Genetic Algorithms for University Course Timetabling Problems

by

Sadaf Naseem Jat

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Department of Computer Science
University of Leicester

May 2012

Declaration of Authorship

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Dr. Shengxiang Yang during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content.

Part of the research work presented in this submission has been published or has been submitted for publication in the following papers:

- S. Yang and S. N. Jat. Multi-objective evolutionary algorithms with guided and local search strategies for the multi-objective university course timetabling problem. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, to be submitted in December 2011. (This paper contains much of the work that is presented in Chapter 7).
- S. N. Jat and S. Yang. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *Journal of Scheduling*, published online first: 3 November 2010. Springer. (The detail of this work is presented in Chapter 6).
- S. N. Jat and S. Yang. A guided search non-dominated sorting genetic algorithm for the multi-objective university course timetabling problem. *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2011)*, Lecture Notes in Computer Science 6622, pp. 1–13, 2011. Springer. (In this paper, the university course timetabling problem is represented as a multi-objective optimization problem, and the work is presented as an initial part of Chapter 7).
- S. Yang and S. N. Jat. Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, 41(1): 93–106, January

2011. IEEE Press. (This paper contains much of the work that is presented in Chapter 5).

- S. N. Jat and S. Yang. A guided search genetic algorithm for the university course timetabling problem. *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009)*, pp. 180-191, 2009. (The work is presented in the first half of Chapter 5).

- S. N. Jat and S. Yang. A memetic algorithm for the university course timetabling problem. *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 427–433, 2008. IEEE Press. (This paper presents the initial work on genetic algorithms with local search for the university course timetabling problem, and the work is presented in Chapter 4).

Abstract

The university course timetabling problem is a difficult optimisation problem due to its highly-constrained nature. Finding an optimal, or even a high quality, timetable is a challenging task, especially when resources (e.g., rooms and time slots) are limited. In the literature, many approaches have been studied to solve this problem. In this thesis, we investigate genetic algorithms to solve the problem because they have been successfully used for a wide range of real-world problems. However, for university course timetabling problems, traditional genetic algorithms are not usually considered as efficient solvers.

In this thesis, we investigate genetic algorithms to acquire good solutions for university course timetabling problems. Several ideas are introduced to increase the general performance of genetic algorithms on these problems. Local search strategies are introduced into the traditional genetic algorithm to enhance its performance for the university course timetabling problem. This differs from many works in the literature because it works on time slots of the timetable rather than events directly. A guided search approach is also introduced into genetic algorithms to produce high quality individuals into the population. In the guided search technique, the best parts of selected individuals from the current population are stored in an extra memory (or data structure) and are re-used to guide the generation of new individuals for subsequent populations.

In addition to solving university course timetabling problems as a single-objective optimisation problem, we also tackle the multi-objective university course timetabling problem. We integrate the above proposed approaches into multi-objective evolutionary algorithms and propose a framework of multi-objective evolutionary algorithms based on local search and guided search strategies for the multi-objective university course timetabling problem. This framework is then instantiated into a set of multi-objective evolutionary algorithms for the multi-objective university course timetabling problem based on a set of multi-objective evolutionary algorithms that are typically used for general multi-objective optimisation problems.

Computational results based on a set of well-known university course timetabling benchmark instances, show the effectiveness of the proposed approaches for both single- and multi-objective university course timetabling problems.

Acknowledgements

In the name of Almighty God, the Most Gracious, the Most Merciful, I thank to God, who always bless me and provided me great people to work with them. I thank them all who made this thesis possible.

First, I would like to express my heartfelt gratitude to my supervisor, Dr Shengxiang Yang. He provided me invaluable guidance, encouragement and fantastic support throughout my PhD study. His guidance, help and suggestion made me more confident to overcome many difficulties during my research. He has surely made a great impact upon me, which shall be with me throughout the future. Without him, this thesis would not have been possible. I am very great full of gratitude to him.

I also express my sincere gratitude to Prof. Thomas Erlebach and Dr. Fer-Jan de Vries for their advice, encouragement, support and assessing my yearly reports presented to them. I would like to say special thank to Prof. Thomas Erlebach for his valuable comments on the completion of Chapter 7 of this thesis. Special thanks to Prof. Rick Thomas for his kindness and support throughout my study by allowing me to bring my three year old daughter to university with me during my hard time of study.

I would like to thanks my examiners, Prof. Graham Kendall and Prof. Rajeev Raman, for enduring the work represented to them and for their recommendations to steer my thesis to an apprehensible one.

It is my pleasure to acknowledge the University of Sindh, Jamshoro, for their sponsorship to my PhD study. I would also like to thanks all the members of the Computer Science Department at the University of Leicester. Special thanks to John Landamore, Gilbert Laycock, and Richard Grant, for their constant technical help during this study period.

I express special thanks to all my PhD colleagues and friends with whom I have spent privileged time during my study in Leicester. Thanks to their kindness and generousness.

Specially, I want to appreciate the support from my family, my parents, my brothers, my aunties, and my uncles. Thanks for their prayers, support, and encouragement.

Each of them in their own way contributed actively toward my well-being. Thank you all for being there for me, as always.

