552.70‡	27.04
150	1000	488.70	488.70	540.70	***438.70***	23.02	446.80	22.46	448.30	211.00	**435.20**	17.58	439.30	18.72
150	2000	271.50	271.50	293.10	***245.70***	13.94	259.40	9.34	255.60	9.05	**241.50**	12.20	242.10‡	12.66
150	3000	191.10	191.10	204.70	***169.20***	10.39	173.40	10.86	175.20	7.89	**168.10**	9.89	168.00‡	9.80
200	250	2095.10	2095.10	2039.20	1962.10	52.30	1934.30	52.62	***1927.00***	54.60	**1924.10**	50.76	1951.70	65.32
200	500	1380.80	1380.80	1389.40	1266.30	40.07	***1259.70***	45.80	1260.80	50.59	**1251.30**	48.03	1269.50	58.69
200	750	1020.50	1020.50	1096.20	939.80	49.22	***938.70***	44.79	940.10	44.97	**927.30**	48.30	935.30‡	44.04
200	1000	809.60	809.60	869.90	***747.80***	16.41	751.20	16.40	753.70	19.30	**731.10**	20.89	736.50‡	16.87
200	2000	467.50	467.50	524.10	***432.90***	17.46	440.20	16.52	444.70	16.61	**417.00**	12.66	422.40‡	14.35
200	3000	334.80	334.80	385.70	***308.50***	12.28	309.90	11.86	315.20	17.44	**294.70**	11.93	297.80‡	14.41
250	250	2906.00	2906.00	NA	2703.40	80.70	***2655.40***	72.91	***2655.40***	74.63	**2653.70**	60.52	2695.50	66.78
250	500	2040.90	2040.90	NA	1878.80	77.85	1850.30	55.49	***1847.90***	68.49	**1835.30**	60.11	1909.30	74.25
250	750	1533.80	1537.00^{--}	NA	1421.10	40.16	***1405.20***	36.56	1405.50	34.09	**1399.20**	41.69	1436.90	58.39
250	1000	1238.80	1238.80	NA	1143.40	43.43	1127.10	43.03	***1122.90***	26.48	**1114.90**	30.70	1137.30	43.68
250	2000	715.80	715.80	NA	656.60	30.73	***672.80***	29.18	676.40	30.88	**637.50**	35.52	657.40‡	33.90
250	3000	500.40	500.40	NA	***469.30***	22.09	474.10	20.02	476.30	21.77	**456.30**	19.49	459.50‡	19.16
250	5000	328.40	328.40	NA	***300.50***	9.65	310.40	12.18	308.70	11.56	**291.80**	5.16	295.50‡	5.42

NA indicates that result is not available for that instance.

500), where ACO-PP-LS obtained better solution quality. For both types of instances, difference in solution quality grows with the number of nodes and the degree of the nodes.

It can also be observed that improvement in solution quality by EA/G-IR for small & medium size instances is less in comparison to large size instances. Actually, for small size instances, the results obtained by all the approaches are either optimal or are very close to the optimal and that is why EA/G-IR either obtained the same solution or not able to improve them much. On the other hand, for large size instances, existing approaches could not scale well and their solution quality deteriorates, and, that is why EA/G-IR obtained noticeable improvement in solution quality on almost all the instances.

Table 3.3: Results of Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G for large size Type I instances.

|V|	|E|	**Heu_A**	**Heu_I**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**		**EA/G-IR**		**EA/G**	
					Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
300	300	3481.50	3481.50	NA	3255.20	74.13	***3198.50***	63.82	3205.90	70.39	3213.70*	75.81	3288.30	80.56
300	500	2697.90	2697.90	NA	2509.80	69.21	2479.20	80.75	***2473.3***	84.44	2474.80*	76.95	2574.90	85.18
300	750	2104.30	2099.40^{++}	NA	1933.90	81.23	***1903.30***	55.08	1913.90	64.69	**1896.30**	55.74	1977.60	78.11
300	1000	1688.00	1686.20^{++}	NA	1560.10	35.27	***1552.50***	32.51	1555.80	36.19	**1531.00**	28.43	1604.30	52.85
300	2000	965.60	965.60	NA	***909.60***	22.85	916.80	25.61	916.50	23.08	**880.10**	21.91	904.40‡	26.61
300	3000	701.00	701.00	NA	***654.90***	24.44	667.80	30.00	670.70	28.00	**638.20**	22.15	645.60‡	26.04
300	5000	459.80	459.80	NA	***428.30***	15.07	437.40	16.59	435.90	16.22	**415.70**	10.32	419.60‡	7.44
500	500	5820.50	5820.50	5476.30	5498.30	113.45	5398.30	100.57	***5387.70***	99.53	**5380.10**	89.37	5630.00	153.32
500	1000	4039.80	4038.10^{++}	4069.80	3798.60	92.08	3714.80	103.77	***3698.30***	85.61	**3695.20**	107.47	3990.20	144.51
500	2000	2484.90	2484.90	2627.50	2338.20	77.75	2277.60	60.20	***2275.90***	65.12	**2264.30**	84.53	2427.30	94.45
500	5000	1177.10	1177.10	1398.50	1122.70	30.96	1115.30	35.79	***1110.20***	41.94	**1083.50**	33.27	1128.00	30.84
500	10000	670.50	670.50	825.70	***641.10***	22.18	652.80	11.81	650.90	119.00	**606.80**	11.57	625.70‡	8.87
800	1000	8472.10	8472.10	8098.90	***8017.70***	141.01	8117.60	162.03	8068.00	178.60	**7792.20**	108.34	8426.60	134.45
800	2000	5641.40	5639.70^{++}	5739.90	***5318.70***	130.02	5389.90	151.14	5389.60	144.43	**5160.70**	76.92	5641.40	154.05
800	5000	2739.90	2739.90	3116.50	2633.40	69.07	26160	66.49	***2607.90***	62.02	**2561.90**	37.51	2702.40	53.32
800	10000	1590.60	1590.60	1923.00	1547.70	45.66	***1525.70***	32.40	1535.30	31.00	**1497.00**	33.41	1559.60	28.70
1000	1000	11666.30	11666.30	***10924.40***	11095.20	147.38	11035.50	174.92	11022.90	129.43	**10771.70**	122.15	11599.10	109.77
1000	5000	4168.50	4168.50	4662.70	***3996.60***	73.73	4012.00	81.91	4029.80	85.90	**3876.30**	64.70	4163.50	101.85
1000	10000	2379.80	2379.80	2890.30	2334.70	64.51	2314.90	64.03	***2306.60***	56.03	**2265.10**	51.68	2365.80	58.32
1000	15000	1704.00	1704.00	2164.30	1687.50	28.29	***1656.30***	44.23	1657.40	40.05	**1629.40**	30.04	1682.60	41.04
1000	20000	1351.00	1351.30	1734.30	1337.20	30.97	***1312.80***	22.52	1315.80	24.10	**1299.90**	19.32	1333.70	22.36

NA indicates that result is not available for that instance.

5. For UDG instances, out of 12 instance groups, average solution quality returned by EA/G-IR is equal to the best among HGA, ACO-LS and ACO-PP-LS on 6 groups and better than all the three on remaining 6 groups.

   All the observations made so far clearly show the superiority of EA/G-IR over existing approaches in terms of solution quality which again vindicates the advantage of using both the global statistical information about the search space as well as location information of the solution generated so far while generating offspring.

6. From Tables 3.7 to 3.11, we can clearly say that EA/G-IR is much faster than HGA, ACO-LS and ACO-PP-LS. From these tables, it can also be realized that as the size of the graph increases, the average execution time taken by HGA, ACO-LS and ACO-PP-LS grow faster than that of EA/G-IR. The superiority of EA/G-IR over HGA in terms of execution time is due to the use of subset encoding in EA/G-IR in place of bit vector encoding which is used in HGA, and efficient implementation of repair operator (see Section 3.4.5). Size of dominating set is much less than the total number of nodes present and as such use of subset encoding wields a significant advantage in terms of time also

Table 3.4: Results of Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G for small & medium size Type II instances.

|V|	|E|	**Heu_A**	**Heu_I**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**		**EA/G-IR**		**EA/G**	
					Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
50	50	64.90	64.90	62.30	***60.80***	5.71	***60.80***	5.71	***60.80***	5.71	60.80	5.71	60.80†	5.71
50	100	100.10	100.10	98.40	***90.30***	17.21	***90.30***	17.21	***90.30***	17.21	90.30	17.21	90.30†	17.21
50	250	165.40	165.40	202.40	***146.70***	40.54	***146.70***	40.54	***146.70***	40.54	146.70	40.54	146.70†	40.54
50	500	205.50	205.50	312.90	***179.90***	63.14	***179.90***	63.14	***179.90***	63.14	179.90	63.13	179.90†	63.13
50	750	178.10	178.10	386.30	***171.10***	913.00	***171.10***	913.00	***171.10***	913.00	171.10	91.33	171.10†	91.33
50	1000	162.80	162.80	NA	***146.50***	97.30	***146.50***	97.3	***146.50***	97.30	146.50	97.30	146.50†	97.30
100	100	133.00	132.10^{++}	126.50	124.50	15.26	123.60	14.77	***123.50***	14.74	123.50	14.74	123.70	14.74
100	250	229.20	229.20	236.60	211.40	21.76	***210.20***	21.95	210.40	22.43	**209.20**	21.33	209.60‡	21.33
100	500	340.80	340.80	404.80	***306.00***	45.17	307.80	44.70	308.40	44.46	**305.70**	45.27	305.70‡	45.27
100	750	437.80	437.80	615.10	***385.30***	82.76	385.70	83.08	386.30	82.64	**384.50**	82.16	384.50‡	82.16
100	1000	470.30	470.30	697.30	***429.10***	76.13	430.30	79.43	430.30	79.43	**427.30**	77.05	427.30‡	77.05
100	2000	615.40	615.40	1193.90	***550.60***	171.77	558.80	169.7	559.80	171.82	550.60	171.77	550.60†	171.77
150	150	198.00	198.30^{--}	190.10	186.00	18.24	***184.70***	17.99	184.90	18.21	**184.50**	17.70	185.40	17.70
150	250	255.40	255.40	253.90	234.90	21.47	***233.20***	21.02	233.40	20.84	**232.80**	20.52	233.50	20.52
150	500	376.80	376.80	443.20	***350.00***	36.99	351.90	38.66	351.90	38.66	**349.70**	37.17	351.30	37.17
150	750	513.50	513.50	623.30	455.80	77.80	456.90	77.96	***454.70***	78.06	**452.40**	77.44	453.60‡	77.44
150	1000	622.80	622.80	825.30	***547.50***	82.12	551.40	83.67	549.00	81.68	548.20*	82.97	551.00	82.97
150	2000	833.40	833.40	1436.40	***720.10***	180.32	725.70	179.45	725.70	179.45	720.10	180.32	720.10†	180.32
150	3000	888.50	888.50	1751.90	***792.60***	218.03	794.00	220.10	806.20	245.43	**792.40**	217.92	793.20	217.92
200	250	297.40	296.70^{++}	293.20	275.10	21.55	***272.60***	20.12	***272.60***	20.31	**272.30**	20.38	274.80	20.38
200	500	427.00	427.00	456.50	390.70	55.66	388.60	56.42	***388.40***	56.64	388.40	56.80	389.00	56.80
200	750	560.50	560.50	657.90	507.00	60.41	501.70	50.44	***501.40***	50.10	**497.20**	51.94	501.70	51.94
200	1000	668.10	668.10	829.20	***601.10***	52.84	605.90	47.97	605.80	49.16	**598.20**	51.77	599.20‡	51.77
200	2000	1004.70	1004.70	1626.00	893.50	150.64	***891.00***	136.47	892.90	133.15	**885.80**	106.64	885.80‡	106.64
200	3000	1140.80	1140.80	2210.30	***1021.30***	162.54	1027.00	164.38	1034.40	167.73	**1019.70**	162.83	1023.80	162.83
250	250	327.20	327.30^{--}	NA	310.10	19.60	***306.50***	17.89	306.70	17.98	306.50	17.47	312.50	17.47
250	500	485.70	485.30^{++}	NA	444.00	30.35	443.80	32.84	***443.20***	32.47	**441.60**	31.53	448.50	31.53
250	750	635.40	635.40	NA	578.20	42.33	***573.10***	40.82	575.90	42.70	**569.20**	40.67	579.90	40.67
250	1000	749.70	749.70	NA	672.80	59.42	***671.80***	58.56	675.10	60.79	**671.70**	62.27	677.40	62.27
250	2000	1154.30	1154.30	NA	***1030.80***	139.83	1033.90	131.02	1031.50	129.71	**1010.30**	126.98	1019.60‡	126.98
250	3000	1438.80	1438.80	NA	***1262.00***	216.63	1288.50	212.74	1277.00	228.16	**1250.60**	214.08	1253.80‡	214.08
250	5000	1741.10	1741.10	NA	***1480.90***	307.17	1493.60	306.43	1520.10	349.30	**1464.20**	278.38	1470.30‡	278.38

NA indicates that result is not available for that instance.

over bit vector encoding. ACO-LS and ACO-PP-LS also uses subset encoding, but they are slower than EA/G-IR, because each solution in ACO-LS and ACO-PP-LS are constructed from scratch by performing a random walk on the graph where probability vector needs to be updated whenever a node is added to the partially constructed solution.

Combining the observation made here regarding execution times with observations made previously regarding the average solution quality of EA/G-IR vis-à-vis HGA, ACO-LS and ACO-PP-LS, we can say that EA/G-IR, in general, returns solutions of better quality in much shorter time when compared with HGA, ACO-LS and ACO-PP-LS.

7. Another important point about EA/G-IR is its fast rate of convergence. EA/G-IR on most

Table 3.5: Results of Heu_A, Heu_I, Raka-ACO, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G for large size Type II instances.

|V|	|E|	**Heu_A**	**Heu_I**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**		**EA/G-IR**		**EA/G**	
					Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
300	300	397.90	**397.80**	NA	375.60	24.79	***371.10***	23.14	***371.10***	23.44	**370.50**	23.14	380.00	24.62
300	500	516.00	516.00	NA	484.20	39.19	***480.80***	40.13	481.2	40.05	**480.00**	41.24	491.00	40.46
300	750	680.70	679.90^{++}	NA	623.80	52.76	621.60	48.76	***618.30***	48.90	**613.80**	52.25	631.50	55.27
300	1000	828.70	828.70	NA	751.10	75.91	744.90	77.80	***743.50***	74.20	**742.20**	72.28	767.10	75.65
300	2000	1240.60	1240.60	NA	***1106.70***	116.09	1111.60	114.61	1107.50	112.03	**1094.90**	106.64	1107.70	113.15
300	3000	1555.30	1555.30	NA	***1382.10***	125.93	1422.80	153.78	1415.30	167.50	**1359.50**	129.06	1371.90‡	141.51
300	5000	1941.10	1941.10	NA	***1686.30***	294.44	1712.10	291.41	1698.60	300.02	**1683.60**	294.89	1686.20‡	295.82
500	500	668.60	668.20^{++}	651.20	632.90	29.54	627.50	30.06	***627.30***	30.13	**625.80**	30.41	650.90	34.35
500	1000	987.50	987.40^{++}	1018.10	919.20	41.71	913.00	35.69	***912.60***	36.56	**906.00**	42.37	955.70	44.21
500	2000	1508.10	1507.50^{++}	1871.80	1398.20	131.90	1384.90	121.03	***1383.90***	121.77	**1376.70**	116.91	1441.30	139.79
500	5000	2668.10	2668.10	4299.80	***2393.20***	222.03	2459.10	272.38	2468.80	260.35	**2340.30**	210.28	2412.50	248.49
500	10000	3723.30	3723.30	8543.50	***3264.90***	4218.00	3377.90	470.35	3369.40	482.89	**3216.40**	389.63	3236.90‡	400.42
800	1000	1193.10	1192.50^{++}	1171.20	1128.20	48.22	1126.40	51.56	***1125.10***	50.79	**1107.90**	45.43	1187.60	50.73
800	2000	1813.00	1811.60^{++}	1938.70	***1679.20***	74.70	1693.70	80.25	1697.90	80.26	**1641.70**	75.40	1797.50	93.04
800	5000	3321.50	3321.50	4439.00	***3003.60***	204.03	3121.90	227.35	3120.90	229.21	**2939.30**	213.91	3200.40	243.65
800	10000	4725.30	4725.30	8951.10	***4268.10***	379.71	4404.10	380.67	4447.90	371.23	**4155.10**	346.83	4353.50	349.66
1000	1000	1338.50	1335.90^{++}	1289.30	1265.20	30.99	1259.30	33.44	***1258.60***	34.35	**1240.80**	30.45	1333.10	38.86
1000	5000	3596.40	3596.40	4720.10	***3320.10***	221.66	3411.60	228.22	3415.10	209.28	**3222.00**	196.89	3586.20	234.63
1000	10000	5432.60	5432.60	9407.70	***4947.50***	330.77	5129.10	308.66	5101.90	306.17	**4798.60**	291.65	5247.00	295.84
1000	15000	6857.00	6857.00	14433.50	***6267.60***	463.09	6454.60	445.76	6470.60	467.53	**5958.10**	427.56	6364.40	498.45
1000	20000	7785.60	7785.60	19172.60	***7088.50***	659.71	7297.40	598.98	7340.80	604.06	**6775.80**	571.69	7066.80‡	498.82

NA indicates that result is not available for that instance.

Table 3.6: Results of Heu_A, Heu_I, HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G for UDG instances.

|V|	**Range**	**Heu_A**	**Heu_I**	**HGA**		**ACO-LS**		**ACO-PP-LS**		**EA/G-IR**		**EA/G**	
				Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
50	150	429.60	429.60	394.30	59.42	***393.90***	58.89	***393.90***	58.89	393.90	58.98	393.90†	58.89
50	200	264.50	264.50	247.80	48.06	***247.80***	48.06	***247.80***	48.06	247.80	48.06	247.80†	48.06
100	150	484.40	484.40	450.40	73.23	***449.70***	73.71	***449.70***	73.71	449.70	73.71	449.70†	73.71
100	200	237.50	237.50	217.30	18.56	***216.00***	18.34	***216.00***	18.34	216.00	18.34	216.00†	18.34
250	150	323.80	323.80	***294.20***	53.42	294.60	52.92	294.50	52.92	**293.70**	52.48	293.90‡	52.86
250	200	135.10	135.10	119.10	17.80	***118.90***	17.95	***118.90***	17.95	**118.70**	18.02	118.70‡	18.02
500	150	183.80	183.80	172.70	36.91	***170.80***	35.88	170.90	35.45	**169.90**	34.66	170.50‡	35.18
500	200	77.40	77.40	68.10	11.29	68.00	11.37	68.00	11.37	68.00	11.37	68.00	11.37
800	150	129.50	129.50	115.90	18.33	114.10	16.58	***114.00***	16.45	**113.90**	16.47	114.50	16.93
800	200	48.20	48.20	42.60	8.04	41.00	7.60	***40.80***	7.47	40.80	7.56	40.90	7.56
1000	150	136.30	136.30	1250	16.17	***121.90***	14.12	121.90	13.64	**121.10**	13.53	121.50‡	13.31
1000	200	53.40	53.40	47.20	6.12	***46.00***	5.72	46.20	5.55	**45.90**	5.70	46.80	6.43

of the instances reached the best solution after few iterations only. As already mentioned, EA/G-IR generates 20000 solutions for each instance, but on most of the instances, it

Table 3.7: Average execution time of HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G in seconds on small & medium size Type I instances.

|V|	|E|	HGA	ACO-LS	ACO-PP-LS	EA/G-IR	EA/G
50	50	0.98	0.44	0.40	0.21	0.15
50	100	0.87	0.33	0.30	0.23	0.15
50	250	0.77	0.20	0.20	0.18	0.12
50	500	0.80	0.20	0.20	0.16	0.13
50	750	0.70	0.10	0.10	0.10	0.13
50	1000	0.70	0.10	0.10	0.10	0.11
100	100	3.45	1.54	1.40	0.76	0.45
100	250	3.30	1.22	1.10	0.72	0.39
100	500	2.61	1.03	0.90	0.60	0.33
100	750	2.02	0.84	0.80	0.52	0.31
100	1000	2.30	0.80	0.80	0.49	0.32
100	2000	2.86	0.70	0.70	0.45	0.35
150	150	9.31	3.94	3.50	1.64	0.81
150	250	8.60	3.55	3.10	1.71	0.78
150	500	6.68	2.66	2.40	1.45	0.67
150	750	5.69	2.33	2.10	1.27	0.61
150	1000	5.90	2.25	2.00	1.11	0.56
150	2000	4.74	1.70	1.70	0.88	0.55
150	3000	4.74	1.71	1.60	0.82	0.59
200	250	17.99	7.63	6.60	3.68	1.26
200	500	14.81	5.85	5.10	3.30	1.11
200	750	13.74	5.18	4.50	2.78	1.01
200	1000	12.45	4.5	4.00	2.39	0.94
200	2000	8.61	3.68	3.40	1.68	0.82
200	3000	8.27	3.27	3.10	1.42	0.81
250	250	31.23	13.96	12.20	4.65	1.85
250	500	27.08	11.35	9.90	4.66	1.67
250	750	26.55	9.65	8.40	4.25	1.52
250	1000	23.34	8.54	7.50	3.69	1.42
250	2000	14.63	6.50	6.00	2.65	1.22
250	3000	13.80	5.70	5.40	2.16	1.15
250	5000	12.92	4.98	4.80	1.85	1.16

Table 3.8: Average execution times of HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G in seconds on small & medium size Type II instances.

|V|	|E|	HGA	ACO-LS	ACO-PP-LS	EA/G-IR	EA/G
50	50	1.267	0.40	0.40	0.19	0.16
50	100	1.04	0.30	0.30	0.27	0.15
50	250	1.46	0.34	0.30	0.23	0.14
50	500	0.93	0.20	0.20	0.09	0.14
50	750	1.25	0.25	0.20	0.07	0.14
50	1000	0.95	0.15	0.10	0.06	0.12
100	100	4.12	1.57	1.50	0.86	0.46
100	250	4.47	1.42	1.30	0.92	0.43
100	500	4.90	1.30	1.20	0.78	0.38
100	750	3.74	1.04	1.00	0.70	0.36
100	1000	4.47	1.05	1.00	0.67	0.35
100	2000	4.68	0.96	0.90	0.54	0.39
150	150	12.00	3.70	3.50	1.85	0.82
150	250	12.68	3.75	3.40	2.03	0.82
150	500	11.46	3.27	2.90	1.95	0.75
150	750	10.12	2.74	2.60	1.78	0.70
150	1000	8.88	2.48	2.30	1.61	0.68
150	2000	8.96	2.25	2.10	1.20	0.64
150	3000	29.94	7.00	7.00	1.07	0.68
200	250	21.77	6.90	6.30	4.38	1.31
200	500	22.22	6.50	5.70	4.51	1.22
200	750	22.33	6.08	5.40	4.18	1.14
200	1000	17.92	5.38	4.90	3.89	1.08
200	2000	16.93	4.72	4.30	2.78	0.98
200	3000	18.37	4.422	4.30	2.16	0.95
250	250	41.76	13.29	12.10	5.26	1.90
250	500	43.77	12.48	11.20	6.10	1.80
250	750	39.30	11.72	10.50	6.09	1.70
250	1000	39.05	10.95	10.00	5.89	1.61
250	2000	31.97	8.95	8.50	4.23	1.41
250	3000	29.76	8.50	8.20	3.50	1.35
250	5000	29.69	7.09	6.90	2.59	1.34

reached the best solution after generating 300 to 10000 solutions. Very few instances took more than 15000 solutions to reach the best solution. Specially on Type II instances, it took a very less number of iterations to reach the best solution. On most of Type II instances, it has generated less than 5000 solutions to reach the best solution.

8. As far as comparison between Heu_I and Heu_A is concerned, out of a total of 106 groups of Type I and Type II instances where each group contains 10 instances with the same number of nodes and edges, the average solution quality of Heu_I is worse than that of Heu_A on 3 groups, equal to Heu_A on 87 groups and better than Heu_A on 16 groups.

For UDG, Heu_I found the same results as Heu_A for all instances. Heu_I could not improve the results of Heu_A for UDG, the reason can be the less number of edges in these graphs. Overall, the results of Heu_I and Heu_A vindicate our two modifications (Section 3.3).

9. When we compare EA/G with EA/G-IR, we can see that EA/G is quite capable of finding a high quality solution on its own on small and medium size instances. On the majority of these instances, it is able to find solutions as good as or better than HGA, ACO-LS and ACO-PP-LS, all of which use a local search. There is not much difference in solution quality obtained by EA/G and EA/G-IR on most of these instances. On the other hand, on large instances the benefits of improvement operator are evident as there is a large difference in solution quality between EA/G and EA/G-IR on all the instances. However, this improvement in solution quality comes at the expense of increased execution time. EA/G-IR is much slower in comparison to EA/G on large instances.

In Table 3.2, there are four instance groups where the average solution quality of EA/G is better than EA/G-IR, which seems to be anomalous. However, this is due to the difference in execution sequence between EA/G and EA/G-IR. Actually, when the solutions obtained by EA/G and EA/G-IR produce offspring through guided mutation, then these may have a different set of white nodes, and as a result, execution sequences in repair operator may be completely different. This difference in execution sequence in some rare cases can produce a solution which is even better than the one obtained through the improvement operator.

Table 3.9: Average execution times of HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G in seconds on large Type I instances.

|V|	|E|	HGA	ACO-LS	ACO-PP-LS	EA/G-IR	EA/G
300	300	52.64	22.27	19.10	8.73	2.38
300	500	49.77	18.72	16.30	7.21	2.18
300	750	44.42	16.21	14.30	6.48	1.95
300	1000	40.54	14.27	12.40	5.70	1.80
300	2000	26.60	10.50	9.50	4.03	1.46
300	3000	22.43	9.26	8.60	3.27	1.34
300	5000	17.12	7.81	7.50	2.59	1.30
500	500	204.83	89.57	77.50	37.52	6.02
500	1000	250.38	100.00	81.00	26.36	5.11
500	2000	145.95	60.13	54.40	25.54	4.05
500	5000	75.35	33.79	30.10	9.02	3.06
500	10000	43.62	25.15	24.40	6.08	2.77
800	1000	841.64	443.92	409.20	129.94	14.63
800	2000	576.34	301.57	292.20	102.09	11.76
800	5000	346.05	165.34	148.90	53.019	7.11
800	10000	162.32	97.28	95.90	31.17	5.12
1000	1000	2193.48	1023.79	922.40	249.82	23.76
1000	5000	626.10	341.12	326.50	107.41	13.03
1000	10000	380.27	207.39	194.70	63.22	9.02
1000	15000	254.64	162.73	150.80	45.86	7.44
1000	20000	174.50	142.40	139.20	36.35	6.57

Table 3.10: Average execution times of HGA, ACO-LS, ACO-PP-LS, EA/G-IR and EA/G in seconds on large Type II instances.

|V|	|E|	HGA	ACO-LS	ACO-PP-LS	EA/G-IR	EA/G
300	300	69.70	21.06	19.20	9.01	2.41
300	500	72.50	20.22	18.10	8.83	2.32
300	750	64.33	19.07	17.00	7.57	2.17
300	1000	59.28	17.68	16.00	8.96	2.06
300	2000	55.25	15.31	14.30	6.67	1.72
300	3000	46.13	15.55	13.60	5.41	1.58
300	5000	44.85	12.24	11.70	4.02	1.50
500	500	284.90	81.38	73.90	31.06	6.36
500	1000	268.61	77.31	68.50	28.27	5.69
500	2000	233.33	72.37	65.00	23.41	4.58
500	5000	122.30	51.34	51.00	17.36	3.17
500	10000	118.96	37.41	37.60	10.80	2.38
800	1000	1091.83	300.80	268.30	132.36	14.58
800	2000	927.69	307.11	282.20	111.84	12.49
800	5000	525.34	227.41	224.00	68.14	8.64
800	10000	387.92	148.32	148.20	40.15	6.02
1000	1000	2149.54	574.95	514.20	202.08	23.57
1000	5000	1112.49	464.20	461.30	132.94	15.05
1000	10000	739.45	323.16	324.40	84.82	10.91
1000	15000	617.16	251.67	251.80	61.64	8.96
1000	20000	536.83	212.02	213.80	59.20	7.78

Table 3.11: Average execution times of HGA, ACO-LS and ACO-PP-LS, EA/G-IR and EA/G in seconds on UDG instances.

|V|	Range	HGA	ACO-LS	ACO-PP-LS	EA/G-IR	EA/G
50	150	0.48	0.30	0.30	0.23	0.13
50	200	0.45	0.30	0.30	0.20	0.12
100	150	1.58	1.10	1.10	0.47	0.36
100	200	1.47	0.90	0.80	0.47	0.31
250	150	8.99	6.20	6.10	2.30	1.16
250	200	5.05	5.20	5.20	1.77	1.03
500	150	27.05	23.50	23.40	3.43	2.48
500	200	22.01	19.80	19.70	2.34	2.51
800	150	79.82	63.90	63.50	10.23	2.99
800	200	64.80	51.40	50.90	7.86	5.42
1000	150	115.31	98.40	97.50	13.76	6.71
1000	200	83.61	79.20	78.40	5.42	3.45

3.6 Conclusions

In this chapter, we have presented a hybrid approach called EA/G-IR combining evolutionary algorithm with guided mutation (EA/G) and an improvement operator for minimum weight dominating set problem. We have compared the performance of our EA/G-IR approach with other state-of-the-art approaches available in the literature on standard benchmark instances comprising general graphs and unit disk graphs. Computational results clearly show the superiority of EA/G-IR over other state-of-the-art approaches as it is able to find better quality solutions in general in much shorter time.

Chapter 4

Dominating Tree Problem

4.1 Introduction

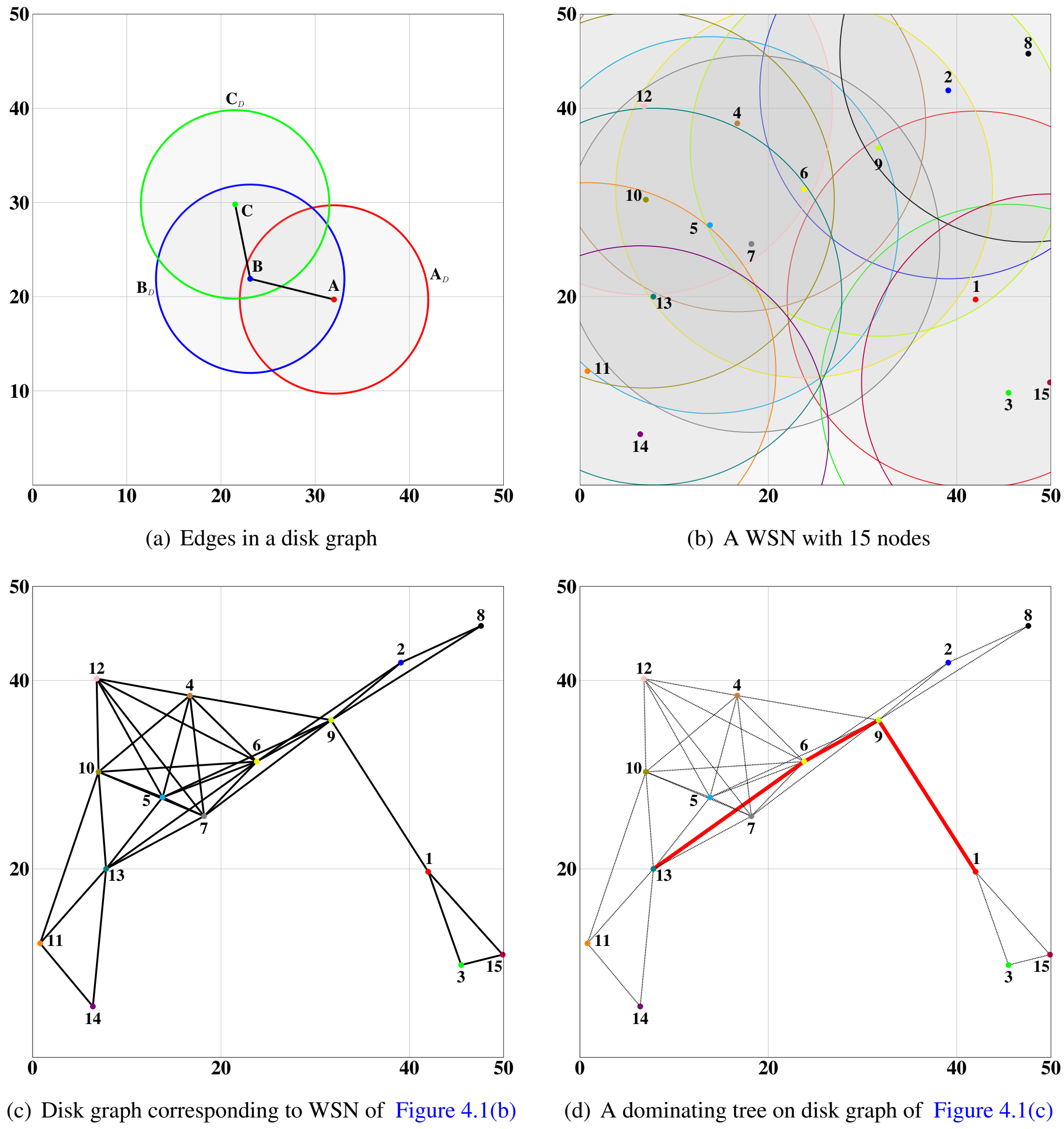
Consider an undirected, connected and edge-weighted graph $G = (V, E)$, where V denotes the set of vertices or nodes and E denotes the set of edges. The dominating tree problem (DTP) is concerned with finding a tree DT of minimum total edge weight on G in such a way that each node $v \in V$ either belongs to DT or is adjacent to a node belonging to DT, i.e., the set of nodes present in DT should form a dominating set. Accordingly, nodes in DT are said to be dominating nodes, whereas nodes which do not belong to DT are said to be non-dominating nodes. DTP is $\mathcal{NP}$-hard in general [87, 88]. However, in some special cases DTP can be solved in polynomial time, e.g. cases where the underlying graph is complete or is a tree. In case of a complete graph, each node is a minimum dominating tree in itself with cost 0. In case the underlying graph G is a tree, the minimum dominating tree is the subgraph (subtree) of G induced by non-leaf nodes.

The DTP, a relatively new problem, finds applications in the area of wireless sensor networks (WSNs). One such application of the DTP is to provide a virtual backbone for routing [82]. In this scheme, routing information is stored only on the dominating nodes after computing a dominating tree DT. Since non-dominating nodes are one hop away from nodes of the DT, in order to forward a message from one node (sender) to another node (receiver), the message can always be first forwarded to the nearest dominating node of the sender, then routed to the nearest dominating node of the receiver with the help of the DT, and finally forwarded to the receiver. Non-dominating nodes only need to know the nearest dominating node. The advantage of this scheme is that the number of dominating nodes is small in comparison to the total nodes [82], thereby, significantly reducing the size of the routing tables. Such a scheme is more resilient to faults also as these tables need to be recalculated only when topological changes in the network affect one of the dominating nodes.

In the literature, the connected dominating set concept has been widely used for constructing a routing backbone in WSNs with minimum energy consumption [89, 90, 91, 92, 93]. However, these approaches focus on the nodes instead of the edges in order to minimize the energy consumption. Actually, the energy consumption at each edge directly affects the energy consumption of routing. Therefore, one has to consider the energy consumption by each edge in order to minimize the energy consumption of routing. With this intention, the DTP was formulated [87, 88]. There exists another related problem called the tree cover problem that has been studied in the literature [94, 95, 96], but this problem is different from the DTP. A tree in the tree cover problem is defined as an edge dominating set, whereas a tree in the DTP is defined as a node dominating set.

Actually, communication in wireless sensor networks can be modelled using disk graphs where each disk around a sensor node represents the transmission range of that node. There exists an edge between a pair of nodes if these two nodes lie in the intersection area of their respective disks. In other words, an edge exists between a pair of nodes only when these two nodes are within the transmission range of each other. Obviously, only those nodes which are connected by an edge can communicate directly with each other. Figure 4.1(a) explains this concept where three nodes A, B and C are placed randomly in a $50m \times 50m$ area. The transmission range of each node is assumed to be $10m$. A$_D$, B$_D$ and C$_D$ are the disks associated with nodes A, B and C respectively. In this figure, for the sake of clarity, each node and its associated disk is represented with the same color which is different from the colors assigned to other nodes and their associated disks. Nodes A & B lie in the intersection area of disks A$_D$ & B$_D$. Similarly, nodes B & C lie in the intersection area of disks B$_D$ & C$_D$. Hence, an edge exists between A & B and another edge exists between B & C. On the other hand, nodes A & C do not lie in the intersection area of their respective disks, viz. A$_D$ & C$_D$, and hence there exists no edge between A & C. From this figure, it can also be observed that if the radius of the disks increases, the number of edges can increase.

Figure 4.1(b) shows a wireless sensor network consisting of 15 nodes, each with a transmission range of $20m$, placed randomly in a $50m \times 50m$ area. The disks associated with each node are also shown. This figure uses the same coloring scheme as used in Figure 4.1(a). The



(a) Edges in a disk graph

(b) A WSN with 15 nodes

(c) Disk graph corresponding to WSN of Figure 4.1(b)

(d) A dominating tree on disk graph of Figure 4.1(c)

Figure 4.1: Illustration of a disk graph and a dominating tree

corresponding disk graph is shown in the Figure 4.1(c). A possible dominating tree on this disk graph is shown in Figure 4.1(d) with thick red color edges.

Shin *et al.* [87] and Zhang *et al.* [88] both proved the $\mathcal{NP}$-hardness of the DTP, provided the inapproximability results and introduced an approximation framework for solving the DTP. Since approximation algorithm is quasi-polynomial ($|V|^{O(lg|V|)}$), each of them developed a polynomial time heuristic for the DTP. Later, Sundar and Singh [97] proposed one more heuristic and two metaheuristic techniques, viz. artificial bee colony (ABC) algorithm and ant colony optimization (ACO) algorithm for the DTP. To the best of our knowledge, only these two metaheuristic approaches have been proposed in the literature for the DTP. The heuristic of [97] outperformed the heuristics proposed in [87] and [88].

In this chapter, we present a heuristic and an evolutionary algorithm with guided mutation (EA/G) for the DTP. Our heuristic is derived from the heuristic proposed in [97]. We have compared our approaches with the previously proposed approaches. Computational results show the effectiveness of our approaches.

The organization of the remaining part of this chapter is as follows: Section 4.2 presents the formal problem formulation and introduces the notational conventions used in this chapter. Section 4.3 describes the modifications proposed in the heuristic of [97]. Section 4.4 describes our EA/G approach for the dominating tree problem. Computational results are presented in Section 4.5. Finally, Section 4.6 presents some concluding remarks.

4.2 Problem formulation

Let $G = (V, E)$ be an undirected connected graph, where V is the set of vertices or nodes and E is the set of edges. As defined already in the previous chapter, two nodes u and v are called neighbors of each other or adjacent to each other, iff, there exists an edge between them, i.e., $(u, v) \in E$. Likewise, two edges $e_{i,j}$ and $e_{k,l}$ are called neighbors of each other or adjacent to each other, iff, they have a node in common. Given a non-negative weight function $w : E \rightarrow \Re^{+}$ associated with the edges of G, the dominating tree problem (DTP) seeks on G a tree DT such that for each node $v \in V$, v is either in DT or adjacent to a node in DT and has a minimum total edge weight among all such trees, i.e.,

$$\textstyle\sum_{e_{i,j} \in DT} w(e_{i,j})$$

is minimum. Nodes in DT are called dominating nodes, whereas nodes not in DT are called non-dominating or dominatee nodes. In this chapter, we will also call any edge belonging to DT a dominating edge, and any edge not in DT a non-dominating or a dominatee edge.

Throughout this chapter, while constructing a dominating tree, we will follow the convention that a node which is neither in the tree nor adjacent to a node in the tree is assumed to have color

WHITE, a node which does not belong to the tree, but is adjacent to a node in the tree is assumed to have color GREY, and a node belonging to the tree is assumed to have color BLACK. Initially, all nodes are assumed to have color WHITE. When construction of dominating tree is completed then nodes belonging to the tree will have BLACK color and all other (non-dominating) nodes will have GREY color.

Important notational conventions used throughout this chapter are given in the Table 4.1. Please note that $On(v)$ and $CN(v)$ are already introduced in the previous chapter. Additional notational conventions will be introduced wherever those will be used.

Table 4.1: Notational convention

Notation	Definition
$e_{u,v}$	Edge between nodes u and v, i.e., $(u, v) \in E$.
$w(e_{u,v})$	weight of the edge $e_{u,v}$.
$ON(v) \subseteq V$	$\{u : u \in V$ and $(u, v) \in E\}$ is called open neighborhood of node $v \in V$.
$CN(v) \subseteq V$	$ON(v) \cup \{v\}$ is called closed neighborhood of node $v \in V$.
$wd(v) \subseteq CN(v)$	Set of WHITE nodes in the closed neighborhood of node $v \in V$.
$c(i, j)$	has value 1 if at least one of i and j have color WHITE, 0 otherwise.

4.3 Heuristic

This section describes our heuristic, which is an improved version of the heuristic H_DT proposed in [97]. We will refer to our heuristic as M_DT hereafter. Similar to the H_DT, the M_DT consists of two phases. The first phase is the initialization phase in which the shortest path between all pairs of nodes in the graph G are computed. The second phase consists of an iterative procedure to construct a dominating tree. At the beginning of the second phase of the M_DT, all nodes are assumed to have color WHITE, and we start with an empty tree DT. During each iteration, an edge $e_{i,j}$ is selected using the following expression (Equation (4.1)).

$$e_{i,j} \leftarrow \underset{e_{u,v} \in E}{\arg\max} \frac{W(e_{u,v}) \times nc(e_{u,v})}{w(e_{u,v})} \tag{4.1}$$

where $W(e_{u,v})$ is $(\sum_{x \in ON(u)} w(e_{u,x}) \times c(u,x) + \sum_{y \in ON(v)} w(e_{v,y}) \times c(v,y) - w(e_{u,v}) \times c(u,v))$, i.e., $W(e_{u,v})$ is the sum of the weights of all those edges which can potentially be avoided in DT in case the edge $e_{u,v}$ is selected and $nc(e_{u,v})$ is $|wd(u) \cup wd(v)|$, i.e., $nc(e_{u,v})$

gives the number of WHITE nodes in the closed neighborhood of nodes u and v. Therefore, expression Equation (4.1) selects an edge considering not only its own characteristics, but also the characteristics of its adjacent edges. The characteristics that are considered are the weight of an edge and the color of its end points. Further processing in the iteration depends on the color of the nodes i and j. Depending on the color of the nodes i and j, following cases can occur:

Case A : *If both the nodes i and j are WHITE*

A shortest path, say SP, between nodes $\{i, j\}$ and the partially constructed dominating tree DT is searched in G. If two or more than two shortest paths exist, then the tie is broken by selecting a path which has the maximum number of WHITE nodes. If a tie occurs in case of the number of WHITE nodes on the paths as well, then arbitrarily one such path is selected. All edges belonging to SP are added to DT and all nodes belonging to SP are recolored BLACK (if not already) and all WHITE neighbors of such nodes are recolored GREY. The edge $e_{i,j}$ will be added to DT only if one of the nodes among i and j which does not lie on the shortest path SP has at least one WHITE neighbor. Otherwise, there is no point in adding the edge $e_{i,j}$ as both of its endpoint nodes are non WHITE after adding the nodes of the SP to DT. If the edge $e_{i,j}$ got added to DT then nodes i and j are recolored BLACK (if not already) and all their WHITE neighbors are recolored GREY.

Case B : *If one node is WHITE and the other is GREY*

Check which one is WHITE node. Suppose node j is WHITE. Now, find a shortest path SP between partially constructed dominating tree DT and node j (ties are broken in the same manner as the previous case). Let k be the node which lies one hop away from node j on SP. All edges belonging to SP except $e_{k,j}$ are added to DT and all nodes belonging to SP except j are recolored BLACK (if not already) and all WHITE neighbors of such nodes are recolored GREY. Now check whether node j has any WHITE neighboring nodes, if yes, then add the edge $e_{k,j}$ into DT and recolor node j BLACK and also recolor GREY all the WHITE neighboring nodes of node j.

Case C : *If both the nodes i and j are GREY.*

C1 : *If both the nodes have WHITE neighboring nodes.*

Find, which one among i and j has shortest path SP from DT (ties are broken arbitrarily). Suppose SP connects i to DT. Add all the edges on the shortest path SP into DT and recolor all the nodes lying on SP BLACK and all the WHITE neighboring nodes of such nodes GREY. Again recheck that node j still has WHITE neighboring nodes, if yes, then proceed as in case C2.

C2 : *If only one has WHITE neighboring nodes.*

Suppose node j has WHITE neighboring nodes. Now, find the shortest path SP between partially constructed dominating tree DT and node j. Add all the edges on the shortest path SP into DT and recolor all the nodes lying on SP BLACK and all the WHITE neighboring nodes of such nodes GREY.

Case D : *If one node is BLACK and the other is GREY with at least one WHITE neighbor.*

Add the edge $e_{i,j}$ into partially constructed dominating tree DT and recolor BLACK the node whose color is GREY and also recolor GREY all its WHITE neighboring nodes.

After this another iteration begins. This process continues till the construction of the dominating tree is complete, i.e., till no WHITE node remains.

After the completion of the second phase of the heuristic, all nodes in DT are reconnected by computing a minimum spanning tree (MST) [98] on the subgraph of G induced by these nodes, thereby possibly reducing the cost further as MST is a spanning tree of least cost among all spanning trees over a graph with a given set of nodes. After computing MST, a pruning operator is called to remove all the redundant nodes from DT. A redundant node is a node such that if we remove that node from DT, DT still satisfy the property of a dominating tree. Detail of pruning operator can be found in Section 4.3.2.

The following points highlight the differences between the H_DT of [97] and our heuristic M_DT.

1. The determination of next edge $e_{i,j}$ to be added into DT differ for the H_DT and the M_DT. In the H_DT, next edge to be added is an edge whose edge weight is least among all available edges. Whereas in M_DT, next edge is selected with the help of expression Equation (4.1).

2. In the heuristic H_DT, the edge $e_{i,j}$ is always included into DT, but in case of heuristic M_DT, the edge $e_{i,j}$ will be added to DT only when absolutely necessary as explained already.

3. The heuristic H_DT applies two times pruning and two times reconnection by computing a minimum spanning tree (MST) in the following order: pruning $\rightarrow$ MST $\rightarrow$ pruning $\rightarrow$ MST. Whereas in our heuristic M_DT, we applied only once the pruning and MST in the order of MST followed by pruning. Actually, if we apply MST first, then we may get a dominating tree of lesser cost in comparison to applying the pruning first because we may loose some nodes in pruning which are vital for reducing the cost of the dominating tree. Empirical observations also favored this strategy. It is computationally less expensive, as well.

Algorithm 4.1 provides the pseudo-code of the M_DT.