The great thanks go to my daughter Eshal, who is the source of my encouragement. Thanks to her for her prayers, for her patience, for being understandable and caring during my PhD period, for all which I need in difficult time.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

EA	**E**volutionary **A**lgorithm
EGSGA	**E**xtended **G**uided **S**earch **G**enetic **A**lgorithm
ϵ-GSMOEA	ϵ-**G**uided **S**earch **M**ulti-**O**bjective **E**volutionary **A**lgorithms
ϵ-MOEA	ϵ-**M**ulti-**O**bjective **E**volutionary **A**lgorithms
GA	**G**enetic **A**lgorithm
GS	**G**uided **S**earch
GSGA	**G**uided **S**earch **G**enetic **A**lgorithm
GSNSGA	**G**uided **S**earch **N**on-dominated **S**orting **G**enetic **A**lgorithm
GSPAES	**G**uided **S**earch **P**areto **A**rchived **E**volutionary **S**trategy
GSSPEA	**G**uided **S**earch **S**trength **P**areto **E**volutionary **A**lgorithm
HGATS	**H**ybrid **G**enetic **A**lgorithm **T**abu **S**earch **A**pproach
ITC-2002	the 2002 **I**nternational **T**imetabling **C**ompetition
ITC-2007	the 2007 **I**nternational **T**imetabling **C**ompetition
LS	**L**ocal **S**earch
MA	**M**emetic **A**lgorithm
MOEA	**M**ulti-**O**bjective **E**volutionary **A**lgorithms
MOOP	**M**ulti-**O**bjective **O**ptimization **P**roblem
MOUCTP	**M**ulti-**O**bjective **U**niversity **C**ourse **T**imetabling **P**roblem
NSGA-II	**E**litist **N**on-dominated **S**orting **G**enetic **A**lgorithm
PAES	**P**areto **A**rchived **E**volutionary **S**trategy
PECTP	**P**ost **E**nrollment **C**ourse **T**imetabling **P**roblem
SPEA-II	**I**mproved **S**trength **P**areto **E**volutionary **A**lgorithm

SSGA	**S**teady **S**tate **G**enetic **A**lgorithm
SSMA	**S**teady **S**tate **M**emetic **A**lgorithm
TS	**T**abu **S**earch
UCTP	**U**niversity **C**ourse **T**imetabling **P**roblem

Symbols

α	percentage of best individuals selected from the current population to construct the MEM data structure
β	percentage of the total number of events that are used to create an offspring through guided search
ϵ	minimum allowable tolerance used in ϵ-MOEA and ϵ-GSMOEA
γ	probability that indicates whether an offspring is created through the guided search strategy or crossover
λ	number of offspring created in each generation in ϵ-MOEA and ϵ-GSMOEA
m	number of objectives in an MOOP ($m = 3$ for the MOUCTP in this thesis)
MEM	**MEM**ory or data structure used for the guided search strategy
N	population size
P_c	crossover probability
P_m	mutation probability
s_{max}	local search step size (maximum number of steps per local search operation)
t_{max}	maximum allowable time for an algorithm to run once on a problem
τ	frequency of updating the MEM data structure in the guided search strategy

Dedicated to my family...

Chapter 1

Introduction

Timetabling is a common scheduling problem, which can be described as the allocation of resources for tasks under predefined constraints so that it maximizes the possibility of allocation or minimizes the violation of constraints [200]. Timetables are usually necessary to organise tasks and resources in different areas of life, such as education, transportation, industries, hospital, and entertainment. This organisation of tasks becomes very challenging when resources (such as time, people, and space) are limited, which usually occurs in our daily life. Timetabling problems are often made complicated by the details of a particular timetabling task. In addition, it is a problem that people of different fields have to face on a fairly regular basis. For example, university timetables often have to be scheduled at the beginning of the semester, and transport timetables have to be modified according to the addition of new bus stops, etc.

Effective timetables greatly affect different real-world problems, and are essential to the growth of work in many fields. Therefore, it is essential constructing such a good or optimal timetable. However, to construct such a good or optimal timetables is a very difficult task due to the highly constrained nature of the problem. A general algorithm approach to one problem may turn out to be not suitable for another problem, because certain special constraints are required in a particular instance of that problem.

Typical timetabling are include educational timetabling [13, 208], sports timetabling [133], transport timetabling [28] and employee timetabling [43], etc. Educational timetabling can be divided into school timetabling, exam timetabling, and course timetabling. In a university context, the course timetabling problem (UCTP) is complex since events (e.g., subjects and courses) have to be allocated into a number of time slots and rooms while satisfying various constraints. It is very difficult to find a general and effective solution for the UCTP due to the diversity of the problem, the variance of constraints, and the particular requirements that change from university to university according to their individual characteristics. There is no known deterministic polynomial time algorithm for the UCTP since it is an NP-hard combinatorial optimization problem [96].

1.1 Background and Motivation

The application of computers to timetabling problems has a long and active history, which dates back almost to the time computers were first built [191, 213]. Conventional computer based timetabling methods are closely related to timetabling by hand. They concern themselves more with simply finding and maintaining the lists of periods to tables [191]. However, these methods are insufficient to satisfy all the required constraints. The solution to such problems using knowledge based or operation research based approaches is also hard to develop. These approaches are often slow and can be inflexible because they are based on specific assumptions about the nature of the problem [24, 35, 191]. The first generation of computer timetabling programmes developed in the early 1960s were largely an attempt to reduce the associated administration work [120]. Thereafter, programs were soon presented with the aim of fitting classes and teachers to periods. In 1964, Broder [41] and Cole [72] both presented heuristic approaches to timetabling. In 1967, Welsh and Powell [206] pointed out the similarity between the timetabling problem and colouring the nodes of a graph. Colouring the graph amounts to placing courses in appropriate periods. The algorithm they presented was similar to Broder's algorithm [41].

Researchers have proposed various timetabling approaches by using constraint-based methods [10], meta-heuristic methods (e.g., tabu search (TS) [185], simulated annealing (SA) [95], genetic algorithms (GAs) [17], ant colony optimization (ACO) [152, 196], particle swarm optimisation (PSO) [131], great deluge [195]), variable neighbourhood search (VNS) [13], hybrid meta-heuristics, and hyper-heuristic [47] approaches, etc. In the last decade, numerous research papers have been published. The earliest algorithms are based on graph colouring heuristics. These algorithms are very efficient when solving small instances of timetabling problems, but have not proved to be very efficient for large problem instances [109, 135, 208]. Later, meta-heuristic algorithms, such as GAs, SA, and TS, etc., were introduced to solve timetabling problems [43].

Generally speaking, there are two types of meta-heuristic algorithms [22]: local area based algorithms and population-based algorithms. Each type has some advantages and disadvantages. A local area based algorithm starts from an initial state/solution and tries to find a better solution in the space of candidate solutions until a stopping criterion is met [105]. Local area based algorithms differ from each other in the method that is used to find a neighbourhood solution in the search space and the criterion to stop the search. Local area based algorithms include SA [20, 204], very large neighbourhood search [13, 14], TS [21, 150], and many more. Usually, local area based algorithms focus more on exploitation (i.e., using collected information to direct further search to the promising search area) rather than exploration (i.e., discovery of new regions in the search space) [22, 71]. They usually work in a non-systematic way that may lead to finding a solution in one direction without performing a wider scan of the search space [22, 105]. Population-based algorithms start with a set of solutions and try to refine them in the hope of obtaining optimal solution(s) in the whole search space and, hence, are global area based algorithms.

Population-based algorithms that are commonly used to tackle timetabling problems include evolutionary algorithms (EAs) [73], ACO algorithms [196], and artificial immune systems (AISs) [151], etc.

In recent years, GAs have been used to solve the UCTP. GAs were first used for timetabling in 1990 [73]. Since then, there have been a number of researchers who have investigated and applied GAs for the UCTP [69, 205]. Generally speaking, when a simple GA is employed for timetabling problems, it may generate invalid timetables that have duplicate and/or missing events. Researchers have enhanced the performance of traditional GAs by using modified genetic operators, heuristics operators, and local search (LS) strategies. For example, Erben and Keppler [97] proposed a GA for constructing weekly course timetables. They used a problem-specific chromosome representation and knowledge-augmented genetic operators. These operators can avoid building invalid timetables. Their approach was tested on real data. Sigl *et al.* [194] used 3D cubes, corresponding to room, day, and time slot, to model the timetable. They enhanced the performance of GAs by using modified genetic operators and tested their algorithm on small and large problem instances.

In general, the quality of a solution produced by population-based algorithms, including GAs, may not be better than that produced by local area based algorithms. There are many reasons behind this. One major reason is due to the premature convergence problem. In that situation, the solving procedure of population-based algorithms is trapped in the sub-optimal state and is unable to generate offspring that are superior to their parents. The main reason for this premature convergence problem in population-based algorithms is due to the high selection pressure used within them. Another reason is that population-based algorithms are usually more concerned with exploration than exploitation [22]. Population-based algorithms perform search in the whole search space without strictly focusing on the good part of an individual within a population, which may lead to the loss of useful information in a good individual [11]. Population-based algorithms have another drawback of requiring more time [71]. However, GAs have several advantages when compared with other optimisation techniques [176]. For example, GAs perform a multi-directional search using a set of candidate solutions [115].

In order to address the problems faced by both local area and global area based algorithms for timetabling problems, various combinations of local area and global area based algorithms have been reported in the literature with some promising results [18, 44, 128, 181, 200]. Basically, GAs do not require any domain knowledge about the problems for which they are deployed to solve. Recently, however, significant interest has been shown in the use of genetic algorithms with domain-specific information because of their flexibility and robustness [37].

So far, most research has treated the UCTP as a single-objective optimization problem. However, inherently, the UCTP has different objectives or constraints. It is very difficult to satisfy all the constraints that vary from university to university. Hence, this complexity requires that the UCTP should also be treated as a multi-objective optimization problem (MOOP), i.e., the multi-objective university course timetabling problem (MOUCTP) needs to be studied.

In this thesis, we aim to combine the good properties of local and global area based algorithms to solve the UCTP of both single-objective and multi-objective versions. We try to make a balance between the exploration ability (global improvement) of GAs and the exploitation ability (local improvement) of LS strategies. In addition, we will introduce a guided search (GS) strategy to enhance the performance of GAs for the UCTP.

1.2 Aims and Objectives

This thesis investigates GAs to solve the UCTP. The overall aim of the thesis is to develop and apply GAs to produce solutions of good quality for the single-objective UCTP as well as the MOUCTP. In order to accomplish this primary aim, several objectives are outlined as follows:

- To learn the concepts of the timetabling problem in general and the UCTP in particular, to understand the challenges behind these problems, and to review relevant approaches that researchers have developed to solve these problems.
- To carry out an experimental analysis of the effectiveness and efficiency of traditional GAs and memetic algorithms (i.e., GAs enhanced with LS strategies) for solving the UCTP.
- To investigate LS strategies to enhance the exploitation ability of GAs for the UCTP and develop memetic algorithms for the UCTP.

- To develop a GS strategy to increase the quality of children in GAs and develop GAs with the GS strategy for the UCTP.
- To develop hybrid GAs that integrate the above GS and LS strategies and other heuristics to solve the UCTP.
- To model the UCTP as a MOOP and design multi-objective EAs (MOEAs) to solve the MOUCTP.
- To implement the above developed GAs and relevant algorithms for the UCTP and MOUCTP, respectively, using the GNU C++ programming language, and to carry out a systematic experimental study based on the implemented algorithms.
- To present a solution methodology that is generic, robust and able to produce good solutions, when compared against the state of the art.