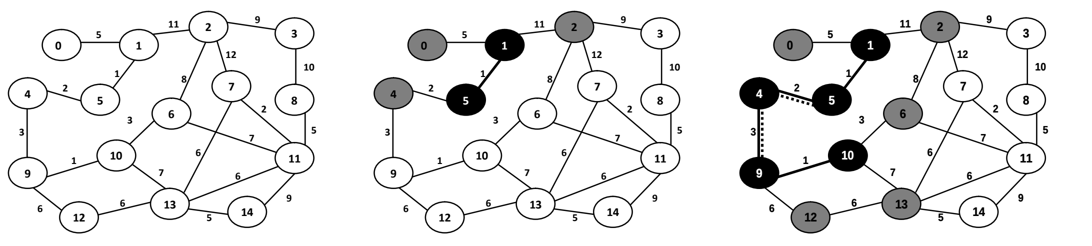
4.3.1 Illustrating H_DT & M_DT with an example

With the help of the Figure 4.2, we demonstrate the process of construction of a dominating tree using the heuristic H_DT in [97]. In the Figure 4.2(a), initially all the nodes are assumed to have color WHITE. The edge $e_{1,5}$ which has the least weight among all the edges, is selected and added into the partially constructed dominating tree DT. Now, the nodes {1, 5} are recolored BLACK and also their neighboring nodes {0, 2, 4} are recolored GREY. This situation is shown in the Figure 4.2(b). In the next iteration, the edge $e_{9,10}$ is added into DT. As a result, the nodes {9, 10} are recolored BLACK and their WHITE neighboring nodes {6, 12, 13} are recolored GREY. The two sub-trees $\{e_{1,5}\}$ and $\{e_{9,10}\}$ are connected via the shortest path between nodes 5 and 9. The node 4 which is on this shortest path is recolored BLACK and the edges $\{e_{5,4}, e_{4,9}\}$ are included into DT. This situation is depicted in the Figure 4.2(c) where the shortest path between two sub-trees is shown with thick and dotted line. Now, the edge $e_{7,11}$ is selected and included into DT which leads to the recoloring of the nodes {7, 11} with BLACK color and their WHITE neighboring nodes {8, 14} with GREY color. The two resulting sub-trees, viz. $\{e_{1,5}, e_{5,4}, e_{4,9}, e_{9,10}\}$ and $\{e_{7,11}\}$ are connected via the shortest path between nodes 10 and 11. The node 6 which is on this shortest path is recolored BLACK and the edges $e_{11,6}$ and $e_{6,10}$ are added into DT. This situation is shown in the Figure 4.2(d). In the last iteration, the edge $e_{2,3}$ is selected and included into DT and the nodes {2, 3} are recolored BLACK. The edge $e_{2,6}$ which constitutes the shortest path between the two sub-trees $\{e_{1,5}, e_{5,4}, e_{4,9}, e_{9,10}, e_{10,6}, e_{6,11}, e_{11,7}\}$ and $\{e_{2,3}\}$ is included into DT yielding a total weight of 36 for DT (Figure 4.2(e)). Now, no WHITE node remains, and as a result, the second phase of heuristic H_DT stops. Hereafter, a *pruning procedure* (Section 4.3.2) is applied to remove all the redundant edges as well as

Algorithm 4.1: The pseudo-code for heuristic the M_DT

```
    ▷ Initially all nodes in $V$ are colored WHITE
 1: $W_n \leftarrow V$;
 2: $DT \leftarrow \emptyset$;
 3: $E_r \leftarrow E$;
 4: $nd \leftarrow \emptyset$;
 5: Compute shortest path between all pairs of nodes in G;
 6: $e_{i,j} \leftarrow \arg\max_{e_{k,l} \in E_r} \frac{W(e_{k,l}) \times nc(e_{k,l})}{w(e_{k,l})}$;        ▷ Equation (4.1)
 7: $nd \leftarrow \{p : p \in ON(i) \cup ON(j) \cap W_n\}$;
 8: make $i$ and $j$ BLACK;
 9: make all nodes $\in nd$ GREY;
10: $W_n \leftarrow W_n \backslash (nd \cup \{i, j\})$;
11: $DT \leftarrow DT \cup \{e_{i,j}\}$;
12: while $W_n \neq \emptyset$ do
13:     $e_{i,j} \leftarrow \arg\max_{e_{k,l} \in E_r} \frac{W(e_{k,l}) \times nc(e_{k,l})}{w(e_{k,l})}$;        ▷ Equation (4.1)
14:     if Both the nodes $i$ and $j$ are WHITE then
15:         Apply *Case A*;
16:     else if One is WHITE and the other is GREY then
17:         Apply *Case B*;
18:     else if Both the nodes $i$ and $j$ are GREY then
19:         Apply *Case C*;
20:     else if One is BLACK and the other is GREY then
21:         Apply *Case D*;
22:     end if
23:     Remove all BLACK and GREY nodes from $W_n$;
24: end while
25: Reconnect nodes in $DT$ via a minimum spanning tree;
26: Apply Pruning operator on nodes in $DT$;
27: return $DT$;
```

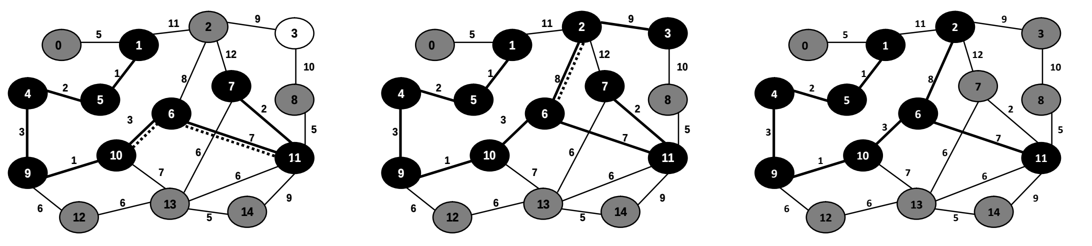
the redundant nodes. In the dominating tree $DT = \{e_{1,5}, e_{5,4}, e_{4,9}, e_{9,10}, e_{10,6}, e_{6,11}, e_{11,7}, e_{2,6}, e_{2,3}\}$, the nodes $\{3, 7\}$ are redundant. After the removal of these two redundant nodes and their corresponding redundant edges $\{e_{2,3}, e_{11,7}\}$, DT becomes $\{e_{1,5}, e_{5,4}, e_{4,9}, e_{9,10}, e_{10,6}, e_{6,11}, e_{6,2}\}$ with total weight 25. After the *pruning procedure* , a minimum spanning tree (MST) is constructed on the subgraph induced by the set of nodes in DT to explore the possibility of reconnecting these nodes via this MST in case it leads to reduction in cost. In this example, nodes in DT are already connected via a MST, so MST procedure fails to reduce the cost of DT any further. Once again the *pruning procedure* is applied, but in vain as there are no redundant nodes. Finally, MST procedure is also applied unsuccessfully on the nodes of DT and then the heuristic H_DT stops. The final dominating tree with weight 25 is shown in the Figure 4.2(f).



(a) Initially, all nodes are colored WHITE and $weight = 0$

(b) Edge $e_{1,5}$ is selected by heuristics H_DT and $weight = 1$

(c) Edge $e_{9,10}$ is selected by heuristics H_DT and $weight = 7$



(d) Edge $e_{7,11}$ is selected by heuristics H_DT and $weight = 19$

(e) Edge $e_{2,3}$ is selected by heuristics H_DT and $weight = 36$

(f) After pruning, MST, pruning and MST $weight = 25$

Figure 4.2: Illustrating H_DT heuristic

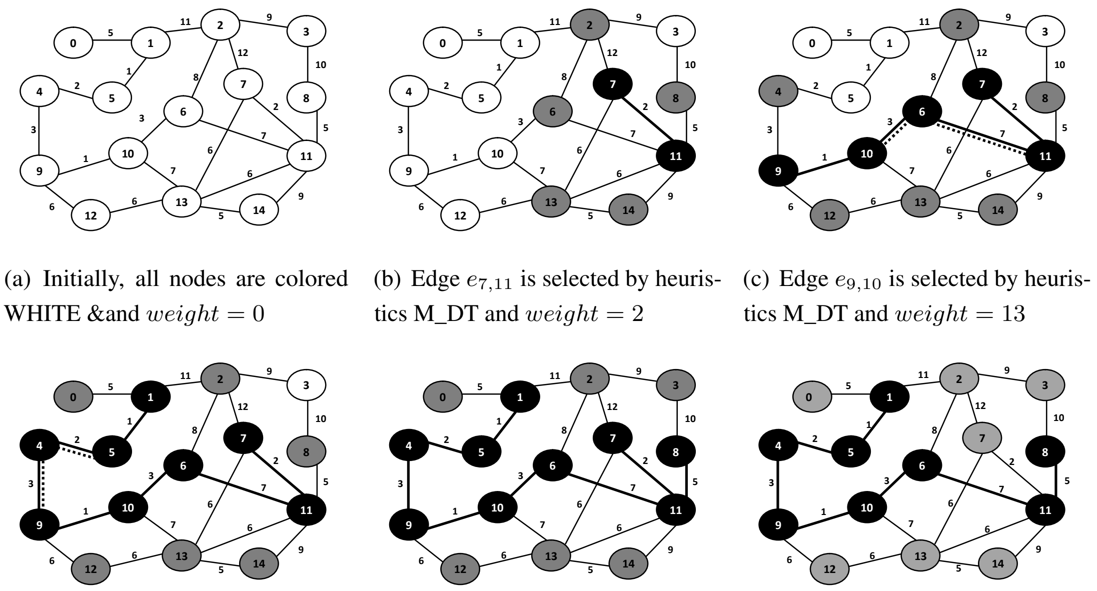
To illustrate the M_DT, the same input graph with 15 nodes as used for the H_DT is taken. Initially, all nodes are assumed to have color WHITE as shown in the Figure 4.3(a). At the first iteration, the edge $e_{7,11}$ is selected by the expression Equation (4.1) as this edge has the maximum ratio. Now, the edge $e_{7,11}$ is included into empty dominating tree DT. The nodes {7, 11} are colored BLACK and also all the WHITE neighboring nodes {2, 6, 8, 13, 14} of the nodes {7, 11} are colored GREY. This situation is shown in the Figure 4.3(b). In the next iteration, expression Equation (4.1) returns the edge $e_{9,10}$. Here both the nodes, viz. 9 and 10 are WHITE, and therefore, the *Case A* is applicable. Now, to connect the sub-trees $\{e_{7,11}\}$ and $\{e_{9,10}\}$ a shortest path between these two sub-trees is searched. Here the shortest path is between the nodes 11 and 10 with node 6 as the only intermediate node. The edges lying on the shortest path, viz. $e_{11,6}$ and $e_{6,10}$ are added into DT. The nodes 6 and 10 are recolored BLACK. Now, the $|wd(9)|$ is calculated and $|wd(9)| \geq 1$, therefore, the edge $e_{10,9}$ is added into DT and the node 9 is recolored BLACK and also all the WHITE neighboring nodes of the node 9, viz. 4 and 12 are recolored GREY. This situation is shown in Figure 4.3(c).

Next, the edge $e_{1,5}$ is returned by the expression Equation (4.1) leading to the *Case A* again. The edges $e_{9,4}$, $e_{4,5}$ and $e_{5,1}$ are added into the partially constructed dominating tree DT, the nodes 4, 5 and 1 are colored BLACK and their WHITE neighbors GREY (in this case only the node 0). This is shown in the Figure 4.3(d).

In the last iteration, the edge $e_{2,3}$ is returned by the expression Equation (4.1), because the ratio of edges $e_{2,3}$, $e_{8,11}$, $e_{3,8}$, $e_{2,6}$, $e_{1,2}$ and $e_{2,7}$ are 2.11, 2, 1.9, 1.12, 0.81 and 0.75 respectively and the remaining edges have ratio zero. Here the node 2 is GREY and the node 3 is WHITE. So the *Case B* is applicable. As a result, the shortest path between neighboring nodes {2, 8} of node 3 and the partially constructed tree $\{e_{1,5}, e_{5,4}, e_{4,9}, e_{9,10}, e_{10,6}, e_{6,11}, e_{11,7}\}$ is computed. This shortest path is between the nodes 11 and 8 without any intermediate node. The edge $e_{11,8}$ is included into DT, and the node 3, which is the only WHITE neighbor of the node 8 is recolored GREY. Here $|wd(3)|$ is zero, therefore the edge $e_{8,3}$ is not included into DT. At this point M_DT stops as no WHITE node remained, and the construction of the dominating tree DT is complete. This dominating tree has cost 24 (Figure 4.3(e)). Comparing Figure 4.2(e) with Figure 4.3(e), we can see that the cost of the dominating tree returned by the heuristic H_DT is 36, whereas the M_DT returns a dominating tree with cost 24 on the same graph. Thus, we can say that our heuristic M_DT can perform better than the heuristic H_DT of [97]. Hereafter, we compute a minimum spanning tree (MST) on the set of nodes in DT to reconnect these nodes via this MST in a bid to reduce the cost of DT further. As the nodes of DT are already connected via a MST, so cost of DT remains the same. Next the pruning operator is applied to remove the redundant nodes as well as redundant edges which removes the node 7 and the edge $e_{11,7}$ leading to the final DT with cost 22 as shown in Figure 4.3(f).

4.3.2 Pruning operator

Our Pruning operator is similar to the pruning procedure of [97]. Pruning operator removes the redundant nodes of dominating tree DT. Let DN be the set of nodes belonging to DT. A node $v \in DN$ is redundant if $|E_v \cap DT| = 1$ and $CN(v) \subseteq (\cup_{u \in DN \setminus \{v\}} CN(u))$, where $E_v = \{e_{v,x} \forall x \in ON(v)$. In other words, a node $v \in DN$ is redundant if the degree of the node v in DT must be one, v is adjacent to at least one other node in DN and each node in $ON(v)$ either belongs to DN or is adjacent to some node other than node v in DN. If the node v is redundant, then it can be removed from DN without affecting the dominating tree characteristic of DT. Our pruning operator begins by computing the set R_n of redundant nodes and then an iterative process starts where during each iteration a node is selected and removed from DN and



(a) Initially, all nodes are colored WHITE &and $weight = 0$

(b) Edge $e_{7,11}$ is selected by heuristics M_DT and $weight = 2$

(c) Edge $e_{9,10}$ is selected by heuristics M_DT and $weight = 13$

(d) Edge $e_{1,5}$ is selected by heuristics M_DT and $weight = 19$

(e) Edge $e_{2,3}$ is selected by heuristics M_DT and $weight = 24$

(f) After MST and pruning $weight = 22$

Figure 4.3: Illustrating M_DT heuristic

the set R_n is recomputed. We have selected a node for removal from R_n according to the order in which it is added into DN . We have also tried selecting a node from R_n according to the non-increasing order of the cost of their sole incident edge or according to the non-decreasing order of the number of non-dominating nodes covered by each node in R_n. Experimentally, we observed that solutions obtained through different ordering schemes did not differ much in quality and no ordering scheme has an ultimate advantage over others. Therefore, we settled for a simpler ordering scheme. Iterative process stops when the set R_n becomes empty. The pseudo-code of the pruning operator is presented in Algorithm 4.2, where $Select\_Node(R_n)$ is a function that returns a node from R_n which was added first into DN among all the nodes currently presented in R_n.

4.4 Hybrid EA/G Approach for DTP

We have developed a hybrid EA/G approach for DTP. The solutions obtained through the EA/G approach are further improved through the use of the same two procedures as used in M_DT, i.e.,

Algorithm 4.2: The pseudo-code of Pruning operator

```
1: $R_n \leftarrow \{v : v \in DN \; and \; |E_v \cap DT| = 1 \; and \; CN(v) \subseteq (\cup_{u \in DN \setminus \{v\}} CN(u))\}$;
2: while ($R_n \neq \emptyset$) do
3:     $v \leftarrow Select\_Node(R_n)$;
4:     $DN \leftarrow DN \setminus \{v\}$;
5:     $R_n \leftarrow \{v : v \in DN \; and \; |E_v \cap DT| = 1 \; and \; CN(v) \subseteq (\cup_{u \in DN \setminus \{v\}} CN(u))\}$;
6: end while
7: return $DN$;
```

reconnecting the nodes of the solution via minimum spanning tree (MST) and pruning operator. However, each of these procedures are applied twice in the order MST $\rightarrow$ pruning $\rightarrow$ MST $\rightarrow$ pruning. Hereafter, our hybrid EA/G approach with MST and pruning operator will be referred to as EA/G-MP (EA/G with MST and pruning operator).

Before starting our EA/G-MP approach, we pre-compute the set of neighboring nodes for each node $v \in V$ and the shortest paths between all pairs of nodes in V. Subsequent subsections describe other salient features of our approach.

4.4.1 Solution encoding

Edge-set encoding has been used to represent a solution, i.e., each dominating tree is represented directly be the set of the edges it contains. The edge-set encoding was introduced by [99] for representing a spanning tree. It is to be noted that a spanning tree always has $|V| - 1$ edges, whereas the number of edges in a dominating tree varies.

4.4.2 Initial solution

Our initial solution generation method is derived from the initial solution generation method used in [97]. First, we will describe the initial solution generation method of [97] and then introduce our modifications. Let W_n be the set of WHITE nodes, which is initialized to V ($W_n = V$ initially). Let I_u be the set of non-dominating nodes, DN be the set of dominating nodes and DT be the partially constructed dominating tree. Initially, these three sets, viz. I_u, DN and DT are empty. Randomly, a node say v is selected from W_n and added into DN and recolored BLACK and node v is removed from set W_n. Now, consider a set n_b of WHITE neighboring nodes of node v, i.e., $n_b = ON(v) \cap W_n$. Remove the nodes in n_b and also from W_n. After that, all the nodes in n_b are recolored GREY and added into the set I_u. From here onwards, at each step an edge is selected by the following one of the two strategies. With probability φ, the first strategy is followed where a least cost edge $e_{v,u}$ connecting a node v in DN and a node u in I_u is selected. Otherwise, second strategy is followed where an edge $e_{v,u}$ connecting a node v in DN to a node u in I_u is selected through roulette wheel selection method where the probability of selection of an edge is inversely proportional to its cost. Here φ is a parameter to be determined empirically.

Algorithm 4.3: The pseudo-code for initial solution

```
    ▷ Initially all nodes in $V$ are colored WHITE
 1: $W_n \leftarrow V$;
 2: $DT \leftarrow \emptyset$;
 3: $DN \leftarrow \emptyset$;
 4: $I_u \leftarrow \emptyset$;
 5: $v \leftarrow random(W_n)$;
 6: $DN \leftarrow DN \cup \{v\}$;
 7: make $v$ BLACK;
 8: $n_b \leftarrow ON(v) \cap W_n$;
 9: $W_n \leftarrow W_n \backslash (n_b \cup \{v\})$;
10: make all nodes $\in n_b$ GREY;
11: $I_u \leftarrow n_b$;
12: **while** $W_n \neq \emptyset$ **do**
13:     Generate a random number $u_{01}$ such that $0 \leq u_{01} \leq 1$;
14:     **if** $u_{01} < \varphi$ **then**
15:         $(v,u) \leftarrow \underset{v \in DN, \{u \in I_u,\ |ON(u) \cap W_n| \geq 1\}}{\arg\min} w(v,u)$;
16:     **else**
17:         $v \leftarrow random(DN)$
18:         $u \leftarrow \{u : random(I_u)\ and\ |ON(u) \cap W_n| \geq 1\}$;
19:     **end if**
20:     $DT \leftarrow DT \cup \{e_{v,u}\}$;
21:     $DN \leftarrow DN \cup \{u\}$;
22:     make $u$ BLACK
23:     $I_u \leftarrow I_u \backslash \{u\}$;
24:     $n_b \leftarrow \{u : u \in ON(u) \cap W_n\}$;
25:     make all nodes $\in n_b$ GREY;
26:     $I_u \leftarrow I_u \cup n_b$;
27:     $W_n \leftarrow W_n \backslash n_b$;
28: **end while**
29: Apply pruning operator on nodes in $DT$;
30: Reconnect nodes in $DT$ via a minimum spanning tree;
31: **return** $DT$;
```

Clearly, first strategy aims at quality, whereas the latter strategy aims at diversity. Therefore, φ governs the balance between the quality and the diversity of initial solutions. We have made two modifications in the initial solution generation method of [97]. Our first modification here is that only those edges $e_{v,u}$ are considered where $u \in I_u$ has at least one WHITE neighboring node, i.e., where $|ON(u) \cap W_n| \geq 1$. Whereas in [97], those nodes in I_u are also considered which have no WHITE neighboring nodes, and as a result, some edges might be unnecessarily inserted into DT. In addition, with probability $1 - \varphi$, we are selecting the edges uniformly at random instead of using roulette wheel selection method. Now, node u is added into set DN and removed from I_u and an edge $e_{v,u}$ is added into DT. Nodes in $n_b = ON(u) \cap W_n$ are added into I_u and nodes in n_b are removed from W_n. After that, all the WHITE neighboring nodes of the node u are recolored GREY. This whole process is repeated until the set W_n becomes empty. After construction of a feasible dominating tree DT, a pruning operator (as described in Section 4.3.2) is applied to remove the redundant nodes from DN and the redundant edges from DT. After an application of the pruning operator, a MST is constructed on the set of nodes in DN and these nodes are reconnected via this MST. Pseudo-code of the construction of an initial solution is presented in the Algorithm 4.3. Here, we have applied pruning first and then MST with the intention of generating more diverse solutions.

4.4.3 Initialization and update of the probability vector

Our EA/G-MP, like the approaches described in previous chapters models the distribution of promising solutions in the search space through the use of univariate marginal distribution (UMD) model. In this model, a probability vector $p = \{p_1, p_2, \dots, p_{|V|}\} \in [0, 1]^{|V|}$ is used to characterize the distribution of the promising solutions in the search space, where $|V|$ is the cardinality of the set V, i.e., the number of nodes in the graph G. p_v is the probability of the node $v \in V$ to be present in a dominating tree. The probability vector is initialized using N_p initial solutions. The probability of each node is initialized to the ratio of the number of initial solutions containing that node to the total number of initial solutions. The pseudo-code for initializing the probability vector p for the DTP is presented in Algorithm 4.4.

Algorithm 4.4: The pseudo-code for initializing a probability vector p

```
1: Compute $n_v \leftarrow$ number of initial solutions containing node $v$, $\forall v \in V$;
2: Compute $p_v \leftarrow \frac{n_v}{N_p}$, $\forall v \in V$;
```

At each generation g, a parent set *parent(g)* is formed by selecting the best L solutions from the current population *pop(g)*. Once the *parent(g)* is formed, it is used for updating the probability vector p. The pseudo-code for updating the probability vector is given in Algorithm 4.5. The probability of a node increases after update if the ratio of solutions containing this node in *parent(g)* to the total number of solutions in *parent(g)* is more than its current probability. The probability decreases in case this ratio is less than its current value. The probability remains the same in case this ratio is exactly equal to its current value.

Algorithm 4.5: The pseudo-code for updating the probability vector p in generation g

```
1: Compute $n_v$ ← number of solutions in parent(g) containing node $v$, $\forall v \in V$;
2: Compute $p_v \leftarrow (1-\lambda)p_v + \lambda\frac{n_v}{L}$, $\forall v \in V$;
```

4.4.4 Guided mutation (GM) Operator

The *GM* operator uses both the global statistical information stored in the form of probability vector p and the location information of the parent solution for generating new offspring. In [55] as well as in two previous chapters, GM operator is applied M times on the best solution of the current population to generate M offspring. On the other hand, our GM operator is applied on M best solutions of current population *pop*(g) to generate M new offspring. In other words, on the set of M best solutions $\{m_1, m_2, \ldots, m_M\}$, GM is applied once on each $m_i, i = 1, 2, \ldots, M$ to generate $\{o_1, o_2, \ldots, o_M\}$ offspring. We found that this strategy provides better results in case of DTP than the original strategy used in [55]. The pseudo-code of our GM operator is presented in Algorithm 4.6, where $\beta \in [0, 1]$ is an adjustable parameter and DT is a new offspring constructed through the GM operator whose nodes are either sampled randomly from the probability vector p or directly copied from the solution m_i in *pop*(g). In case of sampling from probability vector p, a node is copied only when either its color is WHITE according to partially constructed DT or its color is GREY and it has at least one WHITE neighbor. Whereas in case a node is to be directly copied from the solution m_i, it is copied only when it is a dominating node in m_i, its color is GREY according to partially constructed DT and it has at least one WHITE neighbor. Such a policy helps in getting some more edges in DT from m_i. As m_i is among the best M solutions, this may help in improving the solution quality. There is no guarantee of the feasibility of the offspring generated through the GM operator, i.e., it may not be a dominating tree. Therefore, each infeasible offspring generated through the GM operator is passed through a repair operator (Section 4.4.5) so that it can be made feasible.

Algorithm 4.6: The pseudo-code of generating a solution through *GM* operator

```
1:  Set the color of all nodes in V to WHITE;
2:  $DT \leftarrow \emptyset$;
3:  for node $v \in V$ in some random order do
4:      Generate a random number $r_1$ such that $0 \leq r_1 \leq 1$;
5:      if $r_1 < \beta$ then
6:          Generate a random number $r_2$ such that $0 \leq r_2 \leq 1$;
7:          if ($r_2 < p_v$) and (($v$ is WHITE) or ($v$ is GREY with at least one WHITE neighbor)) then
8:              Find the shortest path $SP$ between node $v$ and a node $u$ in $DT$;
9:              Add all the edges of $SP$ into $DT$;
10:             Color BLACK all the nodes on the path $SP$;
11:             Color GREY all the WHITE neighboring nodes of nodes on the path $SP$;
12:         end if
13:     else
14:         if $v$ is a dominating node in $m_i$ and $v$ is GREY with at least one WHITE neighbor then
15:             Find the shortest path $SP$ between node $v$ and a node $u$ in $DT$;
16:             Add all the edges of $SP$ into $DT$;
17:             Color BLACK all the nodes on the path $SP$;
18:             Color GREY all the WHITE neighboring nodes of nodes on the path $SP$;
19:         end if
20:     end if
21: end for
22: return $DT$;
```

4.4.5 Repair operator

Repair operator is applied only on an infeasible offspring generated through the GM operator. After the application of GM operator, there is a possibility that some nodes remain WHITE, i.e., some nodes may remain uncovered. Let U_{cn} be the set of such WHITE nodes. Such WHITE nodes are covered by making use of the repair operator which follows an iterative procedure. During each iteration, a node with the highest number of WHITE neighboring nodes is selected from U_{cn} (ties are broken in favor of the node having lower index). If none of the nodes in U_{cn} has WHITE neighboring nodes, then the node with the lowest index is selected from U_{cn}. After selecting the node i from U_{cn} , a shortest path between the node i and the partially constructed tree DT is found, and, all the edges on this path are added into DT. All the nodes on this path are recolored BLACK (if not already) and their neighboring nodes GREY. Then all BLACK and GREY nodes are removed from U_{cn}. After this another iteration begins. This whole process is repeated until set U_{cn} becomes empty. The pseudo-code of the repair operator is given in Algorithm 4.7.

Algorithm 4.7: The pseudo-code of repair operator

```
1: while $U_{cn} \neq \emptyset$ do
2:     $v \leftarrow \arg\max_{u \in U_{cn}}(|wd(u)|)$;
3:     Find a shortest path $SP$ between node $v$ and a node $u$ in $DT$;
4:     Add all the edges on the path $SP$ into $DT$;
5:     Make BLACK all nodes $\in SP$;
6:     Make GREY all WHITE neighboring nodes of nodes $\in SP$;
7:     Remove all BLACK and GREY nodes from $U_{cn}$;
8: end while
9: return $DT$;
```

4.4.6 Others features

Zhang *et al.* [55] kept best $\frac{N_p}{2}$ solutions of $pop(g)$ into the $parent(g)$ and generated $\frac{N_p}{2}$ new offspring through the GM operator in each generation (iteration). The population of the next generation is formed by using $\frac{N_p}{2}$ newly created offspring through the GM operator and the best $\frac{N_p}{2}$ solutions of $pop(g)$. Therefore, in each next generation the population size remains the same as in the previous generation. Same approach has been followed in Chapter 2. On the other hand, in the approach of previous chapter, we have kept best $\frac{N_p}{4}$ solutions of $pop(g)$ into the $parent(g)$ and generated $\frac{N_p}{4}$ new offspring through the GM operator in each generation (iteration). The population of the next generation is formed by using $\frac{N_p}{4}$ newly created offspring through the GM operator and the best $\frac{3N_p}{4}$ solutions of $pop(g)$. Generalizing this further, here $parent(g)$ is formed by using the best L solutions of $pop(g)$ and M new offspring are generated through the GM operator in each generation. The best $N_p - M$ solutions of $pop(g)$ along with M newly generated offspring constitute $pop(g + 1)$. Therefore, also in this case population size remains the same throughout the execution of the algorithm.

In case of DTP, we never found all the solutions of the population to be same. We also observed that the best solution does not improve for a large number of generations. Therefore, to avoid getting stuck into a local optima, if the best solution does not improve over S_c number of generations, then except for the best solution, we reinitialize the entire population in the same manner as described in Section 4.4.2. So, in a way, we have followed the 1-elitism policy as the best solution is always retained.

The pseudo-code of our EA/G-MP approach for DTP is given in Algorithm 4.8.

Algorithm 4.8: EA/G-MP Approach for DTP

```
1: At generation $g \leftarrow 0$, an initial population *pop(g)* consisting of $N_p$ solutions, is generated randomly;
2: Initialize the probability vector $p$ for all nodes using Algorithm 4.4;
3: Select best $L$ solutions from *pop(g)* to form a parent set *parent(g)*, and then update the probability vector $p$ using Algorithm 4.5;
4: Apply the *GM* operator once on each of the $M$ best solutions in *pop(g)* in order to generate $M$ new solutions. A repair operator is applied to each generated solution, if necessary, and then MST, pruning operator, MST and pruning operator are applied to each generated solution to improve its fitness. Add all $M$ newly generated solutions along with $N_p - M$ best solutions in *pop(g)* to form *pop(g+1)*. If the stopping condition is met, return the dominating tree with minimum weight found so far ;
5: $g \leftarrow g + 1$ ;
6: If the best solution of the population did not improve over $S_c$ generations, then reinitialize entire *pop(g)* except for the best solution, and then go to step 2 ;
7: Go to step 3 ;
```

4.5 Computational results

Our approaches, viz. M_DT and EA/G-MP have been implemented in C and executed on an Intel Core 2 Duo processor based system with 2 GB RAM running under Fedora 12 at 3.0 GHz which is exactly the same system as used for executing the approaches in [97]. `gcc 4.4.4-10` compiler with `O3` flag has been used to compile the C programs of our approaches. We have used a super set of test instances used in [97] to test our approaches. Due to unavailability of the test instances used in [87, 88], [97] generated a set of 18 test instances in the same manner as in [87, 88]. These instances were generated considering a disk graph $G = (V, E)$, where each disk around a node represents the transmission range of that node. There exists an edge between a pair of nodes if these two nodes are within the transmission range of each other. The weight on each edge $e_{i,j}$ in E is assigned through a weight function $w : E \rightarrow \Re^+$ which is defined as $w(e_{i,j}) = d_{i,j}^2$, where $d_{i,j}$ is the Euclidean distance between the nodes i and j. It was assumed that nodes in $|V|$ 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

Table 4.2: Results of MST_L, Heu_DT1, Heu_DT2, H_DT and M_DT on the instances with transmission range 100m

Instance	MST_L		Heu_DT1		Heu_DT2		H_DT		M_DT		% Improvement			
	Value	NDN	Value	NDN	Value	NDN	Value	NDN	Value	NDN	MST_L	Heu_DT1	Heu_DT2	H_DT
50_1	1860.67	38	1608.11	30	1819.91	35	1321.83	20	1288.80	20	30.73	19.86	29.18	2.50
50_2	1780.66	37	1564.05	33	1795.89	35	1427.65	26	1467.03	26	17.61	6.20	18.31	-2.75
50_3	1860.12	39	1659.70	34	1863.01	35	1494.12	25	1429.76	24	23.14	13.85	23.26	4.31
100_1	2491.23	76	1836.57	54	2290.28	61	1852.86	28	1482.11	26	40.51	19.30	35.29	20.00
100_2	2515.82	78	2096.97	62	2265.68	62	1449.25	23	1454.16	25	42.20	30.65	35.82	-0.32
100_3	2670.84	77	2213.15	60	2488.15	59	1732.35	29	1704.19	31	36.17	22.97	31.48	1.58
200_1	3652.20	154	2530.57	110	3093.01	115	1880.50	30	1766.97	35	51.62	30.18	42.87	6.04
200_2	3597.99	150	2709.42	113	3437.79	125	1909.86	32	1695.43	34	52.88	37.42	50.68	11.23
200_3	3592.74	152	2561.99	110	3132.56	112	1587.48	27	1589.81	31	55.75	37.95	49.25	-0.14
300_1	4445.38	231	2932.26	154	3653.64	165	1929.91	34	1695.08	30	61.87	42.20	53.61	12.17
300_2	4498.58	233	3480.73	178	4136.57	183	1781.00	31	1773.32	35	60.58	49.05	57.13	0.43
300_3	4673.49	239	3640.35	184	3990.55	170	1815.28	33	1673.31	33	64.20	54.03	58.07	7.82
400_1	5110.49	311	3776.71	230	4524.29	228	2017.50	32	1587.43	30	68.94	57.97	64.91	21.32
400_2	5225.01	310	4004.41	243	4744.41	248	1972.89	36	1904.82	37	63.54	52.43	59.85	3.45
400_3	5227.94	314	4026.04	241	4394.95	218	1907.05	29	1883.79	32	63.97	53.21	57.14	1.22
500_1	5761.72	390	4276.57	291	4534.93	257	1795.28	27	1771.82	34	69.25	58.57	60.93	1.31
500_2	5953.15	398	4399.44	296	5251.35	309	1824.03	34	1683.54	29	71.62	61.60	67.83	7.38
500_3	5840.50	390	4629.12	304	4944.21	269	1903.86	29	1837.40	30	68.54	60.31	62.84	3.49

Table 4.3: Results of MST_L, Heu_DT1, Heu_DT2, H_DT and M_DT on the instances with transmission range 125m

Instance	MST_L		Heu_DT1		Heu_DT2		H_DT		M_DT		% Improvement			
	Value	NDN	Value	NDN	Value	NDN	Value	NDN	Value	NDN	MST_L	Heu_DT1	Heu_DT2	H_DT
50_1	1860.67	37	1404.49	26	1679.80	29	982.61	14	1047.25	13	43.72	25.59	37.66	-6.58
50_2	1780.66	36	1407.53	26	1705.67	31	1165.63	16	1179.19	19	33.78	16.22	30.87	-1.16
50_3	1860.12	38	1488.20	29	1774.12	32	1154.05	14	1201.88	19	35.39	19.24	32.25	-4.14
100_1	2517.76	75	1737.47	50	2258.10	56	1442.11	18	1331.99	20	47.10	23.34	41.01	7.64
100_2	2515.82	77	1969.81	59	2372.79	60	1511.53	21	1238.10	20	50.79	37.15	47.82	18.09
100_3	2670.84	76	2213.15	59	2402.43	59	1445.39	20	1311.60	21	50.89	40.74	45.41	9.26
200_1	3652.20	153	2510.88	109	2990.67	100	1639.11	20	1355.67	24	62.88	46.01	54.67	17.29
200_2	3597.99	149	2733.43	113	3074.32	115	1436.93	23	1367.08	20	62.00	49.98	55.53	4.85
200_3	3592.74	151	2540.70	108	3118.57	109	1345.50	21	1307.22	20	63.61	48.55	58.08	2.85
300_1	4445.38	230	2899.16	153	3537.02	150	1454.30	19	1516.07	26	65.90	47.71	57.14	-4.25
300_2	4498.58	232	3500.06	180	4310.97	189	1739.35	23	1387.87	19	69.15	60.35	67.81	20.21
300_3	4673.49	238	3612.75	183	3802.58	160	1541.75	21	1370.75	20	70.67	62.06	63.95	11.09
400_1	5110.49	310	3758.23	228	4211.58	210	1654.87	23	1528.15	24	70.10	59.34	63.72	7.66
400_2	5225.01	309	3901.81	233	4596.69	235	1739.40	25	1539.59	23	70.53	60.54	66.51	11.49
400_3	5227.94	313	3981.63	238	4421.78	213	1630.39	23	1524.44	24	70.84	61.71	65.52	6.50
500_1	5761.72	389	4354.11	298	4472.74	245	1563.24	23	1551.67	26	73.07	64.36	65.31	0.74
500_2	5953.15	397	4471.75	299	5005.20	298	1638.64	23	1548.49	23	73.99	65.37	69.06	5.50
500_3	5840.50	389	4508.65	297	4715.44	259	1731.32	24	1344.64	23	76.98	70.18	71.48	22.33

instances were generated leading to a total of 18 instances. In addition to the transmission range of 100m, we consider two more values for the transmission range of each node, viz. 125m and

Table 4.4: Results of MST_L, Heu_DT1, Heu_DT2, H_DT and M_DT on the instances with transmission range 150m

Instance	MST_L		Heu_DT1		Heu_DT2		H_DT		M_DT		% Improvement			
	Value	NDN	Value	NDN	Value	NDN	Value	NDN	Value	NDN	MST_L	Heu_DT1	Heu_DT2	H_DT
50_1	1860.67	37	1408.01	26	1766.43	31	1058.69	12	784.03	11	57.86	44.32	55.61	25.94
50_2	1780.66	36	1309.87	27	1728.53	33	1061.02	12	1047.55	15	41.17	20.03	39.40	1.27
50_3	1860.12	38	1389.86	28	1887.77	32	1104.85	14	1010.84	12	45.66	27.27	46.45	8.51
100_1	2517.76	75	1737.47	50	2219.63	53	1420.84	14	1278.74	17	49.21	26.40	42.39	10.00
100_2	2515.82	77	2014.17	63	2276.44	57	1009.99	13	964.16	13	61.68	52.13	57.65	4.54
100_3	2670.84	76	2148.23	58	2331.72	55	1124.55	13	1184.35	15	55.66	44.87	49.21	-5.32
200_1	3652.20	153	2530.52	109	2911.73	95	1319.86	15	1286.15	16	64.78	49.17	55.83	2.55
200_2	3597.99	149	2703.77	114	3327.77	112	1300.02	19	1203.19	15	66.56	55.50	63.84	7.45
200_3	3592.74	151	2561.99	109	3112.19	108	1258.29	16	1224.62	17	65.91	52.24	60.65	2.68
300_1	4445.38	230	2908.12	153	3344.34	143	1170.23	16	1144.65	15	74.25	60.64	65.77	2.19
300_2	4498.58	232	3493.12	180	3740.08	160	1324.59	19	1224.33	13	72.78	64.95	67.26	7.57
300_3	4673.49	238	3589.75	183	4016.39	151	1382.82	18	1195.94	13	74.41	66.68	70.22	13.51
400_1	5110.49	310	3790.75	231	3704.20	187	1295.98	15	1166.44	17	77.18	69.23	68.51	10.00
400_2	5225.01	309	4017.53	243	4350.94	218	1174.12	13	1171.00	17	77.59	70.85	73.09	0.27
400_3	5227.94	313	4006.47	238	4304.85	197	1335.58	17	1272.84	17	75.65	68.23	70.43	4.70
500_1	5761.72	389	4253.35	288	4540.70	249	1252.10	15	1089.90	18	81.08	74.38	76.00	12.95
500_2	5953.15	397	4379.35	296	5269.14	303	1286.67	15	1279.42	17	78.51	70.79	75.72	0.56
500_3	5840.50	389	4618.27	300	4767.25	250	1474.32	17	1300.60	16	77.73	71.84	72.72	11.78

150m and generated three different test instances for each combination of values of V mentioned above and one of these two values of the transmission range. This results in generation of 36 additional instances leading to a grand total of 54 instances. All these 54 test instances can be downloaded from `http://dcis.uohyd.ernet.in/~alokcs/dtp.zip`. Actually, density of a disk graph depends on the transmission range of its constituent nodes. The longer the transmission range of nodes, the higher will be the density of the corresponding disk graph. Therefore, to show that effectiveness of our proposed approaches is not limited to graphs with a particular density, it is necessary to consider different values of the transmission range. The values of transmission range that we have considered leads to difficult randomly generated feasible DTP instances. We have also considered the transmission range of 75m and 200m. At the transmission range of 75m, not all instances with 50 nodes were connected. At the transmission range of 200m, generated instances were highly dense, and therefore, all dominating trees on these instances had few edges only, and as a result,finding a minimum dominating tree among them was not that difficult. As the C programs for the approaches considered in [97] were available, we have executed them on these additional 36 instances under the same setup as used for our approaches.

For EA/G-MP, we have used a population size of 60, i.e., $N_p = 60$, generated $M = 25$ new solutions through guided mutation and used $L = 15$ the best solutions of the current population to update probability vector. The value of β is set to 0.50 in the guided mutation. The value $\lambda = 0.50$ is used in the update of probability vector and the value $\varphi = 0.20$ is used in the initial solution generation. If the best solution does not improve over $S_c = 400$ generations, entire population minus the best solution and the probability vector are reinitialized. We have allowed our EA/G-MP approach to execute till the best solution does not improve over 3000 generations and it has executed at least for a total of 10000 generations. All these parameters are set empirically after a large number of trials. These parameter values provide good results on all instances, though they may not be optimal for all instances. Like ABC_DT and ACO_DT approaches of [97], EA/G-MP has been executed 20 independent times on each test instance.

We first present the results of M_DT and other problem specific heuristics. Tables 4.2 to 4.4 report the results of M_DT on instances with the transmission range 100m, 125m and 150m respectively and compare them with previously proposed heuristic approaches, viz. heuristics of [88], [87] and [97], which will be referred to as Heu_DT1, Heu_DT2 and H_DT respectively. In addition, we have also included a simple heuristic which computes a MST on the input graph and then removes leaf nodes from the computed MST to obtain a dominating tree. This heuristic, which will be referred to as MST-L was used in [88] and [87] for comparison against their respective heuristics. For each heuristic, these tables report the cost of the dominating tree obtained (column labelled Value) and the number of nodes in the dominating tree (column labelled NDN) on each instance. In Table 4.2, data for Heu_DT1, Heu_DT2, H_DT and MST-L are taken from [97]. Whereas Tables 4.3 and 4.4 contain the results obtained after executing various approaches on 36 new instances. These three tables also report the % improvement in cost of the dominating tree obtained by M_DT over other approaches. Though not the objective of DTP, we have reported the number of nodes in the dominating tree due to past precedences. [88], [87] and [97], all reported the number of nodes in the dominating tree obtained by various approaches. These tables clearly show the superiority of M_DT over other approaches in terms of the cost of the dominating tree obtained. Except for 8 instances (3 with transmission range 100m, 4 with transmission range 125m and 1 with transmission range 150m) where H_DT has slightly better cost (as indicated with a negative value for % improvement of M_DT over H_DT in Tables 4.2 to 4.4), the cost of the dominating tree obtained by M_DT is always better than all the other approaches. As far as number of dominating nodes in a solution is concerned, performance of M_DT is far superior in comparison to Heu_DT1, Heu_DT2 and MST-L on all instances. However, H_DT performs slightly better on this count on most of the instances.

Execution times of various heuristics are not reported as all of them hardly need a second on any instance.

Tables 4.5 to 4.7 report the results of EA/G-MP on instances with the transmission range 100m, 125m and 150m respectively, and compare them with ABC_DT and ACO_DT approaches of [97]. For each test instance, these tables report the best solution (column Best), average solution quality (column Avg), standard deviation of solution values (column SD), average number of dominating nodes (column ANDN) and average execution time in seconds (column AET) obtained over 20 runs for EA/G-MP, ABC_DT and ACO_DT. Data for ABC_DT and ACO_DT is taken from [97] for Table 4.5. On the other hand, Tables 4.6 and 4.7 report the results obtained after executing ABC_DT and ACO_DT on 36 new instances. These three tables also report the results of Mann-Whitney U test between EA/G-MP & ABC_DT (column ABC_EA/G) and between EA/G-MP & ACO_DT (column ACO_EA/G) on each instance as best and average solution quality of these approaches are close to each other. For Mann-Whitney U test, we have used the online calculator available at `http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx`. For this test, we have used two-tailed hypothesis and 5% significance criterion (p-value ≤ 0.05) leading to a critical U-value of 127.

Comparison of EA/G-MP, ABC_DT and ACO_DT approaches on instances with transmission range 100m: Out of 18 test instances with the transmission range 100m, EA/G-MP is better than ABC_DT on 10 test instances and equal to ABC_DT on 8 test instances in terms of quality of the best solution found. Whereas in terms of average solution quality, EA/G-MP is better than ABC_DT on 12 test instances, worse than ABC_DT on 3 test instances and on the remaining 3 test instances EA/G-MP is equal to ABC_DT. Results of Mann-Whitney U test between EA/G-MP and ABC_DT indicate that out of 18 test instances, results of EA/G-MP is statistically significant on 12 test instances in comparison to ABC_DT, whereas on 3 test instances (100_2, 200_1 and 200_2) results are not significant. On the remaining 3 instances, results of Mann-Whitney U test are not meaningful as both the approaches obtained the same results in all 20 runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ABC_DT on 5 test instances, worse than ABC_DT on 9 test instances and equal to ABC_DT on 4 test instances. Our EA/G-MP approach is much faster than ABC_DT approach. On all 18 test instances the average execution time (AET) of EA/G-MP is better than ABC_DT. From Table 4.5, it can be observed that as the size of the input graph increases, the gap in terms of computational time between ABC_DT and EA/G-MP increases as well. On an average, EA/G-MP is 3 times faster than ABC_DT on the instances with 50 nodes, 3 times faster than ABC_DT on the instances with 100 nodes, 4 times faster than ABC_DT on the instances with 200 nodes, 5 times faster than ABC_DT on the instances with 300 nodes, 6 times faster than ABC_DT on the instances with 400 nodes and 8 times faster than ABC_DT on the instance with 500 nodes.

Now, we compare EA/G-MP with ACO_DT. Out of 18 test instances, the best solution of EA/G-MP is better than ACO_DT on 11 test instances, worse than ACO_DT on 2 test instances and on the remaining 5 test instances both approaches obtained the same best solution. In terms of average solution quality, EA/G-MP is better than ACO_DT on 10 test instances, worse than ACO_DT on 5 test instances and on the remaining 3 test instances both the approaches have the same average solution quality. Results of Mann-Whitney U test between EA/G-MP and ACO_DT show that out of 18 test instances, results of the EA/G-MP is statistically significant in comparison to ACO_DT on 12 test instances, whereas on 3 test instances (100_1, 400_3 and 500_3) results are not significant. On the remaining 3 instances, results of Mann-Whitney U test are not meaningful as both the approaches obtained the same results in all 20 independent runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ACO_DT on 10 test instances, worse than ACO_DT on 4 test instances and equal to ACO_DT on 4 test instances. From Table 4.5, it can be observed that as the size of the input graph increases, the gap in terms of computational time between ACO_DT and EA/G-MP increases as well. On an average, EA/G-MP is 4 times faster than ACO_DT on the instances with 200 nodes, 10 times faster than ACO_DT on the instances with 300 nodes, 14 times faster than ACO_DT on the instances with 400 nodes and 22 times faster than ACO_DT on the instance with 500 nodes. On the other hand, EA/G-MP is 3 times slower than ACO_DT on the instances with 50 nodes and slightly slower than ACO_DT on the instances with 100 nodes.

Comparison of EA/G-MP, ABC_DT and ACO_DT approaches on instances with transmission range 125m: Out of 18 test instances with the transmission range 125m, the best solution obtained by EA/G-MP is better than ABC_DT on 10 test instances, worse than ABC_DT on 1 test instance and equal to ABC_DT on 7 test instances. Whereas in terms of average solution quality, EA/G-MP is better than ABC_DT on 11 test instances, worse than ABC_DT on 3 test instances and on the remaining 4 test instances EA/G-MP is equal to ABC_DT. Results of Mann-Whitney U test between EA/G-MP and ABC_DT indicate that out of 18 test instances, results of EA/G-MP is statistically significant on 12 test instances in comparison to ABC_DT, whereas on 2 test instances (200_2 and 300_3) results are not significant. On the remaining 4 instances, results of Mann-Whitney U test are meaningless as both the approaches obtained the same results in all 20 independent runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ABC_DT on 6 test instances, worse than ABC_DT on 6 test instances and equal to ABC_DT on 6 test instances. Our EA/G-MP approach is much faster than ABC_DT approach. On all 18 test instances, the average execution time (AET) of EA/G-MP is better than ABC_DT. From Table 4.6, it can be observed that as the size of the input graph increases, execution time of ABC_DT increases at a faster pace than EA/G-MP. On an average, EA/G-MP is 2 times faster than ABC_DT on the instances with 50 and 100 nodes, 4 times faster than ABC_DT on the instances with 200 nodes, 5 times faster than ABC_DT on the instances with 300 nodes, 7 times faster than ABC_DT on the instances with 400 nodes and 8 times faster than ABC_DT on the instance with 500 nodes.

As far as comparison between EA/G-MP and ACO_DT is concerned, out of 18 test instances, the best solution obtained by EA/G-MP is better than ACO_DT on 11 test instances, worse than ACO_DT on 1 test instance and on the remaining 6 test instances both the approaches obtained the same best solution. In terms of average solution quality, EA/G-MP is better than ACO_DT on 16 test instances, worse than ACO_DT on 1 test instance and on the remaining 1 test instance, both the approaches have the same average solution quality. Results of Mann-Whitney U test between EA/G-MP and ACO_DT show that out of 18 test instances, results of the EA/G-MP is statistically significant in comparison to ACO_DT on 15 test instances whereas on 2 test instances (50_1 and 200_1) results are not significant. On the remaining 1 instance, result of Mann-Whitney U test is not meaningful as both the approaches obtained the same results in all 20 independent runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ACO_DT on 10 test instances, worse than ACO_DT on 5 test instances and equal to ACO_DT on 3 test instances. From Table 4.6, it can be observed that as the size of the input graph increases, execution time of ACO_DT increases at a faster pace in comparison to EA/G-MP. On an average, EA/G-MP is 3 times faster than ACO_DT on the instances with 200 nodes, 6 times faster than ACO_DT on the instances with 300 nodes, 9 times faster than ACO_DT on the instances with 400 nodes and 10 times faster than ACO_DT on the instance with 500 nodes. On smallest instances with 50 nodes, EA/G-MP is 3 times slower than ACO_DT. On instances with 100 nodes, EA/G-MP is slightly slower than ACO_DT.

Comparison of EA/G-MP, ABC_DT and ACO_DT approaches on instances with transmission range 150m: On 18 test instances with the transmission range 150m, EA/G-MP is better than ABC_DT on 7 test instances, worse than ABC_DT on 1 test instance and equal to ABC_DT on 10 test instances in terms of quality of the best solution obtained. On the other hand, average solution quality of EA/G-MP is better than ABC_DT on 10 test instances, worse than ABC_DT on 4 test instances and same as ABC_DT on the remaining 4 instances. Results of Mann-Whitney U test between EA/G-MP and ABC_DT indicate that out of 18 test instances, results of EA/G-MP is statistically significant on 6 test instances in comparison to ABC_DT, whereas on 8 test instances (100_1, 100_2, 200_1, 200_2, 200_3, 300_1, 300_2 and 400_2) results are not significant. On the remaining 4 instances, results of Mann-Whitney U test are not meaningful as both the approaches obtained the same results in all 20 independent runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ABC_DT on 7 test instances, worse than ABC_DT on 5 test instances and equal to ABC_DT on 6 test instances. Our EA/G-MP approach is much faster than ABC_DT approach. On all 18 test instances, the average execution time (AET) of EA/G-MP is better than ABC_DT. From Table 4.7, it can be observed that as the size of the input graph increases, the gap in terms of computational time between ABC_DT and EA/G-MP increases as well. On an average, EA/G-MP is 2 times faster than ABC_DT on the instances with 50 nodes, 3 times faster than ABC_DT on the instances with 100 nodes, 5 times faster than ABC_DT on the instances with 200 nodes, 8 times faster than ABC_DT on the instances with 300 nodes, 11 times faster than ABC_DT on the instances with 400 nodes and 16 times faster than ABC_DT on the instance with 500 nodes.