1.3 Methodology

Timetabling in general and university course timetabling in particular are challenging problems due to their highly-constrained nature. In this thesis, we aim to develop efficient approaches, especially efficient GAs, to solve the UCTP. To improve the performance of traditional GAs for solving the UCTP, several different approaches are proposed in this thesis to avoid some disadvantages of GAs and enhance their searching power, e.g., applying LS strategies to enhance GA's exploitation capacity, developing a GS strategy to speed up GA's ability to locate good solutions and balance its exploration and exploitation capacities, and introducing multi-objective approaches to address the UCTP as an MOOP to reflect the nature of multiple objectives in real-world UCTPs, respectively.

In order to develop efficient GAs to tackle a complicated problem such as the UCTP, it is quite natural for us to follow a spiral procedure, starting from the basic case and eventually making progress towards complicated cases. So, we will start the research from studying the performance of traditional GAs for the simple benchmark UCTP. Then, we will develop and add strategies, e.g., LS and GS strategies, to enhance the performance of traditional GAs for the standard UCTP and a more challenging specialised UCTP, the post enrolment course timetabling problem (PECTP). With LS strategy, we try to enhance GA's exploitation capacity and with a GS strategy we try to speed up GA's ability to locate good solutions and balance its exploration and exploitation capacities. Finally, we will address the UCTP as an MOOP and utilise the above developed LS and GS strategies to enhance the performance of MOEAs that are used for general MOOPs to solve the MOUCTP, which is a more real-world oriented UCTP.

It is very hard to give a formal analysis of the performance of GAs (e.g., the convergence property and the ability to achieve optimality within a reasonable amount of time) for the UCTP due to the stochastic features in GAs and the complexity of the UCTP. However, even if we cannot give a formal analysis of GAs for the UCTP, we can experimentally analyse some search behaviour of GAs for solving the UCTP.

An important method of testing the performance of an algorithm for the UCTP is to carry out experimental studies based on benchmark problem instances. Normally, to test an algorithm's performance, we are required to choose benchmarks with different properties. In this thesis, all proposed GAs are tested on a certain number of state-of-the-art UCTP benchmarks. In addition, comparing the performance of an algorithm with other state-of-the-art algorithms under the same conditions is also a useful method to test the algorithm. This is because, usually, the state-of-the-art algorithms have been examined by many researchers and they normally have some distinguished performance on some benchmark problems. Therefore, the performance of an algorithm can be shown by comparison with these state-of-the-art algorithms. All algorithms proposed in the thesis are experimentally compared with other state-of-the-art algorithms for solving the UCTP to further evaluate their performance.

1.4 Why Are GAs Used in This Study?

Before we go into details, the question may arise here as to why we chose GAs to solve the UCTP. In the literature, stochastic search methods, such as GAs, simulated annealing (SA), and tabu search (TS), have been deemed particularly suitable for solving hard and complex combinatorial optimization problems [164]. The following are some advantages of GAs over traditional optimization methods that make them more suitable for tackling the UCTP, which is a complex combinatorial optimization problem.

- GAs are parallel, direct, stochastic methods for global search and optimization. Due to the nature of the UCTP, it is often impossible to search the whole solution space with traditional optimization methods. Traditional optimization methods are usually sequential and explore the solution space to a problem in one direction at a time. For these methods, if a sub-optimal solution is found, then usually all previous work is abandoned and the process starts again. GAs maintain and evolve a population of solutions. They are intrinsically parallel and can explore the solution space to a problem in multiple directions at the same time. If they find a path with a dead end, they simply remove or eliminate it and continue to work on other paths. This approach gives them more opportunities to find optimal solutions [84]

- GAs are very useful for complex or loosely defined problems. The inductive nature of GAs makes them more suitable for optimization problems like the UCTP because this problem usually has no predefined definition and it varies from department to department. The inductive nature of GAs means that they do not need to know any rules of the problem. GAs usually work by their own internal rules via genetic operators [130].

- GAs work very well on mixed (continuous and discrete) combinatorial problems. They are less susceptible to getting stuck at local optima than gradient and other search methods [130].

- One more advantage of using GAs is that bad proposals do not significantly affect the end solution negatively, as in every generation the least fit members of the population are less likely to get selected for reproduction and are simply discarded or die out, so the chance of getting a global optimum is increased in each generation [108].

- Through GAs, we can scan a vast search space, as they are able to scan a large solution set of problems. In a single run, we are able to get multiple solutions to a problem because GAs work with a set (population) of potential solutions instead of trying to improve a single solution [188].

- GAs are typically characterized as applying probabilistic transition rules, instead of deterministic rules [211].

1.5 Scientific Contributions

The majority of the scientific investigation in this thesis will be conducted using well-known benchmark UCTP instances, which will be described in Chapter 3. This thesis examines the performance of different algorithms, including those GAs developed in this thesis, based on these UCTP instances.

From our studies, the following scientific contributions are made:

- An experimental analysis concerning the suitability of GAs for the UCTP is conducted. From this analysis, we conclude that traditional GAs with simple crossover and mutation operators and without any LS strategies behave in quite different ways with different problem sizes. For example, for the small problem instances, GAs are able to produce good solutions, but for medium and large sizes, the performance is not quite so good. Hence, we introduce a new LS strategy for the GA for solving the UCTP. This LS strategy finds the high penalty time slot in a timetable and tries to rearrange the events of the time slot. This powerful LS strategy prevents the GA from getting stuck in local optima.

- A new GS strategy is proposed for GAs to address the UCTP. This GS strategy uses an extra memory (data structure) to store the best parts of solutions found during the searching process and reuses them to guide the generation of new offspring for the subsequent generations. The developed GSGA is investigated on different benchmark UCTP instances and is compared with other approaches from the literature. The results show that GSGAs are able to produce good results on different benchmark UCTP instances.