Next, we compare EA/G-MP with ACO_DT. Out of 18 test instances, the best solution of EA/G-MP is better than ACO_DT on 8 test instances and on the remaining 10 test instances both approaches obtained the same best solution. In terms of average solution quality, EA/G-MP is better than ACO_DT on 8 test instances, worse than ACO_DT on 5 test instances and on the remaining 5 test instances both the approaches have the same average solution quality. Results of Mann-Whitney U test between EA/G-MP and ACO_DT show that out of 18 test instances, results of EA/G-MP is statistically significant in comparison to ACO_DT on 9 test instances whereas on 5 test instances (100_2, 200_2, 300_1, 400_1 and 500_3) results are not significant. On the remaining 4 instances, results of Mann-Whitney U test are meaningless as both the approaches obtained the same results in all 20 independent runs. As far as comparison in terms of average number of dominating nodes is concerned, EA/G-MP is better than ACO_DT on 10 test instances, worse than ACO_DT on 3 test instances and equal to ACO_DT on remaining 5 test instances.

ACO_DT is also slower than EA/G-MP on larger instances. From Table 4.7, it can also be observed that as the size of the graph increases, execution time of ACO_DT increases at a much faster rate compared to EA/G-MP. On an average, EA/G-MP is 3 times faster than ACO_DT on the instances with 200 nodes, 5 times faster than ACO_DT on the instances with 300 nodes, 7 times faster than ACO_DT on the instances with 400 nodes and 10 times faster than ACO_DT on the instance with 500 nodes. On the other hand, EA/G-MP is 3 times slower than ACO_DT on the instances with 50 nodes and slightly slower than ACO_DT on the instances with 100 nodes.

Table 4.5: Results of ABC_DT, ACO_DT and EA/G-MP on the instances with transmission range 100m

Instance	ABC_DT					ACO_DT					EA/G-MP					ABC_EA/G		ACO_EA/G	
	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	*U-value*	*p-value*	*U-value*	*p-value*
50_1	1204.41	1204.41	0.00	19.00	25.57	1204.41	1204.41	0.00	19.00	2.41	1204.41	1204.41	0.00	19.00	6.75	200.00	0.99202[a]	200.0	0.99202[a]
50_2	1340.44	1340.44	0.00	21.00	21.46	1340.44	1340.44	0.00	21.00	4.18	1340.44	1340.44	0.00	21.00	8.13	200.00	0.99202[a]	200.0	0.99202[a]
50_3	1316.39	1316.39	0.00	19.00	22.99	1316.39	1316.39	0.00	19.00	2.50	1316.39	1316.39	0.00	19.00	6.82	200.00	0.99202[a]	200.0	0.99202[a]
100_1	1217.47	1218.15	0.69	18.45	28.64	1217.47	1217.47	0.00	19.00	12.71	1217.47	1217.61	0.43	18.90	11.06	112.00	0.01778	180.00	0.59612
100_2	1128.40	1128.42	0.09	17.90	27.58	1152.85	1152.85	0.00	17.00	10.86	1128.40	1128.54	0.20	17.30	9.93	143.50	0.13104	0.00	0.00000
100_3	1252.99	1253.14	0.23	19.70	28.39	1253.49	1253.49	0.00	19.00	8.96	1253.49	1257.37	4.29	19.00	11.97	33.00	0.00000	110.00	0.01552
200_1	1206.79	1209.52	2.69	18.25	84.10	1206.79	1207.61	3.58	18.05	81.13	1206.79	1208.26	2.10	18.55	19.76	128.00	0.05360	5.00	0.00000
200_2	1216.41	1219.74	2.15	18.90	87.78	1216.23	1217.73	2.61	17.65	78.72	1216.41	1222.23	7.67	18.95	21.55	167.50	0.38430	65.00	0.00028
200_3	1253.02	1258.06	3.42	22.15	90.44	1247.25	1248.94	2.99	20.90	97.93	1247.63	1250.78	2.37	21.80	21.31	14.00	0.00000	89.00	0.00278
300_1	1229.97	1237.47	2.89	21.75	145.17	1228.24	1243.70	9.71	22.85	352.89	1225.22	1230.48	3.86	21.75	29.57	41.00	0.00000	37.00	0.00000
300_2	1182.52	1200.79	7.82	19.60	162.59	1176.45	1193.95	10.51	21.10	260.30	1170.85	1171.30	0.69	19.00	28.56	0.00	0.00000	0.00	0.00000
300_3	1257.21	1271.20	6.74	20.50	145.75	1261.18	1276.75	9.27	24.60	251.91	1252.14	1260.83	5.61	21.80	29.66	54.00	0.00000	27.00	0.00000
400_1	1223.61	1241.75	7.88	21.90	263.13	1220.62	1237.45	9.50	26.05	600.74	1211.72	1220.79	5.31	22.70	40.26	11.00	0.00000	10.00	0.00000
400_2	1220.54	1235.29	6.97	22.45	249.39	1209.69	1246.14	21.41	24.40	591.44	1199.92	1202.82	1.98	21.30	40.01	0.00	0.00000	0.00	0.00000
400_3	1266.41	1276.80	4.59	22.30	216.95	1254.10	1270.34	9.42	25.85	530.58	1248.29	1268.38	8.80	22.70	39.80	70.00	0.00046	177.00	0.54186
500_1	1233.14	1241.60	4.56	21.40	379.72	1219.66	1240.05	9.17	26.50	1163.20	1206.07	1222.12	11.19	22.30	44.75	33.00	0.00000	37.00	0.00000
500_2	1245.59	1258.33	5.40	22.35	364.04	1273.86	1295.51	13.39	28.65	1031.81	1226.78	1240.62	6.66	23.50	50.11	9.00	0.00000	0.00	0.00000
500_3	1249.17	1278.67	11.96	21.60	338.25	1232.71	1259.08	20.03	24.35	917.73	1232.15	1250.48	16.59	21.65	48.09	37.00	0.00000	160.00	0.28462

[a] Both the approaches obtained the same solution in all the runs.

Table 4.6: Results of ABC_DT, ACO_DT and EA/G-MP on the instances with transmission range 125m

Instance	**ABC_DT**					**ACO_DT**					**EA/G-MP**					**ABC_EA/G**		**ACO_EA/G**	
	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	*U-value*	*p-value*	*U-value*	*p-value*
50_1	802.95	802.95	0.00	10.00	11.26	802.95	803.26	1.36	10.05	2.16	802.95	802.95	0.00	10.00	5.01	200.00	0.99202[a]	190.00	0.79486
50_2	1055.10	1055.10	0.00	12.00	11.50	1055.10	1055.10	0.00	12.00	2.01	1055.10	1055.10	0.00	12.00	4.75	200.00	0.99202[a]	200.00	0.99202[a]
50_3	877.77	877.77	0.00	10.00	9.14	877.77	877.77	0.00	10.00	1.40	877.77	877.77	0.00	10.00	3.90	200.00	0.99202[a]	200.00	0.99202[a]
100_1	943.01	943.01	0.00	12.00	18.12	943.01	946.37	1.86	12.75	7.29	943.01	943.01	0.00	12.00	7.56	200.00	0.99202[a]	40.00	0.00000
100_2	917.00	917.03	0.09	13.00	19.34	935.71	938.71	1.26	12.00	6.81	917.95	917.95	0.00	12.00	7.13	0.00	0.00000	0.00	0.00000
100_3	998.18	998.82	1.18	14.00	18.09	998.18	1006.11	7.57	13.80	7.61	998.18	998.18	0.00	14.00	8.27	100.00	0.00714	60.00	0.00016
200_1	910.17	911.61	0.72	14.00	57.35	910.17	910.50	0.79	14.15	43.25	910.17	910.17	0.00	14.00	14.73	30.00	0.00000	170.00	0.42372
200_2	921.76	922.62	1.05	12.65	54.85	928.84	942.72	5.20	13.15	39.29	921.76	923.03	1.40	12.55	14.29	173.00	0.47152	0.00	0.00000
200_3	942.32	944.93	2.14	14.15	57.36	951.36	959.63	6.83	13.10	41.12	939.58	949.18	3.68	13.00	15.79	72.50	0.00058	4.50	0.00000
300_1	981.31	984.63	1.67	14.30	118.68	978.91	980.11	1.11	16.65	157.13	977.65	981.04	1.35	15.95	22.88	18.00	0.00000	92.00	0.00362
300_2	917.31	926.87	4.14	14.35	117.74	918.40	949.05	11.73	14.55	111.61	913.01	914.08	2.14	14.80	21.40	4.00	0.00000	0.00	0.00000
300_3	974.98	979.95	2.28	13.95	117.45	981.15	981.33	0.14	13.80	121.29	974.85	979.34	2.22	13.65	21.57	153.00	0.20766	75.00	0.00076
400_1	967.34	971.07	2.63	13.25	221.76	968.66	980.60	15.64	15.70	323.06	965.99	966.59	0.39	13.00	28.75	0.00	0.00000	0.00	0.00000
400_2	947.57	952.49	2.77	13.80	200.00	941.52	961.71	21.90	14.70	250.91	941.02	943.53	3.05	14.55	29.49	12.00	0.00000	65.00	0.00028
400_3	1003.24	1007.05	2.59	14.30	200.38	1002.61	1009.07	8.13	13.90	236.62	1002.97	1003.62	0.22	14.90	28.21	20.00	0.00000	119.00	0.02926
500_1	967.32	975.25	3.85	13.80	358.22	986.49	991.85	3.33	17.10	513.13	963.89	963.89	0.00	14.00	36.20	0.00	0.00000	0.00	0.00000
500_2	954.89	965.45	5.96	14.50	314.63	953.77	996.85	14.55	13.95	406.90	948.57	952.96	4.11	15.15	49.80	27.00	0.00000	9.00	0.00000
500_3	992.30	1001.21	4.39	16.15	342.23	1006.23	1007.36	1.53	15.85	456.67	980.67	992.64	7.19	15.90	45.36	70.00	0.00046	0.00	0.00000

[a] Both the approaches obtained the same solution in all the runs.

Table 4.7: Results of ABC_DT, ACO_DT and EA/G-MP on the instances with transmission range 150m

Instance	ABC_DT					ACO_DT					EA/G-MP					ABC_EA/G		ACO_EA/G	
	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	Best	Avg	SD	ANDN	AET	*U-value*	*p-value*	*U-value*	*p-value*
50_1	647.75	647.75	0.00	7.00	6.99	647.75	647.75	0.00	7.00	1.02	647.75	647.75	0.00	7.00	3.47	200.00	0.99202[a]	200.00	0.99202[a]
50_2	863.69	863.69	0.00	11.00	10.05	863.69	863.69	0.00	11.00	1.70	863.69	863.69	0.00	11.00	4.17	200.00	0.99202[a]	200.00	0.99202[a]
50_3	743.74	743.74	0.00	8.00	7.38	743.94	743.94	0.00	8.00	1.21	743.94	743.94	0.00	8.00	3.62	200.00	0.99202[a]	200.00	0.99202[a]
100_1	876.69	876.85	0.39	10.00	16.31	881.37	885.36	5.43	10.65	7.00	876.69	876.69	0.00	10.00	6.61	170.00	0.42372	0.00	0.00000
100_2	657.35	657.35	0.00	8.00	14.55	657.35	657.35	0.00	8.00	4.23	657.35	657.53	0.78	8.00	5.78	190.00	0.79486	190.00	0.79486
100_3	722.87	722.87	0.00	9.00	15.75	722.87	722.87	0.00	9.00	4.80	722.87	722.87	0.00	9.00	5.98	200.00	0.99202[a]	200.00	0.99202[a]
200_1	809.90	809.90	0.00	11.00	62.64	809.90	810.87	0.19	10.20	35.80	809.90	810.49	1.89	10.85	12.71	140.00	0.10740	125.00	0.04338
200_2	736.23	736.27	0.17	8.20	59.44	736.23	736.23	0.00	8.40	28.06	736.23	736.23	0.00	8.00	11.57	190.00	0.79486	200.00	0.99202
200_3	792.73	797.00	1.64	9.75	62.97	792.71	793.73	1.21	9.45	32.15	792.71	795.65	1.64	9.15	12.03	198.50	0.97606	85.50	0.00208
300_1	796.70	797.94	1.37	10.30	154.67	796.70	797.17	1.39	10.00	108.25	796.15	798.12	3.00	10.50	20.09	145.00	0.14156	188.00	0.75656
300_2	741.02	743.20	0.82	10.05	133.79	748.94	752.33	3.51	10.35	76.95	741.02	743.05	1.34	9.85	17.34	185.00	0.69654	0.00	0.00000
300_3	819.76	823.76	2.46	10.10	140.01	826.48	826.56	0.13	10.85	94.81	819.76	821.67	1.75	10.00	18.60	106.50	0.01174	11.00	0.00000
400_1	796.70	801.57	2.77	9.90	314.00	796.70	798.24	1.38	10.50	197.40	795.53	798.82	2.80	10.30	25.71	82.50	0.00158	198.00	0.96810
400_2	781.20	782.28	0.76	9.30	263.35	782.91	787.66	6.62	10.25	175.62	779.63	783.14	2.38	9.25	23.60	188.00	0.75656	70.00	0.00046
400_3	816.53	822.64	2.04	10.25	274.13	826.48	831.32	3.26	10.45	159.38	814.14	817.38	3.55	10.40	25.38	65.00	0.00028	0.00	0.00000
500_1	796.50	800.25	2.11	10.30	490.85	794.47	797.13	2.40	10.45	338.07	792.21	793.59	1.79	10.90	30.52	2.00	0.00000	64.00	0.00024
500_2	779.35	785.10	3.03	9.45	430.80	779.35	791.20	8.35	11.10	293.95	779.35	781.28	2.55	9.20	32.10	51.00	0.00000	58.50	0.00014
500_3	809.65	811.08	1.03	10.25	542.44	808.50	8113.35	3.30	11.55	290.10	808.50	810.27	0.76	10.55	29.65	100.50	0.00736	194.00	0.88076

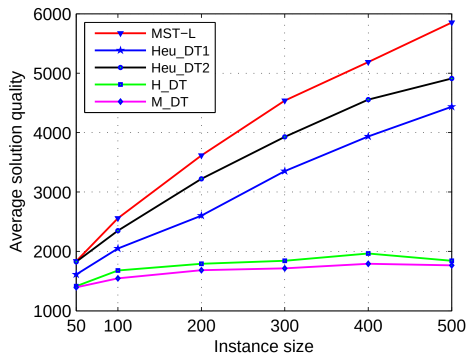
[a] Both the approaches obtained the same solution in all the runs.

The overall picture: Figures 4.4 and 4.5 graphically compare the average solution quality of various approaches over all the instances of each size, viz. 50, 100, 200, 300, 400 and 500 for different transmission ranges. Figure 4.4 compares different problem specific heuristics, viz. MST-L, Heu_DT1, Heu_DT2, H_DT and M_DT, whereas Figure 4.5 compares various metaheuristic approaches, viz. ABC_DT, ACO_DT and EA/G-MP. Figure 4.4 clearly shows that as the problem size increases, the cost of the dominating tree grows rapidly for MST-L, Heu_DT1 and Heu_DT2, thereby restricting the utility of these three heuristics to small size instances only. Like M_DT, H_DT also scales well, but its results are always inferior on an average to those of M_DT. If we look at Figures 4.4(a) to 4.4(c) together, it can be observed that with the increase in transmission range, the average cost of dominating tree returned by H_DT and M_DT decreases for the same node size, which is also expected theoretically. On the other hand, the average cost of the dominating tree returned by other heuristics seem to remain unaffected by the increase in transmission range.

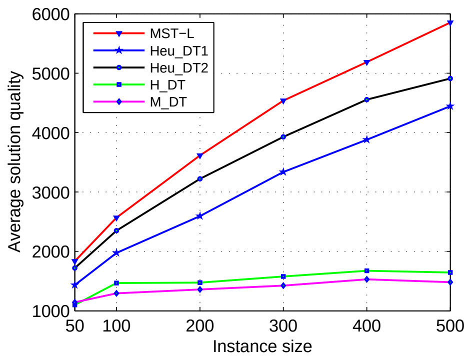
Figure 4.5 shows that all the metaheuristic approaches scale well with regard to average solution quality as the size of the problem increases. Except for instances of size 200 and range 100m, the average solution quality of EA/G-MP is better than ABC_DT and ACO_DT on all other sizes and transmission ranges. Looking at the Figures 4.5(a) to 4.5(c) together, we can observe that the average cost of the dominating tree returned by all the three metaheuristic approaches decreases with increase in transmission range.

Figure 4.6 graphically compares the average execution time of ABC_DT, ACO_DT and EA/G-MP over all the instances of each size for the transmission range 100m, 125m and 150m. The average execution time of ABC_DT and ACO_DT grows rapidly with increase in instance size. Only EA/G-MP scales well with a increase in instance size in terms of average execution time. Some interesting observations can be made if we look at Figures 4.6(a) to 4.6(c) together. Average execution time of ABC_DT increases with increase in transmission range. On the other hand, average execution time of ACO_DT decreases with increase in transmission range. For EA/G-MP also average execution time decreases, but only slightly when compared to ACO_DT.

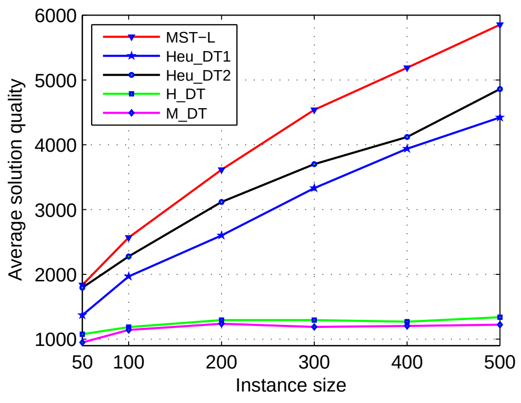
To get an idea about the effect of increase in transmission range on number of dominating nodes, we have found the best value among the average number of dominating nodes returned by ABC_DT, ACO_DT and EA/G-MP on each size for different transmission ranges. We have plotted these best values for each transmission range against node sizes. Resulting plots are shown in Figure 4.6. This figure clearly shows that the number of dominating nodes decreases on an average with increase in transmission range. This is also expected theoretically as with



(a) Transmission range 100m



(b) Transmission range 125m

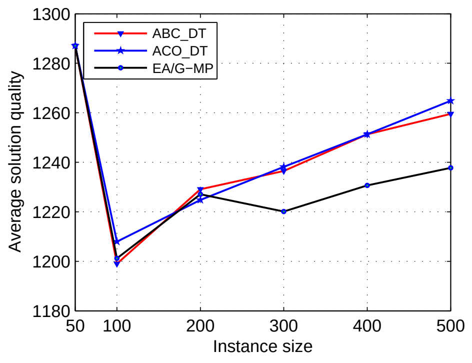


(c) Transmission range 150m

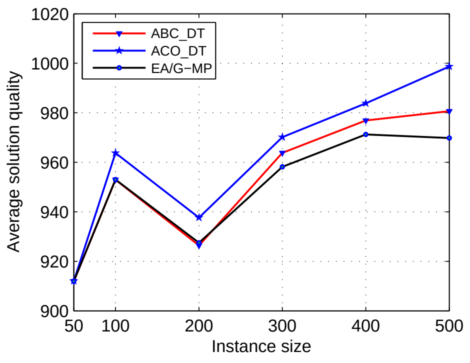
Figure 4.4: Average solution quality of various heuristics over all the instances of each size for different transmission ranges

increase in transmission range, degree of underlying disk graphs increases, leading to dominating trees with lesser number of nodes.

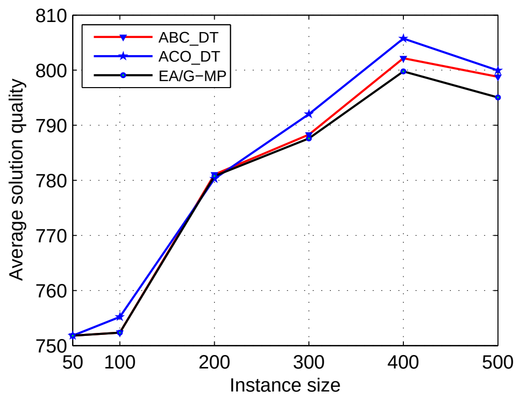
Though better than other approaches in their respective classes in terms of solution quality, both of our approaches obtain dominating tree with slightly more number of nodes on some instances when we compare M_DT to H_DT, and, EA/G-MP to ABC_DT and ACO_DT. Actually, sometimes even when a pair of nodes is connected directly by an edge, it may not be the shortest path between them. Instead a shortest path may involve one or more intermediate nodes. As a result, a dominating tree which has lesser cost than another may not have a lesser number of nodes as well. As our approaches favor shortest paths over direct edges more often in comparison to corresponding previous approaches, dominating trees obtained through our



(a) Transmission range 100m



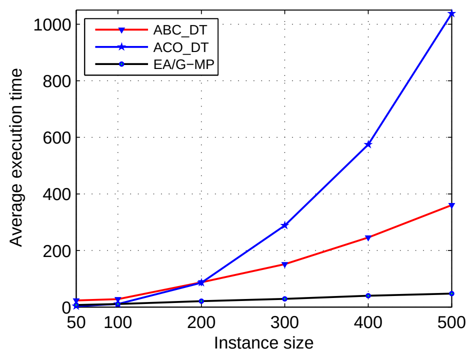
(b) Transmission range 125m



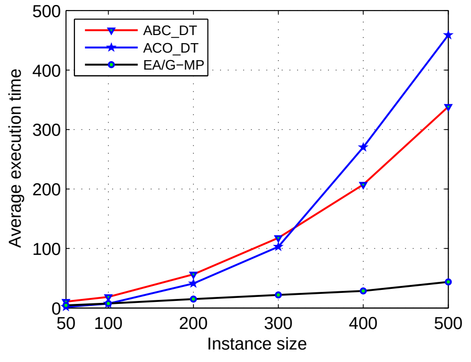
(c) Transmission range 150m

Figure 4.5: Average solution quality of ABC_DT, ACO_DT and EA/G-MP over all the instances of each size for different transmission ranges

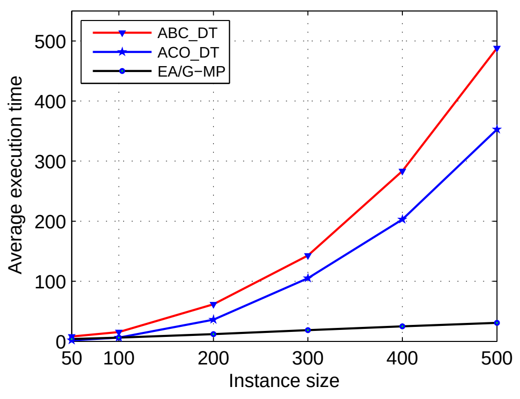
approaches are more likely to have more nodes.



(a) Transmission range 100m



(b) Transmission range 125m



(c) Transmission range 150m

Figure 4.6: Average execution time of ABC_DT, ACO_DT and EA/G-MP over all the instances of each size for different transmission ranges

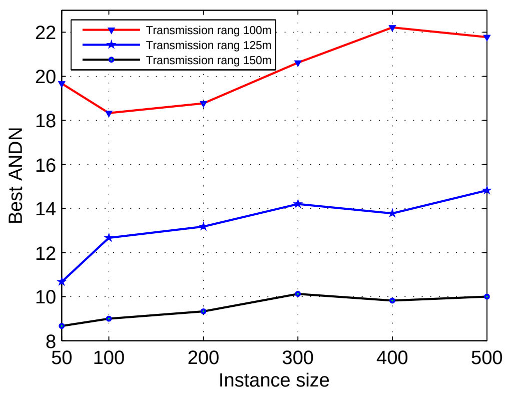


Figure 4.7: Best ANDN among ABC_DT, ACO_DT and EA/G-MP on each problem size for different transmission ranges

4.6 Conclusions

In this chapter, we have presented two approaches, viz. a problem specific heuristic called M_DT and a hybrid approach called EA/G-MP that combines evolutionary algorithm with guided mutation with two improvement procedures for the dominating tree problem (DTP). On 54 benchmark instances of various sizes and transmission ranges, M_DT produced better results in comparison to state-of-the-art problem specific heuristic approaches available in the literature. Similarly, EA/G-MP is able to find better solutions on most of the test instances when compared to two state-of-the-art metaheuristic approaches, viz. ABC_DT and ACO_DT in a much shorter time.

Chapter 5

Order Acceptance and Scheduling

5.1 Introduction

In a make-to-order system, customers put their orders at a time for processing. Due to this an additional wait time is incurred to receive the product. But it gives system the flexibility to customize the processing of these orders according to its capability. Tight order delivery deadline requirements may force a system with limited processing capability to first decide which orders should be accepted and then where to put the accepted orders in the processing sequence.

The order acceptance and scheduling problem (OAS) [100] deals with this class of problems where acceptance and timely delivery of orders play important role in generating revenue for the system. The OAS problem has two folds- first is to decide which orders should be accepted and the second is sequencing of the accepted orders. The OAS problem can be considered as a mix of knapsack and scheduling problems. If there is no scheduling component then the OAS problem reduces to knapsack problem. On the other side, if all the orders are accepted then the OAS problem reduces to a single machine scheduling problem. The coupled decisions of the OAS problem makes it a complex problem, but solving this problem helps in maximizing the net revenue gained through the production system.

A number of diverse versions of OAS problem have been studied in the last two decades. The extensive study of various OAS problem versions can be found in the literature survey of [101] and [102]. This chapter is focused on the OAS problem, where the number of incoming orders is fixed in a single machine environment. In other words, a set of orders are put at a time and the orders are processed sequentially one after another.

Many exact approaches have been proposed to solve the diverse versions of OAS problem. An integer-programming approach was proposed in [103]. Mixed-integer programming approaches were developed in [104, 105]. Further, Yang and Geunes [106], Oguz *et al.* [107], and Nobibon and Leus [108] developed mixed integer linear programming based approaches to solve the OAS problem. Dynamic programming based approaches were proposed in [109, 110, 111]. The generalized version of order acceptance and scheduling problem was solved by two exact branch-and-bound approaches [108]. An optimal branch-and-bound approach using a linear (integer) relaxation for bounding is applied to a problem where the acceptance and the scheduling of orders happen together. Another enhanced branch-and-bound method was proposed in [112].

The complex decision procedure of the OAS problem makes it $\mathcal{NP}$-hard [107, 113, 114]. Due to this, it is not possible to solve large sized problem instances through exact approaches. To get a good solution with a limited computational expense, a number of heuristic algorithms are also proposed. In general, the heuristic algorithms can be classified into two categories, viz. construction heuristics (CHs) and improvement heuristics (IHs). In CHs, an order is added at a time according to some criterion. Once an order is accepted and scheduled, it cannot be reversed. The whole process is repeated until all the orders are processed. The examples of CHs are the methods proposed in [5, 103, 106, 107, 110, 115]. A common shortcoming of all these methods is the non-robustness of the solutions obtained by them. Another limitation of these methods is no one outperforms all other CHs on a given performance criteria on large sized problem instances. On the other hand, an IH method begins with an initial solution and repeatedly try to improve it in a reasonable time. The examples of IH methods are those proposed in [5, 105, 109, 114, 116, 117]. Further, metaheuristic approaches can also be regarded as extension of IH methods where a single solution or a group of solutions are processed to get even better solution(s). Several metaheuristic based algorithms have been proposed to solve the different versions of OAS problem in a single machine environment. The examples of metaheuristic algorithms are simulated annealing (SA) [107], genetic algorithm (GA) [118, 119], extremal optimization (EO) [118], tabu search (TS) [5] and artificial bee colony algorithm (ABC) [120]. The last two approaches, viz. TS and ABC are the two best performing approaches for the OAS problem, and hence, the next two paragraphs provide a brief introduction to these approaches.

Cesaret *et al.* [5] proposed three algorithms for the OAS problem, viz. a modified apparent tardiness cost rule-based approach (called *m-ATCS*), a SA-based algorithm (called *ISFAN*) and a tabu search based algorithm (called *TS*). The OAS problem is also solved by the CPLEX solver with a time limit of 3600 seconds using the mixed integer linear programming (MILP) formulation given in [107]. The tabu search (TS) algorithm uses compound moves to guide the search. Each compound move performs iterative drop, add and insertion operations by using the problem specific information. The TS algorithm maintains a database of two different factors- first is the status of accepted as well as rejected orders and the second is the processing sequence of the orders. Based on the information in the database about these two factors the acceptance and scheduling of orders are done simultaneously rather than sequentially. After each iteration of TS algorithm the solution is passed through a probabilistic local search procedure to improve the solution and also to maintain diversification in the search process.

Lin and Ying [120] proposed an artificial bee colony algorithm (ABC) which is a swarm-based metaheuristic to solve OAS problem. They used permutation based representation of solution. In the ABC approach, the neighbouring solutions are generated using either order crossover [21] or IG algorithm [121]. The IG algorithm consists of destruction and construction phases, and implemented in the same way as Yin and Lin [122] implemented. In destruction phase, randomly a fixed number of orders is deleted, whereas, in construction phase, the orders deleted by destruction phase are reinserted. In the employed bee phase, the neighbouring solutions are generated by using either IG algorithm or order crossover. However, in the onlooker bee phase, neighboring solutions are generated through IG algorithm only. The performance of ABC algorithm was compared with TS approach on the same benchmark instances as used in [5]. Computational results established the superiority of ABC approach over TS.

In this chapter, we present a steady-state genetic algorithm (SSGA) and an evolutionary algorithm with guided mutation (EA/G), both hybridized with a local search for the OAS problem in a single machine environment. We have compared our hybrid SSGA and hybrid EA/G approaches with the TS [5] and ABC [120] approaches on the same benchmark instances as used in [5, 120]. In comparison to these approaches, our hybrid approaches obtained better quality solutions.

The remainder of this chapter is organized as follows: Section 5.2 formally defines the OAS problem studied in this chapter. Section 5.3 and Section 5.4 respectively describe our hybrid steady-state genetic algorithm and hybrid evolutionary algorithm with guided mutation approaches for the OAS problem. Computational results and their analysis are presented in Section 5.5. Finally, Section 5.6 outlines some concluding remarks.

5.2 Problem formulation

The formulation of the order acceptance and scheduling (OAS) problem is same as provided in [120]. In a single machine environment, a set of n incoming orders is to be processed one at a time without any precedence and without preemption. In other words, any order can be processed at any time (after its release date) and can occupy any position in the processing sequence, and, once an order starts processing, it can not be stopped until it finishes. Each order i has the following characteristics:

1. A release date r_i, i.e., processing for order i can not start before its release date.
2. A processing time P_i, i.e., machine takes P_i time to process order i
3. A maximum revenue gain E_i, i.e., if order i is accepted and its completion time C_i is less than or equal to its due date d_i, then the revenue gain for the order i will be E_i. If C_i is greater than d_i then a tardiness penalty is incurred and revenue gain decreases with time and, finally, the revenue gain will be zero when C_i is greater than or equal to the dead line $\overline{d}_i$ for order i. The per unit time tardiness penalty of an order i is calculated as follows w_i = $E_i/(\overline{d}_i - d_i)$ where $d_i < \overline{d}_i$.
4. A sequence dependent setup time S_{ij} $(i \neq j)$ between order i and j is incurred when order i is processed immediately before order j in the sequence.

It is assumed that all the above information about all the orders are known in advance. Based on the above definitions, the objective function of OAS problem is to find a processing sequence of all the accepted orders on the single machine that maximizes the total net revenue (TNR). The objective function for TNR is presented in Equation (5.1).

$$TNR = max \sum_{i=1}^{n} x_i(E_i - w_iT_i) \tag{5.1}$$

Where n is number of orders, $T_i = max(0, C_i - d_i)$ is tardiness of an order i. x_i $\forall i \in \{1, 2, \ldots, n\}$ are boolean variables. $x_i = 1$ means order i is accepted and $x_i = 0$ means order is rejected. Here, it is to be noted that any order j for which $C_j \geq \overline{d}_j$ is rejected by setting its associated boolean variable x_j to zero.

5.3 Hybrid steady-state genetic algorithm

Our genetic algorithm for the OAS problem uses the steady-state population model [21]. During each generation a single offspring is produced which replaces the worst member of the population irrespective of its own finess in case this offspring is different from all the existing population members. This offspring is discarded in case it is found to be identical to any existing population members. Other salient features of our SSGA approach are described in the subsequent subsections.

5.3.1 Solution encoding

We have used the same encoding scheme as used in [120]. In this scheme, each chromosome is a linear permutation of all the n orders. An order at a particular position in the chromosome is accepted for processing only when its deadline constraint is not violated considering all orders preceding it in the solution and its release date.

5.3.2 Fitness evaluation

Fitness function calculates the fitness of only accepted orders in the solution and is same as the objective function (Equation (5.1)).

5.3.3 Initial population generation

Each member of the initial population except the first member is generated in the following manner: Initially, OAS is an empty schedule, C_l is the completion time of the last order in the schedule which is initialized to zero. O is set of orders which are yet to be scheduled. Initially, O contains all the orders. U_o is a set which maintains the rejected orders and is initially empty. Iteratively, an order o is selected from O in one of the two ways - With probability θ, Equation (5.2) is used to select an order, otherwise randomly an order is selected from the set O.

$$o \leftarrow \arg\max_{i \in O} \frac{E_i}{\varphi_i} \tag{5.2}$$

Where φ_i is $(S_{ki} + P_i) \times \overline{d}_i$ and k is the order last added in OAS. Obviously, Equation (5.2) favors an order with higher revenue, lesser sum of setup & processing times and earlier deadline. Here, θ is a parameter to be determined empirically. o is deleted from O and added to OAS at the first available position if it is possible to finish its processing before its deadline. If not, o is

Algorithm 5.1: The pseudo-code for generation of each initial solution.

```
1: $O \leftarrow \{o_1, o_2, \ldots, o_n\}$                    ▷ $n$ is number of orders
2: $OAS \leftarrow \Phi$;
3: $oa_c \leftarrow 1$;
4: $C_t \leftarrow 0$;
5: while ($O \neq \emptyset$) do
6:     Generate a random number $r$ such that $0 \leq r \leq 1$;
7:     if ($r < \theta$) then
8:         $o \leftarrow \arg\max_{o \in O} \frac{E_o}{\varphi_o}$;                ▷ Equation (5.2)
9:     else
10:        $o \leftarrow random(O)$;
11:    end if
12:    if ($R[o] < C_l$ and $(C_l + S[OAS[oa_c - 1]][o] + P_o) < \bar{d}(o)$) then
13:        $OAS[oa_c] \leftarrow o$;
14:        $C_l \leftarrow C_l + S[OAS[oa_c - 1]][o] + P_o$;
15:        $oa_c \leftarrow oa_c + 1$;
16:    else
17:        $U_o \leftarrow U_o \cup \{o\}$;
18:    end if
19:    $O \leftarrow O \backslash \{o\}$;
20: end while
21: Apply *Assignment Operator* to insert the orders from $U_o$ to $OAS$;   ▷ Section 5.3.7
22: Apply *Local Search* to improve $OAS$;                      ▷ Section 5.3.8
23: return $OAS$;
```

added to U_o. This process is repeated till O becomes empty. After that the orders from set U_o are inserted to OAS with the help of *assignment operator* (Section 5.3.7). The first member of the initial population is also generated using the above process except for the fact that Equation (5.2) is used always to select an order from O, i.e., first member is generated by setting $\theta = 1$. Each member of the initial population is passed through a local search to further improve its quality. The pseudo-code of initial solution generation process is given in Algorithm 5.1.

5.3.4 Selection of parent(s)

We have utilized binary tournament selection (BTS) method to choose two parents for crossover and one parent for mutation. The chromosome with better fitness is selected to be a parent with the probability b_t, otherwise the worse one is selected.

5.3.5 Crossover

As OAS problem posses the characteristics of subset selection and permutation problems both, we have developed a special crossover operator for it. Our crossover operator begins with an empty offspring, and then starting from the first position in the offspring, each position in the offspring is considered one-by-one. For each position, one of the two parents is selected uniformly at random, and the order at that position in the selected parent is copied at the same position into the offspring in case this order is not already copied to the offspring at some other position. If the order is already copied to the offspring then this position is left blank and the next position is considered. Once every position is considered, all empty positions are shifted towards the end by moving orders in the offspring towards the start while respecting the relative ordering among them. Now, all unassigned orders will be assigned to this offspring with the help of *Assignment operator* (Section 5.3.7). For example, consider two parents p_1 and p_2 as shown below:

Parent p_1: 3 6 8 10 9 5 7 4 2 1.
Parent p_2: 3 10 9 8 7 4 6 2 1 5.
Position: 1 2 3 4 5 6 7 8 9 10.
Parent Selected: p_1 p_1 p_1 p_2 p_2 p_1 p_2 p_2 p_1 p_1.
Offspring: 3 6 8 – 7 5 – 2 – 1.
Offspring after shifting: 3 6 8 7 5 2 1 – – –.

From the positions 1^{st} to 3^{rd}, parent p_1 is selected and the orders 3^{rd}, 6^{th} and 8^{th} are copied to the offspring at the positions 1^{st}, 2^{nd} and 3^{rd} respectively. At 4^{th} position, parent p_2 is selected, but the order 8^{th} is already included in the child solution. Therefore, this position is left blank and the 5^{th} position is considered. Similarly, at positions 7^{th} and 9^{th}, parents p_2 and p_1 are selected respectively, but the corresponding orders 6^{th} and 2^{nd} are already included in the child solution. Therefore, the next position is considered without including these two orders. Finally, at the last position, parent p_1 is selected and order 1^{st} is included in the child solution. After this, orders already included in the offspring are shifted towards start without disturbing their relative order. Now, the unassigned orders, viz. orders 4, 9 and 10 are assigned with the help of *assignment operator* (Section 5.3.7).

We have also tried UOB crossover for OAS problem, but the crossover operator described above in combination with assignment operator consistently produced better results than UOB crossover. It is pertinent to mention that UOB crossover produces a permutation containing all n orders as offspring and there is no need to use assignment operator, so it is faster than the combination of our crossover operator and assignment operator.

5.3.6 Mutation operator

The purpose of mutation operator is to maintain diversity in the population as well as to prevent the search process from getting stuck into a local optima. The proposed mutation operator starts by randomly deleting mut ($1 \leq mut \leq Del$) number of orders, where Del is some fixed integer which is always less than the number of orders. Like the crossover operator, the remaining orders are shifted towards the beginning while maintaining the relative ordering among orders by shifting the vacant positions towards the end. The deleted orders are re-inserted by the *assignment operator* (Section 5.3.7).

Crossover and mutation are applied in a mutually exclusive manner. With probability π_c crossover is used, otherwise mutation is used.

5.3.7 Assignment operator

Assignment operator assigns leftout orders into the schedule. Our assignment operator is a modified version of the *Construction phase* of [121]. It begins by computing the set of unassigned orders U_o and then an iterative process follows. During each iteration, an order o is selected randomly from the set U_o and inserted at the *best position* in the partial schedule. Here by *best position*, we mean the position at which inserting the job o increases TNR by maximum amount. If no position is available which increases the TNR, then we look for a position at which inserting the job o reduces the completion time of the last order in the schedule while keeping the TNR same. If no such position is available also, then we insert the order o at the end of the partial schedule. This whole process is repeated until U_o becomes empty.

5.3.8 Local search

To further improve the solution quality, each initial solution is passed through a local search. This local search is also applied to the child solution obtained after the application of crossover/-mutation, only when the difference of the fitness of child solution and the fitness of the best solution found so far is less than P_f times the fitness of the best solution found so far. Our local search is a modified version of the local search used in [120], which is based on exchange of orders with subsequent orders. This local search considered only the TNR while exchanging the orders, whereas the modified version also considers the completion time, C_l, of the last order in the schedule in case TNR remains same before and after the exchange. The motivation is to accept an exchange which makes space to accommodate other orders without losing any revenue. The pseudo-code of our local search is presented in Algorithm 5.2.

Algorithm 5.2: The pseudo-code of local search

```
    ▷ $C_{sol}$: Input solution on which local search is to be applied
 1: $C_l \leftarrow$ Completion time of last order in $C_{sol}$;
 2: for ($i \leftarrow 1; i \leq n-1; i++$) do
 3:     for ($j \leftarrow i+1; j \leq n; j++$) do
 4:         Obtain a new solution $N_{sol}$ by exchanging the orders at positions $i$ and $j$ in solution $C_{sol}$;
 5:         $C_{tem} \leftarrow$ Completion time of last order in $N_{sol}$;
 6:         if (($TNR(N_{sol}) > TNR(C_{sol})$) or ($TNR(N_{sol}) = TNR(C_{sol})$ and $C_{tem} < C_l$)) then
 7:             $C_{sol} \leftarrow N_{sol}$;
 8:             $C_l \leftarrow C_{tem}$;
 9:         end if
10:     end for
11: end for
12: return $C_{sol}$;
```

The pseudo-code of our hybrid SSGA approach for the OAS problem is given in Algorithm 5.3, where *BTS()*, *Crossover()*, *Mutation()*, *Assignment_Operator()* and *Local_Search()* functions return the solution as per their description in Section 5.3.4, Section 5.3.5, Section 5.3.6, Section 5.3.7 and Section 5.3.8 respectively. Hereafter, our hybrid steady-state genetic algorithm based approach for OAS problem will be referred to as HSSGA.

5.4 Hybrid EA/G approach

Our hybrid EA/G approach uses the same solution encoding, fitness function, assignment operator and local search as used by HSSGA. Other salient features of our hybrid EA/G approach are described in subsequent subsections. Hereafter, our hybrid EA/G approach will be referred to as EA/G-LS.

5.4.1 Initialization and update of the probability matrix

Like previous chapters, to model the distribution of promising solutions in the search space, a univariate marginal distribution (UMD) model is used. However, unlike previous chapters, a

Algorithm 5.3: The pseudo-code of HSSGA approach

```
1:  Generate initial population of solutions;                        ▷ Section 5.3.3
2:  $Sol_b \leftarrow$ Best solution of initial population;
3:  $f_{Sol_b} \leftarrow fitness(Sol_b)$;
4:  while (termination condition is not satisfied) do
5:      Generate a random number $r01$ such that $0 \leq r01 \leq 1$;
6:      if ($r01 \leq \pi_c$) then
7:          $p_1 \leftarrow BTS()$;                                      ▷ Section 5.3.4
8:          $p_2 \leftarrow BTS()$;                                      ▷ Section 5.3.4
9:          $C_s \leftarrow Crossover(p_1, p_2)$;                        ▷ Section 5.3.5
10:     else
11:         $p \leftarrow BTS()$;                                        ▷ Section 5.3.4
12:         $C_s \leftarrow Mutation(p)$;                                ▷ Section 5.3.6
13:     end if
14:     $C_s \leftarrow Assignment\_Operator(C_s)$;                      ▷ Section 5.3.7
15:     $f_{C_s} \leftarrow fitness(C_s)$;
16:     if (($f_{Sol_b} - f_{C_s}$) < ($f_{Sol_b} \times P_f$)) then
17:         $C_s \leftarrow Local\_Search(C_s)$;                         ▷ Section 5.3.8
18:         $f_{C_s} \leftarrow fitness(C_s)$;
19:     end if
20:     if ($C_s$ is different from current population members) then
21:         Replace worst member of population with $C_s$;
22:         if ($f_{C_s} > f_{Sol_b}$) then
23:             $Sol_b \leftarrow C_s$;
24:             $f_{Sol_b} \leftarrow f_{C_s}$;
25:         end if
26:     end if
27: end while
28: return $Sol_b$;
```

$n \times n$ probability matrix $P(g)$ models the distribution of promising solutions at iteration g. The value p_{ij} gives the probability of order i at the position j in any schedule. Salhi *et al.* [123] used this matrix representation in the context of a flowshop scheduling problem.

$$P(g) = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

This probability matrix is initialised using N_s initial solutions which are generated in the same manner as all initial population members except the first one in genetic algorithm. The pseudo-code for initializing the probability matrix is given in Algorithm 5.4.

Algorithm 5.4: The pseudo-code for initializing the probability matrix $P(g)$

```
1: Compute $n_{ij} \leftarrow$ number of initial solutions containing order $i$ at position $j$, $\forall i \in O$, $j = 1, 2, \ldots, n$;
2: Compute $p_{ij} \leftarrow \frac{n_{ij}}{N_s}, \forall i \in O, j = 1, 2, \ldots, n$;
```

Once the $n \times n$ probability matrix $P(g)$ is formed, a n element vector $B_p(g)$ is computed which contains for each order i, the position j_i which has maximum probability, i.e.,

$$B_p(g) = \begin{bmatrix} j_1 & j_2 & \cdots & j_i & \cdots & j_n \end{bmatrix}$$

where each j_i is computed as follows:

$$j_i \leftarrow \arg\max_{k=1,2,\ldots,n} p_{ik} \tag{5.3}$$

At each generation g, a parent set *parent(g)* is formed by selecting the best L solution from the current population *pop(g)* like the previous chapter. Once *parent(g)* is formed, it is used for updating the probability vector $P(g)$. The pseudo-code for updating the probability vector is given in Algorithm 5.5, where $\lambda \in (0, 1]$ is the learning rate.

Algorithm 5.5: The pseudo-code for updating a probability vector p in generation g

```
1: Compute $n_{ij} \leftarrow$ number of solutions in parent(g) containing order $i$ at position $j$, $\forall i \in O$, $j = 1, 2, \ldots, n$;
2: Compute $p_{ij} \leftarrow (1 - \lambda)p_{ij} + \lambda \frac{n_{ij}}{L}, \forall i \in O, j = 1, 2, \ldots, n$;
```

5.4.2 Guided mutation (GM) operator

As mentioned already in the beginning of this section, *GM* operator uses both the global statistical information about the search space (stored in the form of probability matrix $P(g)$) and the location information of the parent solution to generate a new offspring. Our *GM* operator is applied on a set of M best solutions in the current population *pop*(g). The proposed *GM* operator works as follows: On the set of M best solution $\{s_1, s_2, \ldots, s_M\}$, *GM* operator is applied only once on each $s_t, t = 1, 2, \ldots, M$ to generate $\{c_1, c_2, \ldots, c_M\}$ offspring. The pseudo-code of *GM* operator is presented in Algorithm 5.6, where $\beta \in$ [0, 1] is an adjustable parameter and NS is a new offspring constructed through *GM* operator. Positioning of orders in NS are either sampled randomly from the probability matrix $P(g)$ or directly copied from the solution s_t in $pop(g)$. To sample the positioning of an order i through probability matrix, we first access $B_p(g)$ to get j_i and then make use of entry p_{ij_i} in $P(g)$. In case of sampling from probability vector $P(g)$, the order i is copied to position j_i only when the position j_i in the partial schedule NS is not occupied, otherwise that order is added into the set U_o. Whereas in case an order is to be directly copied from the solution s_t, it is copied from solution s_t only when the corresponding position of that order in NS is not occupied. In other words, suppose the position index of the order i in the solution s_t is l ($1 \leq l \leq n$). Now, the order i will be copied at the position l in NS if that position is not occupied, otherwise add the order i into set U_o. This process is repeated till all the orders are considered. After this the orders in the set U_o are assigned to NS using the assignment operator (Section 5.3.7).

Algorithm 5.6: The pseudo-code of generating a solution through *GM* operator

```
 1: for ($i \leftarrow 1; i \leq n; i++$) do
 2:     $NS[i] \leftarrow 0$;
 3: end for
 4: $U_o \leftarrow \emptyset$;
 5: for (each order $i \in O$) do
 6:     Generate a random number $r_1$ such that $0 \leq r_1 \leq 1$;
 7:     if ($r_1 < \beta$) then
 8:         Generate a random number $r_2$ such that $0 \leq r_2 \leq 1$;
 9:         if ($r_2 < p_{ij_i}$) then
10:             if ($NS[j_i]$=0) then
11:                 $NS[j_i] \leftarrow i$;
12:             else
13:                 $U_o \leftarrow U_o \cup i$;
14:             end if
15:         end if
16:     else
17:         if ($NS[s_t[j_i]]$=0) then
18:             $NS[s_t[j_i]] \leftarrow i$;
19:         else
20:             $U_o \leftarrow U_o \cup i$;
21:         end if
22:     end if
23: end for
24: $NS \leftarrow Assignment\_Operator(NS)$;                ▷ Section 5.3.7
25: return $NS$;
```

5.4.3 Other features

Same local search as used in HSSGA (Section 5.3.8) is applied to each solution obtained through *GM* operator only when the difference of the fitness of the solution under consideration and the fitness of the best solution found so far is less than P_f times the fitness of the best solution found so far.

Like the approach of previous chapter, *parent(g)* is formed by using the best L solutions from *pop(g)*, and M new solutions are produced through applying *guided mutation* operator once on each of the best M solutions in *pop(g)* in each generation. The best $N_s - M$ solutions of *pop(g)* along with M newly produced children constitute *pop(g+1)*. Also like the approach of previous chapter, if the best solution of the population does not improve over R_I generations, we reinitialize the population and probability matrix. For this, we create a new population of N_s solutions. The best solution obtained since the beginning of the algorithm is included in this population as its first member. Remaining $N_s - 1$ members of this population are generated in the same manner as non-first members of initial population are generated in genetic algorithm. Then we make use of Algorithm 5.4 to re-initialize the probability matrix.

The pseudo-code of our EA/G-LS approach is given in Algorithm 5.7.

Algorithm 5.7: EA/G-LS Approach for OAS

```
1: At generation $g \leftarrow 0$, an initial population pop(g) consisting of $N_s$ solutions, is generated randomly;
2: Initialize the probability vector $p$ for all orders using Algorithm 5.4;
3: Select best $L$ solutions from pop(g) to form a parent set parent(g), and then update the probability
   vector $p$ using Algorithm 5.5;
4: Apply the GM operator once on each of the $M$ best solutions in pop(g) in order to generate $M$ new
   solutions. The assignment operator is applied to each generated solution, if necessary, and then
   the local search is applied to each generated solution only in case solution under consideration is
   sufficiently close to the best solution found so far (See Section 5.4.2). Add all $M$ newly generated
   solutions along with $N_s - M$ best solutions in pop(g) to form pop(g+1). If the stopping condition is
   met, return the solution with maximum $TNR$ found so far ;
5: $g \leftarrow g + 1$ ;
6: If the best solution of the population did not improve over $R_I$ generations, then reinitialize entire
   pop(g) except for the best solution, and then go to step 2 ;
7: Go to step 3 ;
```

5.5 Computational results

The proposed approaches have been coded in C language and executed on a Linux based system with 3.10GHz `Intel core i5-2400` processor and 4GB RAM. `gcc 4.6.3-2` compiler with `O3` flag has been used to compile the C programs, and their performances are examined on the same set of test instances as used in [5, 120]. These test instances were generated by Cesaret *et al.* [5] and can be downloaded from `http://home.ku.edu.tr/~coguz/Research/Dataset_OAS.zip`. These test instance were generated considering the three factors: number of orders (n), the tardiness factor (τ) and the due date range (R). Number of orders is set to 10, 15, 20, 25, 50 and 100. The values of τ and R are set to 0.1, 0.3, 0.5, 0.7 and 0.9. For each combination of these three factors, 10 instances were generated. Therefore, total number of instances is 1500 ($6\times5\times5\times10$). The instances were generated in the following way: A maximum revenue gain, E_i and a processing time, P_i, for each order is generated from the uniform distribution [0, 20]. A release date r_i is generated with uniform distribution in the interval of [0, τP_T], where P_T is the total processing time of all orders. The sequence-dependent setup time S_{ji} ($j, i = 1, 2, \ldots, n.j \neq i$) between orders i and j is generated with uniform distribution in the interval of [0, 10]. A due date d_i = r_i + $\max_{j=0,1,\ldots,n} S_{ji}$ + max{slack, P_i} of an order ($i = 1, 2, \ldots, n$) is generated in the interval of [P_T(1-τ-R/2), P_T(1-τ+R/2)]. A dead line of an order i is defined as $\bar{d}_i$=d_i+RP_i. A non-integer tardiness weight of an order i is calculated as w_i=E_i/($\bar{d}_i$-d_i).

Proposed HSSGA parameters: The proposed hybrid steady-state genetic algorithm starts with the population size Pop_s=60, b_t=0.50 is used to choose a better member through the binary tournament selection (BTS), probability for crossover C_o=0.80 (hence mutation is used with probability 0.20 as crossover and mutation are used mutually exclusively). Del=6 is used to delete orders in mutation operator. P_f is set to 0.015, and θ=0.65 is used in initial solution generation. The HSSGA is allowed to execute till the best solution does not improve over 5000 generations and it has executed for at least 10000 generations.

Proposed EA/G-LS parameters: For EA/G-LS, we have used the population size of N_s = 40, $L = 20$, M=20 to generate new offspring through the *guided mutation* operator. The value of β is set to 0.10. λ=0.35 is used to update the probability vector of the promising solutions. The value of θ=0.65 is used in the initial solution generation. P_f is set to 0.015. If the best solution does not improve over R_I=50 generations, the entire population, except the best solution, and the probability matrix are reinitialized. The EA/G-LS is allowed to execute till the best solution does not improve over 100 generations and it has executed for at least 200 generations.

All the above mentioned parameters for EA/G-LS and HSSGA are set after extensively experimenting with different values. These selected parameter values produce good results, though they may not be optimal for all instances.

We have compared our HSSGA and EA/G-LS approaches with Tabu Search (TS) [5] and Artificial bee colony algorithm (ABC) [120] based approaches. Like TS and ABC approaches, our approaches have been executed on each instance once. The criteria to evaluate the performance of various approaches is the percentage deviation of TNR obtained by an approach from the upper bound (UB) on each of the 1500 instances. The percentage deviation on an instance is calculated as follows:

$$\% \textit{ Deviation from } UB = \frac{(UB - TNR)}{UB} \times 100\%$$

Where TNR indicates the TNR obtained by the approach in consideration and UB is the upper bound proposed in Cesaret *et al.* [5]. We have reported the maximum, average and minimum percentage deviations from UB of various approaches on each group of 10 instances with same n, τ and R. Tables 5.1 to 5.6 report the results obtained by TS, ABC, HSSGA and EA/G-LS approaches for instances with 10, 15, 20, 25, 50 and 100 orders respectively. These tables also report the number of proven optimal solutions found over each group of 10 instances by each of the 4 methods and their average execution times. Detailed instance by instance results for TS and ABC approaches have been obtained from authors of [120] through personal communication. We have made use of these detailed results to report the percentage deviations of ABC and TS approaches with an increased precision in these tables than those reported in [5, 120]. TS approach was executed on 3.0 GHz Intel Xeon Processor based system with 4GB of RAM and ABC approach was executed on 3.0 GHz Intel Pentium 4 Processor based system with 4GB of RAM. As TS and ABC approaches were executed on systems whose configuration is different from the system used to execute HSSGA and EA/G-LS approaches, and therefore, execution times of different approaches in these tables can not be compared precisely.

Tables 5.1 to 5.6 show that the HSSGA and EA/G-LS solve more instances optimally in comparison to TS and ABC approaches. For problem size 10, out of 250 test instances TS, ABC, HSSGA and EA/G-LS found 188, 247, 249 and 247 optimal solutions respectively (Table 5.1). For problem size 15, out of 250 test instances TS, ABC, HSSGA and EA/G-LS found 91, 101, 121 and 118 optimal solutions respectively(Table 5.2). Out of 250 test instances with size 20, TS, ABC, HSSGA and EA/G-LS found 47, 58, 76 and 75 optimal solutions respectively(Table 5.3). Table 5.4 shows that for problem size 25, out of 250 test instances TS, ABC, HSSGA and EA/G-LS found 34, 43, 55 and 54 optimal solutions respectively. Table 5.5 shows that for problem size 50, out of 250 test instances TS, ABC, HSSGA and EA/G-LS found 4, 18, 28 and 31 optimal solutions respectively . As per Table 5.6, for problem size 100, out of 250 test instances TS, ABC, HSSGA and EA/G-LS found 5, 21, 30 and 31 optimal solutions respectively.

From these tables, it can be observed that most of the *Max.*, *Avg.* and *Min.* % deviation from *UB* of the proposed HSSGA approach are less than or equal to TS and ABC approaches. Similar observations can be made about EA/G-LS. As mentioned already that systems used to execute TS and ABC approaches had different configuration than the system on which HSSGA and EA/G-LS have been executed, and therefore, execution times of HSSGA and EA/G-LS can not be compared precisely with those of TS and ABC. However, HSSGA and EA/G-LS approaches are definitely slower than TS and ABC on most of the instances.

As far as comparison between our two proposed approaches are concerned, we can observe from Tables 5.1 to 5.6 that on most of the instance groups HSSGA and EA/G-LS obtained results which are very close to each other. But, overall HSSGA shows the dominance over EA/G-LS approach in terms of solution quality. From these tables we can see that the "Total Average" returned by HSSGA is better than EA/G-LS in terms of *Max.*, *Avg.* and *Min.* % deviation from UB, except on instances with 50 orders where the minimum (*Min.*) % deviation from UB is worse in comparison to EA/G-LS. If we compare the number of optimal solutions, HSSGA found more number of optimal solutions than EA/G-LS on all instance groups (except the groups containing 50 orders). However, the computational time taken by HSSGA is always greater than EA/G-LS on all instance groups.

Table 5.1: Performance of various approaches on instances with 10 orders

n=10		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	2.61	0.84	0.09	0.10	0.09	0.09	0.10	0.09	0.09	1.68	0.25	0.09	6	10	10	9	0.00	0.02	0.02	0.00
	0.3	7.89	1.31	0.09	2.33	0.32	0.09	0.10	0.10	0.09	0.10	0.10	0.09	5	9	10	10	0.00	0.02	0.02	0.00
	0.5	7.50	0.84	0.00	6.00	0.69	0.00	6.00	0.67	0.00	6.00	0.67	0.00	8	9	9	9	0.00	0.02	0.02	0.00
	0.7	2.68	0.35	0.00	0.10	0.07	0.00	0.10	0.06	0.00	0.10	0.06	0.00	8	10	10	10	0.01	0.03	0.02	0.00
	0.9	1.24	0.24	0.00	0.09	0.02	0.00	0.09	0.02	0.00	1.24	0.13	0.00	7	10	10	9	0.00	0.02	0.02	0.00
0.3	0.1	1.68	0.25	0.08	0.10	0.09	0.08	0.10	0.09	0.08	0.10	0.09	0.08	9	10	10	10	0.00	0.02	0.02	0.00
	0.3	2.13	0.76	0.09	0.12	0.09	0.09	0.12	0.09	0.09	0.12	0.09	0.09	6	10	10	10	0.00	0.02	0.02	0.00
	0.5	7.89	2.11	0.09	1.56	0.23	0.09	0.10	0.08	0.09	0.10	0.08	0.09	6	9	10	10	0.01	0.03	0.02	0.00
	0.7	2.39	0.63	0.08	0.10	0.09	0.08	0.10	0.09	0.08	0.10	0.09	0.08	6	10	10	10	0.00	0.02	0.02	0.00
	0.9	3.39	0.86	0.08	0.09	0.07	0.00	0.10	0.07	0.00	0.10	0.07	0.00	4	10	10	10	0.00	0.03	0.03	0.00
0.5	0.1	6.06	0.65	0.00	0.09	0.06	0.00	0.09	0.06	0.00	0.09	0.06	0.00	9	10	10	10	0.00	0.03	0.02	0.00
	0.3	5.07	0.95	0.01	0.10	0.07	0.01	0.10	0.07	0.01	0.10	0.07	0.01	8	10	10	10	0.00	0.02	0.02	0.00
	0.5	0.27	0.08	0.00	0.10	0.06	0.00	0.10	0.06	0.00	0.10	0.06	0.00	9	10	10	10	0.00	0.02	0.02	0.00
	0.7	2.15	0.27	0.00	0.10	0.07	0.00	0.10	0.07	0.00	0.10	0.07	0.00	4	10	10	10	0.00	0.03	0.02	0.00
	0.9	1.62	0.23	0.00	0.10	0.08	0.00	0.10	0.08	0.00	0.10	0.08	0.00	7	10	10	10	0.00	0.03	0.02	0.00
0.7	0.1	4.26	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9	10	10	10	0.00	0.03	0.02	0.00
	0.3	0.01	0.01	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	10	10	10	10	0.00	0.02	0.02	0.00
	0.5	1.26	0.13	0.00	0.05	0.00	0.00	0.04	0.00	0.00	0.04	0.00	0.00	9	10	10	10	0.00	0.03	0.02	0.00
	0.7	0.08	0.01	0.00	0.08	0.01	0.00	0.08	0.01	0.00	0.08	0.01	0.00	8	10	10	10	0.00	0.03	0.02	0.00
	0.9	3.60	0.37	0.00	0.06	0.01	0.00	0.07	0.02	0.00	0.07	0.02	0.00	6	10	10	10	0.00	0.03	0.02	0.00
0.9	0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10	10	10	10	0.00	0.03	0.02	0.00
	0.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9	10	10	10	0.00	0.03	0.02	0.00
	0.5	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	8	10	10	10	0.00	0.03	0.02	0.00
	0.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9	10	10	10	0.00	0.03	0.02	0.00
	0.9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8	10	10	10	0.00	0.03	0.02	0.00
Total Average		2.55	0.45	0.02	0.45	0.08	0.02	0.30	0.07	0.02	0.41	0.08	0.02	7.52	9.88	9.96	9.88	0.00	0.02	0.02	0.00
Total														188	247	249	247				

Table 5.2: Performance of various approaches on instances with 15 orders

n=15		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	3.57	2.70	1.19	3.55	2.34	0.00	3.55	1.96	0.00	3.55	2.07	0.00	0	1	1	1	0.04	0.04	0.06	0.02
	0.3	13.73	3.19	0.00	7.19	3.01	1.20	6.21	2.76	0.00	6.54	2.85	0.00	1	0	1	1	0.01	0.03	0.06	0.02
	0.5	3.65	1.71	0.00	3.65	1.59	0.00	3.65	1.46	0.00	3.65	1.49	0.00	1	1	3	3	0.01	0.04	0.07	0.02
	0.7	4.45	1.77	0.00	3.86	0.99	0.00	3.87	1.01	0.00	4.45	1.33	0.00	1	5	6	3	0.01	0.03	0.07	0.02
	0.9	6.08	1.32	0.00	6.08	1.16	0.00	6.08	1.16	0.00	6.08	1.16	0.00	4	5	5	5	0.01	0.04	0.06	0.02
0.3	0.1	8.05	4.42	3.36	5.75	3.21	0.00	5.75	2.77	0.00	6.90	3.06	0.00	0	1	1	1	0.01	0.04	0.06	0.02
	0.3	11.33	4.92	1.57	11.33	3.94	1.08	11.33	3.73	0.00	11.33	3.73	0.00	0	0	1	1	0.01	0.04	0.05	0.02
	0.5	8.04	4.61	1.57	7.76	4.05	1.08	6.88	3.88	0.00	7.76	4.05	0.00	0	0	0	0	0.01	0.03	0.07	0.02
	0.7	7.57	3.97	1.55	7.57	2.98	1.17	7.57	2.91	1.17	7.57	3.03	1.17	0	0	0	0	0.01	0.04	0.07	0.02
	0.9	7.10	3.79	0.00	7.10	2.94	0.00	7.10	2.94	0.00	7.10	2.94	0.00	2	2	3	3	0.01	0.04	0.07	0.02
0.5	0.1	12.56	7.48	2.47	12.56	7.00	2.47	12.56	7.00	2.47	12.56	7.00	2.47	0	0	0	0	0.01	0.04	0.05	0.02
	0.3	13.73	7.68	4.26	13.73	7.51	4.27	13.73	6.89	3.05	13.73	6.89	3.05	0	0	0	0	0.01	0.05	0.05	0.02
	0.5	15.29	9.35	5.81	15.29	9.17	4.07	15.29	9.10	3.88	15.29	9.11	4.07	0	0	0	0	0.01	0.04	0.06	0.02
	0.7	11.00	6.47	1.16	11.00	6.09	0.09	11.00	6.04	0.09	11.00	6.04	0.09	0	0	1	1	0.01	0.05	0.07	0.02
	0.9	18.49	6.36	0.10	18.39	6.09	0.10	18.39	6.09	0.10	18.39	6.09	0.10	1	1	1	1	0.01	0.05	0.06	0.02
0.7	0.1	4.06	0.70	0.00	2.21	0.30	0.05	0.10	0.08	0.00	0.10	0.08	0.00	7	9	10	10	0.01	0.04	0.05	0.02
	0.3	2.78	0.57	0.07	0.10	0.09	0.06	1.25	0.21	0.07	0.10	0.09	0.06	7	10	9	10	0.01	0.05	0.05	0.01
	0.5	2.40	0.43	0.07	0.10	0.08	0.00	0.13	0.09	0.07	0.10	0.08	0.00	7	10	10	10	0.01	0.05	0.06	0.02
	0.7	8.28	2.05	0.08	8.28	1.03	0.03	8.28	0.90	0.03	8.28	1.03	0.03	3	6	9	8	0.01	0.05	0.06	0.02
	0.9	0.27	0.11	0.08	0.10	0.08	-0.01	0.10	0.08	0.00	0.10	0.08	0.00	9	8	10	10	0.01	0.05	0.06	0.01
0.9	0.1	2.70	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9	10	10	10	0.01	0.05	0.05	0.01
	0.3	4.59	0.47	0.00	0.08	0.01	0.00	0.08	0.01	0.00	0.08	0.01	0.00	9	9	10	10	0.01	0.06	0.05	0.01
	0.5	0.08	0.03	0.00	0.08	0.03	0.00	0.08	0.03	0.00	0.08	0.03	0.00	10	8	10	10	0.01	0.05	0.06	0.02
	0.7	0.10	0.03	0.00	0.09	0.03	0.00	0.09	0.03	0.00	0.09	0.03	0.00	10	7	10	10	0.01	0.05	0.05	0.01
	0.9	0.10	0.05	0.00	0.10	0.05	0.00	0.10	0.05	0.00	0.10	0.05	0.00	10	8	10	10	0.01	0.05	0.06	0.01
Total Average		6.80	2.98	0.93	5.84	2.55	0.63	5.73	2.45	0.44	5.80	2.49	0.44	3.64	4.04	4.84	4.72	0.01	0.04	0.06	0.02
Total														91	101	121	118				

Table 5.3: Performance of various approaches on instances with 20 orders

n=20		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	4.57	3.16	1.45	4.11	1.90	0.00	2.74	1.62	0.00	4.11	1.95	0.00	0	2	2	1	0.10	0.06	0.12	0.06
	0.3	9.16	3.27	0.82	2.34	1.70	0.82	2.34	1.66	0.82	2.34	1.66	0.82	0	0	0	0	0.04	0.06	0.11	0.06
	0.5	3.29	1.91	0.82	3.04	1.41	0.00	3.04	1.43	0.73	3.04	1.49	0.77	0	1	0	0	0.03	0.05	0.13	0.05
	0.7	1.92	0.64	0.00	1.32	0.46	0.00	1.28	0.33	0.00	1.28	0.33	0.00	5	6	7	7	0.04	0.05	0.13	0.06
	0.9	3.91	1.16	0.00	3.04	0.54	0.00	2.61	0.40	0.00	3.04	0.41	0.00	3	7	8	8	0.03	0.07	0.14	0.05
0.3	0.1	7.65	4.65	1.65	7.45	3.82	1.65	7.45	3.23	0.00	6.67	3.45	1.65	0	0	1	0	0.03	0.07	0.10	0.05
	0.3	6.22	4.75	3.17	6.22	3.69	1.44	5.33	3.18	1.44	5.33	3.45	1.59	0	0	0	0	0.04	0.07	0.11	0.06
	0.5	7.61	5.21	3.17	5.77	3.70	1.44	7.00	3.71	1.44	7.00	3.93	1.59	0	0	0	0	0.03	0.07	0.12	0.06
	0.7	6.96	3.27	1.29	6.96	2.47	0.62	6.96	2.38	0.55	6.96	2.46	0.55	0	1	0	0	0.04	0.07	0.13	0.06
	0.9	4.69	2.13	0.00	4.69	1.68	0.00	4.69	1.49	0.00	4.69	1.45	0.00	2	2	3	4	0.04	0.08	0.13	0.06
0.5	0.1	7.69	5.75	3.61	5.91	4.60	2.06	5.53	4.17	2.06	5.91	4.33	2.06	0	0	0	0	0.03	0.08	0.10	0.05
	0.3	9.16	6.32	2.83	8.74	5.43	1.42	9.16	5.40	1.42	7.65	5.10	1.42	0	0	0	0	0.07	0.07	0.10	0.05
	0.5	10.09	6.91	3.09	9.21	5.49	2.06	9.21	5.49	2.06	9.21	5.53	2.06	0	0	0	0	0.04	0.09	0.10	0.06
	0.7	10.70	5.32	1.21	10.70	4.65	1.16	10.70	4.74	1.16	10.70	4.85	1.16	0	0	0	0	0.03	0.09	0.13	0.07
	0.9	10.57	6.25	0.00	10.57	5.54	0.00	10.57	5.48	0.00	10.57	5.55	0.00	1	1	1	1	0.05	0.07	0.12	0.06
0.7	0.1	14.85	9.50	5.47	13.97	8.75	5.47	13.54	8.57	5.47	13.97	8.27	1.99	0	0	0	0	0.05	0.09	0.10	0.05
	0.3	14.29	9.21	5.94	12.61	8.45	5.94	11.76	8.23	5.94	11.76	8.44	5.94	0	0	0	0	0.04	0.10	0.11	0.05
	0.5	20.18	9.13	0.09	20.18	8.72	0.09	20.18	8.60	0.09	20.18	8.71	0.09	2	1	2	2	0.04	0.09	0.10	0.05
	0.7	12.17	6.71	0.09	12.17	6.46	0.09	12.17	6.51	0.09	12.17	6.51	0.09	3	2	3	3	0.07	0.10	0.11	0.05
	0.9	12.83	7.90	0.00	12.50	7.11	0.00	12.50	7.11	0.00	12.50	7.11	0.00	1	0	1	1	0.04	0.08	0.10	0.05
0.9	0.1	1.21	0.12	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0	10	10	10	0.06	0.11	0.09	0.02
	0.3	0.74	0.12	0.00	0.74	0.11	0.00	0.09	0.04	0.00	0.09	0.04	0.00	8	8	10	10	0.03	0.12	0.10	0.03
	0.5	2.33	0.48	0.00	0.10	0.07	-0.01	0.10	0.07	0.00	0.10	0.07	0.00	8	7	10	10	0.03	0.11	0.10	0.03
	0.7	1.48	0.44	0.00	0.10	0.08	0.00	0.10	0.08	0.00	0.10	0.08	0.00	8	6	10	10	0.03	0.11	0.12	0.04
	0.9	13.21	1.94	0.00	13.21	1.45	0.00	13.21	1.45	0.00	13.21	1.45	0.00	6	4	8	8	0.03	0.10	0.11	0.05
Total Average		7.90	4.25	1.39	7.03	3.53	0.97	6.89	3.41	0.93	6.90	3.46	0.87	1.88	2.32	3.04	3.00	0.04	0.08	0.11	0.05
Total														47	58	76	75				

Table 5.4: Performance of various approaches on instances with 25 orders

n=25		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	6.28	4.02	1.09	3.85	2.70	1.45	3.16	2.08	1.09	3.16	2.16	1.09	0	0	0	0	0.08	0.09	0.22	0.13
	0.3	9.24	3.31	1.61	5.78	2.18	0.69	3.20	1.66	0.69	3.20	1.58	0.69	0	0	0	0	0.06	0.09	0.22	0.14
	0.5	4.29	2.25	0.87	2.45	1.38	0.00	1.90	0.90	0.00	1.77	0.91	0.00	0	1	2	2	0.06	0.08	0.24	0.13
	0.7	4.07	1.19	0.00	2.68	0.63	0.00	1.67	0.39	0.00	1.67	0.39	0.00	4	6	7	7	0.06	0.10	0.24	0.12
	0.9	2.11	0.94	0.00	2.09	0.35	0.00	2.09	0.35	0.00	2.09	0.35	0.00	4	7	7	7	0.05	0.10	0.24	0.11
0.3	0.1	5.24	3.66	1.71	5.24	3.11	1.21	4.55	2.48	1.21	4.55	2.44	1.21	0	0	0	0	0.10	0.11	0.21	0.14
	0.3	7.33	4.89	2.61	6.04	3.37	1.12	4.62	3.12	1.12	6.04	3.46	2.15	0	0	0	0	0.09	0.10	0.21	0.13
	0.5	5.59	2.99	2.61	5.24	2.40	1.12	3.50	1.52	1.12	3.50	1.85	2.15	0	0	0	0	0.08	0.12	0.29	0.13
	0.7	5.69	2.48	1.06	5.32	1.97	0.88	4.61	1.61	0.00	5.32	1.90	0.00	0	0	1	1	0.08	0.12	0.30	0.14
	0.9	4.05	1.85	0.00	3.38	1.27	0.00	2.82	1.05	0.00	2.82	1.10	0.00	2	2	3	2	0.07	0.12	0.26	0.14
0.5	0.1	7.37	6.35	2.80	6.76	4.88	2.80	6.76	4.17	1.68	6.76	4.11	1.68	0	0	0	0	0.07	0.12	0.20	0.12
	0.3	9.24	5.49	3.17	7.29	4.49	2.85	7.29	4.22	1.82	7.29	4.12	1.82	0	0	0	0	0.08	0.12	0.22	0.14
	0.5	7.89	5.28	1.94	6.02	4.48	1.94	6.08	4.28	0.97	6.08	4.25	0.97	0	0	0	0	0.09	0.16	0.21	0.15
	0.7	10.89	5.70	1.60	7.17	4.20	0.64	7.17	4.16	0.64	7.17	4.19	0.64	0	3	0	0	0.08	0.19	0.21	0.14
	0.9	7.45	4.32	1.03	7.00	3.37	1.03	7.00	3.05	1.03	6.79	3.13	1.03	0	0	0	0	0.08	0.11	0.21	0.13
0.7	0.1	18.24	9.28	3.24	15.88	7.72	0.93	14.86	7.43	0.93	16.22	7.53	0.93	0	0	0	0	0.06	0.14	0.19	0.12
	0.3	14.11	9.74	7.22	12.40	8.64	5.05	12.40	8.49	5.78	12.50	8.53	5.78	0	0	0	0	0.08	0.11	0.19	0.12
	0.5	14.75	11.51	6.61	14.03	10.45	5.37	14.03	10.05	5.37	14.03	10.17	5.37	0	0	0	0	0.08	0.18	0.19	0.12
	0.7	13.75	7.86	1.69	12.08	6.90	1.69	12.08	6.56	1.58	12.08	6.47	1.58	0	0	0	0	0.07	0.15	0.21	0.12
	0.9	14.64	10.24	2.55	13.64	8.12	0.01	13.64	8.12	0.01	13.64	8.27	0.01	0	1	1	1	0.08	0.15	0.20	0.12
															8						
0.9	0.1	6.40	1.28	0.00	4.93	0.59	0.00	4.93	0.49	0.00	4.93	0.49	0.00	5	6	9	9	0.06	0.18	0.18	0.07
	0.3	0.44	0.07	0.00	1.48	0.16	0.00	0.01	0.00	0.00	0.01	0.00	0.00	8	3	10	10	0.06	0.17	0.17	0.07
	0.5	12.43	3.51	0.00	12.30	2.58	0.00	12.30	2.49	0.00	12.30	2.49	0.00	3	4	6	6	0.07	0.19	0.17	0.08
	0.7	24.81	7.53	0.00	20.99	6.81	0.00	20.99	6.67	0.00	20.99	6.67	0.00	4	2	5	5	0.08	0.16	0.18	0.09
	0.9	22.27	6.77	0.00	19.33	6.09	0.00	19.10	5.99	0.00	19.10	5.99	0.00	4		4	4	0.09	0.17	0.21	0.10
Total Average		9.54	4.90	1.74	8.13	3.95	1.15	7.63	3.65	1.00	7.76	3.70	1.08	1.36	1.72	2.20	2.16	0.07	0.13	0.21	0.12
Total														34	43	55	54				

Table 5.5: Performance of various approaches on instances with 50 orders

n=50		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	3.00	2.23	1.20	2.81	1.89	1.20	1.69	1.18	0.51	1.47	1.08	0.51	0	0	0	0	1.19	0.68	1.85	1.35
	0.3	3.92	2.44	1.39	2.33	1.77	1.06	1.57	1.12	0.71	1.57	1.10	0.71	0	0	0	0	0.98	0.60	2.30	1.52
	0.5	2.10	1.61	1.12	2.03	0.96	0.34	0.84	0.43	0.00	0.95	0.45	0.00	0	0	1	2	0.89	0.80	2.06	1.49
	0.7	16.39	2.45	0.00	16.39	1.90	0.00	16.39	1.73	0.00	16.39	1.66	0.00	1	4	7	8	0.58	0.73	1.93	1.15
	0.9	1.90	0.54	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2	10	10	10	0.46	1.14	1.89	1.04
0.3	0.1	3.48	2.63	1.58	3.77	2.59	1.63	2.79	1.80	1.02	2.86	1.93	1.22	0	0	0	0	1.61	0.69	1.71	1.48
	0.3	5.05	3.54	2.66	4.49	2.98	1.94	2.66	2.06	1.55	3.05	2.18	1.54	0	0	0	0	1.49	0.44	2.11	1.46
	0.5	4.84	2.56	2.66	3.88	1.88	1.94	3.88	1.51	1.55	3.68	1.53	1.54	0	0	0	0	1.16	0.80	2.07	1.42
	0.7	3.44	1.40	0.00	1.56	0.75	0.00	1.03	0.39	0.00	1.04	0.35	0.00	1	1	3	4	0.85	0.72	1.87	1.45
	0.9	2.85	1.35	0.40	1.83	0.50	0.00	1.21	0.29	0.00	1.02	0.23	0.00	0	3	6	6	0.68	1.01	2.14	1.40
0.5	0.1	5.34	4.15	2.82	4.17	3.07	1.71	3.20	2.20	1.11	3.38	2.23	1.11	0	0	0	0	1.36	0.64	1.88	1.42
	0.3	8.02	5.69	3.08	6.11	4.45	3.08	5.15	3.43	2.26	5.73	3.53	2.26	0	0	0	0	1.18	0.45	2.10	1.37
	0.5	8.01	4.42	1.55	6.82	3.78	2.07	6.33	3.04	1.03	7.14	3.17	1.03	0	0	0	0	1.35	0.77	2.37	1.66
	0.7	5.16	3.10	1.51	4.24	2.28	0.76	3.87	2.00	0.36	4.24	2.03	0.38	0	0	0	0	1.22	1.01	2.85	1.51
	0.9	5.83	3.17	2.41	4.04	1.71	3.61	4.31	1.54	0.00	4.04	1.57	-4.02	0	0	1	1	1.16	0.72	2.27	1.41
0.7	0.1	8.61	6.61	3.72	5.79	4.57	3.16	5.20	4.17	2.42	5.39	4.18	2.42	0	0	0	0	1.44	0.97	1.63	1.35
	0.3	10.77	7.26	4.25	9.36	5.74	3.51	8.30	5.02	3.01	8.30	5.05	3.17	0	0	0	0	1.52	0.95	1.83	1.35
	0.5	13.17	8.94	6.99	11.39	7.09	5.30	11.03	6.71	5.08	11.03	6.70	5.17	0	0	0	0	1.59	0.94	1.73	1.47
	0.7	17.72	9.23	2.41	15.26	7.56	1.89	15.38	7.18	1.72	14.31	7.09	1.37	0	0	0	0	1.27	0.95	1.72	1.28
	0.9	17.66	10.32	4.27	13.67	8.02	3.69	13.38	7.69	2.91	13.00	7.62	3.30	0	0	0	0	1.15	1.09	2.02	1.30
0.9	0.1	18.49	12.67	8.05	16.77	11.33	7.30	16.77	10.97	6.55	16.77	11.15	7.30	0	0	0	0	1.25	1.48	1.68	1.10
	0.3	20.66	16.33	11.62	17.80	13.77	9.28	17.60	13.49	9.28	17.40	13.47	9.28	0	0	0	0	1.26	1.32	1.34	0.99
	0.5	20.26	15.50	7.21	17.29	12.77	3.28	16.88	12.33	3.28	16.73	12.45	3.46	0	0	0	0	1.42	1.19	1.30	1.03
	0.7	21.05	13.85	7.75	17.85	11.53	5.81	17.85	11.48	5.62	17.85	11.39	5.62	0	0	0	0	1.43	1.24	1.38	1.10
	0.9	18.74	14.37	10.75	16.27	12.38	10.02	15.92	12.37	9.57	16.09	12.30	9.58	0	0	0	0	1.37	1.03	1.78	1.16
Total Average		9.86	6.25	3.58	8.24	5.01	2.90	7.73	4.57	2.38	7.74	4.58	2.28	0.16	0.72	1.12	1.24	1.19	0.89	1.91	1.33
Total														4	18	28	31				

Table 5.6: Performance of various approaches on instances with 100 orders

n=100		% Deviations from UB												Number of optimal solutions				Run times (s)			
τ	R	TS			ABC			HSSGA			EA/G-LS			TS	ABC	HSSGA	EA/G-LS	TS	ABC	HSSGA	EA/G-LS
		Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.								
0.1	0.1	3.28	2.44	1.18	2.21	1.75	1.11	1.32	0.77	0.40	1.22	0.94	0.72	0	0	0	0	16.76	3.98	17.91	13.72
	0.3	3.06	2.10	1.51	1.71	1.35	0.90	0.98	0.71	0.45	1.06	0.74	0.44	0	0	0	0	17.26	8.22	16.68	13.76
	0.5	2.74	1.39	0.36	1.23	0.70	0.36	0.38	0.19	0.00	0.37	0.20	0.00	0	0	2	2	10.87	7.61	17.07	12.82
	0.7	0.80	0.43	0.00	0.35	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1	6	10	10	6.21	6.25	15.08	11.06
	0.9	0.89	0.22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4	10	10	10	3.53	9.08	14.73	10.02
0.3	0.1	3.96	2.65	1.47	2.71	2.11	1.27	1.71	1.18	0.50	1.71	1.25	0.69	0	0	0	0	22.37	4.89	16.13	13.53
	0.3	5.39	2.94	1.94	3.97	2.12	1.40	2.64	1.27	0.66	2.64	1.41	0.66	0	0	0	0	20.10	5.29	19.70	14.33
	0.5	3.81	2.21	1.94	2.45	1.67	1.40	1.52	1.07	0.66	1.69	1.11	0.66	0	0	0	0	16.77	4.88	21.00	13.31
	0.7	2.75	1.65	0.37	1.16	0.63	0.00	0.85	0.35	0.00	0.63	0.24	0.00	0	1	1	4	9.48	6.16	16.60	14.04
	0.9	2.41	0.91	0.19	0.82	0.26	0.00	0.56	0.10	0.00	0.47	0.14	0.00	0	4	7	5	7.58	8.59	18.33	12.10
0.5	0.1	5.27	3.96	2.49	4.33	3.37	2.50	2.99	2.09	1.22	3.27	2.31	1.69	0	0	0	0	25.96	5.27	19.30	12.24
	0.3	5.99	3.95	2.68	4.00	3.01	2.46	2.95	2.29	1.54	2.96	2.43	1.67	0	0	0	0	28.87	6.94	20.96	12.87
	0.5	4.53	3.54	2.57	4.21	3.13	2.26	3.35	2.36	1.22	3.61	2.51	1.48	0	0	0	0	20.56	5.86	21.52	13.75
	0.7	4.04	2.71	1.61	2.99	1.82	0.77	2.33	1.49	0.51	2.24	1.32	0.51	0	0	0	0	15.57	6.57	17.27	11.79
	0.9	4.65	2.10	0.67	3.13	1.42	0.53	2.15	1.04	0.18	2.15	0.90	0.18	0	0	0	0	12.15	6.76	18.66	13.72
0.7	0.1	6.42	4.98	2.70	4.84	3.99	3.20	3.53	2.84	2.09	3.82	2.98	2.27	0	0	0	0	33.60	6.04	23.40	11.75
	0.3	10.83	6.66	3.93	7.16	4.72	2.37	6.07	3.88	1.71	6.26	4.09	1.67	0	0	0	0	26.62	6.25	24.90	12.79
	0.5	12.85	6.45	4.04	11.94	5.27	3.27	10.95	4.62	2.25	10.95	4.69	2.62	0	0	0	0	22.30	6.77	19.89	12.79
	0.7	13.20	7.36	3.36	7.37	5.37	2.77	7.78	5.16	2.18	7.65	5.39	2.77	0	0	0	0	26.36	6.78	24.24	14.33
	0.9	12.95	8.40	5.30	8.33	5.90	3.57	7.49	5.48	3.73	6.88	5.34	3.15	0	0	0	0	17.84	7.42	21.90	15.45
0.9	0.1	11.95	9.03	7.30	9.44	6.62	4.70	9.06	6.27	4.57	9.34	6.45	4.75	0	0	0	0	29.46	13.63	19.59	10.65
	0.3	17.41	13.61	7.32	11.86	9.40	5.39	11.25	9.03	5.11	12.35	9.09	5.39	0	0	0	0	26.32	9.82	19.64	11.75
	0.5	18.38	15.68	10.80	17.25	11.88	8.82	16.29	11.26	8.60	15.84	11.20	8.28	0	0	0	0	21.51	6.46	19.06	11.62
	0.7	19.95	15.22	10.70	12.85	11.19	7.96	13.07	11.23	8.07	13.06	11.07	8.05	0	0	0	0	22.70	7.52	17.61	14.30
	0.9	22.21	15.50	11.23	16.83	11.31	4.11	16.60	11.07	3.57	16.73	11.02	4.07	0	0	0	0	17.48	8.61	17.92	11.04
Total Average		7.99	5.44	3.43	5.73	3.96	2.44	5.03	3.43	1.97	5.08	3.47	2.07	0.20	0.84	1.20	1.24	19.13	7.03	19.16	12.78
Total														5	21	30	31				

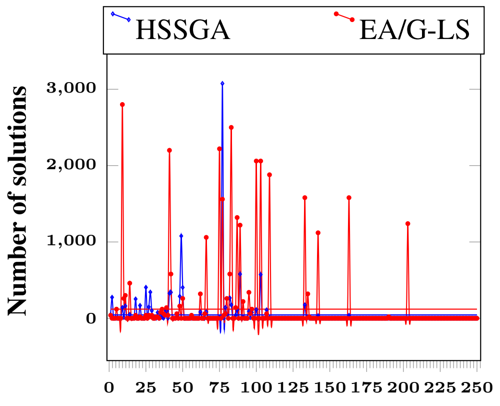
Table 5.7: Comparison of our approaches (HSSGA and EA/G-TS) among themselves and with TS and ABC in terms of number of groups of same size where they perform better(>)/same (=)/worse(<)

n	HSSGA vs. TS			EA/G-LS vs. TS			HSSGA vs. ABC			EA/G-LS vs. ABC			HSSGA vs. EA/G-LS		
	Max_Dev.	Avg_Dev.	Min_Dev.	Max_Dev.	Avg_Dev.	Min_Dev.	Max_Dev.	Avg_Dev.	Min_Dev.	Max_Dev.	Avg_Dev.	Min_Dev.	Max_Dev.	Avg_Dev.	Min_Dev.
	1<	0<	0<	1<	0<	0<	3<	1<	0<	3<	3<	0<	0<	0<	0<
10	**18>**	**19>**	**1>**	**17>**	**19>**	**1>**	**3>**	**4>**	**0>**	**3>**	**4>**	**0>**	**2>**	**2>**	**0>**
	6=	6=	24=	7=	6=	24=	20=	20=	25=	19=	18=	25=	23=	23=	25=
	0<	0<	0<	0<	0<	0<	3<	3<	3<	2<	2<	1<	2<	2<	1<
15	**13>**	**22>**	**10>**	**12>**	**22>**	**12>**	**3>**	**12>**	**6>**	**2>**	**9>**	**5>**	**4>**	**9>**	**1>**
	12=	3=	15=	13=	3=	13=	19=	15=	16=	21=	14=	19=	19=	14=	23=
	0<	0<	0<	0<	0<	0<	2<	4<	2<	1<	7<	4<	2<	3<	1<
20	**17>**	**25>**	**10>**	**18>**	**25>**	**10>**	**8>**	**15>**	**2>**	**6>**	**13>**	**2>**	**4>**	**13>**	**4>**
	8=	0=	15=	7=	0=	15=	15=	6=	21=	18=	6=	19=	19=	9=	21=
	0<	0<	0<	0<	0<	0<	1<	0<	1<	3<	2<	3<	2<	6<	0<
25	**25>**	**25>**	**15>**	**25>**	**25>**	**15>**	**12>**	**23>**	**6>**	**10>**	**22>**	**6>**	**4>**	**12>**	**2>**
	0=	0=	10=	0=	0=	10=	12=	2=	18=	12=	14=	16=	19=	7=	23=
	0<	0<	0<	0<	0<	0<	2<	0<	0<	1<	1<	0<	8<	11<	4<
50	**24>**	**25>**	**22>**	**24>**	**25>**	**22>**	**18>**	**24>**	**19>**	**18>**	**24>**	**18>**	**10>**	**13>**	**8>**
	1=	0=	3= =	1=	0=	3=	5=	1=	6=	6=	0=	7=	7=	14=	13=
	0<	0<	0<	0<	0<	0<	2<	1<	2<	3<	1<	2<	9<	7<	5<
100	**25>**	**25>**	**23>**	**25>**	**25>**	**23>**	**22>**	**23>**	**19>**	**21>**	**23>**	**17>**	**10>**	**16>**	**11>**
	0=	0=	2=	0=	0=	2=	1=	2=	4=	12=	1=	6=	6=	2=	9=
	1<	0<	0<	1<	0<	0<	13<	9<	8<	13<	16<	10<	23<	29<	11<
Total	**122>**	**141>**	**81>**	**121>**	**141>**	**83>**	**66>**	**101>**	**52>**	**60>**	**95>**	**48>**	**34>**	**65>**	**26>**
	27=	9=	69=	28=	9=	67=	71=	30=	90=	77=	39=	92=	93=	56=	113=

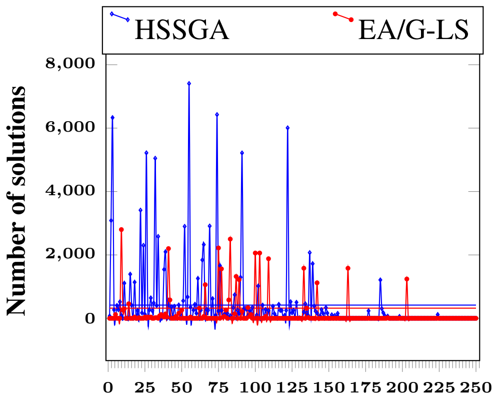
Table 5.7 compares TS, ABC, HSSGA and EA/G-LS approaches over the groups of instances with same problem size, i.e., same number of orders. Each problem size consists of 25 groups, and each group contains 10 instances with the same number of orders, tightness factor (τ) and range factor (R). Results in the corresponding columns indicate that on how many groups, the former approach is less than (Entries marked with <), better than (entries marked with >) and equal to (entries marked with =) to the latter one approach. For example, column "HSSGA vs. TS" reports that, out of 25 groups, on how many groups HSSGA is worse than TS, better than TS and equal to TS for each problem size. The comparison is done in terms of the maximum (Max_Dev.), average (Avg_Dev.) and minimum (Min_Dev.) $\%$ deviation from upper bound. In this table, row "Total" provides the overall count out of all 150 groups (25 groups for each problem size).

We have also studied the convergence behaviour of HSSGA and EA/G-LS on each instance. Figures 5.1(a) to 5.1(f) plot the number of solutions generated by HSSGA and EA/G-LS to reach the best solution on each of the 250 instances of a particular size. In these figures, the horizontal lines indicate the average number of solutions generated by the corresponding approach (HSSGA or EA/G-LS) to reach the best solution. From these figures, it can be observed that HSSGA and EA/G-LS approaches converge very fast on most of the instances.

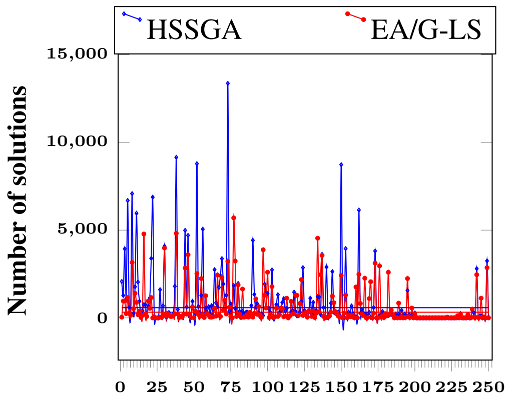
HSSGA generates minimum 10000 solutions, but on most of the instances, it reached the best solution after generating very few solutions only. On problem size 10, HSSGA took on an average 45 and maximum 3075 solutions to reach the best solution, on problem size 15, the average number of solutions is 417, whereas the maximum number of solutions is 7404, on problem size 20, the average number of solutions is 889, whereas the maximum number of solutions is 13368. Similarly, on problem sizes 25, 50 and 100, HSSGA generates on an average 1379, 3950 and 6618 solutions respectively to reach the best solution, whereas the maximum number of solutions generated by HSSGA to reach the best solution are 12223, 22178 and 21561 on instances of size 25, 50 and 100 respectively. Same kind of observations can be made about the convergence behaviour of EA/G-LS. On problem sizes 10, 15, 20, 25, 50 and 100, EA/G-LS generates on an average 120, 320, 580, 820, 1940 and 3520 solutions respectively to reach the best solution. The maximum number of solutions generated by EA/G-LS to reach the best solution are about 2800, 5540, 5700, 6580, 6580 and 12760 on instances of size 10, 15, 20, 25, 50 and 100 respectively.



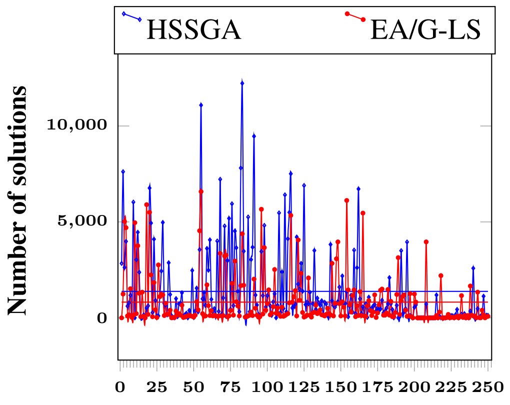
(a) Instances with 10 orders (n=10)



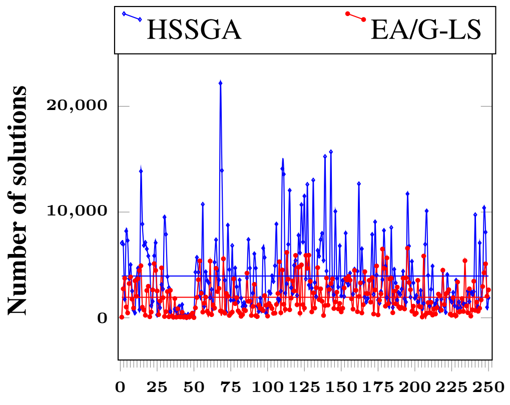
(b) Instances with 15 orders (n=15)



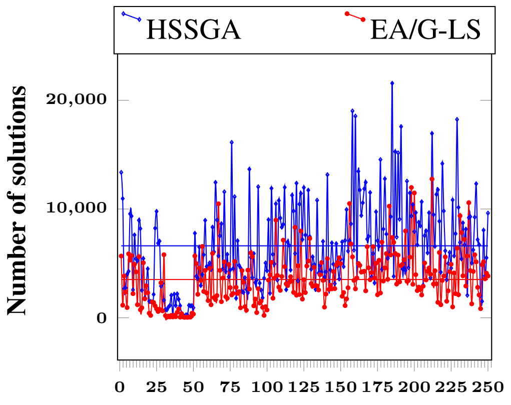
(c) Instances with 20 orders (n=20)



(d) Instances with 25 orders (n=25)



(e) Instances with 50 orders (n=50)



(f) Instances with 100 orders (n=100)

Figure 5.1: Number of solutions generated by HSSGA and EA/G-LS to reach the best solution on each instance with 10, 15, 20, 25, 50 and 100 orders

5.6 Conclusions

This chapter presented two hybrid metaheuristic approaches, viz. HSSGA and EA/G-LS for order acceptance and scheduling (OAS) problem in a single machine environment. We have compared our approaches with two state-of-the-art approaches, viz. TS and ABC for OAS problem and the computational results show the superiority of the proposed approaches in terms of solution quality. However, in terms of computational times, TS and ABC approaches are faster than HSSGA and EA/G-LS approaches on most of the instances. As far as comparison between the two proposed approaches is concerned, HSSGA and EA/G-LS obtained solutions close to each other on most of the instances. Though, HSSGA is slightly better than EA/G-LS in terms of solution quality, it is slower.

Chapter 6

The Single Machine Total Stepwise Tardiness Problem with Release Dates

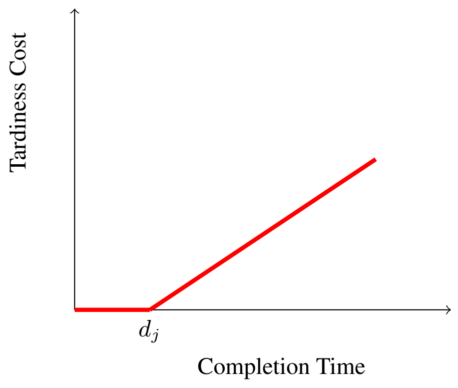
6.1 Introduction

This chapter addresses a single machine scheduling problem where the tardiness cost of a job increases stepwise with various due dates. Using the notational conventions similar to those used in [124], 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. Each job J_j needs a processing time p_j on this machine and has a release date r_j. It also has m due dates $d_{1j}, d_{2j}, \ldots, d_{mj}$ such that $d_{1j} < d_{2j} < \ldots < d_{mj}$. The job J_j can not start before its release date and is considered to be tardy if its completion time C_j exceeds d_{1j}. Each tardy job incurs a tardiness cost. The tardiness cost $\bar{f}_j(C_j)$ of job J_j is determined as follows:

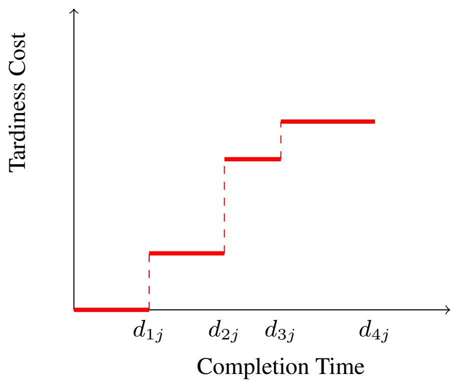
$$\bar{f}_j(C_j) = \begin{cases} 0, & \text{if } C_j \le d_{1j}; \\ w_{1j}, & \text{if } d_{1j} < C_j \le d_{2j}; \\ w_{2j}, & \text{if } d_{2j} < C_j \le d_{3j}; \\ \vdots & \vdots \\ w_{mj}, & \text{if } d_{mj} < C_j. \end{cases} \tag{6.1}$$

where w_{kj} is the tardiness cost of the job J_j in the k^{th} late period and $0 < w_{1j} < w_{2j} < \ldots < w_{mj}$. The objective of the single machine total stepwise tardiness problem with release dates (SMTSTP-R) considered in this chapter is to find a schedule that minimizes

$$\sum_{j=1}^{n} \bar{f}_j(C_j) \tag{6.2}$$



(a) Conventional tardiness



(b) Stepwise tardiness

Figure 6.1: Conventional and stepwise tardiness

Hence, SMTSTP-R seeks a schedule that minimizes the sumtotal of the tardiness costs of all the jobs. We have also considered another version of SMTSTP-R in this chapter, where all release dates are assumed to be zero, i.e., all jobs are available for processing at the beginning. This version will be referred to as SMTSTP subsequently. Both SMTSTP and SMTSTP-R are $\mathcal{NP}$-hard, as they can be considered as generalizations of the weighted tardiness scheduling problem which is $\mathcal{NP}$-hard [125]. Using the standard three field notation [126], SMTSTP and SMTSTP-R can be denoted as $1|\bar{f}_j(C_j)$ and $1|r_j|\bar{f}_j(C_j)$ respectively.