- A hybrid GA that integrates the above LS and GS strategies into a standard GA is proposed to solve the UCTP. This hybrid GA is able to generate some of the best quality solutions for the benchmark UCTP instances used in the experimental study.

- A two-phase approach is developed to solve the PECTP. The first phase consists of a GA with the GS and LS strategies and a new heuristic crossover operator. The second phase employs a TS method that tries to improve the solution obtained from the first phase. The research shows that the proposed algorithm is good enough to tackle the more real-world hard problem instances and that generated solutions are good across all standard benchmarks of the 2007 International Timetabling Competition (ITC-2007)[1] PECTP instances.

- The UCTP is also investigated as an MOOP and a framework of integrating the above LS and GS strategies into general MOEAs is proposed to tackle the MOUCTP.

- The above proposed framework of MOEAs for the MOUCTP is instantiated into several MOEAs based on state-of-the-art MOEAs for general MOOPs to solve the MOUCTP. The obtained MOEAs are tested on different MOUCTP instances and the results show that these MOEAs are good choices for solving the MOUCTP.

[1]For more details, see the ITC-2007 website at http://www.cs.qub.ac.uk/itc2007.

1.6 Thesis Outline

This thesis consists of eight chapters. This chapter presents the background, motivation, aims and objectives, and methodology of the research. It also highlights some scientific contributions that are made during this research. The remainder of this thesis is structured as follows.

Chapter 2 provides an introduction to the timetabling problem in general and the UCTP in particular. The reasons why the UCTP is often hard to solve will be discussed. A review and analysis of various approaches that have been proposed in the literature for the UCTP will then be provided.

In Chapter 3, several special versions of the UCTP, which have recently been used as benchmark instances for a number of relevant works in the literature, are presented. These benchmark instances include the general UCTP instances, the PECTP instances, and the MOUCTP instances, respectively. They will also be used in the experiments carried out in the subsequent chapters in the thesis.

In Chapter 4, an experimental analysis of a simple steady-state GA for the UCTP is conducted. A memetic algorithm that integrates two LS strategies into the GA to enhance the quality of solutions is presented. The possible limitations of the proposed memetic algorithm for the UCTP are also investigated.

In Chapter 5, a guided search strategy is introduced to enhance the searching power of GAs for the UCTP. Based on this GS strategy and the LS strategies described in Chapter 4, several GA variants for solving the UCTP are presented. Based on a set of benchmark UCTP instances, a number of experiments are carried out to investigate and analyse the effect of the GS and LS strategies. The proposed GAs are also experimentally compared with a number of state-of-the-art approaches from the literature for the UCTP.

Chapter 6 is intended to move this work from the benchmark UCTP instances considered in previous chapters towards more real-world oriented problems. The ITC-2007 PECTP instances are used to test our approaches. A two-phase hybrid approach is proposed for the PECTP. The first phase employs a GA with guided and local search strategies and tries to find feasible or optimal solutions for the PECTP. The second phase employs a tabu heuristic to further improve the quality of solutions found by the first phase. In this chapter, arguments are also proposed as to why the two-stage approach may be an effective way of tackling the PECTP.

Moving away from single-objective UCTPs, in Chapter 7, attention is focused on solving the UCTP as an MOOP. In this chapter, a framework of integrating the developed GS and LS strategies with multi-objective EAs (MOEAs) to solve the MOUCTP is presented. This framework is then instantiated to construct several MOEAs for the MOUCTP based on a set of MOEAs that are typically used for general MOOPs. The instantiated MOEAs are then experimentally validated using several benchmark MOUCTP instances and performance metrics that are taken from the literature on MOEAs.

Finally, Chapter 8 concludes this thesis by providing a summary of the major technical contributions made from the research carried out in this thesis to the domain of UCTP research and the major conclusions that can be drawn from the experimental studies carried out in the thesis. Some ideas about possible directions for further research are also discussed in this chapter.

Chapter 2

University Course Timetabling Problems and Solution Approaches

2.1 Introduction

In this chapter, we give a general overview of the timetabling problem, especially the university course timetabling problem (UCTP) and relevant constraints that need to be considered. We will also review the methods that are used for solving the UCTP in the literature.

This chapter has four sections. Section 2.2 describes definitions of timetabling and the educational timetabling problem along with its different types. Section 2.3

describes the UCTP in detail and discusses some fundamental research aspects to solve this problem. In Section 2.4, we review key approaches that have been utilised to handle the UCTP. A summary of the chapter is given in Section 2.5.

2.2 Timetabling

Timetabling problems are a specific type of scheduling problem [214]. Sometimes, the terms "timetabling" and "scheduling" are loosely used as if they were synonymous. However, it has been shown in the literature [48, 208, 214] that there are certain distinctions between them. According to Wren [214], "scheduling" is:

> "the allocation, subject to constraints, of resources to objects being placed in space-time, in such a way as to minimise the total cost of some set of the resources used."

From the above definition, we can say that scheduling will normally include all the information necessary for a process to be carried out. Scheduling in the broadest sense can be described as the process of solving practical problems related to the allocation of resources (subject to constraints) to objects being placed in space-time. For example, a production scheduling problem concerns the allocation of resources (materials, labour, and equipments, etc) to tasks over time on the basis of some constraints [174]. In employee scheduling, we need to place resources into slots in a pattern, under given constraints, where the pattern denotes a set of legal shifts defined in terms of work to be done [214].

On the other hand, "timetabling", according to Wren [214], is defined as:

> "the allocation, subject to constraints, of given resources to objects being placed in space and time, in such a way as to satisfy as nearly as possible a set of desirable objectives."

Based on the above definition, for timetabling problems, we need to consider, whether there are adequate resources available for the given events (people, activities, and vehicles, etc) to take place at their predefined time as well as which resources are allocated. In the real world, timetabling problems may arise in connection with such issues as educational requirements, transport or sport.