Unlike the conventional scheduling problems based on tardiness criterion where there is a single due date d_j for each job J_j and the tardiness cost associated with a job is assumed to be a linear function of its completion time (Figure 6.1(a)), SMTSTP and SMTSTP-R assume that the tardiness cost of a job to be a piecewise constant function of the completion time, i.e., tardiness cost increases in a stepwise manner with various due dates (Figure 6.1(b))[124]. Stepwise tardiness cost occurs in several practical scenarios mostly related to transportation [124, 127, 128]. Transportation services are not always available and a finished job has to wait till the next transportation service is available. Hence, the delivery time of a job finished anywhere after the departure of one service and before the departure of the next service remains unaffected at the end customer. As a result such a job incurs the same tardiness cost irrespective of its exact completion time. Hence, the tardiness cost in this situation increases in a stepwise manner. Another situation where tardiness costs of jobs increase in a stepwise manner arises when modes of shipping (rail, road, air etc.) of jobs depend on their completion times with respect to their various due dates.

Stepwise tardiness criterion also finds application for batch production in manufacturing sector [128]. A batch may contain several customer orders each of which has its own due date. When an order in the batch misses its due date then tardiness cost of the batch increases. Hence, tardiness cost of the batch increases in a stepwise manner depending on how many orders have missed their due dates. Similarly, Detienne *et al.* [129] describes another application of stepwise cost function in semiconductor manufacturing, where inspection operations need to be scheduled on parallel machines in such a manner that a fixed production schedule can be made as reliable as possible.

There exist only a few studies in the literature utilizing stepwise tardiness criterion for scheduling. Curry and Peters [127] produced the seminal work on stepwise tardiness criterion. They used total stepwise tardiness criterion in a parallel machine re-scheduling problem and proposed a branch-and-price algorithm to solve it. Sahin [128] utilized this criterion for hump sequencing problem in railroad yards and developed integer programming formulations and approximation algorithms to solve it.

Detienne *et al.* [129] proposed mixed integer linear programming models for the already described problem of inspection operations scheduling. Later Detienne *et al.* [130] presented an exact approach based on Lagrangian relaxation for SMTSTP. Lower and upper bounds based on linear and Lagrangian relaxation of different mixed integer linear programming formulations were also presented.

Tseng and Chen [124] presented three heuristics and a metaheuristic approach based on electro-magnetism like mechanism (EM) algorithm [131] for SMTSTP. One heuristic was derived by modifying Moore's algorithm for $1|\sum U_j$ [132]. The other two heuristics were based on NEH heuristic for the flow-shop scheduling problem with makespan criterion [133]. Though Tseng and Chen [124] described SMTSTP-R, none of the approaches presented considered the factor of release dates into account. EM algorithm presented was experimentally evaluated for SMTSTP only by ignoring the release dates of all jobs and assuming all jobs are available for processing at time zero. To compare the performance of the proposed EM algorithm, they also implemented approaches based on genetic algorithm (GA), particle swarm optimization (PSO), differential evolution (DE) and an existing EM algorithm for SMTSTP. Each of these approaches was derived from the corresponding approach for a related scheduling problem ([134, 135, 136]), as there was no metaheuristic approach for SMTSTP to compare with the proposed EM algorithm. Computational results showed the superiority of the proposed EM algorithm over all the other algorithms considered for SMTSTP.

To our knowledge, there exists no metaheuristic approaches for SMTSTP-R. This served as the motivation for the present work. In this chapter, we have proposed two metaheuristic approaches viz. a genetic algorithm (GA) and an artificial bee colony (ABC) algorithm for SMTSTP and SMTSTP-R. We have hybridized our approaches with a local search to further improve the quality of the solutions obtained. We have also tried an approach based on EA/G, but its results were always much inferior to those obtained by GA and hence it is not described. We have developed the ABC approach so that the performance of genetic algorithm can be ascertained vis-á-vis an state-of-the-art metaheuristic for SMTSTP-R. We have compared our approaches with those presented in [124] on the same benchmark instances as used in [124] for SMTSTP. As the approaches presented in [124] do not use any local search, so we have also presented the results of our approaches without any local search. Computational results show the superiority of our approaches over other approaches in terms of solution quality and running time both, irrespective of whether local search is used or not. For SMTSTP-R, our approaches being the only approaches are compared among themselves. Results of hybrid GA and hybrid ABC approaches are comparable, though the hybrid ABC approach performs slightly better in terms of solution quality and running time both, not only on SMTSTP-R, but also on SMTSTP.

The rest of the chapter is organized as follows: Section 6.2 describes our genetic algorithm based approach, whereas Section 6.3 describes our artificial bee colony algorithm based approach. Computational results are presented in Section 6.4. Finally, Section 6.5 outlines some concluding remarks.

6.2 Hybrid genetic algorithm

This section describes our hybrid genetic algorithm based approach for SMTSTP and SMTSTP-R. We have taken the factor of release dates into account while designing genetic operators and local search.

We have used permutation encoding to encode the solutions into chromosomes. Hence, each chromosome is a linear permutation of jobs. The order in which jobs need to be scheduled is determined by the positioning of the jobs in the permutation. More precisely, a value of j at the i^{th} position in the permutation indicates that job J_j is the i^{th} job in the schedule represented by that permutation. The decision to choose permutation encoding is based on the fact that SMTSTP and SMTSTP-R are inherently permutation problems, and therefore, permutation encoding is natural for them.

Fitness function is same as the objective function (Equation (6.2)), i.e., a schedule is more fit than the other if its total tardiness cost is less than the other.

We have used binary tournament selection (BTS) method to select two parent chromosomes for crossover. It is also used to select a parent chromosome for our first mutation operator. With probability P_{bt}, the chromosome having better fitness is chosen to be a parent, otherwise the worse one is chosen.

We have employed uniform order based crossover (UOB) where jobs at each position of the first parent is copied with probability P_{uc} to the corresponding position of the child. The UOB crossover is chosen because in the case of SMTSTP and SMTSTP-R, the information about the relative ordering among jobs matters more than the information about adjacency among jobs. P_{uc} is taken to be equal to $\frac{F(p_2)}{F(p_1)+F(p_2)}$, where $F(p_1)$ and $F(p_2)$ are the fitnesses of the first and second parents respectively. Clearly, this scheme favors more number of jobs to be transferred from the best of the two parents to the child [137]. The EM method of [124] also used UOB crossover as one of the two attraction operators. The other attraction operator used was similar block order crossover [138].

We have used crossover and mutation in a mutually exclusive manner, i.e., each child is generated either by the crossover or by mutation, but never through crossover and mutation both. Crossover is applied with probability P_c, otherwise mutation is used. We have used two mutation operators which are also used in a mutually exclusive manner. With probability P_{mut}, the first mutation operator is used, otherwise the second mutation operator is used.

Our first mutation operator consists of two phases. The first phase is same as the first phase of UOB crossover, i.e., it copies jobs at each position of the parent with probability P_{mc} to the child. The second phase first sorts the leftout jobs in a specific order and then copy these jobs in the sorted order to the vacant positions one-by-one beginning with the first vacant position. There are three strategies for sorting. These leftout jobs are sorted according to non-decreasing order of their release dates in the first strategy, according to non-decreasing order of their earliest possible completion times (sum of release date and processing time) in the second strategy and according to non-decreasing order of their first due dates in the third strategy. The first, second, and third strategies are used with probabilities 0.75, 0.20, 0.05 respectively. Considering a small number of leftout jobs, we have used selection sort for sorting. When we sort according to release dates, we swap two jobs with probability 0.5, even when their release dates are equal. This is done to generate diverse solutions. This is specially important for SMTSTP where all release dates are assumed to be zero.

Our second mutation operator consists of multiple swap operations. During each swap operation, it selects two positions in the schedule uniformly at random and swaps the jobs at these two selected positions. Swap operation is applied N_{swap} times. Unlike the first mutation operator, where parent chromosome to be mutated is selected through the BTS, here the parent chromosome to be mutated is selected uniformly at random. Actually, first mutation operator is designed with exploitation in mind, whereas second mutation operator is designed for exploration and maintaining the diversity in the population.

Our genetic algorithm uses the steady-state population model [21], and hence, the newly generated child is discarded in case it is found to be identical to any existing population member, otherwise newly generated child always replaces the worst member of the population irrespective of its own fitness.

The population of the genetic algorithm is initialized with randomly generated permutations. However, to comply with the steady-state population model, the uniqueness of each member of the initial population is ensured. This is done by always comparing the newly generated random permutation with initial population members generated so far, and, adding the newly generated random permutation in the initial population only when it is unique. If the newly generated random permutation is found to be identical to any initial population members generated so far then it is discarded.

The solution obtained through the genetic algorithm is improved further through a series of local searches. Our first local search tries to remove the idle times in the schedule whenever doing so reduces the objective function value. The schedule may contain some idle times at various places because a job in the schedule may finish earlier than the release date of the next job in the schedule minus one. Our local search begins with the first job in the schedule and check each job one-by-one to determine whether there is idle time before the job under consideration (say J_i) can be scheduled. If there is idle time, then we try all jobs following J_i in the schedule one-by-one to determine whether inserting them before J_i and shifting the jobs affected one position towards the end removes the idle time and reduces the objective function value. If J_i is the first job in the schedule, then instead of searching for a job which can remove the idle time completely, we search for the job which can reduce the idle time and objective function both. The first such job say J_j is inserted and the next idle slot in the schedule is considered. If no such job exists, then this idle slot remains and the next idle slot in the schedule is considered. This process is repeated till all idle slots have been considered once. The reason for special processing in case J_i happens to be the first job in the schedule is that there may not exist any job with release date zero.

Our second local search, instead of removing idle times completely, tries to either reduce the idle times or keep them same as long as doing so reduces the objective function value. It is similar to first local search except for the fact that it allows insertion even when idle times remain unchanged provided the objective function value is reducing.

Our third local search consists of successive passes of an adjacent pairwise interchange (API) procedure. During each pass, beginning at first job in the schedule, API procedure considers one-by-one each of the $n - 1$ pairs of adjacent jobs and swaps the jobs in the pair provided doing so either reduces the objective function value or the objective function value remains the same but the completion time of the second job in the pair after the swap reduces. API procedure is applied repeatedly till a complete pass fails to reduce the objective function value and the completion time. Our last local search being computationally expensive is applied only in case the solution obtained is better than the current best solution. This local search also consists of multiple passes through the schedule. Beginning at the first position in the schedule, it considers each position one-by-one and tries to insert the jobs at subsequent positions in the schedule at the position under consideration by shifting the affected jobs one position towards the end. The job that reduces the objective function value is inserted, and the next job in the schedule is tried for insertion at the position under consideration. The next position in the schedule is considered for insertion when all subsequent jobs have been tried. This process is repeated till a complete pass fails to reduce the objective function value.

The pseudo-code of our hybrid genetic algorithm based approach is presented in Algorithm 6.1, where u_{01} is a uniform variate in [0, 1]. In this pseudo-code, $crossover()$, $First\_Mutation()$ and $Second\_Mutation()$ are functions implementing UOB crossover, first mutation operator and second mutation operator respectively. $Local\_Search()$ is another function that implements the series of local searches described above.

Hereafter, our hybrid genetic algorithm based approach will be referred to as HGA, and the version of this approach where no local search is used will be referred to as GA.

6.3 Hybrid artificial bee colony algorithm

As artificial bee colony (ABC) algorithm is a relatively new metaheuristic, so we provide a brief introduction to ABC algorithm before describing our hybrid ABC approach for SMTSTP and

Algorithm 6.1: Pseudo-code of hybrid genetic algorithm approach

```
1:  Generate initial population;
2:  $best \leftarrow$ Best solution in the initial population;
3:  while (Termination condition does not hold) do
4:      if ($u_{01} < P_c$) then
5:          Select two parents $p_1$ and $p_2$ using BTS;
6:          $C \leftarrow Crossover(p_1, p_2)$;
7:      else
8:          if ($u_{01} < P_{mut}$) then
9:              Select a parent $p$ using BTS;
10:             $C \leftarrow First\_Mutation(p)$;
11:         else
12:             Select a parent $p$ randomly;
13:             $C \leftarrow Second\_Mutation(p)$;
14:         end if
15:     end if
16:     $C \leftarrow Local\_Search(C)$;
17:     if ($C$ is unique with respect to the current population) then
18:         Include $C$ in the population in place of the worst member;
19:         if ($C$ is better than $best$) then
20:             $best \leftarrow C$;
21:         end if
22:     end if
23: end while
24: return $best$;
```

SMTSTP-R.

6.3.1 Introduction to ABC algorithm

ABC algorithm is inspired by foraging behavior of honey bees. Since the inception of ABC algorithm in 2005 [13], it has been playing a remarkable role in finding high quality solutions for hard optimization problems. In nature, the forager bees are classified into three categories, viz. scout bees, employed bees and onlooker bees. Scout bees are those bees that search new food sources in the vicinity of the hive, and, as soon as, they find food sources, they become employed bees. Employed bees are those bees that are currently exploiting food sources. Onlooker bees are those bees that wait in the hive for the employed bees to return and communicate with them the information (direction, distance, and nectar content) about their respective food sources through a dance known as Waggle dance. Onlookers tend to select a food source with a probability proportional to the quality of that food source. Obviously, a good quality food source would attract more onlooker bees. Once an onlooker selects a food source, it also becomes employed. If a food source becomes empty, then all employed bees associating with such a food source abandon it and become either scouts or onlookers.

Such collective and highly-coordinated behavior of forager bees have been modelled in ABC algorithm [13, 139, 140]. ABC algorithm also consists of same three categories of artificial bees, i.e., scout, employed and onlooker bees. Each food source represents a feasible solution to the problem under consideration, whereas its nectar content represents the fitness of the solution. Each food source has an associated employed bee, i.e., the number of employed bees is same as the number of food sources. Usually, but not always, the number of onlooker bees are also taken to be equal to the number of employed bees. To start ABC algorithm, a fixed number of solutions (food sources) are generated randomly, and each one of them is associated with an unique employed bee. Hereafter, a search process is carried out iteratively until a termination criterion is satisfied. Each iteration of this search process contains two phases which are described as follows:

1. *Employed bee phase*: Each employed bee determines a solution in the neighborhood of its currently associated solution. If the fitness of this new solution is better than its currently associated solution, then it moves to this new solution, otherwise it continues with the old one. The exact manner in which this neighboring solution is determined varies from problem to problem and even for the same problem from one implementation of ABC algorithm to the other. If the fitness of a solution does not improve for a certain number of iterations (say $limit$ number of iterations), then it is assumed that this solution is completely exploited. In this situation, employed bee associating with this solution becomes a scout bee by discarding this solution. A new solution is generated for this scout bee and its status is again changed back to employed. This new solution is usually (but not always) generated in the same manner as an initial solution.

2. *Onlooker bee phase*: Once all employed bees complete the job of determining new neighboring solutions, they start communicating the information about fitnesses of their corresponding solutions with onlooker bees. Each onlooker bee selects a solution probabilistically with the help of a probability based selection method. Generally, such a

selection method prefers the high quality solutions over the poor ones, resulting in selection of high quality solutions by more and more number of onlooker bees. All onlooker bees, like the employed bees, determine new solutions in the neighborhood of its selected solution. Hereafter, among all new neighboring solutions determined by onlookers that associate with a particular solution i and solution i itself, the best solution will be the new solution i. This phase completes when the new positions of all solutions are determined.

For a latest survey on different variations of ABC algorithm and their applications, interested readers may refer to [141].

6.3.2 Proposed hybrid ABC approach

This section describes the salient features of our hybrid ABC approach for SMTSTP and SMTSTP-R. We have used the same solution encoding and fitness function in our hybrid ABC approach as used in our genetic algorithm. Each initial employed bee solutions is also generated in the same manner as a member of the initial population in the genetic algorithm. Each neighboring solution generated by ABC algorithm is improved through the same series of local searches as in genetic algorithm. We have also chosen binary tournament selection method as the probability based selection method for selecting a food source for each onlooker bee. Here the candidate with better fitness is selected with probability ρ_{bt}

6.3.3 Determination of a neighboring solution

Since SMTSTP and SMTSTP-R are permutation problems and ABC algorithm is a population-based metaheuristic technique, therefore, to exploit the problem structure of this problem in the framework of an ABC algorithm, two different methods are applied in a mutually exclusive way to determine a new solution in the neighborhood of a solution. First method called multi-point insert method [142] considers the idea of utilizing solution components (jobs) from another solution [142, 143]. It is based on a simple observation that if a job is placed at a particular position in a schedule of high quality, then there is a high possibility that this job would be placed exactly at the same position or near to this position in many other high quality schedules. Second method called 3 point swap (3PS) method [144] helps in avoiding a solution from getting trapped in a local optima and also helps in exploring the search space. With probability ρ_{method}, multi-point insert method is applied, otherwise 3PS method is applied.

1. *Multi-point insert method*: In order to determine a new solution (say Z) in the neighborhood of the current solution (say X), another solution (say Y) is selected uniformly at random. X and Y are compared, and in case they are found to be same, then we switch to 3PS method as multi-point-insert method will have no effect in this situation. If X and Y are not same then multi-point insert method is applied. This method initializes Z with an empty schedule (all positions in the schedule are vacant). MPI_n different positions are selected uniformly at random in Y, where MPI_n is a parameter to be determined empirically. All jobs associated with these selected positions of Y are inserted exactly to the same positions in Z. In order to fill the remaining vacant positions in Z, those jobs of X that are not already in Z are sorted according to ascending order of their positions in the schedule represented by X. Beginning at first vacant position in Z, vacant positions are filled one-by-one by these jobs in their sort order. The value of MPI_n should be small (10-20%) in comparison to the total number of jobs, otherwise generated solution will be quite far from the original solution X.

   In order to better understand the multipoint insert operator, let us consider an example where X = {1, 3, 2, 5, 7, 4, 6, 8}, Y = {2, 1, 4, 5, 3, 7, 8, 6}, and MPI_n = 2. Initially, Z is an empty schedule,i.e., Z = {_, _, _, _, _, _, _, _}, where '_' indicates a vacant position. Suppose position 3 and 6 of Y got selected so jobs at these two positions in Y will be copied to the corresponding positions in Z, i.e., Z = {_, _, 4, _, _, 7, _, _} . Positions 1, 2, 4, 5, 7 and 8 are still vacant. Those jobs in solution X that are still not in Z are inserted into Z in the order 1, 3, 2, 5, 6, 8. So jobs 1, 3, 2, 5, 6 and 8 get inserted into Z at positions 1, 2, 4, 5, 7 and 8 respectively. So the new neighboring solution Z is {1, 3, 4, 2, 5, 7, 6, 8}.

2. *3 point swap method*: In order to generate a solution Z in the neighborhood of a solution X, this method creates a copy of X into Z. Then three positions in Z, say i, j and k, are selected uniformly at random. The jobs in positions i and j are swapped, and then the new job in position i is swapped with job at position k. 3 point swap (3PS) method was proposed in [144] in the context of one-dimensional cutting stock problem. 3PS generates a schedule which is at a distance of 2 from the original schedule if we measure the distance between two schedules in terms of number of swap operations required to convert one schedule to the other. In comparison to using only one swap, 3PS may accelerate the search towards optimal.

The pseudo-code of determining a new neighboring solution is given in Algorithm 6.2.

Algorithm 6.2: The pseudo-code of determining a new neighboring solution

```
1: Input: A solution $X$;
2: Output: A new solution $Z$ in the neighborhood of $X$;
3: Select a solution $Y$ randomly;
4: if $((u_{01} < \rho_{method}) \wedge (X \neq Y))$ then
5:     Apply multi-point insert method to generate $Z$ from $X$ and $Y$;
6: else
7:     Create a solution $Z$ which is a copy of $X$ initially;
8:     Apply 3PS method on $Z$;
9: end if
10: return $Z$;
```

If the quality of a solution does not improve for some specified number of iterations, say $limit$, then this solution is assumed to be completely exploited, and employed bee associated with this solution abandons it to become a scout. In order to make this new scout bee again employed a new solution need to be generated. However, instead of generating the new solution randomly like an initial employed bee solution, we have employed 3PS method on the just abandoned solution to generate the new solution. There are two reasons for doing this. First, a new solution generated randomly is expected to have a much worse fitness in comparison to a solution obtained by applying 3PS method on an existing solution, and as a result, a randomly generated solution in general requires several iterations for its fitness to be at par with a solution obtained through 3PS method. Second reason lies in efficiency. It is more efficient to generate a solution through 3PS method than randomly generating it from scratch. $limit$ is a parameter whose value is determined empirically. This parameter plays an important role in ABC algorithm, as it controls the intricate balance between exploration and exploitation. A lower value of the $limit$ favors exploration over exploitation, whereas the reverse is true for a higher value of $limit$.

The pseudo-code of hybrid ABC approach is presented in Algorithm 6.3. In this algorithm, E and On are the number of employed bees and onlooker bees respectively. *DNS(X)* is a function that determines a new solution Z in the neighborhood of the solution X and returns Z. Pseudo-code for *DNS(X)* is given in Algorithm 6.2. Function *BTSM*(e_1, e_2,…,e_E) implements the binary tournament selection method. This function returns the index of the selected solution.

Hereafter, our hybrid artificial bee colony algorithm based approach will be referred to as HABC, and the version of this approach where no local search is used will be referred to as

Algorithm 6.3: The pseudo-code of hybrid ABC approach

```
1:  Initialize $E$ solutions $e_1$, $e_2$,...,$e_E$;
2:  $best \leftarrow$ Best solution $\in \{e_1, e_2, \ldots, e_E\}$;
3:  while (Termination criteria is not satisfied) do
4:      for ($i \leftarrow 1$ to $E$) do
5:          $Z \leftarrow DNS(e_i)$;                                   ▷ Algorithm 6.2
6:          Apply local search procedure on $Z$;
7:          if ($Z$ is better than $e_i$) then
8:              $e_i \leftarrow Z$;
9:          else if ($e_i$ has not improved for $limit$ iterations) then
10:             Replace $e_i$ with a randomly generated solution;
11:         end if
12:         if $e_i$ is better than $best$ then
13:             $best \leftarrow e_i$;
14:         end if
15:     end for
16:     for ($i \leftarrow 1$ to $On$) do
17:         $I_i \leftarrow BSM(e_1, e_2, \ldots, e_E)$;
18:         $Z_i \leftarrow DNS(e_{I_i})$;                              ▷ Algorithm 6.2
19:         Apply local search procedure on $Z_i$;
20:         if ($Z_i$ is better than $best$) then
21:             $best \leftarrow Z_i$;
22:         end if
23:     end for
24:     for ($i \leftarrow 1$ to $On$) do
25:         if $Z_i$ is better than $e_{I_i}$ then
26:             $e_{I_i} \leftarrow Z_i$;
27:         end if
28:     end for
29: end while
```

ABC.

6.4 Computational results

All the proposed approaches have been coded in C and executed on a Linux based system with 2.83 GHz Intel Core 2 Quad processor and 4 GB RAM. In our genetic algorithm, we have used a population of 600 chromosomes. For crossover, first parent is selected with probability P_{bt} = 0.8, whereas second parent is selected with probability P_{bt} = 0.7. Crossover is applied with probability P_c = 0.65, otherwise mutation is used. When mutation is used, first mutation operator is used with P_{mut} = 0.80, otherwise the second mutation operator is used. In the first mutation operator, we have used P_{mc} = 0.90 and P_{bt} = 0.80, whereas N_{swap} is set to $0.1 \times n$ in the second mutation operator. In case of artificial bee colony algorithm, we have used a population of 50 bees with 25 employed bees and 25 onlooker bees, i.e., E = 25 and On = 25. We have set ρ_{bt} = 0.85, ρ_{method} = 0.5, MPI_n = 0.2×n, and $limit$ = 50. All these parameter values are chosen after a large number of trials. Results obtained using these parameter values are good on most of the instances, though these parameter values may not be optimal for all the instances. We have set the termination condition in our genetic algorithm and ABC algorithm in such a manner that they generate roughly the same number of solutions. Our genetic algorithm terminates when the best solution has not improved over $300 \times n$ iterations and it has been executed for at least 30000 iterations. Our ABC algorithm terminates when the best solution has not improved over $6 \times n$ iterations and it has been executed for at least 611 iterations. It is to be noted that our genetic algorithm being steady-state generates a single solution in an iteration, whereas ABC algorithm generates fifty solutions in an iteration.

We have used the same test instances as used in [124] to test our approaches. These test instances are divided into two sets, viz. Set I and Set II. Set I consists of instances with number of jobs $n \in \{10, 30, 50, 100, 200\}$ and number of due dates $m \in \{2, 3, 4, 9\}$. Tseng ad Chen [124] generated instances in Set I so that they have the similar characteristics as the instances used in [130, 134, 145]. The processing time p_j, release date r_j and m due dates in these instances are drawn from the uniform distributions $U(1, 100)$, $U(0, n \times K_1)$ and $U(r_j + p_j, r_j + p_j + n \times K_2)$ respectively, where K_1 and K_2 are two parameters which can take values in $\{5, 10, 20\}$. Tardiness increase for each due date of each job is drawn from $U(1, 100)$. For each combination of values of n, m, K_1 and K_2, 10 instances were generated leading to a grandtotal of 1800 instances in the Set I.

Set II contains instances with number of jobs $n \in \{50, 100, 150, 200\}$ and number of due dates $m \in \{3, 4, 9\}$. Tseng ad Chen [124] followed the procedures used in [146, 147] for generating Set II instances. The processing time p_j is drawn from the normal distribution $N(\mu_p, \sigma_p^2)$ and tardiness cost in first late period w_{1j} is drawn from normal distribution $N(\mu_w, \sigma_w^2)$. The release date r_j is drawn from a uniform distribution $U(0, n \times K_1)$, where K_1 can be either 5 or 20. The first due date d_{1j} is drawn from uniform distribution with mean μ_d and range R_d. μ_d and μ_w are set to 100 and 20 respectively in these instances. Remaining four parameters σ_p, σ_w, μ_d and R_d are specified indirectly in the form of following factors:

- Relative variation of processing times f_1: $\frac{\sigma_p}{\mu_p}$ is set to 0.80.
- Relative variation of tardiness costs f_2: $\frac{\sigma_w}{\mu_w}$ is set to 0.80.
- Tardiness factor f_3: $1 - \frac{\mu_d}{n \times \mu_p}$ is set to either 0.40 or 0.65.
- Relative range of due dates f_4: $1 - \frac{R_d}{n \times \mu_p}$ is set to either 0.30 or 0.80

The time differences between various due dates are generated by multiplying a random number $K \in [0.1, 0.25]$ with a uniform distribution with mean μ_d and range R_d. The cost differences between stepwise tardiness are drawn from a normal distribution $\frac{1}{2} \times N(\mu_w, \sigma_w^2)$. All data are rounded off to nearest integer. For each combination of values of n, m, f_1, f_2, f_3, f_4, K_1 and K, 10 instances were generated leading to a grandtotal of 960 instances in the Set II. Like the approaches of [124], our approaches are also executed 10 independent times on each instance in Set I and Set II.

First, we compare the performance of GA, ABC, HGA and HABC approaches with EM and PSO approaches of [124] for SMTSTP on Set I and Set II instances. For SMTSTP, release dates of these instances are ignored and all jobs are assumed to be available for processing at time zero. We have ignored other approaches presented in [124] in this comparison as their results are even inferior. We have also ignored Set I instances with 10 jobs, as these instances are too small for comparing the performance of metaheuristics. Table 6.1 compares these six approaches on Set I, whereas Table 6.2 does the same on Set II. For each group of instances with same n and m, these tables report the following for each of the six methods:

- Average percentage deviation (APD) in the quality of the best solution obtained by a method with respect to the overall best solution among six methods over all the instances in a group. Suppose the best solution obtained by a method M over 10 runs on a particular instance is B_M, and B_O is the overall best solution obtained by any of the six methods over their respective 10 runs on this instance, the percentage deviation in the quality of the best solution obtained by method M on this instance is $100 \times \frac{(B_M - B_O)}{B_O}$. APD in the quality of the best solution obtained by a method M is the average of these percentage deviations over all the instances in the group.
- APD in the quality of average solution obtained by a method with respect to the best average solution among six methods over all the instances in a group. Suppose the average solution obtained by a method M over 10 runs on a particular instance is A_M, and A_O

is the best average solution obtained by any of the six methods over their respective 10 runs on this instance, the percentage deviation in the quality of average solution obtained by method M on this instance is $100 \times \frac{(A_M - A_O)}{A_O}$. APD in the quality of the average solution obtained by a method M is the average of these percentage deviations over all the instances in the group.

- Average execution time (AET) in seconds over all the runs of all the instances in a group.

Results of EM and PSO approaches for each of the 10 runs on each instance were obtained from C.-T. Tseng through personal communication, and we have processed these results to get various APD and AET values.

Tables 6.1 and 6.2 clearly show the superiority of our approaches over EM and PSO in terms of best and average APD values. Even the results of our approaches without local search, viz. GA and ABC are much better when we compare their best and average APD values with those of EM and PSO. Among all the six approaches, HABC performed the best followed by HGA. For Set II, the performance of HGA is also quite close to HABC. However, this is not the case for Set I. For Set I, even ABC performs better than HGA on some groups of instances with 200 jobs. If we compare GA with HGA and ABC with HABC, it can be clearly seen that the use of local search enhances the solution quality, but at the expense of increased computation times. As far as the comparison of AET values are concerned, EM and PSO approaches were executed on a system with 2.83 GHz Intel Core 2 Quad processor and 2 GB RAM, whereas our approaches are executed on a system with slightly different configuration, viz. 2.83 GHz Intel Core 2 Quad processor and 4 GB RAM. Even with this slight difference in configuration, we can safely say that GA, ABC, HGA and HABC approaches are several times faster than EM and PSO. ABC is faster than GA and HABC is faster than HGGA.

Table 6.1: Comparison of various approaches for SMTSTP on Set I instances

$n \times m$	**EM**			**PSO**			**GA**			**ABC**			**HGA**			**HABC**		
	APD		ATA	APD		ATA	APD		ATA	APD		ATA	APD		ATA	APD		ATA
	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)
30×2	3.00	6.31	1.63	18.36	24.91	1.62	0.76	1.92	0.14	0.42	0.95	0.03	0.09	0.38	0.22	0.00	0.01	0.11
30×3	2.21	5.28	1.63	17.86	23.80	1.62	0.59	1.88	0.14	0.26	0.73	0.03	0.04	0.21	0.25	0.03	0.01	0.12
30×4	1.85	3.96	1.64	15.79	20.53	1.64	0.57	1.21	0.15	0.30	0.58	0.03	0.06	0.22	0.27	0.01	0.01	0.14
30×9	1.37	2.88	1.71	14.13	17.56	1.72	0.33	0.84	0.16	0.11	0.35	0.04	0.03	0.06	0.36	0.00	0.01	0.22
50×2	6.01	9.60	4.34	21.47	27.16	4.34	1.26	2.36	0.22	0.98	1.57	0.04	0.33	0.60	0.34	0.02	0.02	0.20
50×3	4.73	7.66	4.37	17.78	23.22	4.38	1.06	1.83	0.22	0.63	1.10	0.05	0.15	0.47	0.38	0.01	0.00	0.23
50×4	4.45	7.15	4.41	17.72	22.73	4.42	1.26	1.91	0.23	0.66	1.16	0.05	0.18	0.49	0.42	0.01	0.01	0.26
50×9	3.00	4.85	4.61	15.05	19.31	4.62	0.58	1.13	0.26	0.33	0.69	0.07	0.09	0.21	0.64	0.00	0.00	0.42
100×2	11.34	14.67	18.24	35.85	41.90	18.26	1.73	2.34	0.71	1.28	1.57	0.18	0.71	1.13	1.45	0.03	0.00	0.87
100×3	9.78	12.79	18.73	31.91	37.40	18.76	1.62	2.17	0.72	1.16	1.45	0.20	0.70	1.05	1.67	0.01	0.00	1.03
100×4	8.78	11.33	19.13	27.95	32.78	19.17	1.46	1.83	0.73	0.90	1.21	0.21	0.53	0.81	1.89	0.01	0.00	1.16
100×9	6.22	8.19	20.37	21.95	25.61	20.40	1.02	1.36	0.84	0.74	1.06	0.28	0.27	0.55	3.11	0.01	0.00	1.91
200×2	19.69	23.71	84.14	64.21	67.07	84.39	2.27	2.78	2.44	1.26	1.35	0.87	1.79	2.62	6.14	0.00	0.00	5.23
200×3	16.88	20.59	88.20	55.02	58.18	88.46	1.89	2.25	2.54	1.08	1.28	0.93	1.54	2.26	7.24	0.00	0.00	6.26
200×4	15.60	18.78	91.25	49.98	53.18	91.51	1.66	1.88	2.62	1.03	1.19	0.97	1.35	1.86	8.06	0.00	0.00	6.90
200×9	11.58	13.87	99.03	38.71	41.58	99.29	1.17	1.32	3.07	0.90	1.07	1.37	0.83	1.20	13.81	0.00	0.00	11.85
Overall	7.90	10.73	28.96	28.98	33.56	29.04	1.20	1.81	0.95	0.75	1.08	0.33	0.54	0.88	2.89	0.01	0.00	2.31

Table 6.2: Comparison of various approaches for SMTSTP on Set II instances

$n \times m$	**EM**			**PSO**			**GA**			**ABC**			**HGA**			**HABC**		
	APD		ATA	APD		ATA	APD		ATA	APD		ATA	APD		ATA	APD		ATA
	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)
50×3	0.60	1.49	4.10	10.05	11.31	4.09	0.15	0.36	0.17	0.05	0.21	0.04	0.00	0.05	0.33	0.00	0.01	0.20
50×4	0.64	1.46	4.17	9.28	10.44	4.17	0.16	0.38	0.17	0.08	0.25	0.04	0.02	0.06	0.37	0.00	0.01	0.24
50×9	0.77	1.53	4.47	8.85	10.34	4.48	0.13	0.30	0.22	0.07	0.24	0.07	0.01	0.05	0.59	0.00	0.01	0.43
100×3	0.86	1.76	16.71	10.79	11.39	16.73	0.15	0.33	0.45	0.07	0.28	0.14	0.02	0.05	0.95	0.00	0.02	0.58
100×4	0.89	1.76	17.17	9.83	10.40	17.20	0.16	0.28	0.46	0.10	0.27	0.14	0.01	0.04	1.09	0.00	0.01	0.69
100×9	0.89	1.51	19.27	9.86	10.69	19.30	0.11	0.21	0.60	0.10	0.26	0.23	0.01	0.04	2.15	0.00	0.01	1.39
150×3	1.19	2.09	36.16	10.40	10.77	36.26	0.17	0.27	0.84	0.13	0.30	0.31	0.03	0.03	2.13	0.01	0.02	1.47
150×4	1.12	2.01	37.78	10.47	10.84	37.89	0.13	0.21	0.87	0.10	0.27	0.33	0.02	0.03	2.48	0.00	0.02	1.69
150×9	1.05	1.74	43.31	9.70	10.33	43.42	0.11	0.15	1.16	0.12	0.25	0.55	0.02	0.02	4.98	0.00	0.01	3.53
200×3	1.35	2.26	72.22	10.34	10.58	72.49	0.13	0.20	1.37	0.14	0.31	0.58	0.02	0.02	3.83	0.01	0.03	2.86
200×4	1.29	2.14	75.94	9.89	10.22	76.21	0.13	0.18	1.43	0.16	0.30	0.64	0.02	0.02	4.50	0.01	0.02	3.35
200×9	1.21	1.91	89.08	9.62	10.06	89.35	0.09	0.12	1.89	0.14	0.26	1.07	0.02	0.01	9.40	0.01	0.01	6.89
Overall	0.99	1.81	35.03	9.92	10.62	35.13	0.13	0.25	0.80	0.10	0.27	0.35	0.02	0.03	2.73	0.00	0.01	1.94

Now, we compare the performance of HGA and HABC for SMTSTP-R on Set I and Set II instances. On Set I instances with 10 jobs, both of our approaches obtained the optimal solution in all 10 runs of all the instances. Therefore, results of these instances are not reported. The optimal solution of these instances were obtained from C.-T. Tseng through personal communication. These optimal solutions were found using exhaustive enumeration [124].

Table 6.3: Comparison of HGA and HABC for SMTSTP-R on Set I instances

$n \times m$	**EM**			**PSO**			**HGA </=/>HABC**					
	APD		ATA	APD		ATA	Best			Avg.		
	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	<	=	>	<	=	>
30×2	0.09	0.38	0.22	0.00	0.01	0.11	00	84	6	8	39	43
30×3	0.04	0.21	0.25	0.03	0.01	0.12	10	84	5	6	43	41
30×4	0.06	0.22	0.27	0.01	0.01	0.14	2	81	7	4	42	44
30×9	0.03	0.06	0.36	0.00	0.01	0.22	0	87	3	10	54	26
50×2	0.33	0.60	0.34	0.02	0.02	0.20	4	50	36	9	3	78
50×3	0.15	0.47	0.38	0.01	0.00	0.23	4	56	30	6	5	79
50×4	0.18	0.49	0.42	0.01	0.01	0.26	5	53	32	8	2	80
50×9	0.09	0.21	0.64	0.00	0.00	0.42	3	68	19	10	8	72
100×2	0.71	1.13	1.45	0.03	0.00	0.87	7	5	78	1	0	89
100×3	0.70	1.05	1.67	0.01	0.00	1.03	3	3	84	0	0	90
100×4	0.53	0.81	1.89	0.01	0.00	1.16	5	4	81	1	0	89
100×9	0.27	0.55	3.11	0.01	0.00	1.91	8	10	72	0	0	90
200×2	1.79	2.62	6.14	0.00	0.00	5.23	1	0	89	0	0	90
200×3	1.54	2.26	7.24	0.00	0.00	6.26	0	0	90	0	0	90
200×4	1.35	1.86	8.06	0.00	0.00	6.90	1	0	89	0	0	90
200×9	0.83	1.20	13.81	0.00	0.00	11.85	6	1	83	0	0	90
Overall	0.54	0.88	2.89	0.01	0.00	2.31	50	586	804	63	196	1181

Table 6.3 and Table 6.4 compare HGA and HABC for SMTSTP-R on Set I and Set II instances respectively. In addition to reporting APD and AET values, we have also reported the number of instances on which HGA is better (<), equal (=), or worse (>) than HABC in terms of best and average solution quality obtained over 10 runs. Please note that there are 90 instances in each group of instances having same n and m in Set I and 80 instances in each group having same n and m in Set II. These tables clearly show the superiority of HABC over HGA in terms of solution quality and running time both. Again, the performance of HGA is more close to

HABC on Set II instances than on Set I instances.

Table 6.4: Comparison of HGA and HABC for SMTSTP-R on Set II instances

$n \times m$	**EM**			**PSO**			**HGA </=/>HABC**					
	APD		ATA	APD		ATA	Best			Avg.		
	Best	Avg.	(Sec.)	Best	Avg.	(Sec.)	<	=	>	<	=	>
50×3	0.00	0.05	0.33	0.00	0.01	0.20	1	78	1	13	27	40
50×4	0.02	0.06	0.37	0.00	0.01	0.24	0	75	5	16	23	41
50×9	0.01	0.05	0.59	0.00	0.01	0.43	1	72	7	17	23	40
100×3	0.02	0.05	0.95	0.00	0.02	0.58	2	66	12	24	5	51
100×4	0.01	0.04	1.09	0.00	0.01	0.69	1	70	9	28	5	47
100×9	0.01	0.04	2.15	0.00	0.01	1.39	4	57	19	22	4	54
150×3	0.03	0.03	2.13	0.01	0.02	1.47	6	47	27	38	2	40
150×4	0.02	0.03	2.48	0.00	0.02	1.69	3	54	23	34	1	45
150×9	0.02	0.02	4.98	0.00	0.01	3.53	11	39	30	35	1	44
200×3	0.02	0.02	3.83	0.01	0.03	2.86	12	47	21	51	0	29
200×4	0.02	0.02	4.50	0.01	0.02	3.35	12	45	23	51	1	28
200×9	0.02	0.01	9.40	0.01	0.01	6.89	22	26	32	48	0	32
Overall	0.02	0.03	2.73	0.00	0.01	1.94	75	676	209	377	92	491

6.5 Conclusions

This chapter presented two hybrid metaheuristic approaches, viz. a genetic algorithm and an artificial bee colony algorithm for SMTSTP and SMTSTP-R. The solutions obtained through our metaheuristic approaches were improved further by a series of local searches. To the best of our knowledge, our approaches are the first metaheuristic approaches for SMTSTP-R. For SMTSTP, computational results established the superiority of our hybrid approaches over state-of-the-art approaches available in the literature in terms of solution quality and running time both. For SMTSTP-R, as no other metaheuristic approach exists, we have compared our two hybrid approaches and found that hybrid artificial bee colony algorithm based approach outperformed the hybrid genetic algorithm based approach in terms of solution quality and running time both.

Chapter 7

Registration Area Planning Problem

7.1 Introduction

In the last two decades the popularity of wireless network, particularly, cellular wireless networks has increased manifolds. The mobility of wireless devices within the network facilitated the user to remain connected with their personal and business activities all the time in an affordable price. This attracted a huge number of users to use these devices. This leads to an increased burden on the limited communication bandwidth available to the cellular wireless networks [148]. Actually, in cellular-based wireless communication networks, the wireless network is divided into contiguous geographical regions called cells and all mobile users must be located in one of these cells in order to remain connected with the network. Each cell is served by a base station, which is responsible for providing services to all mobile devices within that cell. A basic idea for utilizing the available bandwidth efficiently is to decrease the size of cells in the network which enables increased reuse of frequency among cells that are sufficiently far apart. But this increased reuse of frequency also increases the total number of control functions that a network has to perform in order to provide connectivity to all mobile devices using that network. This is due to the fact that as the cell size decreases, the number of cells needs to be increased for covering the same area. As a result, the number of mobile devices moving from one cell to another increases. Therefore, it has been always motivating for the researchers to design efficient mobility management schemes.

One such scheme, partitions all the cells of the network into clusters called registration areas (RAs). Each registration area (RA) consists of cells belonging to a contiguous geographical region. In this scheme whenever a mobile device is called, the exact cell in which the called mobile device is presently located is determined using a two stage function called location function. In the first stage, the location function determines the RA in which the called device is located, and in the second stage, the exact cell in this RA [149]. To implement this scheme, a centralized location database is maintained which keeps track of the current RA for each mobile device. This database is updated using a location update function whenever a mobile device migrates from one RA to another to reflect the new location of this mobile device. Whenever a mobile device is called, the RA in which the called device currently resides is determined first by accessing the location database. After that a process called paging determines the exact cell within the determined RA where the called device is located. During paging, all the base stations within the determined RA make an attempt to establish the connection with the called device. The called mobile device responds to the base station closest to it, thereby identifying the cell in which it currently resides. So the process of locating a mobile device generates quite a lot control traffic. The cost or efficiency of a location function depends on the amount of control traffic generated which in turn depends on the manner in which RAs are designed, i.e., the grouping of cells into RAs impacts significantly the costs incurred in location update (update cost) and paging (paging cost). The sum of update cost and paging cost is known as the location cost. The registration area planning problem (RAP) seeks a partitioning of cells into RAs that minimizes the total location cost. The RAP problem is $\mathcal{NP}$-hard because the graph partitioning problem, which has been proved $\mathcal{NP}$-hard in [57], reduces to it in polynomial time [74]. The update cost is proportional to the number of mobile devices moving from one RA to the other. The paging cost of an RA is proportional to the number of incoming calls to the mobile devices currently residing in that RA. When the size of RAs decreases, the paging cost also decreases, but the update cost increases. On the contrary, when the size of RAs increases, the paging cost increases, but the update cost decreases. If we have only one cell in each RA then paging cost is zero, but the update cost is highest. On the other hand, if only a single RA encompasses the whole network then update cost is zero, but paging cost will be highest. Hence, these two costs are inversely related and any strategy for partitioning the cells into RAs has to perform a balancing act between these two costs. However, as pointed out in [150, 151], these two costs are incomparable and any attempt to assign relative weights to these two costs will also be in vain as these weights show lots of variation from one part of the network to the other and from one network to the other. Therefore, standard practice in the literature is to include the paging cost in the constraints and minimize the update cost. As a result, RAP problem reduces to the problem of partitioning the cells into RAs in such a manner that minimizes the update cost without violating the paging bound of any of the RAs.

In the literature, a number of heuristics and metaheuristics have been proposed for the RAP and other related problems. Gamst [152] developed a graph theory based model to represent registration area planning and presented a greedy approach to solve it. But this greedy approach, in general, provided solutions far from the optimal. Markoulidakis and Sykas [153] proposed a model based on estimating hand-off and location update rates. It was related to subscriber crossing rates at the boundaries of RAs and cells. They used real data to check their model. Plehn [154] introduced a weighted greedy algorithm and showed the proposed algorithm to be better than the previously proposed approaches. Bejerano and Cidon [155, 156], Biesterfeld and Jobmann [157] worked on the concept of incorporating location prediction with the location management scheme. Bhattacharjee *et al.* [158] worked on two approaches called sequential intelligent paging (SIP) and parallel-to-sequential intelligent paging (PSIP), and used an occupancy probability vector to find out the cells to be paged out. These approaches were capable of reducing the paging signalling load, however, this reduction comes at the cost of extra processing power. Later, Bhattacharjee *et al.* [159] presented a two-stage solution approach that considers recurring costs and hybrid costs. The first stage tries to minimize the recurring costs defined as the partitioning of cells into location areas in such a way that it attempts to minimize the location update and paging cost, whereas second stage related to hybrid costs or network planning factors involve in handling hand-off costs between switches and cabling from the base station to the switch. Bhattacharjee *et al.* [160] also worked on integrating estimation of the RA boundary with identification of the cells in an RA. Wan and Lin [90] introduced a dynamic paging scheme that is based on movement information of an individual rather than a group of portables. This scheme relaxes the real-time location tracking to semi-real-time to balance the reduction in cost with ease of use. Bejerano *et al.* [161] developed a polynomial time approximation algorithm for the minimum bandwidth location management. Their algorithm based on relaxing the integrality constraints in their integer programming formulation is capable of finding a solution that is a lower bound on the optimal solution. A near optimal solution is obtained by rounding the fractional decision variable values.

Among the metaheuristics, a tabu search method [162], simulating annealing methods [150, 151, 162, 163] and genetic algorithms (GAs) [74, 162, 164] were developed for solving the RAP problem. A genetic algorithm was presented in [165] to solve a related problem of assigning cells to switches in the design phase of cellular wireless networks. Shyu *et al.* [84]

developed an ant colony optimization based approach to a related problem called cell assignment problem where cells are partitioned to minimize a hybrid cost comprising hand-off and cabling costs.

Since RAP is a grouping problem, Vroblefski and Brown [166] and James *et al.* [167] developed grouping genetic algorithms for the solving the RAP problem. Both these GAs are based on grouping genetic algorithm proposed in [30] and have several common features such as initial population generation, crossover, selection method of parents for crossover. Both GAs use the rank-based roulette wheel selection method and employ two point crossover designed by Falkenauer for grouping problems. However, the mutation was used in [166], whereas it was not used in [167]. In addition, [167] used a local search method, i.e., improvement operator to further improve the solution quality. Throughout this chapter, generational genetic algorithm of [167] without local search will be referred as GGA and with local search will be referred as HGGA, and the generational genetic algorithm of [166] will be referred as GGARAP.

In this chapter, we present another hybrid grouping genetic algorithm, i.e., a steady-state grouping genetic algorithm (SSGGA) for the RAP problem. Solution encoding, selection method, genetic operators, viz. crossover and mutation, population replacement policy applied in SSGGA are all different from those of GAs of [166] and [167]. We have also applied a local search to further improve the solution quality of certain selected solutions obtained by SSGGA. We have compared SSGGA with GGA, HGGA and GGARAP. Computational results clearly show the benefit of our approach, especially on large instances over these approaches in terms of solution quality and execution time both.

The rest of this chapter is organized as follows: Section 7.2 formally defines the RAP problem. Section 7.3 describes SSGGA for the RAP problem. Section 7.4 reports and analyzes the computational results. Finally, Section 7.5 outlines some concluding remarks.

7.2 Problem definition

In this section, we follow the same notational conventions as used in [166] and [167] for defining the RAP problem formally. Suppose PB is the paging bound for any RA, i.e., the maximum number of incoming calls allowed to any RA per unit of time and $page_i$ is the average number of incoming calls to cell i per unit of time. Also, suppose the average number of wireless devices crossing the boundary between cells i and j per unit of time is denoted by w_{ij}. The location update cost is proportional to the number of wireless devices crossing RA borders. The

RAP problem seeks a partition of cells into RAs that minimizes the location update cost for the network while satisfying the paging bound for each RA and preset constraints, if there is any. Each preset constraint dictates a pair of cells to be either always in the same RA or always in the different RAs. Actually, preset constraints arise from practical considerations. For example, if a pair of adjacent cells have a very high frequency of devices crossing the boundary between them, then the two cells belonging to this pair need to be together in the same RA. On the other hand, future expansion plans may force the two cells to be always in different RAs. We have represented preset constraints by v_{ij}s in our formulation. $v_{ij} = 0$ in case cells i and j have to be together in the same RA, $v_{ij} = 1$ in case they have to be accommodated in different RAs, and $v_{ij} = *$ otherwise. We have also made use of binary variables x_{ir} to indicate whether cell i belongs to RA r ($x_{ir} = 1$) or not ($x_{ir} = 0$) and binary variables y_{ij} to indicate whether cells i and j are in the same RA ($y_{ij} = 0$) or not ($y_{ij} = 1$). Finally, assume N represents the number of cells in the network and K represents the number of RAs in a solution. With the help of these notational conventions, the RAP problem can be formally defined as follows:

$$Minimize\ \ Cost\ = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} y_{ij}\, w_{ij} \tag{7.1}$$

subject to

$$\sum_{i} x_{ir} page_i \leq PB, \quad r = 1, \ldots, k \tag{7.2}$$

$$y_{ij} = v_{ij}, \qquad \forall v_{ij} = 0 \ \ or \ \ 1 \tag{7.3}$$

$$y_{ij} = 1 - \sum_{r} x_{ir}\, x_{jr}, \quad i, j = 1, \ldots, N \tag{7.4}$$

$$\sum_{r} x_{ir} = 1, \quad i = 1, \ldots, N \tag{7.5}$$

$$x_{ir} \in \{0, 1\} \tag{7.6}$$

Equation (7.1) is the objective function of the RAP problem that seeks to minimize the total location update cost. Equation (7.2) enforces the constraint that the total paging cost of each RA should not exceed the given paging bound PB. Preset constraints, if any, are enforced by

Equation (7.3). Equation (7.4) ensures that $y_{ij} = 0$ if cells i and j are in the same RA, otherwise $y_{ij} = 1$. Equation (7.5) enforces the constraint that each cell must belongs to one and only one RA. Equation (7.6) restricts variables x_{ir} to binary values, i.e., 0 & 1 only. Preset constraints are considered by [166] in their problem formulation. On the other hand, [167] ignored the preset constraints altogether.

7.3 Steady-state grouping genetic algorithm for RAP problem

Our approach is combination of steady-state grouping genetic algorithm (SSGGA), repair heuristic and the local search. The salient features of our SSGGA approach for RAP are described in subsequent subsections.

7.3.1 Chromosome representation

We have adapted the representation scheme used in [31] for the one-dimensional bin-packing problem to the RAP problem. Each solution in this scheme is represented as a set of non-empty RAs in the network, and each RA itself is a set of cells. There is no ordering among RAs in a solution and also no ordering among cells belonging to the same RA. In a solution, each and every cell is assigned to exactly one of the RAs. Number of cells in an RA can vary from one RA to the other and number of RAs can vary from one solution to the other. For example, suppose a solution for a network, comprising 10 cells, consists of three RAs where cells 1, 3 and 9 belong to one RA, cells 2, 4 and 7 belong to another RA, and cells 5, 6, 8 and 10 belong to yet another RA. This solution is represented as { {1, 3, 9}, {2, 4, 7}, {5, 6, 8, 10} }. With such a scheme there is no redundancy, i.e., each solution is represented uniquely. The representation schemes used in [166] and [167] suffer from the problem of redundancy as a numbering of different RAs present in the solution is arbitrary, i.e., interchanging the numbers assigned to two RAs in group part and accordingly changing the numbers assigned to their respective cells in the cell part will not change the solution represented, so a solution can be represented in more than one way.

Moreover, our genetic operators, viz. crossover and mutation are designed in such a manner so that the child solution obtained after their application does not depend on the ordering among RAs in the parent solution(s). As the genetic operators used in [166] and [167] are based on the standard genetic operators presented by [29] for grouping problems, the child solution obtained after the application of these genetic operators does depend on the ordering of RAs in the parent solution(s).

7.3.2 Fitness

Fitness function used in our genetic algorithm is same as the objective function of the RAP problem (Equation (7.1)). Hence, a solution is considered to be more fit than the other, if it has a lesser value in comparison to other as per the objective function. $fitness(S)$ function returns the fitness of the solution S.

7.3.3 Crossover

Our crossover operator is inspired by the crossover operator presented in [31] for one dimensional bin packing problem. Our crossover operator starts with an empty child, and, then an iterative process follows. During each iteration, one of the two parents is selected uniformly at random and most tightly filled connected RA in the selected parent is deleted from it and copied to the child. Cells belonging to the just copied RA are also deleted from various RAs of other parent. By a most tightly filled connected RA, we mean an RA that has the maximum sum total of average number of incoming calls per unit of time to its constituent cells and any cell belonging to this RA is adjacent to at least one more cell belonging to this RA. If no connected RA exists in the selected parent then iterative process moves to the next iteration. Anyway, iterative process is repeated for a maximum of $min(R_1, R_2) - R_{left}$, where R_1 and R_2 are the number of RAs in two selected parents p_1 and p_2 respectively and R_{left} is some fixed constant. Here, the purpose of R_{left} is to always leave some cells unassigned. Clearly, our crossover operator tries to preserve as far as possible the best RAs while creating child solution. It is pertinent to mention here that RAs consist of contiguous geographical regions and that is why only connected RAs are transferred to the partial child solution.

After our crossover operator, some cells are left unassigned in the generated child. Such left out cells are assigned to the child solution with the help of *repair heuristic* (Algorithm 7.3). The pseudo-code of crossover operator is given in Algorithm 7.1, whereas Figure 7.1 explains the process of generating a child solution through the crossover operator with the help of an example.

With the help of Figure 7.1, we illustrate our crossover operator. In this figure, each hexagon represents a cell and the number inside the cell represents the index of that cell. In the 19 cell network shown, each cell number and the neighbours of each cell are assigned through the H-mesh [168] construction method. PC means paging cost for corresponding cell, and PB means paging bound for an RA. First of all, the figure shows the two parents *Parent 1* and *Parent*

Algorithm 7.1: The pseudo-code of crossover operator

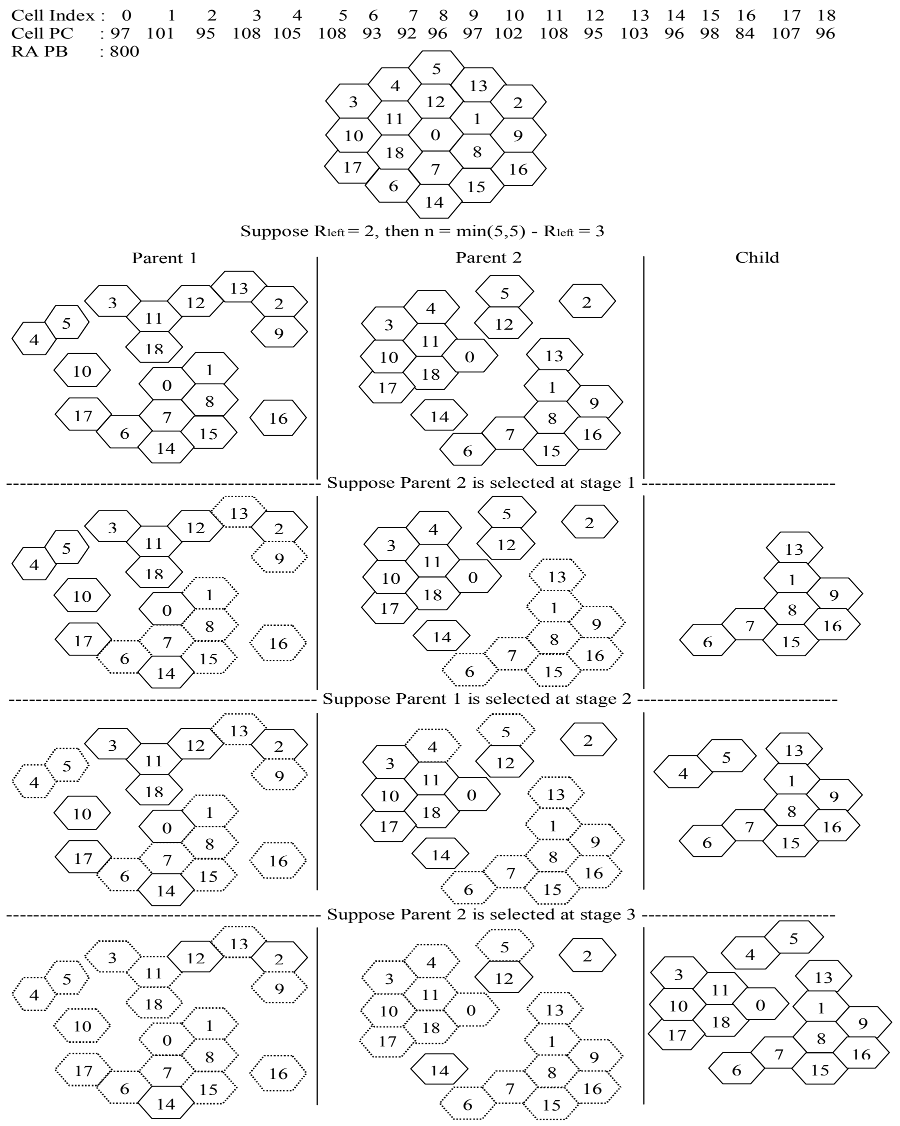
```
1: $csol \leftarrow \phi$;
2: $n \leftarrow min(R_1, R_2) - R_{left}$;
3: $i \leftarrow 1$;
4: **while** $(i \leq n)$ **do**
5:     Generate a random number $u01$ such that $0 \leq u01 \leq 1$;
6:     **if** $(u01 \leq 0.5)$ **then**
7:         **if** (A connected RA exists in $p_1$) **then**
8:             Select the most tightly filled and connected RA $r$ from parent $p_1$;
9:             $p_1 \leftarrow p_1 - \{r\}$;
10:            Delete the cells belonging to RA $r$ from various RAs of parent $p_2$;
11:        **end if**
12:    **else**
13:        **if** (A connected RA exists in $p_2$) **then**
14:            Select the most tightly filled and connected RA $r$ from parent $p_2$;
15:            $p_2 \leftarrow p_2 - \{r\}$;
16:            Delete the cells belonging to RA $r$ from various RAs of parent $p_1$;
17:        **end if**
18:    **end if**
19:    $csol \leftarrow csol \cup \{r\}$;
20:    $i \leftarrow i + 1$;
21: **end while**
22: **return** $csol$;
```

2 selected through binary tournament selection (BTS) method. Suppose grouping of cells into RAs in two parents are as follows.

Parent 1: RA_1= {4, 5}, RA_2= {2, 3, 9, 11, 12, 13, 18}, RA_3= {10} RA_4= {16}, RA_5= {0, 1, 6, 7, 8, 14, 15, 17}

Parent 2: RA_1= {5, 12}, RA_2= {0, 3, 4, 10, 11, 17, 18}, RA_3= {14} RA_4= {2}, RA_5= {1, 6, 7, 8, 9, 13, 15, 16}

RAs in these two parents are numbered for the sake of illustration only, otherwise, there is no ordering among RAs. Suppose at stage 1, *Parent 2* is selected at random, and the most tightly filled connected RA of it viz. RA_5 is selected and added to the child solution. Now, RA_5 is removed from *Parent 2*, and all the cells which belong to this RA are removed from *Parent*



Cell Index :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Cell PC	: 97	101	95	108	105	108	93	92	96	97	102	108	95	103	96	98	84	107	96
RA PB	: 800																		

Suppose Rleft = 2, then n = min(5,5) - Rleft = 3
Parent 1
Parent 2
Child
Suppose Parent 2 is selected at stage 1
Suppose Parent 1 is selected at stage 2
Suppose Parent 2 is selected at stage 3

Figure 7.1: Creation of child chromosome from parents using crossover operator

1. In the Figure 7.1, dotted cell indicates that this cell has been removed from corresponding RA. At stage 2, suppose *Parent 1* is selected. Here the most tightly filled connected RA is RA_1, because RA_2 and RA_5 became disconnected after stage 1. Finally at stage 3, suppose *Parent 2* is selected, and RA_2 which is most tightly filled connected RA is copied to the child. As $n = 3$, the crossover stops after this stage.

Our crossover operator can be considered as an extension of the uniform crossover operator for grouping problem which uses greedy criterion while choosing RAs, whereas crossover operator used in [166] and [167] is a traditional 2-point crossover designed by [29] for grouping problems. We have already discussed the advantages of uniform crossover over traditional 1- or 2-point crossover in Chapter 1.

7.3.4 Mutation

Our mutation operator starts by deleting all the singleton RAs (RAs having only one cell) in the parent solution. After that *25%* RAs from remaining RAs are chosen uniformly at random and deleted from parent solution. Now, remaining RAs are copied to the child solution. Obviously, this will leave many cells unassigned in child solution. Such left out cells are reassigned with the help of *repair heuristic*.

Our genetic algorithm uses crossover and mutation in a mutually exclusive way, i.e, during each generation either crossover is used or mutation is used, but not both. Crossover is used with probability π_c, otherwise mutation is used. The reason for this decision lies in the fact that from their very design crossover and mutation are complimentary. Whereas crossover tries to pass, as far as possible, good RAs from parents to child in a bid to create high quality solution, mutation disregards quality of an RA while deleting it in a bid to generate a diverse solution. If both are used together one after the other then it is highly likely that the resulting solution has neither high quality nor has sufficient diversity.

7.3.5 Selection of parent(s)

A single parent for mutation and two parents for crossover are selected with the help of binary tournament selection method where the candidate with better fitness has the probability of selection π_{bt}. A copy of the solution selected is created, and this copy is used as a parent in mutation or crossover as the case may be.

7.3.6 Replacement policy

We have adopted steady-state population model [21] for our genetic algorithm. In this model, during each generation a single solution is generated which is checked for uniqueness against the existing solutions in the population. If the newly generated solution is different from all existing population members, then it is included into the population by replacing the worst solution of population, otherwise this newly generated solution is discarded. This is quite different from the traditional generational replacement strategy used in [166] and [167] where the entire parent population is replaced during each generation with an equal number of newly created child solutions.

7.3.7 Initial population generation

Our initial solution generation method is a modified version of the initial solution generation method used in [167]. In [167], for GGA each initial solution which contains some RAs whose paging level are less than or equal to half of the paging bound, is passed through the *repair operator* which tries to reassign the cells of such RAs into other RAs who can accommodate them. Additionally, in case of HGGA, each initial solution is passed through the *improvement operator* in order to further improve the solution quality. But in our method, we have not applied *repair operator* and *improvement operator* on generated initial solution. Actually, the manner in which an initial solution is generated, *repair operator* can never improve it. Hence, applying *repair operator* to an initial solution was a mistake in [167] which unnecessarily increases the computational burden. We have also observed that applying *improvement operator* on initial population leads to a significant decrease in diversity of initial population, and hence we decided against its use on initial population.

Each initial solution is generated by an iterative process. Initially, we start with an empty solution, I_s, and a set C of unassigned cells contains all the cells. During each iteration, a cell is picked uniformly at random from C and an RA is created containing that cell. Neighbouring unassigned cells of this cell is determined and added to an empty set, say U. One of the cells, which can still fit into this RA without violating any constraint is deleted from U and added into this RA. Unassigned neighbours of this newly added cell are added to U. Again, One of the cells which can still fits into the RA without violating any constraint is deleted from U and added to this RA. This process continues till either it is not possible to add any more cells from U to RA without violating any constraint or U becomes empty. After that, another iteration begins. This process which creates a new RA during each iteration continues till set of unassigned cells becomes empty, i.e., all cells are assigned. Basically an unassigned cell has to satisfy three constraints before it can be added to the RA. First, it should be adjacent to at least one cell in the RA. Second, the paging level of RA should not exceed the paging bound after addition of the cell to the RA. Third, addition of the cell should not violate any preset constraints (if any). As we always add a cell from U which is the set of those unassigned cells which are adjacent to at least one cell in RA under construction, so first constraint is automatically taken care of. However, we need to explicitly ensure the second and third constraints.

Each newly generated solution is checked for uniqueness against the initial population members generated so far, and if it is unique, then it is included in the population, otherwise it is discarded. The pseudo-code for generating an initial solution is given in Algorithm 7.2, where *random(C)* is a function that returns a cell from the set C randomly.

Algorithm 7.2: Pseudo-code for generating an initial solution

```
    ▷ $I_s$ is the new initial solution to be generated
    ▷ $N_b[c]$ gives all the neighbourhood cells of cells $c$
 1: $I_s \leftarrow \Phi$;
 2: $r \leftarrow 0$;
 3: while ($C \neq \emptyset$) do
 4:     $U \leftarrow \Phi$;
 5:     $r \leftarrow r + 1$;
 6:     $RA_r \leftarrow \Phi$;
 7:     $c \leftarrow random(C)$;
 8:     $RA_r \leftarrow RA_r \cup \{c\}$;
 9:     $PC(RA_r) \leftarrow PC(c)$;
10:     $U \leftarrow \{N_b[c] \cap C\}$;
11:     $C \leftarrow C \backslash \{c\}$;
12:     while ($U \neq \Phi$ and $PC(RA_r) \leq PB$) do
13:         $\bar{c} \leftarrow random(U)$;
14:         if ($PC(RA_r) + PC(\bar{c}) \leq PB$) then
15:             $RA_r \leftarrow RA_r \cup \{\bar{c}\}$;
16:             $PC(RA_r) \leftarrow PC(RA_r) + PC(\bar{c})$;
17:             $C \leftarrow C \backslash \{\bar{c}\}$;
18:             $U \leftarrow U \backslash \{\bar{c}\}$;
19:             $U \leftarrow U \cup \{N_b[\bar{c}] \cap C\}$;
20:         else
21:             $U \leftarrow U \backslash \{\bar{c}\}$;
22:         end if
23:     end while
24:     $I_s \leftarrow I_s \cup RA_r$;
25: end while
26: return $I_s$;
```

7.3.8 Repair heuristic

Both the genetic operators viz. crossover and mutation leave some cells unassigned in the newly generated offspring. Repair heuristic assigns these left out cells to the newly generated

Algorithm 7.3: Repair heuristic

```
    ▷ $S$ is a solution in which cells to be assigned
    ▷ $C$ is a set of cells in the network
    ▷ $N_n[c]$ gives all the neighbourhood cells of cells $c$
1:  $W_s \leftarrow C - S$;
2:  while ($W_s \neq \emptyset$) do
3:      $U_n \leftarrow \Phi$;
4:      $c \leftarrow random(W_s)$;
5:      $r \leftarrow best\_ra(c, S)$;
6:      $r \leftarrow r \cup \{c\}$;
7:      $PL(r) \leftarrow PL(r) + PC(c)$;
8:      $W_s \leftarrow W_s \backslash \{c\}$;
9:      $U_n \leftarrow N_n[c] \cap W_s$;
10:     while ($U_n \neq \emptyset$ $and$ $PL(r) \leq PB$) do
11:         $c \leftarrow random(U_n)$;
12:         if ($PC(r) + PC(c) \leq PB$) then
13:             $r \leftarrow r \cup \{c\}$;
14:             $PL(r) \leftarrow PL(r) + PC(c)$;
15:             $W_s \leftarrow W_s \backslash \{c\}$;
16:             $U_n \leftarrow U_n \backslash \{c\}$;
17:             $U_n \leftarrow U_n \cup N_n[c] \cap W_s$;
18:         else
19:             $U_n \leftarrow U_n \backslash \{c\}$;
20:         end if
21:     end while
22: end while
23: return $S$;
```

offspring S. However, it begins by deleting all singleton RAs, if there exists any, from S and then computing the set of unassigned cells W_s. After this an iterative process ensues. During each iteration, an unassigned cell c is selected uniformly at random from W_s and assigned to an RA r that has at least one cell adjacent to c and where c fits the best without violating any constraints, i.e., where gap left between current paging level and paging bound after assignment is smallest and no preset constraint, if there is any, gets violated. If no such RA exists then a new RA r initially containing only c is created and added to the solution. After that all those unassigned neighbouring cells of c, whose paging cost is less than or equal to the difference between current paging level of r and paging bound are added to a set say U_n. All cells in U_n are tried one-by-one in some random order for insertion into r. If any cell from U_n gets inserted into r then all its unassigned neighbouring cells are also included into U_n. All cells in U_n are now rechecked to determine whether they can still fit in r. If a cell can no longer fit into r, then it is removed from U_n. This process continues till U_n becomes empty. After that another iteration of the repair heuristic starts. This iterative process continues as long as not all cells are assigned, i.e., as long as W_s is non-empty.

The pseudo-code of repair heuristic is given in the Algorithm 7.3 where $random(U_n)$ is a function that returns a cell of the set U_n randomly and $PL(r)$ gives the current paging level of RA r. $best\_ra(c, S)$ is another function that returns an RA r in solution S where c fits best without violating any constraint as discussed in the previous paragraph. If no such RA exists then a new RA containing single cell c is created and added to S by this function which also returns this newly added RA.

7.3.9 Local search

To further improve the solution quality after application of genetic operators, a local search is applied. Our local search is a modified version of *improvement operator* of [167]. Local search follows an iterative process. Each iteration of local search begins by determining all the current border cells (a cell which shares the border with at least one cell belonging to another RA is called a border cell) in the solution under consideration. All the border cells are tried for possible movement to the neighbouring RAs and a feasible move that reduces the cost of the solution by maximum amount is performed. Movement of a cell from its current RA to its neighbouring RA is considered feasible if it respects all the constraints, viz. the paging level of RA r in which cell will be added must be less than or equal to paging bound after addition, the RA from which the cell will be moved must remain connected after movement and no present constraint, if any, gets violated in the process. If there exists no feasible move that can reduce the cost of the solution then local search stops, otherwise another iteration of local search begins. We have allowed our local search to execute for a maximum of Max_{it} iterations. The pseudo-code of local search is given in Algorithm 7.4. In Algorithm 7.4, the function *Border_cell(S)* returns all the border cells of solution S and *Best_move(br_c)* returns the best feasible move from all the border cells br_c.

As mentioned in the previous paragraph our local search is a modified version of improvement operator used in [167]. Here we will highlight the difference between our local search and improvement operator. In [167], improvement operator is applied till no improving move is possible whereas our local search is applied for a maximum of Max_{it} iterations. To block the reverse movement of border cells which are moved in recent iterations, a tabu list was used

Algorithm 7.4: Local search(S)

```
    ▷ S is solution on which local search to be applied
 1: $i \leftarrow 1$;
 2: $N_f \leftarrow 0$;
 3: $C_f \leftarrow fitness(S)$;
 4: $br_c \leftarrow Border\_cell(S)$;
 5: $b_m \leftarrow Best\_move(br_c)$;
 6: $N_f \leftarrow fitness(b_m)$;
 7: while ($i \leq Max_{it}$) do
 8:     if ($N_f < C_f$) then
 9:         $S \leftarrow b_m$;
10:         $i \leftarrow i + 1$;
11:         $C_f \leftarrow N_f$;
12:     else
13:         $return\ S$;
14:     end if
15:     $br_c \leftarrow Border\_cell(b_m)$;
16:     $b_m \leftarrow Best\_move(br_c)$;
17:     $N_f \leftarrow fitness(b_m)$;
18: end while
19: return $S$;
```

in the improvement operator. Only those border cells are considered for movement which are not in the tabu list. Our local search does not use any tabu list and reverse movement of cell is possible without any restriction. In [167], each initial solution and generated offspring is passed through the improvement operator, whereas our local search is applied only on a solution obtained after application of genetic operators and that too when the difference of fitness of this solution and the fitness of the best solution found so far is less than or equal to IM times the fitness of the best solution found so far. As mentioned already, we did not use local search while generating initial population because of diversity considerations. All these modifications reduce the proportion of time spent on local search while executing our approach.

The pseudo-code of our SSGGA approach for the registration area planning problem is given in Algorithm 7.5, where *Crossover()* operator returns the solution as per Algorithm 7.1, *Mutation()* operator is used to maintain diversity in the population and explained in Section 7.3.4, *Repair()* operator repairs the child solution as per Algorithm 7.3 and the *Local search()* function returns the solution by using Algorithm 7.4.

Algorithm 7.5: The pseudo-code of grouping genetic algorithm for RAP

```
1:  Generate initial population;
2:  $B_{sol} \leftarrow$ Best solution of initial population;
3:  $f_{B_{sol}} \leftarrow fitness(B_{sol})$;
4:  while (Termination condition does not hold) do
5:      Generate a random number $r01$ such that $0 \leq r01 \leq 1$;
6:      if ($r01 \leq \pi_c$) then
7:          Select two parents $p_1$ and $p_2$ with the help of binary tournament selection method;
8:          $C_s \leftarrow Crossover(p_1, p_2)$;
9:      else
10:         Select a parent $p$ with the help of binary tournament selection method;
11:         $C_s \leftarrow Mutation(p)$;
12:     end if
13:     $C_s \leftarrow Repair(C_s)$;
14:     $f_{C_s} \leftarrow fitness(C_s)$;
15:     if (($f_{B_{sol}} - f_{C_s}) < (f_{B_{sol}} \times IM$)) then
16:         $C_s \leftarrow Local\ search(C_s)$;
17:         $f_{C_s} \leftarrow fitness(C_s)$;
18:     end if
19:     if ($C_s$ is different from current population members ) then
20:         Replace worst member of population with $C_s$;
21:         if ($f_{C_s} < f_{B_{sol}}$) then
22:             $B_{sol} \leftarrow C_s$;
23:             $f_{B_{sol}} \leftarrow f_{C_s}$;
24:         end if
25:     end if
26: end while
27: return $B_{sol}$;
```

7.4 Computational results

In this section, we present the computational results of three versions of our steady-state grouping genetic algorithm (SSGGA) for the registration area planning (RAP) problem and compare them with those of three different genetic algorithms proposed in [166] and [167]. We have implemented our SSGGA versions in C language and executed them on a Linux operating system based computer with `Intel core i5-2400` processor and 4GB RAM. `gcc 4.6.3-2` compiler with `O3` flag has been used to compile the C programs for different SSGGA versions. With all the three SSGGA versions, we have used a population of $Pop = 400$ solutions, the probability of selection of a better solution in binary tournament selection π_{bt} is set to 0.9, crossover probability π_c is set to 0.8 (hence mutation is used with probability 0.2 as crossover and mutation are used mutually exclusively), R_{left} in crossover operator is set to 2, Max_{it} and IM in local search are set to 6 and 0.02 respectively. All these parameter values have been chosen empirically after a large number of trials. These parameter values provide good results on all instances, though they may not be the optimal parameter values for all instances.

As GGARAP of [166] considered the preset constrains, did not employ any improvement operator and generated 6000 solutions in each run. In order to compare our SSGGA approach with GGARAP, we have also considered the preset constraints, did not apply the local search and allowed our genetic algorithm to generate 6000 solutions only. Hereafter, this version of SSGGA approach, which considers preset constraints, does not use local search and terminates after generating 6000 solutions, will be referred to as SGGARAP. On the other hand, GGA of [167] ignored the preset constraints, did not apply the improvement operator and generated 10000 solutions in each run. The corresponding SSGGA version that ignores preset constraints, does not use local search and terminates after generating 10000 solutions will be referred to as SGGA hereafter. Whereas HGGA approach of [167] ignored the preset constraints, but uses the improvement operator which is described already. HGGA also generated 10000 solutions in each run. The corresponding SSGGA version that ignores preset constraints, uses the local search and generates 10000 solutions will be referred to as SGGA_LS in the sequel. Like GGARAP, GGA and HGGA, all the three SSGGA versions (SGGARAP, SGGA and SGGA_LS) are executed 10 independent times on each instance considered in this chapter.

Due to unavailability of test instances used in [166] and [167], we have generated a similar set of instances by simulating H-mesh [168] for various network sizes. [166] and [167] considered instances with 19, 37, 61 and 91 cells for testing their approaches. In this chapter, we have extended the network size up to 127 cells. So instances of 5 different network sizes 19, 37, 61, 91 and 127 cells have been considered for evaluating different approaches in this chapter. The layout of 19 cells H-mesh is shown in Figure 7.1. While generating each instance, exactly like [166, 167], it is assumed that paging cost of each cell and crossing intensity at any border between two cells follow a normal distribution with mean =100 & variance in {20, 40 and 60}, paging bound for each RA is taken to be 800 and either 0 or 1 or 2 preset constraints are inserted randomly. For each combination of network size and variance, 10 different instances are generated leading to a grand total of 150 instances. All these instances are publicly available at `http://dcis.uohyd.ernet.in/~alokcs/RAP.zip`

To ensure a fair comparison, we have re-implemented the GGARAP approach of [166] and GGA & HGGA approach of [167] in C and executed these approaches on the test instances created by us on the same system after compiling them with the same compiler with the same compile options as used for our approaches. For all the three approaches, all the parameters are set to the same values as reported in their respective papers.

Computational results comparing SGGARAP with GGARAP, SGGA with GGA and SGGA_LS with HGGA are presented in Table 7.1, Table 7.2 and Table 7.3 respectively. These tables report on each of the 10 instances for each combination of network size (column N) & variance and for each of the two methods compared, the best solution (column Best), average solution (column Mean), standard deviation of solution values (column SD) and the average execution time (AET) over 10 independent runs. In these tables, the best solution and average solution marked with an asterisk ('*') indicate that our approach is worse than the respective approach for that particular instance, whereas the solutions in bold indicate that our approach is better than the respective approach and rest of the results indicate that results are same for both the approaches.

To show the statistical significance of the results obtained by three SSGGA versions over those of their respective competitors, we have performed the *Mann-Whitney U test*. Table 7.4 presents the results of two-tailed *Mann-Whitney U test* between SGGARAP & GGARAP, between SGGA & GGA and between SGGA_LS & HGGA for instances with variance 20, 40 and 60 respectively. For performing this test, we have used the online calculator available at `http://www.socscistatistics.com/tests/mannwhitney/Default2.aspx`. We have taken the significance criteria to be equal to 5% (p-$value \leq 0.05$) and the corresponding critical U-*value* is 23. In Table 7.4, for the sake of brevity, U-*value* and p-*value* between GGARAP and SGGARAP is reported under the heading G_SRAP, between GGA and SGGA under the heading G_S and between HGGA and SGGA_LS under the heading G_SLS. The results with mark 'a' indicate that the two corresponding approaches found same value in all 10 runs rendering the result of *Mann-Whitney U test* meaningless.

Table 7.1 clearly shows the superiority of SGGARAP over GGARAP. In terms of best solution quality, out of a total of 150 instances, SGGARAP is better than GGARAP on 102 instances, worse than GGARAP on 1 instance and equal to GGARAP on remaining 49 instances. In terms of average solution quality, SGGARAP is better than GGARAP on 147 instances and equal to GGARAP on remaining 3 instances. SGGARAP is nearly two times faster than GGARAP on large instances. Standard deviation of solution values for SGGARAP is less than

GGARAP showing the robustness of SGGARAP. If we look under the heading G_SRAP in Table 7.4 for each of the three different variances 20, 40 and 60, then we can observe that out of 150 instances, results of SGGARAP are statistically significant with respect to GGARAP on 123 instances, on 24 instances they are not significant and on remaining 3 instances results of *Mann-Whitney U test* are meaningless.

The computational results reported in Table 7.2 make the superiority of SGGA in comparison to GGA quite evident. Out of 150 instances, SGGA is better than GGA on 118 instances in terms of the best solution, whereas on remaining 32 instances, both the approaches obtained the same value for the best solution. In terms of average solution quality, SGGA is better than GGA on 140 instances and on remaining 10 instances have the same value. Average execution time taken by SGGA is also less, when compared to GGA, on most of the instances. SGGA also has less standard deviation of solution values. *Mann-Whitney U test* between SGGA and GGA (Table 7.4) show that the results of SGGA with respect to GGA are statistically significant on 122 instances, whereas on 12 instances they are not statistically significant. On remaining instances results of *Mann-Whitney U test* are meaningless.

The computational results of SGGA_LS and HGGA are shown in Table 7.3. In terms of best solution, out of 150 instances, the best solution obtained by SGGA_LS is better than HGGA on 48 instances, worse than HGGA on 2 instances and same as HGGA on remaining 100 instances. The average solution quality of SGGA_LS is better than HGGA on 98 instances, worse than HGGA on 3 instances and same as HGGA on remaining 49 instances. SGGA_LS is more than two times faster than HGGA on large instances. Standard deviation of solution values is also less for SGGA_LS on most instances. The *Mann-Whitney U test* between SGGA_LS and HGGA (Table 7.4) indicates that results of SGGA_LS with respect to HGGA are statistically significant on 57 instances, whereas on 44 instances they are not significant. On remaining 49 instances the results of *Mann-Whitney U test* are meaningless as both the approaches obtained the same solution in all 10 runs. Actually, use of local search makes the difference between the performance of two approaches blurred on small instances as both the approaches obtain either the same results or nearly the same results. However, the results are clearly significant on large instances. If we consider the largest instances with 91 and 127 cells, results of SGGA_LS are significant on 43 instances and are not significant on remaining 17 instances out of a total of 60 instances. Here it is pertinent to mention that solution quality is not the only parameter that favour our approach. Our approach is also faster on large instances.

Table 7.1: Results of GGARAP and SGGARAP on problem size 19, 37, 61, 91 and 127 for variance 20, 40 and 60

N	Variance 20								Variance 40								Variance 60							
	GGARAP				SGGARAP				GGARAP				SGGARAP				GGARAP				SGGARAP			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
19	1244	1247.00	6.00	0.02	1244	**1244.00**	0.00	0.04	1214	1219.30	8.67	0.02	1214	**1214.00**	0.00	0.04	1168	1179.00	17.37	0.02	1168	**1168.00**	0.00	0.04
19	1301	1310.00	27.00	0.03	1301	**1301.00**	0.00	0.04	1276	1279.10	4.85	0.02	1276	**1276.00**	0.00	0.04	1229	1243.20	14.29	0.02	1229	**1229.00**	0.00	0.04
19	1283	1298.50	41.44	0.02	1283	**1283.00**	0.00	0.04	1259	1268.40	6.26	0.02	1259	**1259.00**	0.00	0.04	1251	1253.10	3.21	0.02	1251	**1251.00**	0.00	0.04
19	1266	1266.30	0.90	0.02	1266	**1266.00**	0.00	0.04	1290	1290.20	0.40	0.02	1290	**1290.00**	0.00	0.03	1231	1250.20	19.79	0.02	1231	**1231.00**	0.00	0.04
19	1275	1277.30	5.02	0.02	1275	**1275.00**	0.00	0.04	1270	1306.60	54.74	0.03	1270	**1270.00**	0.00	0.05	1248	1264.00	40.91	0.02	1248	**1248.00**	0.00	0.04
19	1309	1312.90	4.25	0.02	1309	**1309.00**	0.00	0.03	1235	1239.90	9.65	0.02	1235	**1235.00**	0.00	0.03	1264	1269.70	9.87	0.02	1264	**1264.60**	1.20	0.03
19	1395	1395.00	0.00	0.02	1395	1395.00	0.00	0.04	1248	1249.00	3.00	0.02	1248	**1248.00**	0.00	0.04	1338	1388.90	40.91	0.03	1338	**1338.00**	0.00	0.04
19	1293	1312.50	29.83	0.02	1293	**1293.00**	0.00	0.04	1286	1286.00	0.00	0.02	1286	1286.00	0.00	0.04	1221	1227.00	11.65	0.02	1221	**1221.00**	0.00	0.04
19	1276	1279.50	3.07	0.02	1276	**1276.00**	0.00	0.04	1250	1258.70	13.39	0.02	1250	**1250.00**	0.00	0.04	1244	1253.90	10.38	0.02	1244	**1244.00**	0.00	0.03
19	1257	1267.90	9.88	0.02	1257	**1257.00**	0.00	0.04	1255	1255.00	0.00	0.02	1255	1255.00	0.00	0.04	1215	1216.50	3.38	0.02	1215	**1215.00**	0.00	0.04
37	2961	3026.30	56.95	0.06	**2927**	**2929.40**	4.80	0.04	2871	2922.40	45.45	0.05	2871	**2873.70**	3.93	0.06	2900	2984.20	54.04	0.05	2900	**2900.00**	0.00	0.05
37	3003	3073.80	61.54	0.06	**2981**	**2988.10**	8.41	0.05	2999	3063.20	44.74	0.06	**2991**	**2992.00**	1.00	0.05	2899	3142.90	124.77	0.06	2899	**2899.00**	0.00	0.06
37	2952	3028.50	82.35	0.05	2952	**2953.20**	3.60	0.05	2973	3064.00	72.52	0.06	2973	**2973.00**	0.00	0.05	2932	3018.30	78.05	0.06	2932	**2937.00**	15.00	0.05
37	2992	3055.90	57.26	0.06	**2987**	**2987.50**	1.50	0.05	2937	3050.00	72.91	0.06	2937	**2937.40**	1.20	0.05	2939	3041.30	73.79	0.05	2939	**2939.80**	1.60	0.05
37	3053	3168.50	50.94	0.06	3053	**3060.20**	14.40	0.05	2998	3066.60	96.69	0.06	2998	**2998.00**	0.00	0.09	2962	3062.20	64.51	0.06	**2944**	**2951.20**	8.82	0.06
37	2942	3010.70	59.28	0.08	2942	**2942.00**	0.00	0.05	2975	3106.50	55.62	0.06	2975	**2999.00**	48.30	0.06	2915	2982.60	36.85	0.06	2915	**2920.50**	11.06	0.05
37	2931	3011.40	71.01	0.06	2940*	**2943.60**	1.20	0.05	2972	3047.60	87.01	0.05	**2974**	**2976.70**	8.10	0.05	2899	2994.90	96.09	0.06	**2886**	**2886.00**	0.00	0.05
37	2977	3042.50	38.98	0.06	**2946**	**2954.30**	7.21	0.07	2965	3054.10	59.70	0.06	2965	**2965.00**	0.00	0.06	2950	3028.30	55.19	0.06	**2945**	**2945.60**	0.92	0.05
37	2957	2994.90	37.34	0.06	**2949**	**2949.00**	0.00	0.05	3040	3154.50	70.59	0.06	**2978**	**2978.00**	0.00	0.05	2984	3038.00	68.44	0.06	**2887**	**2926.60**	48.50	0.05
37	2974	3008.40	34.66	0.06	**2960**	**2965.50**	3.11	0.05	2964	3027.20	49.53	0.06	2964	**2964.00**	0.00	0.05	2939	2985.50	43.93	0.06	2939	**2940.70**	2.90	0.06
61	5712	5927.90	172.97	0.12	**5555**	**5559.80**	7.33	0.08	5424	5707.50	217.71	0.12	**5402**	**5410.10**	11.89	0.08	5471	5885.70	196.73	0.12	**5427**	**5435.00**	15.51	0.08
61	5536	5807.20	188.05	0.12	**5448**	**5475.00**	32.87	0.08	5608	5854.00	155.24	0.13	**5433**	**5455.60**	19.61	0.07	5420	5667.80	217.53	0.12	**5356**	**5399.10**	27.86	0.08
61	5554	5829.90	194.42	0.12	**5365**	**5421.50**	28.70	0.08	5645	5883.60	125.56	0.12	**5432**	**5432.90**	2.70	0.08	5384	5596.10	195.53	0.12	**5362**	**5383.30**	16.57	0.08
61	5564	5806.10	152.31	0.12	**5410**	**5469.50**	54.81	0.08	5604	5834.90	142.34	0.12	**5572**	**5580.70**	14.32	0.08	5500	5698.30	244.38	0.12	**5349**	**5367.10**	21.19	0.08
61	5633	5842.20	211.59	0.12	**5407**	**5460.90**	44.61	0.08	5589	5955.50	176.78	0.12	**5422**	**5450.80**	40.69	0.08	5550	5768.30	195.48	0.12	**5372**	**5412.80**	22.14	0.07
61	5705	5882.80	161.32	0.12	**5604**	**5644.70**	36.62	0.08	5418	5581.20	192.19	0.12	**5273**	**5324.00**	52.46	0.07	5377	5559.30	172.58	0.12	**5162**	**5202.70**	71.55	0.08
61	5438	5691.80	165.82	0.12	5438	**5453.60**	14.93	0.08	5776	5969.10	150.94	0.12	**5684**	**5718.10**	37.86	0.08	5400	5610.20	196.94	0.12	**5282**	**5327.40**	40.79	0.08
61	5633	5892.80	215.98	0.12	**5496**	**5510.30**	34.09	0.08	5475	5757.80	158.74	0.12	**5468**	**5515.40**	31.91	0.08	5228	5547.00	272.11	0.12	**5163**	**5163.00**	0.00	0.08
61	5534	5726.30	160.14	0.12	**5428**	**5479.20**	55.23	0.08	5348	5676.20	208.46	0.12	**5314**	**5356.40**	26.16	0.08	5665	5884.10	177.45	0.12	**5495**	**5533.90**	45.96	0.08
61	5421	5791.70	202.49	0.12	**5413**	**5431.10**	26.35	0.08	5425	5636.90	133.95	0.12	**5382**	**5422.60**	21.75	0.07	5550	5772.90	140.76	0.12	**5433**	**5479.40**	29.44	0.08

Continued on next page

Table 7.1 – continued from previous page

N	Variance 20								Variance 40								Variance 60							
	GGARAP				SGGARAP				GGARAP				SGGARAP				GGARAP				SGGARAP			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
91	9089	9326.10	171.21	0.23	**8642**	**8755.40**	62.91	0.12	9517	9747.30	191.51	0.23	**8922**	**9011.90**	95.51	0.12	8900	9176.60	159.16	0.23	**8588**	**8679.70**	53.11	0.12
91	9022	9408.90	279.02	0.23	**8632**	**8684.70**	38.25	0.12	9006	9196.90	176.94	0.23	**8689**	**8715.80**	23.46	0.12	8956	9455.70	248.91	0.23	**8594**	**8793.00**	92.54	0.12
91	9222	9588.70	267.83	0.23	**8844**	**8954.10**	73.95	0.12	9411	9763.20	285.23	0.22	**8868**	**8999.80**	75.59	0.13	9082	9539.80	251.92	0.23	**8731**	**8789.40**	33.92	0.12
91	8907	9462.20	356.09	0.23	**8594**	**8853.50**	112.59	0.12	8923	9542.20	313.27	0.24	**8676**	**8772.10**	68.20	0.12	8934	9474.70	342.50	0.23	**8689**	**8789.80**	50.19	0.12
91	9228	9523.30	205.15	0.23	**8841**	**8870.80**	28.65	0.12	9176	9456.10	238.85	0.23	**8675**	**8740.30**	40.48	0.12	9145	9445.80	277.96	0.23	**8746**	**8854.90**	61.70	0.12
91	8962	9483.40	349.86	0.23	**8759**	**8797.50**	30.85	0.12	9108	9523.90	264.27	0.23	**8787**	**8859.70**	99.49	0.12	8974	9574.50	358.08	0.23	**8600**	**8676.20**	74.37	0.12
91	9245	9608.00	205.15	0.23	**8830**	**8957.20**	100.30	0.12	9235	9555.10	151.72	0.23	**8857**	**8939.30**	55.55	0.12	9013	9484.10	245.64	0.23	**8675**	**8718.50**	52.09	0.12
91	9135	9615.60	199.25	0.23	**8853**	**8953.70**	75.62	0.12	9328	9583.30	113.68	0.23	**9010**	**9093.40**	78.50	0.13	9321	9732.10	282.45	0.24	**8733**	**8937.60**	80.50	0.12
91	8887	9336.70	311.82	0.23	**8748**	**8795.80**	41.62	0.12	8936	9501.70	297.36	0.23	**8757**	**8861.90**	71.41	0.12	9055	9510.60	325.28	0.23	**8742**	**8804.10**	55.72	0.12
91	9346	9601.20	175.20	0.23	**8784**	**8923.30**	78.75	0.12	8983	9305.40	212.36	0.23	**8788**	**8869.20**	71.75	0.12	9040	9385.20	180.89	0.23	**8522**	**8709.90**	92.77	0.12
127	13952	14259.30	310.31	0.39	**13113**	**13227.70**	77.57	0.20	13366	13822.40	308.94	0.39	**12802**	**12914.00**	100.48	0.20	13567	14150.40	447.61	0.40	**13022**	**13171.50**	86.71	0.20
127	13116	13986.40	396.49	0.40	**13032**	**13159.50**	90.71	0.20	13784	14430.80	280.63	0.40	**13032**	**13245.10**	158.48	0.20	13642	14347.90	319.95	0.40	**13277**	**13488.40**	159.43	0.20
127	13493	14071.90	352.68	0.41	**12842**	**12998.60**	104.54	0.20	13767	14259.90	272.51	0.40	**13071**	**13187.20**	80.57	0.20	13526	14058.00	331.11	0.40	**12976**	**13172.90**	108.17	0.20
127	13914	14195.30	268.61	0.40	**12991**	**13143.20**	113.95	0.20	13564	14236.20	451.33	0.40	**12823**	**13098.00**	159.19	0.20	13640	14272.40	339.64	0.40	**12841**	**13086.30**	138.07	0.20
127	13552	13960.50	332.15	0.40	**13086**	**13191.20**	88.34	0.20	13767	14197.40	294.21	0.40	**12954**	**13152.30**	135.60	0.20	13452	14088.00	483.64	0.41	**12957**	**13036.70**	47.17	0.20
127	13185	13946.50	511.73	0.40	**12815**	**13017.30**	82.01	0.20	13737	14092.60	275.19	0.40	**12857**	**12983.70**	76.39	0.20	13703	14338.70	400.86	0.40	**12947**	**13174.40**	115.50	0.20
127	13590	14296.60	291.03	0.40	**13026**	**13214.80**	87.93	0.20	13106	13835.30	300.11	0.39	**12904**	**13038.30**	106.93	0.20	13684	14208.00	336.61	0.41	**12855**	**13131.70**	117.55	0.20
127	13829	14123.50	172.26	0.40	**13165**	**13351.70**	77.98	0.20	13544	14109.70	240.92	0.40	**13004**	**13130.90**	102.10	0.20	13896	14235.90	244.06	0.39	**13042**	**13329.50**	126.13	0.20
127	14107	14524.30	231.46	0.40	**13302**	**13490.90**	109.60	0.20	14146	14447.60	201.34	0.39	**13047**	**13273.20**	158.35	0.20	13778	14218.50	329.41	0.40	**12871**	**13055.20**	129.52	0.20
127	13685	13986.30	263.12	0.41	**12770**	**12829.50**	59.36	0.20	13585	14094.60	283.40	0.39	**12896**	**13095.50**	107.61	0.20	14038	14554.90	399.14	0.38	**13233**	**13446.50**	95.31	0.20

* Indicates that GGARAP is better for this instance and solution in bold indicates that SGGARAP is better than GGARAP on that instance.