In the following sub-section, we describe education timetabling problems, particularly the course timetabling problem.

2.2.1 Educational Timetabling Problems

In general, an educational timetabling problem can be defined as the task of assigning a number of events, such as lectures, exams, meetings, and so on, to a limited number of resources, while satisfying different constraints. Schearf [190] classified educational timetabling into the following three main categories:

- School timetabling
- Examination timetabling
- University course timetabling

They share the same basic characteristics of the general timetabling problem but can still have significant differences between them. Each one of them has its own constraints, requirements, and rules. More details on educational timetabling can be found in different survey papers, see [48, 142, 190]. In the following sub-sections, different kinds of educational timetabling problems and their properties are briefly discussed.

2.2.1.1 School Timetabling

According to Schearf [190], the school timetabling problem is defined as:

> "the weekly scheduling for all the classes of a school, avoiding teachers meeting two classes at the same time, and vice versa."

The school timetabling problem is concerned with the weekly scheduling for all the lessons of a school. The problem consists of a set of teachers, classes, subjects/-lessons, and weekly periods. These weekly periods are pre-defined. This problem tries to assign lessons to periods and, each teacher to a particular class at a given time while satisfying a set of constraints in order to produce a feasible timetable.

2.2.1.2 Examination Timetabling

According to Carter and Laporte [60], the examination timetabling problem is defined as:

> "the assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes."

The examination timetabling problem refers to the assignment of timeslots and rooms so that students can take examinations without clashes.

2.2.1.3 University Course Timetabling

According to Carter and Laporte [69], the UCTP is defined as:

> "a multi-dimensional assignment problem, in which students and teachers (or faculty members) are assigned to courses, course sections or classes, and events (individual meetings between students and teachers) are assigned to classrooms and timeslots".

The UCTP is the problem of scheduling a set of events to specific timeslots such that no person or resource is expected to be in more than one location at the same time and there is enough space available in each location for the number of people expected to be there. It should also be kept in mind that all classes of one subject should be dispersed throughout the week to provide a conducive learning environment.

This thesis focuses on the UCTP. The details of the UCTP are discussed below.

2.3 University Course Timetabling Problems

2.3.1 Constraints

UCTPs, as well as other scheduling problems, define a class of hard-to-solve combinatorial optimization problems, which are usually highly constrained [184]. There are two types of constraints, soft constraints and hard constraints, defined in a UCTP. These are described below:

2.3.1.1 Hard Constraints

Hard constraints are those constraints that have to be satisfied for a *feasible solution* and should never be violated by any cost. The following are some typical examples of hard constraints in a UCTP:

- Lectures having students in common cannot take place at the same time.
- Lectures must take place in a room suitable for them in term of facilities and student capacity

- No two lectures can take place at the same time in the same room.
- No teacher may be assigned to different events at the same time.
- There must be a maximum number of time periods per day, which may not be exceeded.

2.3.1.2 Soft Constraints

Soft constraints are less important than hard constraints, and it is not necessary to satisfy them all. If both hard and soft constraints are satisfied in a solution, then we would say that the solution is an *optimal solution*. Following are some typical examples of soft constraints in a UCTP:

- Every teacher has their own availability schedule or submits a plan with desirable time periods that suits them best.
- Every teacher has a minimum and maximum limit of weekly work hours.
- Lectures on the same subject should be dispersed across the week to provide a conducive learning environment.
- A student or teacher should not attend more than two lectures in a row.
- Students should not have one lecture in any given day.
- Some timeslots are reserved for specific activities (non-academic or outside class activities such as sports); therefore, they are not available for lectures.

- The travel time of teachers and students between rooms within the campus should be minimised.

2.3.2 Quality of Solutions

From the explanation of constraints, a solution to the problem can be described as feasible, infeasible, and optimal solution. A *feasible* solution means that the solution satisfies all hard constraints of a problem but not necessarily all soft constraints. An *infeasible* solution is one that fails to satisfy all hard constraints. An *optimal* solution fulfils all requirements of the constraints and there is no violation of soft and hard constraints. Throughout this thesis, we will use these terms to represent the quality of a solution.

2.3.3 Types of UCTPs

According to the ITC-2007 organisers, UCTPs can be divided into two tracks: post enrolment-based course timetabling and curriculum-based course timetabling. These two tracks are distinct and are used in institutes with varying constraints.

2.3.3.1 Post Enrolment Based Course Timetabling

This track is also called “class timetabling”, “event timetabling”, or “general university course timetabling”. It is the only track of course timetabling problems in the 2003 International Timetabling Competition. In this type of UCTPs, after student enrolment, the timetable is constructed in such a way that all students can attend the events on which they are enrolled. According to the organisers of the competition, this UCTP has extra hard constraints that move this type further in the direction of real-world timetabling.

2.3.3.2 Curriculum Based Course Timetabling

The curriculum-based courses timetabling problem (CBCTP) deals with the weekly scheduling of lectures for several university courses within a given number of rooms and time periods, where conflicts between courses are set according to the curricula published by a university and not on the basis of the enrolment data.

In this thesis, basically we only deal with the PECTP, with either three or five hard constraints, which are called UCTPs and PECTPs, respectively. Soft constraints on both problem definitions are the same. We will not deal with the CBCTP in this thesis.