Table 7.2: Results of GGA and SGGA on problem size 19, 37, 61, 91 and 127 for variance 20, 40 and 60

N	Variance 20								Variance 40								Variance 60							
	GGA				SGGA				GGA				SGGA				GGA				SGGA			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
19	1244	1244.00	0.00	0.04	1244	1244.00	0.00	0.03	1214	1215.60	3.20	0.04	1214	**1214.00**	0.00	0.03	1190	1191.10	3.30	0.04	**1168**	**1168.00**	0.00	0.03
19	1256	1256.00	0.00	0.03	1256	1256.00	0.00	0.03	1276	1276.50	1.50	0.04	1276	**1276.00**	0.00	0.03	1229	1229.60	1.20	0.04	1229	**1229.00**	0.00	0.03
19	1258	1258.60	1.20	0.04	1258	**1258.00**	0.00	0.03	1259	1260.20	3.60	0.04	1259	**1259.00**	0.00	0.03	1234	1234.00	0.00	0.04	1234	1234.00	0.00	0.03
19	1254	1254.00	0.00	0.04	1254	1254.00	0.00	0.03	1290	1290.60	0.49	0.04	1290	**1290.00**	0.00	0.03	1231	1231.00	0.00	0.04	1231	1231.00	0.00	0.03
19	1267	1267.30	0.90	0.04	1267	**1267.00**	0.00	0.03	1194	1194.00	0.00	0.04	1194	1194.00	0.00	0.03	1227	1227.00	0.00	0.04	1227	1227.00	0.00	0.03
19	1309	1309.40	1.20	0.04	1309	**1309.00**	0.00	0.03	1235	1235.10	0.30	0.04	1235	**1235.00**	0.00	0.03	1264	1264.30	0.90	0.04	1264	**1264.00**	0.00	0.03
19	1276	1276.10	0.30	0.04	1276	**1276.00**	0.00	0.03	1248	1248.00	0.00	0.04	1248	1248.00	0.00	0.03	1271	1272.80	3.60	0.04	1271	**1271.00**	0.00	0.03
19	1234	1234.00	0.00	0.04	1234	1234.00	0.00	0.03	1279	1279.70	2.10	0.04	**1270**	**1270.00**	0.00	0.03	1221	1221.00	0.00	0.04	1221	1221.00	0.00	0.03
19	1276	1276.00	0.00	0.04	1276	1276.00	0.00	0.03	1250	1253.40	6.36	0.04	1250	**1250.00**	0.00	0.03	1244	1244.00	0.00	0.04	1244	1244.00	0.00	0.03
19	1252	1252.00	0.00	0.04	1252	1252.00	0.00	0.03	1255	1255.00	0.00	0.03	1255	1255.00	0.00	0.03	1215	1215.00	0.00	0.04	1215	1215.00	0.00	0.03
37	2927	3038.80	68.42	0.08	2927	**2928.20**	3.60	0.06	2921	2996.80	45.84	0.08	**2871**	**2875.10**	7.23	0.06	2891	3012.00	94.83	0.08	2891	**2893.70**	4.12	0.06
37	2995	3057.60	47.81	0.08	**2959**	**2959.40**	1.20	0.06	2982	3057.40	58.37	0.08	**2939**	**2956.00**	17.00	0.06	2967	3063.60	46.57	0.08	**2899**	**2913.80**	18.13	0.06
37	3007	3082.70	45.08	0.08	**2952**	**2963.40**	10.20	0.06	2974	3033.80	41.07	0.08	**2945**	**2947.60**	6.58	0.06	2932	3071.10	72.24	0.08	2932	**2932.00**	0.00	0.06
37	2979	3058.10	60.11	0.08	**2977**	**2978.00**	3.00	0.06	2976	3063.70	52.53	0.08	**2937**	**2937.60**	0.92	0.06	2943	3061.80	73.26	0.08	**2939**	**2942.30**	8.65	0.06
37	2949	3062.40	90.93	0.08	**2946**	**2946.00**	0.00	0.06	3012	3116.20	50.38	0.08	**2998**	**3002.10**	5.03	0.06	3010	3113.80	59.69	0.08	**2944**	**2949.40**	8.25	0.06
37	2961	3033.10	50.17	0.08	**2910**	**2910.00**	0.00	0.06	2975	3044.50	47.45	0.08	**2942**	**2942.00**	0.00	0.06	2958	3043.80	53.38	0.08	**2915**	**2919.40**	5.39	0.06
37	2944	2988.20	35.08	0.08	**2931**	**2938.90**	9.88	0.06	2974	3057.70	47.80	0.08	**2972**	**2972.00**	0.00	0.06	2900	2974.50	52.75	0.08	**2886**	**2890.80**	7.37	0.06
37	2954	3038.40	70.77	0.08	**2946**	**2953.40**	2.54	0.06	2910	2977.90	57.25	0.08	**2877**	**2884.80**	6.37	0.06	2986	3055.20	33.69	0.08	**2945**	**2947.10**	5.66	0.06
37	2973	3028.90	40.50	0.08	**2949**	**2949.00**	0.00	0.06	2978	3149.20	98.05	0.08	2978	**2996.40**	39.17	0.06	2957	3037.50	35.77	0.08	**2855**	**2876.30**	32.54	0.06
37	2972	3010.70	41.72	0.08	**2960**	**2965.50**	3.11	0.06	2981	3031.70	41.32	0.08	**2964**	**2965.80**	3.63	0.06	2943	3057.80	73.70	0.08	**2939**	**2939.80**	1.60	0.06
61	5881	6051.90	131.01	0.14	**5412**	**5419.20**	9.85	0.11	5706	5859.30	131.79	0.14	**5318**	**5333.80**	31.63	0.11	5789	6031.00	141.64	0.14	**5427**	**5443.10**	19.77	0.11
61	5689	6021.00	185.65	0.14	**5448**	**5485.60**	39.00	0.11	5641	5864.10	127.27	0.14	**5245**	**5297.20**	33.37	0.11	5752	5958.20	124.45	0.14	**5356**	**5415.50**	30.94	0.11
61	5711	5888.10	75.67	0.14	**5349**	**5360.70**	6.69	0.11	5857	6035.70	100.51	0.14	**5339**	**5374.10**	55.82	0.12	5574	5820.70	126.45	0.14	**5325**	**5363.20**	22.54	0.11
61	5603	5949.60	155.62	0.14	**5410**	**5452.60**	24.83	0.12	5802	5962.10	123.24	0.14	**5392**	**5412.80**	11.00	0.11	5601	5930.10	166.61	0.14	**5349**	**5367.00**	26.76	0.11
61	5647	5951.70	145.74	0.14	**5393**	**5435.90**	40.50	0.11	5763	6030.00	127.67	0.14	**5422**	**5442.50**	41.79	0.11	5351	5854.20	186.95	0.14	**5243**	**5250.20**	8.20	0.11
61	5843	6029.50	86.27	0.14	**5604**	**5627.40**	39.57	0.12	5622	5887.40	109.09	0.14	**5273**	**5296.00**	45.90	0.11	5438	5776.90	137.09	0.14	**5162**	**5201.20**	34.39	0.11
61	5676	5929.40	144.52	0.14	**5430**	**5455.00**	15.37	0.11	6067	6222.50	102.44	0.14	**5684**	**5713.40**	26.89	0.12	5564	5895.00	167.75	0.14	**5282**	**5319.40**	29.67	0.11
61	5726	5953.00	142.71	0.14	**5363**	**5422.30**	73.27	0.12	5563	5974.80	203.05	0.14	**5468**	**5502.40**	35.29	0.11	5576	5783.90	133.91	0.14	**5163**	**5182.50**	25.17	0.11
61	5542	5845.70	188.55	0.14	**5327**	**5339.90**	12.42	0.11	5579	5853.30	134.54	0.14	**5208**	**5229.50**	34.44	0.11	5716	6007.10	124.86	0.14	**5495**	**5550.60**	48.33	0.12
61	5658	5955.70	183.41	0.14	**5390**	**5435.30**	25.05	0.12	5735	5875.20	106.80	0.14	**5319**	**5323.40**	9.09	0.11	5882	6007.10	92.02	0.14	**5347**	**5359.40**	20.49	0.11

Continued on next page

Table 7.2 – continued from previous page

N	Variance 20								Variance 40								Variance 60							
	GGA				SGGA				GGA				SGGA				GGA				SGGA			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
91	8834	9723.60	318.97	0.24	**8612**	**8777.30**	72.40	0.19	9475	9923.20	210.72	0.24	**8903**	**8992.60**	64.31	0.20	9457	9691.80	149.70	0.24	**8569**	**8641.80**	50.28	0.20
91	9439	9692.30	168.28	0.23	**8621**	**8710.10**	48.08	0.19	9604	9778.30	143.51	0.24	**8670**	**8703.20**	28.36	0.19	9637	9787.60	126.70	0.24	**8586**	**8776.10**	112.00	0.20
91	9538	9863.70	129.96	0.24	**8799**	**8949.10**	72.27	0.20	9460	9800.60	161.46	0.24	**8884**	**8971.10**	50.33	0.20	9549	9860.30	162.83	0.24	**8669**	**8734.90**	62.30	0.20
91	9420	9866.10	222.88	0.24	**8754**	**8841.20**	55.87	0.20	9291	9643.40	160.39	0.24	**8641**	**8700.60**	55.43	0.20	9287	9757.50	220.68	0.24	**8692**	**8732.20**	34.41	0.19
91	9288	9886.20	284.17	0.24	**8740**	**8765.10**	46.20	0.19	9509	9678.20	127.18	0.24	**8694**	**8734.00**	33.15	0.19	9569	9835.20	140.44	0.24	**8727**	**8818.40**	64.10	0.20
91	9463	9834.20	220.36	0.24	**8699**	**8755.50**	49.32	0.19	9597	9967.50	145.27	0.24	**8764**	**8813.10**	48.64	0.20	9470	9791.10	189.86	0.24	**8599**	**8646.10**	66.30	0.20
91	9560	9980.90	178.55	0.24	**8830**	**8886.70**	44.00	0.20	9680	9975.30	175.74	0.24	**8757**	**8880.30**	67.31	0.20	9381	9916.20	207.47	0.24	**8675**	**8688.00**	35.03	0.19
91	9611	9888.30	154.59	0.24	**8746**	**8786.00**	35.94	0.19	9961	10106.00	111.09	0.24	**9010**	**9061.40**	60.62	0.20	9707	10015.70	188.01	0.24	**8733**	**8811.00**	81.30	0.19
91	9263	9847.90	253.60	0.24	**8729**	**8777.20**	47.37	0.19	9632	9892.10	132.50	0.24	**8742**	**8839.80**	63.98	0.19	9348	9771.90	219.41	0.24	**8631**	**8719.80**	105.43	0.20
91	9529	9863.20	143.26	0.24	**8782**	**8830.70**	44.08	0.20	9667	9820.90	108.27	0.24	**8596**	**8723.60**	63.33	0.19	9190	9641.00	188.47	0.24	**8481**	**8631.80**	102.69	0.19
127	14294	14678.20	212.43	0.38	**13019**	**13101.80**	66.76	0.32	13956	14522.20	305.56	0.38	**12732**	**12874.50**	92.27	0.31	14157	14624.90	207.25	0.38	**12852**	**12972.20**	84.19	0.31
127	14190	14592.80	242.18	0.38	**12787**	**13004.80**	120.76	0.31	14636	14825.80	145.41	0.38	**13022**	**13095.20**	46.30	0.32	14240	14857.50	333.27	0.38	**13364**	**13496.60**	65.12	0.32
127	14076	14690.10	233.09	0.38	**12869**	**12973.50**	54.85	0.31	14176	14702.60	233.81	0.38	**12948**	**13074.10**	134.54	0.32	14072	14447.90	267.87	0.38	**12698**	**12975.30**	144.51	0.32
127	14545	14757.10	148.57	0.38	**12884**	**13035.30**	95.70	0.31	14079	14697.40	266.18	0.38	**13006**	**13130.60**	110.45	0.32	14115	14502.90	197.04	0.38	**12762**	**12891.20**	127.56	0.31
127	14491	14726.80	168.92	0.38	**13001**	**13067.60**	49.38	0.32	14097	14708.90	259.31	0.38	**12942**	**13023.50**	50.68	0.31	14444	14717.70	168.51	0.38	**12786**	**12890.40**	79.84	0.31
127	14168	14452.60	181.67	0.38	**12753**	**12952.50**	84.36	0.31	14303	14694.30	204.17	0.38	**12656**	**12906.20**	140.14	0.31	14624	14928.60	180.68	0.38	**12901**	**13081.30**	125.79	0.32
127	14049	14495.80	213.53	0.38	**12895**	**13062.20**	112.33	0.32	14233	14644.50	204.73	0.38	**12792**	**12924.90**	91.14	0.32	14431	14703.10	150.05	0.38	**12860**	**13007.00**	108.71	0.32
127	14207	14725.10	261.95	0.38	**12957**	**13039.80**	63.68	0.31	14282	14571.10	160.57	0.38	**12732**	**12997.00**	144.36	0.32	14818	14992.00	117.43	0.38	**13045**	**13169.70**	81.37	0.32
127	14463	14819.50	205.06	0.38	**13110**	**13288.20**	105.32	0.32	14467	14711.60	158.00	0.38	**12993**	**13083.00**	79.46	0.32	14227	14631.60	178.38	0.38	**12846**	**12961.00**	78.02	0.32
127	14261	14551.00	226.29	0.38	**12791**	**12845.20**	56.55	0.32	13895	14650.10	304.89	0.38	**12901**	**13022.70**	99.35	0.31	14689	14966.20	157.20	0.38	**13145**	**13305.30**	120.80	0.32

Solution in bold indicates that SGGA is better than GGA on that instance.

Table 7.3: Results of HGGA and SGGA_LS on problem size 19, 37, 61, 91 and 127 for variance 20, 40 and 60

N	Variance 20								Variance 40								Variance 60							
	HGGA				SGGA_LS				HGGA				SGGA_LS				HGGA				SGGA_LS			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
19	1244	1244.00	0.00	0.08	1244	1244.00	0.00	0.03	1214	1214.00	0.00	0.07	1214	1214.00	0.00	0.03	1168	1168.00	0.00	0.12	1168	1168.00	0.00	0.03
19	1256	1256.00	0.00	0.13	1256	1256.00	0.00	0.03	1276	1276.00	0.00	0.10	1276	1276.00	0.00	0.03	1229	1229.00	0.00	0.07	1229	1229.00	0.00	0.03
19	1258	1258.00	0.00	0.06	1258	1258.00	0.00	0.03	1259	1259.00	0.00	0.10	1259	1259.00	0.00	0.03	1234	1234.00	0.00	0.10	1234	1234.00	0.00	0.03
19	1254	1254.00	0.00	0.07	1254	1254.00	0.00	0.03	1290	1290.00	0.00	0.08	1290	1290.00	0.00	0.03	1231	1231.00	0.00	0.07	1231	1231.00	0.00	0.03
19	1267	1267.00	0.00	0.07	1267	1267.00	0.00	0.03	1194	1194.00	0.00	0.07	1194	1194.00	0.00	0.03	1227	1227.00	0.00	0.06	1227	1227.00	0.00	0.03
19	1309	1309.00	0.00	0.08	1309	1309.00	0.00	0.03	1235	1235.00	0.00	0.17	1235	1235.00	0.00	0.03	1264	1264.00	0.00	0.08	1264	1264.00	0.00	0.03
19	1276	1276.00	0.00	0.07	1276	1276.00	0.00	0.03	1248	1248.00	0.00	0.13	1248	1248.00	0.00	0.03	1271	1271.00	0.00	0.09	1271	1271.00	0.00	0.03
19	1234	1234.00	0.00	0.09	1234	1234.00	0.00	0.03	1270	1270.00	0.00	0.13	1270	1270.00	0.00	0.03	1221	1221.00	0.00	0.11	1221	1221.00	0.00	0.03
19	1276	1276.00	0.00	0.10	1276	1276.00	0.00	0.03	1250	1250.00	0.00	0.12	1250	1250.00	0.00	0.03	1244	1244.00	0.00	0.09	1244	1244.00	0.00	0.03
19	1252	1252.00	0.00	0.13	1252	1252.00	0.00	0.03	1255	1255.00	0.00	0.07	1255	1255.00	0.00	0.03	1215	1215.00	0.00	0.10	1215	1215.00	0.00	0.03
37	2927	2931.60	10.43	0.71	2927	**2927.00**	0.00	0.59	2871	2871.00	0.00	0.46	2871	2871.00	0.00	0.66	2891	2891.00	0.00	0.35	2891	2891.00	0.00	0.46
37	2959	2959.00	0.00	0.69	2959	2959.00	0.00	0.55	2939	2939.00	0.00	0.96	2939	2939.00	0.00	0.65	2899	2902.70	11.10	0.77	2899	**2899.00**	0.00	0.49
37	2952	2956.50	7.26	0.79	2952	**2952.00**	0.00	0.64	2945	2945.00	0.00	0.67	2945	2945.00	0.00	0.53	2932	2937.00	15.00	0.90	2932	**2932.00**	0.00	0.57
37	2977	2977.00	0.00	0.61	2977	2977.00	0.00	0.67	2937	2937.00	0.00	1.26	2937	2937.20*	0.40	0.61	2939	2939.00	0.00	0.68	2939	2939.00	0.00	0.50
37	2946	2946.00	0.00	0.81	2946	2946.00	0.00	0.54	2998	3004.00	4.90	0.85	2998	**2999.00**	3.00	0.50	2944	2947.60	7.20	0.71	2944	**2944.00**	0.00	0.56
37	2910	2910.00	0.00	0.45	2910	2910.00	0.00	0.54	2942	2942.00	0.00	0.54	2942	2942.00	0.00	0.51	2915	2915.00	0.00	0.76	2915	2915.00	0.00	0.59
37	2931	2931.00	0.00	0.82	2931	2931.80*	2.40	0.59	2972	2974.00	5.37	0.98	2972	**2972.00**	0.00	0.52	2886	2886.00	0.00	0.41	2886	2886.00	0.00	0.57
37	2946	2947.60	3.20	0.63	2946	**2946.00**	0.00	0.48	2877	2877.00	0.00	0.55	2877	2877.00	0.00	0.46	2945	2945.20	0.60	0.84	2945	**2945.00**	0.00	0.59
37	2949	2949.00	0.00	0.43	2949	2949.00	0.00	0.59	2978	3000.00	45.19	1.16	2978	**2978.00**	0.00	0.54	2855	2855.00	0.00	0.46	2855	2855.00	0.00	0.51
37	2960	2960.60	1.80	0.76	2960	2961.20*	2.40	0.54	2964	2964.00	0.00	0.80	2964	2964.00	0.00	0.51	2939	2939.00	0.00	0.48	2939	2939.00	0.00	0.54
61	5412	5419.80	19.19	3.46	5412	**5412.00**	0.00	1.31	5318	5318.00	0.00	2.27	5318	5318.00	0.00	1.64	5427	5462.10	11.70	2.21	5427	**5438.70**	17.87	1.36
61	5500	5514.00	21.78	4.92	**5448**	**5453.00**	15.00	1.49	5245	5256.30	33.90	2.35	5245	**5245.00**	0.00	1.40	5356	5398.10	14.16	2.52	5356	**5386.40**	21.26	1.11
61	5349	5398.20	31.64	3.50	5349	**5366.40**	28.46	1.59	5339	5413.00	74.00	3.52	5339	**5339.00**	0.00	0.88	5325	5356.80	21.99	2.64	5325	**5345.80**	20.99	1.27
61	5440	5494.80	32.42	3.95	**5410**	**5430.40**	18.04	1.54	5414	5462.30	37.58	3.97	**5392**	**5411.00**	9.89	1.26	5349	5358.50	13.57	4.38	5349	**5353.00**	2.00	1.32
61	5393	5461.50	43.01	4.73	**5391**	**5399.00**	14.81	1.61	5422	5430.10	20.38	2.49	5422	**5422.00**	0.00	1.40	5253	5260.30	6.60	2.97	**5243**	**5243.00**	0.00	1.90
61	5641	5653.60	9.25	3.48	**5604**	**5626.80**	17.61	1.66	5273	5273.90	2.70	2.42	5273	**5273.00**	0.00	1.93	5162	5215.50	28.30	3.31	5162	**5194.90**	30.08	1.49
61	5430	5460.90	25.40	3.56	5430	**5433.60**	10.80	1.37	5669	5692.10	27.53	4.87	5669	**5672.00**	6.00	1.84	5282	5314.90	20.32	3.39	5282	**5286.50**	11.93	1.84
61	5437	5511.40	40.04	4.01	**5363**	**5396.10**	51.33	1.35	5435	5474.40	31.97	4.14	5435	**5453.70**	19.71	1.55	5163	5171.70	13.29	3.17	5163	**5163.00**	0.00	1.24
61	5357	5389.00	40.50	3.37	**5327**	**5338.50**	15.53	1.59	5208	5223.60	31.20	2.40	5208	**5208.00**	0.00	1.11	5524	5539.10	11.59	3.56	**5495**	**5505.40**	16.48	1.80
61	5390	5442.90	47.19	4.06	5390	**5390.00**	0.00	1.53	5319	5326.20	9.06	3.04	5319	**5320.90**	2.02	1.82	5347	5349.40	4.80	3.90	5347	**5347.00**	0.00	1.47

Continued on next page

Table 7.3 – continued from previous page

N	Variance 20								Variance 40								Variance 60							
	HGGA				SGGA_LS				HGGA				SGGA_LS				HGGA				SGGA_LS			
	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET	Best	Mean	SD	AET
91	8612	8612.80	2.40	9.04	8612	**8612.00**	0.00	5.53	8971	8972.60	0.80	14.09	**8912**	**8951.90**	29.29	7.59	8524	8580.50	68.66	16.80	**8523**	**8536.80**	20.17	6.15
91	8683	8710.00	32.54	13.98	**8621**	**8654.00**	22.73	6.11	8663	8718.80	39.24	15.02	**8623**	**8663.30**	28.01	6.01	8586	8618.30	40.70	13.58	8586	**8606.80**	41.60	6.16
91	8910	9014.60	40.12	18.89	**8799**	**8905.40**	50.69	5.87	8875	8975.70	48.49	17.79	8880*	**8891.90**	20.73	5.53	8669	8720.40	48.83	17.24	8669	**8675.40**	10.11	6.16
91	8594	8790.30	104.79	14.97	8594	**8670.20**	95.31	5.24	8598	8643.10	23.44	13.07	8598	**8627.60**	19.76	6.58	8650	8696.40	49.08	17.74	8650	**8695.40**	43.73	5.39
91	8704	8748.80	72.41	13.19	8704	**8712.10**	14.20	4.60	8660	8713.00	39.92	16.56	**8650**	**8656.00**	4.90	7.07	8710	8803.10	53.92	20.57	**8670**	**8737.70**	46.93	6.92
91	8699	8701.40	1.96	9.63	8699	**8700.20**	1.83	5.81	8826	8918.00	61.22	13.43	**8747**	**8774.40**	22.70	4.53	8588	8682.50	72.90	14.78	8588	**8594.30**	6.33	4.92
91	8865	9031.70	64.00	20.99	**8797**	**8828.80**	25.17	5.59	8757	8827.70	56.04	12.58	8757	**8757.00**	0.00	5.46	8725	8827.50	67.31	20.57	**8660**	**8662.30**	4.71	4.77
91	8746	8810.60	79.91	12.76	**8733**	**8743.40**	5.20	4.94	9102	9110.50	14.50	12.33	**9010**	**9036.50**	28.73	6.75	8715	8765.70	44.98	16.40	8715	**8720.40**	8.25	5.53
91	8686	8765.80	76.18	13.12	**8717**	**8728.70**	6.91	7.12	8706	8804.70	86.12	15.83	8706	**8710.40**	13.20	5.22	8631	8720.60	75.17	15.10	8631	**8660.40**	66.61	5.24
91	8828	8903.60	45.46	14.37	**8755**	**8827.50**	49.66	6.07	8549	8587.40	60.18	14.79	8549	**8549.00**	0.00	5.19	8481	8494.80	41.40	14.96	8481	**8505.20**	48.47	5.70
127	12959	13081.50	72.92	52.93	**12896**	**12949.10**	43.14	19.16	12637	12753.20	60.80	55.83	**12562**	**12632.10**	58.09	17.89	12800	12846.10	41.75	51.07	**12740**	**12781.10**	30.81	22.24
127	12807	12843.90	48.30	46.38	**12786**	**12826.00**	45.77	19.27	12918	13052.70	98.54	41.38	**12910**	**12917.80**	2.86	16.15	13095	13251.50	114.41	65.92	**13086**	**13137.90**	33.81	20.25
127	12819	12913.90	45.22	49.43	**12776**	**12810.10**	38.67	18.32	12856	12990.30	74.00	56.02	**12792**	**12841.70**	47.32	20.42	12801	12921.30	71.10	68.75	**12709**	**12769.00**	66.05	20.80
127	12799	12978.30	71.36	45.57	12852*	**12867.40**	29.87	16.23	12898	12995.20	85.23	62.30	**12823**	**12865.40**	39.53	19.83	12670	12773.80	74.02	52.67	**12608**	**12655.70**	25.57	19.36
127	12828	12904.20	49.29	48.09	**12825**	**12859.10**	24.46	20.99	12800	12933.10	61.90	59.34	**12790**	**12840.00**	45.70	18.60	12667	12749.30	45.86	46.98	12667	**12722.30**	41.02	18.67
127	12732	12775.80	66.69	49.24	**12667**	**12728.10**	37.15	17.70	12729	12814.60	62.92	53.93	**12718**	**12765.60**	48.46	20.36	12934	12975.50	29.62	46.13	**12757**	**12844.50**	72.82	17.60
127	12763	12870.70	82.01	43.48	**12735**	**12817.80**	72.56	20.40	12738	12831.00	61.75	42.76	**12678**	**12727.90**	23.90	20.02	12750	12834.80	57.21	52.32	**12651**	**12698.50**	40.75	19.27
127	12968	13038.60	52.31	53.80	**12805**	**12873.60**	36.31	19.43	12657	12712.70	81.57	49.41	12657	**12657.00**	0.00	19.00	13068	13185.30	78.41	74.18	**12864**	**12965.60**	63.07	20.29
127	13147	13259.00	43.44	65.48	**13072**	**13102.80**	28.84	21.98	12858	13124.40	148.01	65.83	12858	**12897.40**	34.55	16.16	12706	12786.30	79.00	49.71	**12670**	**12715.80**	41.15	20.16
127	12692	12762.70	58.44	44.57	**12624**	**12668.90**	31.79	18.51	12919	12993.80	85.06	62.42	**12795**	**12815.40**	19.74	17.73	13117	13210.30	62.95	50.21	**13055**	**13111.00**	41 .46	19.13

* Indicates that HGGA is better for this instance and solution in bold indicates that SGGA_LS is better than HGGA on that instance.

Table 7.4: *Mann-Whitney U test* of SGGARAP, SGGA and SGGA_LS with GGARAP, GGA and HGGA respectively on problem size 19, 37, 61, 91 and 127 for variance 20, 40 and 60

N	Variance 20						Variance 40						Variance 60					
	G_SRAP		G_S		G_SLS		G_SRAP		G_S		G_SLS		G_SRAP		G_S		G_SLS	
	U-value	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*
19	35.00	0.27134	50.00	0.96810[a]	50.00	0.96810[a]	30.00	0.14156	40.00	0.47152	50.00	0.96810[a]	35.00	0.27134	0.00	0.00018	50.00	0.96810[a]
19	45.00	0.72786	50.00	0.96810[a]	50.00	0.96810[a]	30.00	0.14156	45.00	0.72786	50.00	0.96810[a]	25.00	0.06432	40.00	0.47152	50.00	0.96810[a]
19	40.00	0.47152	40.00	0.47152	50.00	0.96810[a]	15.00	0.00906	45.00	0.72786	50.00	0.96810[a]	35.00	0.27134	50.00	0.96810[a]	50.00	0.96810[a]
19	45.00	0.72786	50.00	0.96810[a]	50.00	0.96810[a]	40.00	0.47152	20.00	0.02574	50.00	0.96810[a]	25.00	0.06432	50.00	0.96810[a]	50.00	0.96810[a]
19	40.00	0.47152	45.00	0.72786	50.00	0.96810[a]	30.00	0.14156	50.00	0.96810[a]	50.00	0.96810[a]	40.00	0.00018	50.00	0.96810[a]	50.00	0.96810[a]
19	25.00	0.06432	45.00	0.72786	50.00	0.96810[a]	35.00	0.27134	45.00	0.72786	50.00	0.96810[a]	42.00	0.00018	45.00	0.7278	50.00	0.96810[a]
19	50.00	0.96810[a]	45.00	0.72786	50.00	0.96810[a]	45.00	0.72786	50.00	0.96810[a]	50.00	0.96810[a]	15.00	0.00906	40.00	0.47152	50.00	0.96810[a]
19	35.00	0.27134	50.00	0.96810[a]	50.00	0.96810[a]	50.00	0.96810[a]	0.00	0.00018	50.00	0.96810[a]	35.00	0.27134	50.00	0.96810[a]	50.00	0.96810[a]
19	15.00	0.00906	50.00	0.96810[a]	50.00	0.96810[a]	35.00	0.27134	25.00	0.06432	50.00	0.96810[a]	25.00	0.06432	50.00	0.96810[a]	50.00	0.96810[a]
19	20.00	0.02574	50.00	0.96810[a]	50.00	0.96810[a]	50.00	0.96810[a]	50.00	0.96810[a]	50.00	0.96810[a]	40.00	0.47152	50.00	0.96810[a]	50.00	0.96810[a]
37	0.00	0.00018	5.50	0.00086	40.00	0.47152	15.50	0.01016	0.00	0.00018	50.00	0.96810[a]	5.00	0.00078	6.50	0.00116	50.00	0.96810[a]
37	2.00	0.00034	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	0.00	0.00018	50.00	0.96810[a]	5.00	0.00078	0.00	0.00018	45.00	0.72786
37	11.50	0.00410	0.00	0.00018	35.00	0.27134	5.00	0.00078	0.00	0.00018	50.00	0.96810[a]	22.00	0.03752	5.00	0.00078	45.00	0.72786
37	0.50	0.00022	1.00	0.00024	50.00	0.96810[a]	6.50	0.00116	0.00	0.00018	40.00	0.47152	6.00	0.00100	1.50	0.00028	50.00	0.96810[a]
37	7.00	0.00132	0.00	0.00018	50.00	0.96810[a]	15.00	0.00906	0.00	0.00018	25.00	0.06432	2.00	0.00034	0.00	0.00018	40.00	0.47152
37	15.00	0.00906	0.00	0.00018	50.00	0.96810[a]	9.00	0.00222	0.00	0.00018	50.00	0.96810[a]	12.00	0.00466	0.00	0.00018	50.00	0.96810[a]
37	10.00	0.00278	3.00	0.00044	45.00	0.72786	45.00	0.76418	0.00	0.00018	40.00	0.47152	0.00	0.00018	2.50	0.00038	50.00	0.96810[a]
37	0.00	0.00018	5.50	0.00086	40.00	0.47152	10.00	0.00018	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	0.00	0.00018	45.00	0.72786
37	0.00	0.00018	10.00	0.00174	50.00	0.96810[a]	0.00	0.00018	9.00	0.00222	40.00	0.47152	0.00	0.00018	0.00	0.00018	50.00	0.96810[a]
37	0.00	0.00018	0.00	0.00018	45.00	0.72786	15.00	0.00906	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	1.00	0.00024	50.00	0.96810[a]
61	0.00	0.00018	0.00	0.00018	40.00	0.47152	2.00	0.00034	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	0.00	0.00018	20.00	0.02574
61	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	0.00	0.00018	37.00	0.34722
61	0.00	0.00018	0.00	0.00018	28.00	0.10310	0.00	0.00018	0.00	0.00018	25.00	0.06432	0.00	0.00018	0.00	0.00018	36.00	0.30772
61	2.00	0.00034	0.00	0.00018	3.50	0.00050	1.00	0.00024	0.00	0.00018	11.50	0.00410	0.00	0.00018	0.00	0.00018	42.00	0.56868
61	0.00	0.00018	0.00	0.00018	7.00	0.00132	0.00	0.00018	0.00	0.00018	30.00	0.14156	0.00	0.00018	0.00	0.00018	0.00	0.00018
61	1.00	0.00024	0.00	0.00018	11.50	0.00410	3.00	0.00044	0.00	0.00018	45.00	0.72786	0.00	0.00018	0.00	0.00018	32.00	0.18684
61	9.00	0.00222	0.00	0.00018	16.00	0.01140	1.00	0.00024	0.00	0.00018	31.00	0.16152	0.00	0.00018	0.00	0.00018	10.50	0.00318
61	0.00	0.00018	0.00	0.00018	6.50	0.00116	9.00	0.00222	1.00	0.00024	33.00	0.21130	0.00	0.00018	0.00	0.00018	35.00	0.27134
61	6.00	0.00100	0.00	0.00018	5.00	0.00078	6.50	0.00116	0.00	0.00018	40.00	0.47152	0.00	0.00018	0.00	0.00018	8.50	0.00194
61	4.00	0.00058	0.00	0.00018	20.00	0.02574	2.50	0.00038	0.00	0.00018	18.00	0.01732	0.00	0.00018	0.00	0.00018	40.00	0.47152

Continued on next page

Table 7.4 – continued from previous page

N	Variance 20						Variance 40						Variance 60					
	G_SRAP		G_S		G_SLS		G_SRAP		G_S		G_SLS		G_SRAP		G_S		G_SLS	
	U-value	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*	*U-value*	*p-value*
91	0.00	0.00018	2.00	0.00034	45.00	0.72786	0.00	0.00018	0.00	0.00018	30.00	0.14156	0.00	0.00018	0.00	0.00018	22.00	0.03752
91	0.00	0.00018	0.00	0.00018	7.50	0.00152	0.00	0.00018	0.00	0.00018	12.50	0.00512	0.00	0.00018	0.00	0.00018	42.00	0.56868
91	0.00	0.00018	0.00	0.00018	5.00	0.00078	0.00	0.00018	0.00	0.00018	12.00	0.00466	0.00	0.00018	0.00	0.00018	18.00	0.01732
91	3.00	0.00044	0.00	0.00018	19.00	0.02088	0.00	0.00018	0.00	0.00018	33.50	0.22628	0.00	0.00018	0.00	0.00018	48.50	0.93624
91	0.00	0.00018	0.00	0.00018	38.50	0.40654	0.00	0.00018	0.00	0.00018	6.00	0.00100	0.00	0.00018	0.00	0.00018	20.00	0.02574
91	0.00	0.00018	0.00	0.00018	35.00	0.27134	0.00	0.00018	0.00	0.00018	2.00	0.00034	0.00	0.00018	0.00	0.00018	9.50	0.00252
91	0.00	0.00018	0.00	0.00018	1.00	0.00024	0.00	0.00018	0.00	0.00018	15.00	0.00906	0.00	0.00018	0.00	0.00018	0.00	0.00018
91	0.00	0.00018	0.00	0.00018	12.00	0.00466	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	9.50	0.00252
91	0.00	0.00018	0.00	0.00018	41.00	0.52218	2.00	0.00034	0.00	0.00018	22.00	0.03752	0.00	0.00018	0.00	0.00018	20.00	0.02574
91	0.00	0.00018	0.00	0.00018	17.50	0.01552	1.00	0.00024	0.00	0.00018	25.00	0.06432	0.00	0.00018	0.00	0.00018	46.00	0.79486
127	0.00	0.00018	0.00	0.00018	7.00	0.00132	0.00	0.00018	0.00	0.00018	9.50	0.00252	0.00	0.00018	0.00	0.00018	16.50	0.01278
127	6.00	0.00100	0.00	0.00018	27.00	0.08914	0.00	0.00018	0.00	0.00018	5.50	0.00086	0.00	0.00018	0.00	0.00018	26.00	0.07508
127	0.00	0.00018	0.00	0.00018	7.00	0.00132	0.00	0.00018	0.00	0.00018	8.00	0.00168	0.00	0.00018	0.00	0.00018	7.00	0.00132
127	0.00	0.00018	0.00	0.00018	10.00	0.00278	0.00	0.00018	0.00	0.00018	8.00	0.00168	0.00	0.00018	0.00	0.00018	3.00	0.00044
127	0.00	0.00018	0.00	0.00018	18.00	0.01732	0.00	0.00018	0.00	0.00018	10.50	0.00318	0.00	0.00018	0.00	0.00018	33.50	0.22628
127	0.00	0.00018	0.00	0.00018	36.00	0.30772	0.00	0.00018	0.00	0.00018	25.00	0.06432	0.00	0.00018	0.00	0.00018	7.00	0.00132
127	0.00	0.00018	0.00	0.00018	32.50	0.19706	3.00	0.00044	0.00	0.00018	3.50	0.00050	0.00	0.00018	0.00	0.00018	1.50	0.00028
127	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	30.00	0.14156	0.00	0.00018	0.00	0.00018	2.00	0.00034
127	0.00	0.00018	50.00	0.96810[a]	0.00	0.00018	0.00	0.00018	0.00	0.00018	10.00	0.00278	0.00	0.00018	0.00	0.00018	17.00	0.01390
127	0.00	0.00018	0.00	0.00018	7.50	0.00152	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	0.00	0.00018	7.50	0.00152

[a] Both the approaches obtained the same solution in all the runs.

Table 7.5 reports the overall average solution quality and average execution time on various network sizes for all the approaches considered in this chapter, i.e., average solution quality (column AAvg) and average AET (column AAET) of HGGA, SGGA_LS, GGA, SGGA, GGARAP and SGGARAP on all 30 instances of each size. From Table 7.5, we can clearly observe that overall average solution quality of our SGGA_LS, SGGA and SGGARAP approaches are better than their respective competitors HGGA, GGA and GGARAP.

Table 7.5: Overall average solution quality and average AET of SGGARAP, SGGA, SGGA_LS, GGARAP, GGA and HGGA on various network sizes

N	HGGA		SGGA_LS		GGA		SGGA		GGARAP		SGGARAP	
	AAvg	AAET	AAvg	AAET	AAvg	AAET	AAvg	AAET	AAvg	AAET	AAvg	AAET
19	1247.37	0.09	1247.37	0.03	1248.84	0.04	1247.37	0.03	1272.15	0.02	1263.05	0.04
37	2935.96	0.71	2934.27	0.55	3047.30	0.08	2939.19	0.06	3041.84	0.06	2952.53	0.05
61	5401.41	3.42	5372.40	1.49	5940.14	0.14	5393.70	0.11	5768.17	0.12	5442.53	0.08
91	8775.70	15.14	8713.17	5.79	9836.67	0.24	8789.96	0.20	9496.94	0.23	8838.75	0.12
127	12938.79	53.54	12828.82	19.20	14686.33	0.38	13041.75	0.32	14178.33	0.40	13161.19	0.20

Table 7.6 reports the average percentage improvement in solution quality (API) by various approaches over other comparable approaches on various network sizes. In this table, column HG_LS reports the API by SGGA_LS over HGGA, column G_S reports API by SGGA over GGA, column S_LS reports API by SGGA_LS over SGGA, column S_HG reports API by HGGA over SGGA, column G_LS reports API by SGGA_LS over GGA, column G_HG reports API by HGGA over GGA and column GP_SP reports API by SGGARAP over GGARAP. Some interesting observations can be made with the help of this table. Columns HG_LS, G_S and GP_SP respectively show the superiority of SGGA_LS over HGGA, SGGA over GGA and SGGARAP over GGARAP. If we observe the columns S_LS and G_HG, we can see that the performance of HGGA depends more on *improvement operator* than SGGA_LS on local search. In column S_HG, the negative value indicates that even SGGA is overall better than HGGA for the network of size 61. Overall, this column indicates that solutions obtained by SGGA which does not use local search are quite close to the solutions obtained by HGGA. On the other hand, column G_LS indicates that our SGGA_LS improves substantially the results obtained by GGA.

Figures 7.2(a) to 7.2(e) and 7.3(a) to 7.3(e) graphically compare the average solution quality of several approaches over all the instances of each variance, viz. 20, 40 and 60.

Figures 7.2(a) to 7.2(e) graphically compare the average solution quality of GGARAP and SGGARAP over 10 instances of each network size and variance. These figures clearly demonstrate the superiority of SGGARAP over GGARAP for all network sizes and variances.

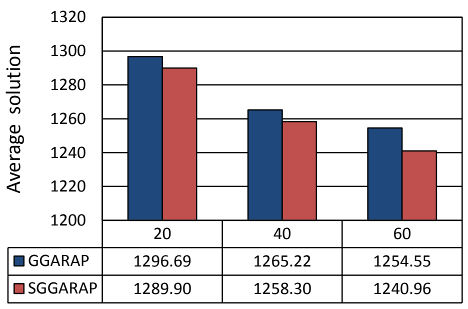
Table 7.6: Average percentage improvement on problem size 19, 37, 61, 91 and 127

N	HG_LS	G_S	S_LS	S_HG	G_LS	G_HG	GP_SP
19	0.00	0.12	0.00	0.00	0.12	0.12	0.72
37	0.06	3.55	0.17	0.11	3.71	3.65	2.94
61	0.54	9.20	0.39	-0.14	9.56	9.07	5.65
91	0.71	10.64	0.87	0.16	11.42	10.79	6.93
127	0.85	11.20	1.63	0.79	12.65	11.90	7.17

It can also be observed that difference in solution quality increases in general as the network size increases.

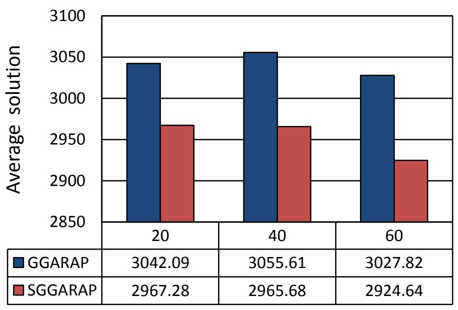
Figures 7.3(a) to 7.3(e) graphically compare the average solution quality of GGA, HGGA, SGGA and SGGA_LS over 10 instances of various network sizes and variances. From these figures, it can be easily observed that SGGA outperformed GGA and SGGA_LS outperformed HGGA on all network sizes as well as all variances. If we observe the differences in solution quality between SGGA_LS and SGGA and between HGGA and GGA, we find that solution quality of SGGA is closer to SGGA_LS compare to that of GGA with HGGA on all network sizes as well as variances.

Total number of solution evaluations: HGGA and SSGA_LS approaches use local search. When local search is used, then evaluation of each of its move constitutes a solution evaluation. As the number of moves available varies from one iteration of the local search to the other and depends on the input solution, HGGA and SSGA_LS may perform a different number of solution evaluations on each instance. As the quality of the solution returned depends on the total number of solution evaluations, therefore, the superior performance of SSGA_LS can be attributed to more number of solution evaluations performed by it in comparison to HGGA. To show that this is not the case, we have counted the solution evaluations performed by the two approaches on each of the 10 runs on each instance. Figure 7.4 plots the average number of solution evaluations over 10 runs performed by HGGA and SSGA_LS on each instance. In Figure 7.4, instance numbers 1 to 50, 51 to 100 and 101 to 150 are for the instances with variance 20, 40 and 60 respectively. Instances numbers 1 to 10, 51 to 60 and 101 to 110 are for the instances with size 19. Instance numbers 11 to 20, 61 to 70 and 111 to 120 are for the instances with size 37. Instance numbers 21 to 30, 71 to 80 and 121 to 130 are for the instances



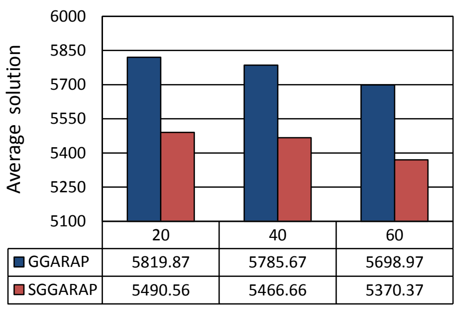
	20	40	60
🟦 GGARAP	1296.69	1265.22	1254.55
🟥 SGGARAP	1289.90	1258.30	1240.96

(a) Comparison on instances with 19 cells



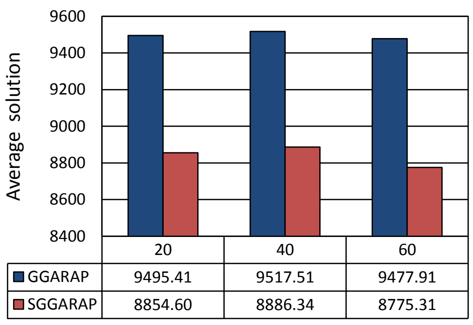
	20	40	60
🟦 GGARAP	3042.09	3055.61	3027.82
🟥 SGGARAP	2967.28	2965.68	2924.64

(b) Comparison on instances with 37 cells



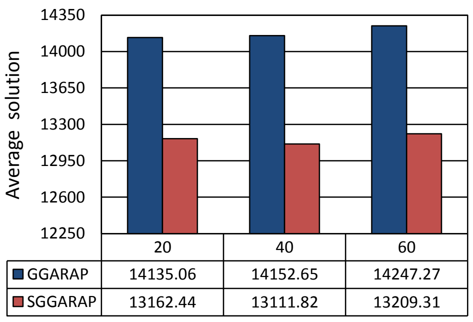
	20	40	60
🟦 GGARAP	5819.87	5785.67	5698.97
🟥 SGGARAP	5490.56	5466.66	5370.37

(c) Comparison on instances with 61 cells



	20	40	60
🟦 GGARAP	9495.41	9517.51	9477.91
🟥 SGGARAP	8854.60	8886.34	8775.31

(d) Comparison on instances with 91 cells

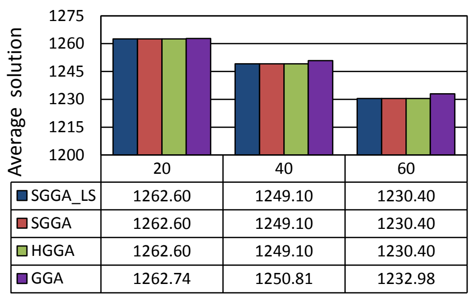


	20	40	60
🟦 GGARAP	14135.06	14152.65	14247.27
🟥 SGGARAP	13162.44	13111.82	13209.31

(e) Comparison on instances with 127 cells

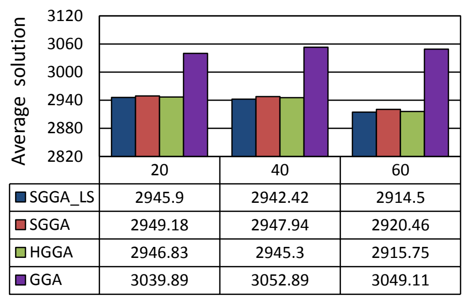
Figure 7.2: Comparison of results GGARAP and SGGARAP on network sizes 19, 37, 61, 91 and 127

with size 61. Instance numbers from 31 to 40, 81 to 90 and 131 to 140 are for the instances with size 91. Instance numbers from 41 to 50, 91 to 100 and 141 to 150 are for the instances with size 127. From Figure 7.4, it can be clearly observed that as the size of the instance increases, the number of solution evaluation also increases for both the approaches. It can also bee seen



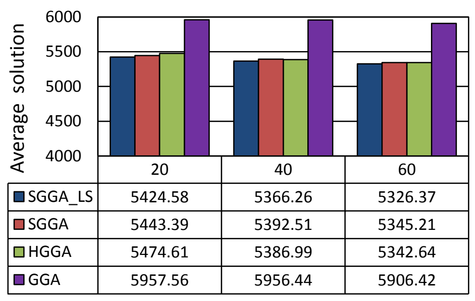
	20	40	60
🟦 SGGA_LS	1262.60	1249.10	1230.40
🟥 SGGA	1262.60	1249.10	1230.40
🟩 HGGA	1262.60	1249.10	1230.40
🟪 GGA	1262.74	1250.81	1232.98

(a) Comparison on instances with 19 cells



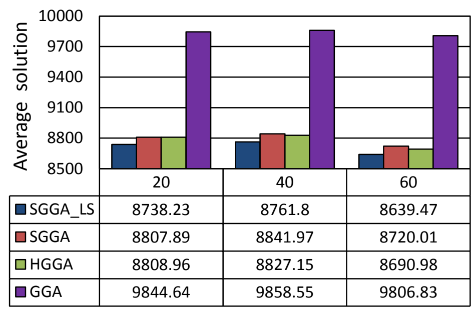
	20	40	60
🟦 SGGA_LS	2945.9	2942.42	2914.5
🟥 SGGA	2949.18	2947.94	2920.46
🟩 HGGA	2946.83	2945.3	2915.75
🟪 GGA	3039.89	3052.89	3049.11

(b) Comparison on instances with 37 cells



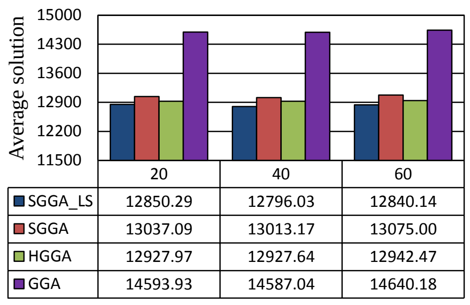
	20	40	60
🟦 SGGA_LS	5424.58	5366.26	5326.37
🟥 SGGA	5443.39	5392.51	5345.21
🟩 HGGA	5474.61	5386.99	5342.64
🟪 GGA	5957.56	5956.44	5906.42

(c) Comparison on instances with 61 cells



	20	40	60
🟦 SGGA_LS	8738.23	8761.8	8639.47
🟥 SGGA	8807.89	8841.97	8720.01
🟩 HGGA	8808.96	8827.15	8690.98
🟪 GGA	9844.64	9858.55	9806.83

(d) Comparison on instances with 91 cells



	20	40	60
🟦 SGGA_LS	12850.29	12796.03	12840.14
🟥 SGGA	13037.09	13013.17	13075.00
🟩 HGGA	12927.97	12927.64	12942.47
🟪 GGA	14593.93	14587.04	14640.18

(e) Comparison on instances with 127 cells

Figure 7.3: Comparison of results SGGA_Ls, SGGA, HGGA and GGA on network sizes 19, 37, 61, 91 and 127

clearly that the variance of instances does not have any significant impact on the total number of solution evaluations performed by the two approaches. Except for small instances with size 19 and 37, where SSGA_LS performed slightly more solution evaluations than HGGA, SSGA_LS outperformed HGGA on remaining instances. On the largest 60 instances, i.e., instances with size 91 and 127, the difference between the number of solution evaluations performed by the two approaches is quite high. Here, It is to be noted that SGGARAP and GGARAP approaches considered in this section do not use any local search and generate 6000 solutions overall so both of these approaches perform 6000 solution evaluations also. Similarly, SGGA and GGA considered in this section do not use any local search and generate 10000 solutions overall, so both of these approaches perform 10000 solution evaluations also.

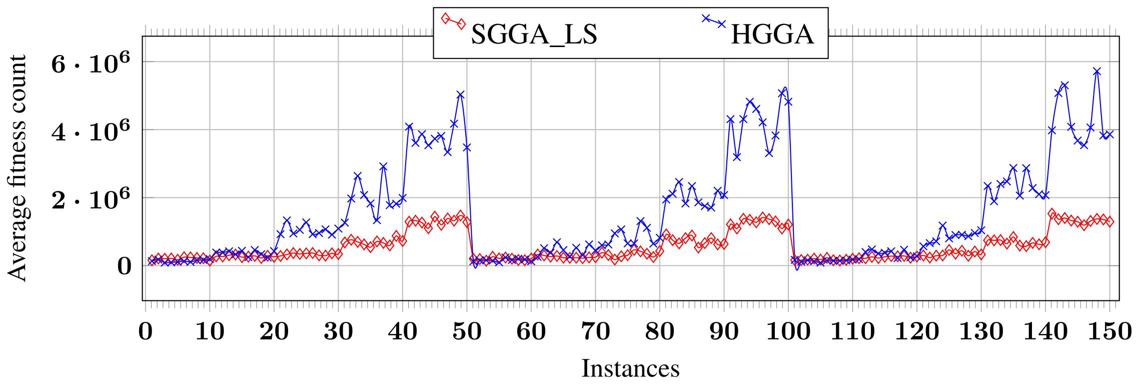


Figure 7.4: Fitness function count for all instances

Influence of parameter settings on solution quality: In order to investigate the influence of parameter settings on the solution quality, we have taken four different instances comprising different variances and network sizes. These instances are represented as Pb(instance index)_v(variance)_n(network size) viz. Pb04_v40_n61, Pb05_v60_n91, Pb08_v20_n127 and Pb07_v40_n127. We have varied all the parameters one by one, and keeping all the other parameters unchanged. In doing so, all other parameters were set to their values reported at the start of this section. The results are reported in Table 7.7. In this table, values in bold show the results with original parameter values which are used in all the experiments involving our approach. From Table 7.7, it can be observed that for all the parameters, the values chosen by us after a large number of trials, give the best possible results in terms of best as well as average solution quality both. The average solution quality for all the four instances vary with parameter values. Note, for parameter IM, it was observed that values greater than 0.02 did not affect the solution quality.