2.3.4 Modelling UCTPs

In this sub-section, we describe the models of UCTPs. De Werra [207, 208] has carried out research to model a timetabling problem in different ways. The most common and well-known models of timetabling problems are the "graph colouring" model and the mathematical model [202, 210, 214]. In the graph colouring model, a

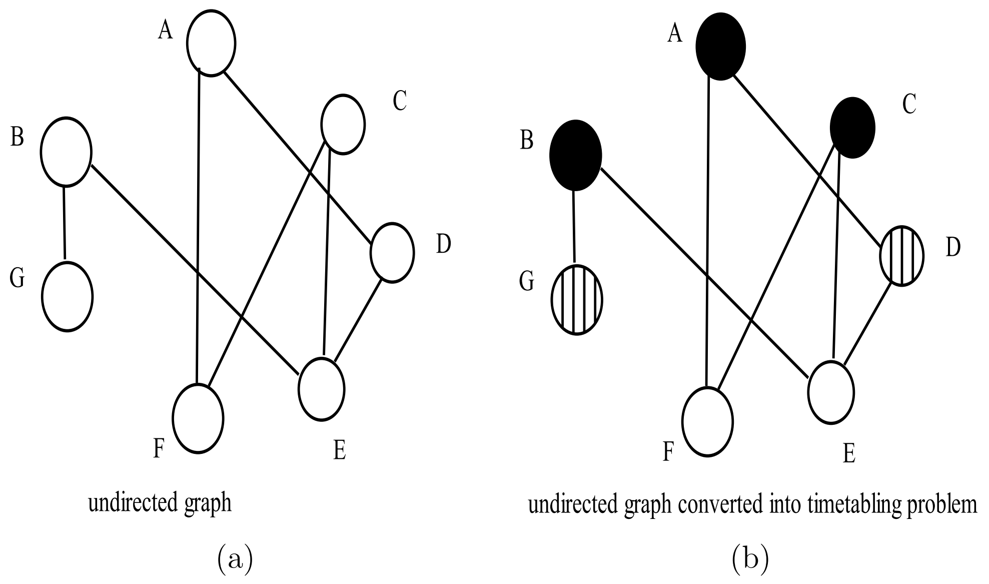

FIGURE 2.1: The graph colouring model: (a) an undirected graph, and (b) a simple timetabling problem represented by the graph colouring model.

simple undirected graph G is used, which has a set of n vertices $V = \{v_1, v_2, \cdots, e_n\}$ and a set of t edges $E = \{e_1, e_2, \cdots, e_t\}$. Figure 2.1(a) shows the representation of an undirected graph on the vertex set $\{A, B, C, D, E, F, G\}$ and the edge set $\{(A, D), (A, F), (D, E), (F, C), (E, C), (B, G), (B, E)\}$. In a graph colouring problem, vertices need to be coloured in such a way that: (a) no pair of vertices, that are connected with the same edge, are assigned the same colour, and (b) the number of colours being used is minimised.

For a better understanding of the relationship between the graph colouring problem and the timetabling problem, we take a simple timetabling problem as an example. Figure 2.1(b) shows a simple course timetabling problem represented using the graph colouring model. In Figure 2.1(b), there are seven different courses, denoded as A, B, C, D, E, F, and G, respectively. One possible goal is to find the minimum number of timeslots that are needed to schedule the seven courses, where some events have clashes with each other on time and cannot be placed on the same timeslot. A set of edges $\{(A, D), (A, F), (D, E), (F, C), (E, C), (B, G), (B, E)\}$ represents the clashes between courses. If there is an edge between two vertices, it means that these two courses cannot be scheduled in the same timeslot.

Figure 2.1(b) shows a possible assignment of three timeslots to the seven events, where the events are represented with three colours/patterns: white, black, and filled. These three patterns represent different timeslots. As we can see, event A cannot be placed at the same timeslot as event D and F, so these three events can be coloured as black, white, and filled, respectively. Now, event E cannot be scheduled on the same timeslot as C and D, so we can give E a different colour from C and D. However, C can be scheduled on the same timeslot as D, so C can use the same colour as D, and so on. From Figure 2.1(b), we can see that the placement of timeslots/colours is according to the constraints of events. The minimum number of timeslots that can be used is three for this example. This model can be further extended for more constraints. Many researchers have proposed graph colouring heuristics that can produce clash-free timetables for small problem instances [207, 209, 210].

Many others have also presented mathematical models for a general timetabling problem [202, 207]. Tripathy [202] formulated the problem using an integer linear programming formulation. Mulvey [163] proposed a classroom/time assignment model for this problem. The simplest one given here is taken from De Werra [207].

Suppose there are a set of p courses $E = \{E_1, E_2, \cdots, E_p\}$, where each course E_e consists of k_e lectures, a set of r course groups $G = \{G_1, G_2, \cdots, G_r\}$, where each course group G_i $(i = 1, 2, \cdots, r)$ consists of those courses that have common students and hence must be scheduled in different timeslots, a set of q periods/timeslots $T = \{t_1, t_2, \cdots, t_q\}$, and l_k is the maximum number of lectures (i.e., the number of rooms available at timeslot k) that can be scheduled at period k. The formulation of a timetabling problem can be given as follows:

$$find \quad x_{ek} \quad (e = 1, \cdots, p; k = 1, \cdots, q) \tag{2.1}$$

$$s.t. \quad \sum_{k=1}^{q} x_{ek} = k_e \quad (e = 1, \cdots, p) \tag{2.2}$$

$$\sum_{e=1}^{p} x_{ek} \leq l_k \qquad (k = 1, \cdots, q) \tag{2.3}$$

$$\sum_{e \in G_l} x_{ek} \leq 1 \qquad (l = 1, \cdots, r; k = 1, \cdots, q) \tag{2.4}$$

$$x_{ek} = 0 \text{ or } 1 \qquad (e = 1, \cdots, p; k = 1, \cdots, q), \tag{2.5}$$

where $x_{ek} = 1$ if a lecture of course E_e is scheduled at period k, and $x_{ek} = 0$, otherwise. Eqs. (2.2), (2.3), and (2.4) represent different constraints of the problem. Eq. (2.2) enforces that each course is composed of a correct number of lectures. Constraint 2.3 shows that for each timeslot there are no more events than rooms. Eq. (2.4) ensures that courses that have the same students must be placed in different periods. The work of other researchers can be found in [40, 193, 203].