Table 7.7: Influence of parameter settings on solution quality

		Pb04_v40_n61		Pb05_v60_n91		Pb08_v20_n127		Pb07_v40_n127	
Parameter	Value	Best	Mean	Best	Mean	Best	Mean	Best	Mean
	600	5392	5416.40	8670	8741.20	12825	12879.20	12729	12762.00
	500	5414	5415.60	8727	8752.00	12845	12891.40	12691	12743.30
Pop	**400**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	300	5414	5417.80	8724	8769.50	12805	12894.50	12689	12764.40
	200	5392	5441.60	8727	8786.40	12879	12911.40	12729	12771.50
	0.95	5414	5416.00	8724	8763.50	12845	12873.80	12729	12753.00
π_{bt}	**0.90**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	0.85	5414	5417.60	8723	8772.30	12818	12882.10	12711	12747.20
	0.80	5414	5417.40	8710	8745.10	12817	12887.80	12729	12752.00
	0.75	5414	5419.00	8670	8764.90	12818	12865.90	12678	12743.30
	0.90	5392	5413.80	8765	8781.40	12845	12899.20	12717	12763.20
	0.85	5414	5416.40	8746	8771.30	12854	12896.60	12729	12753.50
π_c	**0.80**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	0.75	5414	5421.90	8710	8751.10	12795	12868.50	12729	12740.40
	0.70	5392	5416.40	8724	8758.80	12805	12865.60	12729	12757.60
	0	5414	5417.00	8710	8753.90	12843	12906.40	12691	12761.10
	1	5392	5426.20	8670	8751.50	12845	12899.00	12678	12747.80
R_{left}	**2**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	3	5414	5418.60	8727	8760.90	12845	12881.30	12729	12750.80
	4	5414	5419.00	8670	8738.60	12845	12884.80	12678	12751.40
	4	5392	5413.10	8670	8744.00	12818	12904.00	12678	12747.30
	5	5414	5419.60	8710	8767.20	12817	12885.30	12636	12729.00
Max_{it}	**6**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	7	5392	5415.20	8745	8770.10	12862	12899.20	12689	12747.80
	8	5414	5419.60	8710	8767.20	12817	12885.30	12636	12729.00
	0.05	5392	5411.00	8710	8748.30	12805	12875.50	12711	12763.30
	0.01	5392	5411.00	8710	8751.20	12817	12869.50	12678	12754.30
IM	**0.02**	**5392**	**5411.00**	**8670**	**8737.70**	**12805**	**12873.60**	**12678**	**12727.90**
	0.03	5392	5411.00	8670	8737.70	12805	12873.60	12678	12727.90
	0.04	5392	5411.00	8670	8737.70	12805	12873.60	12678	12727.90

7.5 Conclusions

In this chapter, we have presented a steady-state grouping genetic algorithm (SSGGA) to solve the registration area planning (RAP) problem. We have compared our SSGGA approach with the state-of-the-art approaches for the RAP problem. Computational results show the superiority of our SSGGA approach in all the aspects, viz. quality of best solution found, average solution quality, average execution time and standard deviation of solution values. Performance of our approach improves only slightly with the use of local search which is not the case with one of the state-of-the-art approaches which shows significant improvement with the use of local search.

Chapter 8

Conclusions and Directions for Future Research

The field of solving combinatorial optimization problems using metaheuristic techniques has been matured to the extent that no new approach for solving any combinatorial optimization problem can compete with the state-of-the-art approaches unless it makes proper use of problem specific knowledge. This problem specific knowledge can be used anywhere from solution encoding, design of metaheuristic operators (e.g., crossover and mutation in case of genetic algorithm) to the design of repair operator and local search. Evolutionary techniques are no exception. As a result, most of the new development in the field of solving combinatorial optimization problems using evolutionary techniques are taking place in the context of particular problems only. New evolutionary techniques or a part thereof (e.g., new solution encoding, new genetic operators, new local search heuristics) are proposed and tested in the context of particular problems only. Some of these techniques have wider applicability, but these techniques are either never extended and tested for other problems or these are under explored. EA/G is one such technique. Through our work, we have investigated the capabilities of EA/G in solving some $\mathcal{NP}$-hard combinatorial optimization problems. A major portion of our work has been focussed around developing EA/G based approaches for four $\mathcal{NP}$-hard problems and making these approaches perform as good as or better than the state-of-the-art approaches for the same problem. We have also developed genetic algorithm based approaches for three $\mathcal{NP}$-hard problems. These are the major contributions of this thesis.

In addition, we have modified problem-specific heuristics and local search procedures for some problems to enhance their performance. An artificial bee colony algorithm is also developed for the problem of single machine scheduling with stepwise tardiness.

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

In Chapter 2, we have proposed a hybrid approach combining EA/G with a local search for the set packing problem (SPP) and compared it with two state-of-the-art approaches. On the benchmark instances considered, our approach has outperformed other two approaches in terms of best and average solution quality both. Though one of the two approaches is faster than our approach, this approach performs worst in terms of solution quality among all the approaches.

The method that we have presented in this chapter for initializing the probability vector utilizes the information about the individual objects in addition to the information about composition of initial solutions. This method makes use of the ratio of the weight of object i to the cardinality of its conflict set in addition to using the information about the number of initial solutions containing object i, while initializing the probability for object i. It is to be noted that Zhang *et al.* [55] utilized information about composition of initial solutions only for initializing the probability vector. Similar methods for initializing the probability vector can be developed for other problems also. However, benefit of such a scheme should be ascertained over the one used in [55] before using it for any particular problem. Similarly, when all solutions in the population become identical, instead of reinitializing the population with randomly generated solution, we have randomly perturbed existing solutions (which are same) to reinitialize the population. This helps in finding high quality solutions faster. Usually, randomly generated solutions are much worse than the solutions obtained through perturbation. As a result, when population is reinitialized with randomly generated solutions, search process gets hampered for a while. This reinitialization scheme can be used for other problems also in case empirical observations favor its use for them. Ideas presented in this chapter can be easily adapted for developing EA/G based approaches for other related $\mathcal{NP}$-hard problems such as set covering problem, target coverage problem in wireless sensor networks, multidimensional knapsack problem etc.

Chapter 3 described a hybrid EA/G approach for the minimum weight dominating set (MWDS) problem. In addition to the hybrid EA/G approach, an improved version of a greedy heuristic available in the literature is also presented which has been used in our hybrid approach for initial solution generation and repairing an infeasible solution. The repair operator that we have designed for this problem is computationally more efficient than the repair operators available in the literature. We have compared our hybrid EA/G approach with the best approaches known so far on standard benchmark instances comprising general graphs and unit disk graphs. In comparison to these approaches, our approach obtained better quality solutions in shorter time. Hybrid EA/G approach developed in this chapter can be easily extended to the capacitated minimum weight dominating set problem and the connected minimum weight dominating set problem. A similar approach can be designed for the minimum weight vertex cover problem.

Chapter 4 presented two approaches, viz. a problem specific heuristic and a hybrid EA/G approach for the dominating tree problem (DTP). On disk graph instances of various sizes and transmission range, the problem-specific heuristic produced better results in comparison with existing problem-specific heuristics for this problem. Our hybrid EA/G approach also obtained better results on most of the instances in a much shorter time in comparison to two previously proposed metaheuristic approaches, which are the only metaheuristic approaches proposed so far for DTP.

DTP was introduced with the sole intention of minimizing the energy consumption in a routing scheme. However, as mentioned already in Chapter 4, a DTP solution whose cost is less in comparison to another solution may not have lesser number of nodes also. Actually, lesser number of nodes are desirable from fault tolerance point of view. Therefore, there is a trade-off involving energy consumption and fault tolerance. A possible future work is to study this trade-off by considering a bi-objective version of the DTP, where first objective is same as the objective function of the DTP and second objective is the number of nodes in the dominating tree.

In the original EA/G algorithm, Zhang *et al.* [55] updated the probability vector with the best $\frac{N_p}{2}$ solutions in the current population and created $\frac{N_p}{2}$ solutions by applying guided mutation on the best solution of the population in a generation, where N_p is the population size. These newly created solutions along with the best $\frac{N_p}{2}$ solutions in the current population constitutes the population for the next generation. In Chapter 3, we have updated the probability vector with the best $\frac{N_p}{4}$ solutions in the current population and created $\frac{N_p}{4}$ solutions by applying guided mutation on the best solution of the population in a generation where N_p is the population size. These newly created solutions along with the best $\frac{3N_p}{4}$ solutions of the population in the previous generation constitutes the population in the next generation. Generalizing this, in Chapter 4 we have used L best solutions in the current population to update the probability vector and applied guided mutation to create M new solutions. However, instead of applying guided mutation M times on the best solution to create M new solutions, it is applied once on each of the M best solutions. These M new solutions along with the best $N_p - M$ solutions in the current population constitute the population for the next generation. N_p, L and M were set empirically to 60, 15 and 25 respectively in this chapter. The number of best solutions used to initialize the probability vector and the number of solutions generated through guided mutation are independent, and hence, these two parameters should be set individually. On the other hand, the decision about whether to apply guided mutation M times on the best solution of the population or apply once on each of the M best solutions in the population is problem specific and should be made based on some experimentations only. If we use the former strategy, the population is expected to be less diverse in comparison to the population in the latter strategy. However, the optimal balance between exploration and exploitation is not same for all problems and varies from one problem to another, and as a result, none of the two strategies wields an absolute advantage over the other.

In Chapter 4, we have reinitialized the whole population (minus the best solution) and the probability vector, when best solution has not improved over some number of generations. Actually, for DTP, we never found all population members to be same, a condition used in [55] to reinitialize the population and probability vector. At the same time, we also observed that best solution did not improve over a long duration, and therefore, we tried this strategy of reinitializing the population and the probability vector, and found significantly better results. Similar strategies can be used for other problems.

Chapter 5 addressed the order acceptance and scheduling (OAS) problem in a single machine environment via two hybrid approaches, viz. a hybrid genetic algorithm and a hybrid EA/G. In comparison against two state-of-the-art approaches available in the literature, both of our approaches obtained better quality solutions. However, our approaches are slower than the existing approaches. As far as comparison between the two proposed hybrid approaches is concerned, hybrid genetic algorithm based approach performed slightly better in terms of solution quality than hybrid EA/G approach, but the former approach is slower than the latter approach. A possible future work is to extend our approaches to the version of the OAS problem where orders are processed in a multiple machine environment.

As OAS problem has the characteristics of both subset selection and permutation problems, we have developed a special crossover for this problem. This crossover operator leaves some orders unassigned which are assigned to the solution through an assignment operator. This crossover operator in combination with assignment operator consistently produced better results than the UOB crossover which was also tried. When no position exists where inserting the order increases the total net revenue, then our assignment operator tries to insert an order at a position where inserting the order keeps the total net revenue unaltered and the completion time of the last order reduces. Only when this is also not possible, then the order is inserted at the end. A similar strategy is used in the local search for exchanging two orders. The motivation behind these strategies is to accept an insertion/exchange which makes room to accommodate more orders without causing any loss of revenue. Previously no such strategy was used for OAS problem. Though not presented explicitly, we have also devised a greedy heuristic for the OAS problem and used it to generate the first member of the initial population.

Guided mutation operator in our hybrid EA/G approach for OAS problem tries to insert an order at the position whose probability is maximum when sampling an order from the probability distribution. Similar strategies may be tried for other related scheduling problems also.

Chapter 6 described two hybrid evolutionary approaches for a single machine scheduling problem where tardiness costs of jobs increase in a stepwise manner with respect to various due dates. Our first approach is a hybrid genetic algorithm, whereas the second approach is based on artificial bee colony algorithm. Two versions of the problem are considered. In the first version, all jobs are available for processing at the beginning, whereas in the second version, jobs are supposed to have release dates. A special mutation operator is designed for use in our genetic algorithm based approach. The solutions obtained by our two approaches are improved through a series of local searches for both the versions. Our hybrid approaches outperformed the best approaches available in the literature in terms of solution quality and execution time both on the first version. Even the results of our approaches without the local search are better than the previous approaches. As no other metaheuristic approach is available in the literature for the second version of the problem, we have compared our two approaches among themselves for this version. For both the problem versions, artificial bee colony algorithm based approach outperformed the genetic algorithm based approach in terms of solution quality and running time both. A possible future work is to extend our approaches to the version of the problem where jobs have sequence dependent setup costs.

Probably, our genetic operators are too disruptive for this problem in comparison to two neighboring solution generation operators used in artificial bee colony algorithm, and that is why, artificial bee colony algorithm performs better. We have tried to develop a hybrid EA/G approach also for this problem, but it performed quite worse even in comparison with genetic algorithm, and hence, this approach is not reported. The hybrid EA/G approach presented in Chapter 5 for OAS problem also performed slightly worse than our hybrid genetic algorithm for OAS problem. EA/G uses univariate marginal distribution (UMD) to characterise the distribution of promising solutions in the search space. UMD model works well for subset selection problems, but this seems to be not the case for permutation problems. A possible future work is to explore some other probability models like those used in [169, 170, 171, 172] in EA/G framework for permutation problems.

Chapter 7 presents a hybrid grouping genetic algorithm based approach for the registration area planning (RAP) problem in mobile wireless networks. Though there already exists two grouping genetic algorithms for the RAP problem, our approach is quite different from these. Solution encoding, genetic operators, selection mechanism, population replacement policy used in our approach are all different from previous approaches. Computational results showed the superiority of our approach over these two approaches in terms of solution quality and execution times both. Superior performance of our grouping genetic algorithm based approach over two previously proposed grouping genetic algorithms can be attributed to better solution encoding and proper use of problem specific knowledge in the design of genetic operators. A possible direction for future research is to consider a bi-objective version of RAP problem where one objective is location update cost and another objective is paging cost.

We have not tried any approach based on EA/G for the RAP problem as we have not found any approach based on estimation of distribution algorithms in the literature for any grouping problem among the best approaches for that problem. Though there exists few approaches for grouping problems like those proposed in [173, 174], these approaches are still far from being considered as matured as these approaches are inadequately tested on a small set of benchmark instances of their respective problems. In case of [174], this small set of instances is not even a subset of standard instances. It seems that distribution of promising solutions in the search space is more complex for a grouping problem than for a subset selection or permutation problem and no extant probability distribution model approximates it well.

In all, six problems have been considered in this thesis. Chapters 2 to 4 deal with subset selection problems, Chapter 5 is devoted to a problem that has characteristics of subset selection and permutation problems both, Chapter 6 is focussed on a permutation problem and Chapter 7 addresses a grouping problem. So in this thesis, we have attempted all three kinds of combinatorial optimization problems which is necessary for considering combinatorial optimization problems in totality and for gaining some insight about solving them through any metaheuristic approach in general and evolutionary approaches in particular. Afterall, solution encoding and design of any metaheuristic operator depend completely on the nature of combinatorial optimization problem.

Genetic algorithm is the most widely used evolutionary approach for solving combinatorial optimization problems. In fact, it is one among the most successful metaheuristic approaches. Based on the work reported in this thesis, we can say that EA/G provides an attractive alternative to genetic algorithm for subset selection problems. However, same can not be said about the performance of EA/G on other two kinds of combinatorial optimization problems. For permutation problems, already some estimation of distribution algorithms exist which provide good results in comparison to state-of-the-art approaches. So it will be worthwhile to combine guided mutation with these estimation of distribution algorithms and investigate the performance of the combined approach in solving different permutation problems. As mentioned already, the field of estimation of distribution algorithms for solving grouping problems is still in nascent stage and it needs to progress a lot before one can think about combining guided mutation with estimation of distribution algorithms for grouping problems. Even for subset selection problems, performance of guided mutation in combination with other probability distribution models should be investigated in an EA/G framework for further improving its performance.

At the end, we would again like to stress the point that without the proper use of problem-specific knowledge, no new metaheuristic approach for any problem can compete with state-of-the-art approaches for the same problem. All the approaches presented in this thesis make use of problem-specific knowledge and perform as good as or better than existing state-of-the-art approaches. However, this fact does not preclude the possibility that for each problem there might be some unexplored characteristics that need to be unearthed and analyzed further. This, in turn, will provide us further opportunities to develop new heuristic & metaheuristic approaches and improve existing ones for each problem.

References

[1] X. Delorme, X. Gandibleux, and J. Rodriguez. **GRASP for set packing problems**. *European Journal of Operational Research*, **153**:564–580, 2004. (xi, 33, 34, 39, 40, 42, 43, 44, 45, 46)

[2] X. Gandibleux, X. Delorme, and V. T'Kindt. **An ant colony optimisation algorithm for the set packing problem**. In *ANTS 2004*, **3172**, pages 49–60, Berlin, 2004. (xi, 33, 34, 39, 40, 41, 42, 43, 44, 45, 46)

[3] L.V. Kantorovich. **Mathematical methods of organizing and planning production**. *Management Science*, **6**:366–422, 1939. (1)

[4] L.V. Kantorovich. **George B. Dantzig: Operations research icon**. *Operations Research*, **53**:892–898, 2005. (1)

[5] B. Cesaret, C. Oğuz, and F.S. Salman. **A tabu search algorithm for order acceptance and scheduling**. *Computers & Operations Research*, **39**:1197–1205, 2012. (3, 107, 108, 119, 120)

[6] DT Eliiyi and M. Azizoğlu. **Spread time considerations in operational fixed job scheduling**. *International Journal of Production Research*, **44**:4343–4365, 2006. (3)

[7] M. Dorigo, V.O Maniezzo, and A. Colorni. **Ant system: Optimization by a colony of cooperating agents**. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, **26**:29–41, 1996. (4)

[8] J.H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications in biology, control and artificial intelligence*. University of Michigan Press, Ann Arbor, MI, 1975. (4, 5, 7, 8, 11, 14)

[9] D. GOLDBERG. *Genetic algorithm in search, optimization and machine learning*. Reading, MA :Addison-Wesley, 1989. (4, 9, 15)

[10] F. GLOVER. **Tabu search - part 1**. *ORSA Journal on Computing*, **1**:190–206, 1989. (4)

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

[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. (4)

[13] D. KARABOGA. **An idea based on honey bee swarm for numerical optimization**, 2005. Computer Engineering Department, Erciyes University, Turkey. (4, 139, 140)

[14] J.A. JOINES AND C.R. HOUCK. **On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs**. In *Proceedings Evolutionary Computation*, **2**, pages 579–585, Piscataway, New Jersey, 1994. (5)

[15] I. RECHENBERG. **Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution**. Frommann-Holzboog Verlag, Stuttgart, 1973. (5)

[16] A.E. EIBEN AND J.E. SMITH. *Introduction to evolutionary computing*. Springer, New york, 2003. (5, 10, 16)

[17] J.D. SCHAFFER, D. WHITLEY, AND L.J. ESHELMAN. **Combinations of genetic algorithms and neural networks: A survey of the state of the art**, 1992. IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks. (9)

[18] G. SYSWERDA. **Uniform crossover in genetic algorithms**. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1989. (12)

[19] V.M. SPEARS AND K.A.D. JONG. **On the virtues of parameterized uniform crossover**. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, USA, 1991. Morgan Kauman. (12)

[20] K.M. BURJORJEE. **Explaining optimization in genetic algorithms with uniform crossover**. In *Proceedings of the 2013 Conference on Foundations of Genetic Algorithms*, pages 37–50, Adelaide, Australia, 2013. ACM. (12)

[21] L. DAVIS. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. (13, 17, 19, 108, 110, 137, 161)

[22] J.E. BAKER. **Reducing bias and inefficiency in the selection algorithm**. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, 1987. (15)

[23] M. MITCHELL. *An introduction to genetic algorithms*. Bradford Books, 1998. (15)

[24] D.E. GOLDBERG AND K. DEB. **A comparative analysis of selection schemes used in genetic algorithms**. In *Foundations of Gentic Algorithms*, pages 69–93. Morgan Kaufmann, 1990. (16)

[25] J.C. BEAN. **Genetic algorithms and random keys for sequencing and optimization**. *ORSA Journal on Computing*, **6**:154–160, 1994. (17, 26)

[26] D.E. GOLDBERG AND R. LINGLE. **Alleles, loci and the traveling salesman problem**. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc. (17)

[27] I.M. OLIVER, D.J. SMITH, AND J.R.C. HOLLAND. **A study of permutation crossover operators on the travelling salesman problem**. In *Proceedings of the 2^{st} International Conference on Genetic Algorithms*, pages 224–230, 1987. (18)

[28] H. MÜHLENBIN. **Parallel genetic algorithms in combinatorial optimization**. In *Computer Science and Operations Research*, pages 441–456. Pergamon Press, 1992. (21)

[29] E. FALKENAUER. **The grouping genetic algorithms: widening the scope of the GAs**. *JORBEL : Belgaian Journal of Operations Research, Statistic and Computer Science*, **33**:79–102, 1993. (21, 157, 161)

[30] E. FALKENAUER. *Genetic algorithms and grouping problems*. John Wiley & Sons, Chicester, 1998. (21, 22, 23, 155)

[31] A. Singh and A.K. Gupta. **Two heuristic for the one-dimensional bin-packing problem**. *Operation Research Spectrum*, **29**:765–781, 2007. (22, 23, 157, 158)

[32] M. Hauschild and M. Pelikan. **An introduction and survey of estimation of distribution algorithms**. *Swarm and Evolutionary Computation*, **1**:111–128, 2011. (24, 26)

[33] S. Baluja. **Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning**. Technical report, Carnegin Mellon University, Pittsburgh, PA, 1994. (24, 25, 27)

[34] P. Larrañaga and J.A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, **2**. Springer, 2002. (24)

[35] H. Mühlenbein and G. Paass. **From recombination of genes to the estimation of distributions I. binary parameters**. In *Parallel Problem Solving from Nature-PPSN IV*, pages 178–187. Springer-Verlag, 1996. (24, 25)

[36] M. Pelikan, D.E. Goldberg, and F.G. Lobo. **A survey of optimization by building and using probabilistic models**. *Computational optimization and applications*, **21**:5–20, 2002. (24)

[37] H. Muhlenbein and T. Mahnig. **Convergence theory and applications of the factorized distribution algorithm**. *Journal of Computing and Information Technology*, **7**:19–32, 1999. (24)

[38] Q. Zhang and H. Muhlenbein. **On the convergence of a class of estimation of distribution algorithms**. *IEEE Transactions on Evolutionary Computation*, **8**:127–136, 2004. (24)

[39] Q. Zhang. **On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm**. *IEEE Transactions on Evolutionary Computation*, **8**:80–93, 2004. (24)

[40] J.S.D. Bonet, C.L. Isbell, and Jr.P. Viola. **MIMIC: Finding optima by estimating probability densities**. *Advances in neural information processing systems*, pages 424–430, 1997. (25)

[41] S. Baluja and S. Davies. **Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space**. Technical report, DTIC Document, 1997. (25)

[42] M. Pelikan and H. Mühlenbein. **The bivariate marginal distribution algorithm**. In *Advances in Soft Computing*, pages 521–535. Springer, 1999. (25)

[43] G. Harik and G. Harik. **Linkage learning via probabilistic modeling in the ecga**. Technical report, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999. (26)

[44] H. Mühlenbein and T. Mahnig. **FDA-a scalable evolutionary algorithm for the optimization of additively decomposed functions**. *Evolutionary computation*, **7**:353–376, 1999. (26)

[45] M. Pelikan, D.E. Goldberg, and S. Tsutsui. **Hierarchical bayesian optimization algorithm: toward a new generation of evolutionary algorithms**. In *SICE 2003 Annual Conference*, **3**, pages 2738–2743, 2003. (26)

[46] S. Shakya and R. Santana. **An EDA based on local markov property and gibbs sampling**. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 475–476, New York, NY, USA, 2008. ACM. (26)

[47] J.F. Lemmer and L.N. Kanal. **Propagating uncertainty in bayesian networks by probabilistic logic sampling**. *Uncertainty in Artificial Intelligence 2*, pages 149–163, 2014. (26)

[48] P. Larrañaga, R. Etxeberria, J.A. Lozano, and J.M. Peña. **Optimization in continuous domains by learning and simulation of Gaussian networks**. In *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the Genetic and Evolutionary Computation Conference*, pages 201–204. Springer, 2000. (26)

[49] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres. **Inexact graph matching by means of estimation of distribution algorithms**. *Pattern Recognition*, **35**:2867–2880, 2002. (26)

[50] M. Pelikan, S. Tsutsui, and R. Kalapala. **Dependency trees, permutations, and quadratic assignment problem**, 2007. (26)

[51] S. Tsutsui. **Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram**. In *Parallel Problem Solving from Nature — PPSN VII*, **2439** of *LNCS*, pages 224–233. Springer, 2002. (26)

[52] S. Tsutsui, M. Pelikan, and D.E. Goldberg. **Node histogram vs. edge histogram: A comparison of pmbgas in permutation domains**. Technical report, 2006. (26)

[53] J. Ceberio, E. Irurozki, A. Mendiburu, and J.A. Lozano. **A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem**. *IEEE Transactions on Evolutionary Computation*, **18**:286–300, 2014. (26)

[54] M.A. Fligner and J.S. Verducci. **Distance based ranking models**. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 359–369, 1986. (26)

[55] Q. Zhang, J. Sun, and E. Tsang. **An evolutionary algorithm with guided mutation for the maximum clique problem**. *IEEE Transactions on Evolutionary Computation*, **9**:192–200, 2005. (26, 27, 35, 36, 59, 86, 88, 187, 188, 189)

[56] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. (27)

[57] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. Freeman, San Francisco, 1979. (34, 47, 50, 153)

[58] M.W. Padberg. **On the facial structure of set packing polyhedra**. *Mathematical Programming*, **5**:199–215, 1973. (34)

[59] X.Gandibleux, S. Angibaud, X. Delorme, and J. Rodriguez. **An ant algorithm for measuring and optimizing the capacity of a railway infrastructure**. In *Artificial Ants: From Collective Intelligence to Real-life Optimization and Beyond*, **1**. ISTE Ltd and John Wiley & Sons Inc, 2010. (34)

[60] Aurélien Merel, Xavier Gandibleux, and Sophie Demassey. **A collaborative combination between column generation and ant colony optimization for solving**

set packing problems. In *MIC 2011: The IX Metaheuristics International Conference*, pages 25–28, Italy, 2011. (34)

[61] M. RÖNNQVIST. **A method for the cutting stock problem with different qualities**. *European Journal of Operational Research*, **83**:57–68, 1995. (34)

[62] P.J. ZWANEVELD, L.G. KROON, H.E. ROMEIJN, M. SALOMON, S. DAUZÈRE-PÉRÈS, S.P.M.V. HOESEL, AND H.W. AMBERGEN. **Routing trains through railway stations: Model formulation and algorithms**. *Transportation Science*, **30**:181–194, 1996. (34)

[63] S.-H. KIM AND K.-K. LEE. **An optimization-based decision support system for ship scheduling**. *Computers and Industrial Engineering*, **33**:689–692, 1997. (34)

[64] A. MINGOZZI, V. MANIEZZO, S. RICCIARDELLI, AND L. BIANCO. **An exact algorithm for the project scheduling with resource constraints based on a new mathematical formulation**. *Management Science*, **44**:714–729, 1998. (34)

[65] A. TAJIMA AND S. MISONO. **Using a set packing formulation to solve airline seat allocation/reallocation problems**. *Journal of the Operations Research Society of Japan*, **42**:32–44, 1999. (34)

[66] X. DELORME, J. RODRIGUEZ, AND X. GANDIBLEUX. **Heuristics for railway infrastructure saturation**. Electronic Notes in Theoretical Computer Science, 2001. (34)

[67] F. ROSSI AND S. SMRIGLIO. **A set packing model for the ground holding problem in congested networks**. *European Journal of Operational Research*, **131**:400–416, 2001. (34)

[68] R. LUSBY, J. LARSEN, D. RYAN, AND M. EHRGOTT. **Routing trains through railway junctions: A new set packing approach**. *Transportation Science*, **45**:228–245, 2011. (34)

[69] S. BASAGNI. **Distributed clustering for ad hoc networks**. In *Proceedings ISPAN - 99 International Symposium on Parallel Architectures Algorithms and Networks*, pages 310–315, 1999. (47)

[70] K. ERCIYES, O. DAGDEVIREN, D. COKUSLU, AND D. OZSOYELLER. **Graph theoretic clustering algorithms in mobile ad hoc networks and wireless sensor networks**. *Applied and Computational Mathematics*, **2**:162–180, 2007. (47)

[71] F.G. Nocetti, J.S. Gonzalez, and I. Stojmenovic. **Connectivity-based k-hop clustering in wireless networks**. *Telecommunication Systems*, **22**:205–220, 2003. (47)

[72] D. Bevan and B. Vaduvur. **Routing in ad-hoc networks using minimum connected dominating sets**. In *Proceedings of the 1997 IEEE International Conference on Communications (ICC '97)*, pages 376–380. IEEE, 1997. (47)

[73] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. **Query selection techniques for efficient crawling of structured web sources**. In *Proceedings of the 22nd International Conference on Data Engineering*, page 47, ICDE'06, 2006. (47)

[74] Y. Wang, W. Wang, and X.-Z. Li. **Efficient distributed low-cost backbone formation for wireless networks**. *IEEE Transactions on Parallel and Distributed Systems*, **17**:681–693, 2006. (47, 153, 154)

[75] D. Dai and C. Yu. **A $5+\epsilon$ -approximation algorithm for minimum weighted dominating set in unit disk graph**. *Theoretical Computer Science*, **410**:756–765, 2009. (48)

[76] F. Zou, Y. Wang, X.-H. Xu, X. Li, H. Du, P. Wan, and W. Wu. **New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs**. *Theoretical Computer Science*, **412**:198–208, 2011. (48)

[77] X. Zhu, W. Wang, S. Shan, Z. Wang, and W. Wu. **A PTAS for the minimum weighted dominating set problem with smooth weights on unit disk graphs**. *Journal of Combinatorial Optimization*, **23**:443–450, 2012. (48)

[78] B. Aoun, R. Boutaba, Y. Iraqi, and G. Kenward. **Gateway placement optimization in wireless mesh networks with QoS constraints**. *IEEE Journal on Selected Areas in Communications*, **24**:2127–2136, 2006. (48)

[79] M.E. Houmaidi and M.A. Bassiouni. **K-weighted minimum dominating sets for sparse wavelength converters placement under non-uniform traffic**. In *Proceedings International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, **MASCOTS'03**, pages 56–61, 2003. (48)

[80] D. Subhadrabandhu, S. Sarkar, and F. Anjum. **Efficacy of misuse detection in ad-hoc networks**. In *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, page 97–107, 2004. (48)

[81] C. Shen and T. Li. **Multi-document summarization via the minimum dominating set**. In *Proceedings 23rd International Conference on Computational Linguistics*, pages 984–992, Coling 2010, 2010. (48)

[82] 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 (DIALM 1999), Seattle, USA*, pages 7–14, 1999. (48, 71)

[83] R. Jovanovic, M. Tuba, and D. Simian. **Ant colony optimization applied to minimum weight dominating set problem**. In *Proceedings of the 12th WSEAS international conference on Automatic control, modelling and simulation (ACMOS'10)*, pages 322–326, 2010. (49, 50, 59, 61)

[84] S.J. Shyu, P.-Y. Yin, and B.M. Lin. **An ant colony optimization algorithm for the minimum weight vertex cover problem**. *Annals of Operation Research*, **13**:283–304, 2004. (49, 154)

[85] A. Potluri and A. Singh. **Hybrid metaheuristic algorithms for minimum weight dominating set**. *Applied Soft Computing*, **13**:76–88, 2013. (49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61)

[86] M. Mastrogiovanni. **The clustering simulation framework: A simple manual**. `http://www.michele-mastrogiovanni.net/software/download/README.pdf.`, 2007. (60)

[87] I. Shin, Y. Shen, and M.T. Thai. **On approximation of dominating tree in wireless sensor networks**. *Optimization Letters*, **4**:393–403, 2010. (71, 72, 73, 74, 89, 92)

[88] N. Zhang, I. Shin, B. Li, C. Boyaci, R. Tiwari, and M.T. Thaiand. **New approximation for minimum-weight routing backbone in wireless sensor network**. In *Wireless Algorithms, Systems, and Applications*, **5258**, pages 96–108. Springer, 2008. (71, 72, 73, 74, 89, 92)

[89] 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. (72)

[90] G. Wan and E. Lin. **Cost reduction in location management using semi-realtime movement information**. *Wireless Network*, **5**:245–256, 1999. (72, 154)

[91] S. Guha and S. Khuller. **Approximation algorithms for connected dominating sets**. *Algorithmica*, **20**:374–387, 1998. (72)

[92] M.A. Park, J. Willson, C. Wang, M. Thai, W. Wu, and A. Farago. **A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges**. In *Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '07, pages 22–31, New York, NY, USA, 2007. ACM. (72)

[93] M.T. Thaiand, 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**:1098–1109, 2008. (72)

[94] E.M. Arkin, M.M. Halldórsson, and R. Hassin. **Approximating the tree and tour covers of a graph**. *Information Processing Letters*, **47**:275–282, 1993. (72)

[95] T. Fujito. **On approximability of the independent/connected edge dominating set problems**. *Information Processing Letters*, **79**:261–266, 2001. (72)

[96] T. Fujito. **How to trim an mst: A 2-approximation algorithm for minimum cost tree cover**. In *Automata, Languages and Programming*, **4051** of *LNCS*, pages 431–442. Springer, 2006. (72)

[97] S. Sundar and A. Singh. **New heuristic approaches for the dominating tree problem**. *Applied Soft Computing*, **13**:4695–4703, 2013. (74, 75, 77, 78, 81, 83, 85, 89, 91, 92, 93)

[98] R.C. Prim. **Shortest connection networks and some generalizations**. *Bell Systems Technical Journal*, **36**:1389–1401, 1957. (77)

[99] 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. (83)

[100] H.H. Guerrero and G.M. Kern. **How to more effectively accept and refuse orders**. *Production and Inventory Management*, **29**:59–62, 1988. (106)

[101] P. Keskinocak and S. Tayur. **Due date management policies**. *Handbook of Quantitative Supply Chain Analysis International Series in Operations Research & Management Science*, **74**:485–554, 2004. (106)

[102] S.A. Slotnick. **Order acceptance and scheduling: A taxonomy and review**. *European Journal of Operational Research*, **212**:1–11, 2011. (106)

[103] H. Stern and Z. Avivi. **The selection and scheduling of textile orders with due dates**. *European Journal of Operational Research*, **44**:11–16, 1990. (107)

[104] K. Charnsirisakskul, P.M. Griffin, and P. Keskinocak. **Order selection and scheduling with leadtime flexibility**. *IIE Transactions*, **36**:697–707, 2004. (107)

[105] K. Charnsirisakskul, P.M. Griffin, and P. Keskinocak. **Pricing and scheduling decisions with leadtime flexibility**. *European Journal of Operational Research*, **171**:153–169, 2006. (107)

[106] J. Geunes B. Yang. **Heuristic approaches for solving single resource scheduling problems with job-selection flexibility**. Technical report, Technical Report, Department of Industrial & Systems Engineering, University of Florida, 2003. (107)

[107] C. Oğuza, F.S. Salmana, and Z.B. Yalçin. **Order acceptance and scheduling decisions in make-to-order systems**. *International Journal of Production Economics*, **125**:200–211, 2010. (107, 108)

[108] F.T. Nobibon and R. Leus. **Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment**. *Computers & Operations Research*, **38**:367–378, 2011. (107)

[109] H.F. Lewis and S.A. Slotnick. **Multi-period job selection: Planning work loads to maximize profit**. *Computers & Operations Research*, **29**:1081–1098, 2002. (107)

[110] D.W. Engelsa, D.R. Kargerb, S.G. Kolliopoulosc, S. Senguptab, R.N. Umad, and J. Weine. **Techniques for scheduling with rejection**. *Journal of Algorithms*, **49**:175–191, 2003. (107)

[111] V.S. Gordona and V.A. Strusevich. **Single machine scheduling and due date assignment with positionally dependent processing times**. *Computers & Operations Research*, **198**:57–62, 2009. (107)

[112] S. Nguyen, M. Zhang, and M. Johnston. **Enhancing branch-and-bound algorithms for order acceptance and scheduling with genetic programming**. In *Genetic Programming, LNCS*, **8599**, pages 124–136, 2014. (107)

[113] J.B. Ghosh. **Job selection in a heavily loaded shop**. *Computers & Operations Research*, **24**:141–145, 1997. (107)

[114] S.A. Slotnick and T.E. Morton. **Order acceptance with weighted tardiness**. *Computers & Operations Research*, **34**:3029–3042, 2007. (107)

[115] I.S. Lee and C.S. Sung. **Single machine scheduling with outsourcing allowed**. *International Journal of Production Economics*, **111**:623–634, 2008. (107)

[116] C. Akkan. **Finite-capacity scheduling-based planning for revenue-based capacity management**. *European Journal of Operational Research*, **100**:170–179, 1997. (107)

[117] B. Yang and J. Geunes. **A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times**. *Computers & Industrial Engineering*, **53**:420–432, 2007. (107)

[118] Y.-W. Chen, Y.-Z. Lu, and G.-K. Yang. **Hybrid evolutionary algorithm with marriage of genetic algorithm and extremal optimization for production scheduling**. *The International Journal of Advanced Manufacturing Technology*, **36**:959–968, 2008. (107)

[119] W.O. Rom and S.A. Slotnick. **Order acceptance using genetic algorithms**. *Computers & Operations Research*, **36**:1758–1767, 2009. (107)

[120] W. LIN AND K.-C. YING. **Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm**. *journal of the Operational Research Society*, **64**:293–311, 2013. (107, 108, 109, 110, 113, 119, 120)

[121] R. RUIZ AND T. STÜTZLE. **A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem**. *European Journal Operational Research*, **177**:2033–2049, 2007. (108, 113)

[122] K.-C. YING AND S.-W. LIN. **Unrelated parallel machine scheduling with sequence and machine-dependent setup times and due date constraints**. *International Journal of Innovative Computing, Information and Control*, **8**:3279–3297, 2012. (108)

[123] A. SALHI, J.A.V. RODRIGUEZ, AND Q. ZHANG. **An estimation of distribution algorithm with guided mutation for a complex flow shop scheduling problem**. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 570–576, 2007. (115)

[124] C.-T. TSENG AND K.-H. CHEN. **An electromagnetism-like mechanism for the single machine total stepwise tardiness problem with release dates**. *Engineering Optimization*, **45**:1431–1448, 2013. (132, 133, 134, 135, 136, 145, 146, 150)

[125] J.K. LENSTRA, A.H.G. RINNOOY KAN, AND P. BRUCKER. **Complexity of machine scheduling problems**. *Annals of Discrete Mathematics*, **1**:343–362, 1977. (133)

[126] R.L. GRAHAM, E.L. LAWLER, J.K. LENSTRA, AND A.H.G. RINNOOY KAN. **Optimization and approximation in deterministic sequencing and scheduling**. *Annals of Discrete Mathematics*, **4**:287–326, 1979. (133)

[127] J. CURRY AND B. PETERS. **Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives**. *International Journal of Production Research*, **43**:3231–3246, 2005. (133, 134)

[128] G. SAHIN. **New combinatorial approaches for solving rail road planning and scheduling problems. Ph.D. Dissertation**, 2006. University of Florida. (133, 134)

[129] B. DETIENNE, S. DAUZÈRE-PÉRÈS, AND C. YUGMA. **Scheduling in section operations subject to a fixed production schedule**. In *proceedings of the 4th Multidisciplinary International Conference (MISTA2009)*, pages 581–593, Dublin, Ireland, 2009. (134)

[130] B. DETIENNE, S. DAUZÈRE-PÉRÈS, AND C. YUGMA. **An exact approach for scheduling jobs with regular step cost functions on a single machine**. *Computers & Operations Research*, **39**:1033–1043, 2012. (134, 145)

[131] I. BIRBIL AND S.-C. FANG. **An electromagnetism-like mechanism for global optimization**. *Journal of Global Optimization*, **30**:263–282, 2003. (134)

[132] M.J. MOORE. **An n job, one machine sequencing algorithm for minimizing the number of late jobs**. *Management Science*, **15**:102–109, 1968. (134)

[133] M. NAWAZ, E.E. ENSCORE JR, AND I. HAM. **A heuristic algorithm for the m-machine, n-job flowshop sequencing problem**. *Omega*, **11**:91–95, 1983. (134)

[134] M. SEVAUX AND S. DAUZÈRE-PÉRÈS. **Genetic algorithms to minimize the weighted number of late jobs on a single machine**. *European Journal of Operational Research*, **151**:296–306, 2003. (134, 145)

[135] M.F. TASGETIREN, Y.-C. LIANG, M. SEVKLI, AND G. GENCYILMAZ. **Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem**. *International Journal of Production Research*, **44**:4737–4754, 2006. (134)

[136] P.-C. CHANG, S.-H. CHEN, AND C.Y. FAN. **A hybrid electromagnetism-like algorithm for single machine scheduling problem**. *Expert Systems with Applications*, **36**:1259–1267, 2009. (134)

[137] J.E. BEASLEY AND P.C. CHU. **A genetic algorithm for the set covering problem**. *European Journal of Operational Research*, **94**:392–404, October (2)1996. (136)

[138] R. RUIZ, C. MAROTO, AND J. ALCARAZ. **Two new robust genetic algorithms for the flowshop scheduling problem**. *Omega*, **34**:461–476, 2006. (136)

[139] D. KARABOGA AND B. BASTURK. **On the performance of artificial bee colony (ABC) algorithm**. *Applied Soft Computing*, **8**:687–697, 2008. (140)

[140] D. KARABOGA AND B. BASTURK. **A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm**. *Journal of Global Optimization*, **39**:459–471, 2007. (140)

[141] D. KARABOGA, B. GORKEMLI, C. OZTURK, AND N. KARABOGA. **A comprehensive survey: artificial bee colony (ABC) algorithm and applications**. *Artificial Intelligence Review*, **42**:21–57, 2014. (141)

[142] S. SUNDAR AND A. SINGH. **A swarm intelligence approach to the early/tardy scheduling problem**. *Swarm and Evolutionary Computation*, **4**:25–32, 2012. (141)

[143] A. SINGH. **An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem**. *Applied Soft Computing*, **9**:625–631, 2009. (141)

[144] 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. (141, 142)

[145] R. M'HALLAH AND R.L. BULFIN. **Minimizing the weighted number of tardy jobs on a single machine with release dates**. *European Journal of Operational Research*, **176**:727–744, 2007. (145)

[146] F.J. VILLAREAL AND R.L. BULFIN. **Scheduling a single machine to minimize the weighted number of tardy jobs**. *IIE Transactions*, **15**:337–343, 1983. (145)

[147] R. M'HALLAH AND R.L. BULFIN. **Minimizing the weighted number of tardy jobs on a single machine**. *European Journal of Operational Research*, **145**:45–56, 2003. (145)

[148] J. XU, D.L. LEE, AND B. LI. **On bandwidth allocation for data dissemination in cellular mobile networks**. *Wireless Networks*, **9**:103–116, 2003. (152)

[149] A. KUMAR, M.N.UMESH, AND R. JHA. **Mobility modeling of rush hour traffic location area design in cellular networks**. In *Proceedings of the Third ACM International Workshop on Wireless Mobile Multimedia*, pages 48–54, Bostan, MA, 2000. (153)

[150] I.Demirkol, C. Ersoy, M. Ufuk Çağlayan, and H.D. Deliç. **Location area planning and cell-to-switch assignment in cellular networks**. *IEEE Transactions on Wireless Communications*, **3**:880–890, 2004. (153, 154)

[151] I. Demirkol, C. Ersoy, M. Ufuk Çağlayan, and H.D. Deliç. **Location area planning in cellular networks using simulated annealing**. In *Proceedings of the* 23^{rd} *Conference of the IEEE Communications Society*, **1**, pages 13–20, Anchorage, AK, 2001. (153, 154)

[152] A. Gamst. **Application of graph theoretical methods to GSM radio network planning**. In *Proceedings of the IEEE Symposium on Circuits and Systems*, pages 942–950, Sydney, Australia, 1991. (154)

[153] J.G. Markoulidakis and E.D. Sykas. **Method for efficient location area planning in mobile telecommunications**. In *Electronics Letters*, **29**, pages 2165–2166, 1993. (154)

[154] J. Plehn. **The design location areas in a GSM-network**. In *Proceeding of the IEEE* 45^{th} *Vehicular Technology Conference*, **2**, pages 871–885, Chicago, IL, 1995. (154)

[155] Y. Bejerano and I. Cidon. **Efficient location management based on moving location areas**. In *Proceedings of the* 23^{rd} *Conference of the IEEE Communications Society*, **1**, pages 3–12, Anchorage, AK, 2001. (154)

[156] Y. Bejerano and I. Cidon. **An efficient mobility management strategy for personal communication systems**. In *Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking*, pages 215–222, Dallas, TX, 1998. (154)

[157] J. Biesterfeld and K. Jobmann. **The use of prediction areas to improve mobility management algorithms**. In *Proceedings of the International Conference on Telecommunications*, **4**, pages 446–450, Chaldiki, 1998. (154)

[158] P.S. Bhattacharjee, D. Saha, and A. Mukherjee. **Intelligent paging strategies for personal communication services network**. In *Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access*, pages 36–43, Seattle, WA, 1999. (154)

[159] P.S. Bhattacharjee, D. Saha, and A. Mukherjee. **An approach for location area planning in a personal communication services network (PCSN)**. *IEEE Transactions on Wireless Communications*, **3**:1176–1187, 2004. (154)

[160] P.S. Bhattacharjee, D. Saha, and M. Maitra. **Location area planning for personal communication services networks**. In *Proceedings of Second ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, pages 95–98, W.A. Seattle, 1999. (154)

[161] Y. Bejerano, N. Immorlica, J. Naor, and M.A. Smith. **Efficient location area planning for personal communication systems**. In *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking*, pages 109–121, San Diego, CA, 2003. (154)

[162] P. Demestichas, E. Tzifa, V. Demesticha, N. Georgantas, G. Kotsakis, M. Kilanioti, M. Striki, M.E. Anagnostou, and M.E. Theologou. **Control of the location update and paging signaling load in cellular systems by means of planning tools**. In *Proceedings of the* 50^{th} *Vehicular Technology Conference*, **4**, pages 2119–2123, Amsterdam, 1999. (154)

[163] C.U. Saraydar and C. Rose. **Location area design using population and traffic data**. In *Proceedings of the* 32^{nd} *Annual Conference on Information Science and System*, pages 739–744, Princeton, 1998. (154)

[164] P.R.L. Gondim. **Genetic algorithms and the location area partition problem in cellular networks**. In *Proceedings of the IEEE* 46^{th} *Vehicular Technology Conference*, **3**, pages 1835–1838, Atlanta, GA, 1996. (154)

[165] C. Hedible and S. Pierre. **A genetic algorithm for assigning cells to switches in personal communication networks**. *IEEE Canadian Review*, **44**:21–24, 2003. (154)

[166] M. Vroblefski and E.C. Brown. **A grouping genetic algorithm for registration area planning**. *Omega-International Journal of Management Science*, **34**:220–230, 2006. (155, 157, 161, 167, 168, 169)

[167] T. James, M. Vroblefski, and Q. Nottingham. **A hybrid grouping genetic algorithm for the registration area planning problem**. *Computer Communications*, **30**:2180–2190, 2007. (155, 157, 161, 162, 165, 166, 167, 168, 169)

[168] Y.-B. LIN AND V.W. MAK. **Eliminating the boundary effect of a large-scale personal communication service network simulation**. *ACM Transactions on Modeling and Computer Simulation*, **4**:165–190, 1994. (158, 168)

[169] S. TSUTSUI. **Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram**. In *Parallel Problem Solving from Nature — PPSN VII*, **2439**, pages 224–233. Springer, 2002. (191)

[170] S. TSUTSUI. **Node histogram vs. edge histogram: A comparison of PMBGAs in permutation domains**. In *IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada*. 2006. (191)

[171] J. CEBERIO, E. IRUROZKI, A. MENDIBURU, AND J.A. LOZANO. **A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems**. *Progress in Artificial Intelligence*, **1**:103–117, 2012. (191)

[172] J. CEBERIO, A. MENDIBURU, AND J.A. LOZANO. **Introducing the mallows model on estimation of distribution algorithms**. In *Neural Information Processing*, **7063**, pages 461–470. Springer, 2011. (191)

[173] Q. ZHANG, B. LIU, L. BI, Z. WANG, AND B. MA. **Estimation of distribution algorithms for the machine-part cell formation**. In *Advances in Computation and Intelligence*, **5821** of *LNCS*, pages 82–91. Springer, 2009. (191)

[174] Y. CAI, H. CHEN, R. XU, H. SHAO, AND X. LI. **An estimation of distribution algorithm for the 3D bin packing problem with various bin sizes**. In *Intelligent Data Engineering and Automated Learning*, **8206** of *LNCS*, pages 401–408. Springer, 2013. (191)

List of Publications

[1] SACHCHIDA NAND CHAURASIA, ALOK SINGH. **A hybrid evolutionary approach to the registration area planning problem.** *Applied Intelligence, Springer, Volume -41, Issue- 4, Pages-1127-1149, 2014.*

[2] SACHCHIDA NAND CHAURASIA, SHYAM SUNDAR, ALOK SINGH. **A hybrid evolutionary approach for set packing problem.** Available online 29 Jul 2014, in the journal *OPSEARCH*. http://dx.doi.org/10.1007/s12597-014-0184-3. Published by Springer.

[3] SACHCHIDA NAND CHAURASIA, ALOK SINGH. **A hybrid heuristic for dominating tree problem.** Available online 13 Nov 2014 , in the journal *Soft Computing*. http://dx.doi.org/10.1007/s00500-014-1513-4. Published by Springer.

[4] SACHCHIDA NAND CHAURASIA, ALOK SINGH. **A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set.** Available online 23 Apr 2015 , in the journal *Applied Intelligence*. http://dx.doi.org/10.1007/s10489-015-0654-1s. Published by Springer.

[5] SACHCHIDA NAND CHAURASIA, ALOK SINGH. **Hybrid evolutionary approaches for the single machine order acceptance and scheduling problem.** Communicated to *Applied soft computing, Elsevier, 2015.*

[6] SACHCHIDA NAND CHAURASIA, SHYAM SUNDAR, ALOK SINGH. **Hybrid metaheuristic approaches for the single machine total stepwise tardiness problem with release dates.** Communicated to *Operational Research, Springer, 2015.*