2.3.5 Why are UCTPs Difficult to Solve?

There are two major reasons for the question why UCTPs are difficult to solve. Firstly, as there is no unified definition of the problem, any solver will only address some of the many possible variants. As we discussed earlier, the problem itself does not have a widely approved definition; different universities have their own idiosyncratic sets of constraints. These variations in constraints from university to university make this problem different to most others. The problem definition of one university may directly oppose to that of another university. From the research point of view, this distinguishing nature of the problem makes it difficult to give a meaningful and general formulation for the problem [143].

Secondly, almost all variants of UCTPs are NP-complete and therefore difficult to solve computationally. We can say that problems that can be solved by a deterministic computer in polynomial time (depending on the input size of the problem) are called "P-problems". Similarly, those problems that can be solved by a non-deterministic computer in polynomial time according to the input size are called "NP-problems". More importantly, any problem that can be solved in deterministic polynomial time can also be solved in non-deterministic polynomial time. So, P is a subset of NP. However, there are many problems in NP for which no deterministic polynomial time algorithm is known. The hardest problems in NP, in the sense that any other problem in NP can be reduced to such a problem in polynomial time, are called *NP-complete*. It is widely believed that NP-complete problems do not admit polynomial time deterministic algorithms.

The UCTP has been proved to be NP-complete [96, 111, 112, 190] in most of its variants. Like all NP-complete problems, the solution space for UCTP is too large to be explored through exhaustive search. Let's take an example of a university that has 1500 events to be placed in 45 timeslots and 100 rooms. The possible number of timetables is $(45 * 100)^{1500}$ and most of them are infeasible. As the timetabling problem is NP-complete, we do not know any efficient methods for finding the best solution in this huge search space in polynomial time. In contrast, if a university has a small number of events that need to be placed in an abundant number of rooms, then it is possible that good or optimal solution(s) can be found easily in the search space. Thus, for difficult or harder (in the sense of complexity or constraints) problems, it is difficult to find the solution in a reasonable time. Hence, a powerful or robust approach is required to tackle these problems.

2.4 Approaches for Solving UCTPs

UCTPs are generally complex, large scale, constrained, and multi-objective in nature. Researchers have developed different approaches to address UCTPs. Several survey papers have been published that classify or discuss major solution methodologies for them [55, 69, 178, 190, 207]. Carter and Laport [69] discussed the major approaches and divided them into four catagories. Schaerf [190] also conducted a survey on this problem along with mathematical description of variants of the problem and solution methodologies. Subsequently, Petrovic and Burke [173] presented two approaches to solve UCTPs. Based on the survey papers discussed above, we divide the solving methodologies into the following groups.

2.4.1 Constraint-Based Methods

Generally, a constraint-based method models a problem by a set of n variables x_i ($i = 1, \cdots, n$) with a corresponding set of domains D_i that declare the allowable values for each variable x_i, and a set of e constraints c_j ($j = 1, \cdots, e$) over the variables which restrict the allowable combinations of variable values [33]. More precisely constraint-based methods deal with the assignment of constraints from its domain to each variable such that all constraints are satisfied [38].

For the solution of timetabling problems, a problem is modelled as a set of variables (i.e., events) to which values (i.e., resources such as rooms and time periods) have to be assigned to satisfy a number of constraints. Some rules are also defined for assigning values to events. When no rule is applicable to the current partial solution, a backtracking operation is performed until a solution is found that satisfies all constraints [161]. Constraint-based solution methods, such as constraint propagation, domain reduction, and backtracking, are well suited for many industrial applications [103]. These methods are combined with classic techniques, such as linear, integer, and mixed integer programming, to yield powerful tools for solving constraint-based problems [103].

Considerable research has been carried place to solve timetabling problems as a constraint satisfaction problem. Abdennadher and Marte [10] applied constraint-handling rules (i.e. a powerful special-purpose declarative programming language for writing application-oriented constraint solvers) and created a solver to generate a timetable for the Department of Computer Science at Munich University. First, they modelled the problem as a constraint satisfaction problem. They, then used constraint-handling rules to implement a finite domain solver, which performs hard and soft constraints propagation. One of the main characteristics of their work is that, they clearly distinguished between hard and soft constraints, which could not be possible in traditional constraint satisfaction approaches. This limitation usually forces us to treat soft constraints as hard constraints, which sometimes leads an over-constrained constraint satisfaction process, ending up without solutions.

Many researchers have applied constraints logic programming to solve this problem. Zervoudakis and Stamatopoulus [217] applied a constraint programming generic object-oriented model using the ILOG SOLVER C++ library for the UCTP. They used real data instances from the Department of Informatics and Telecommunications at the University of Athens to test their approach. A variety of search methods (for example, the depth first search) and variable ordering heuristics were used in their technique. The quality of solution depends on the search methods. Of all of the search methods, although the depth first search was capable of finding feasible solutions in the early stage of search, no further improvement were seen in the solutions even when the algorithm was run for a long time.

Deris *et al.* [87] modelled the timetabling problem as a constraint satisfaction problem and then used a constraint-based reasoning technique (any reasoning technique that utilizes an arc-consistency algorithm for the purpose of constraint propagation in problem solving) to solve the problem. In order to facilitate the search for solutions, timetabling problems were represented as a graph tree organization. Due to the large search space, they used a backtracking search method incorporated with constraint propagation. They also employed variable orderings (variable ordering helps to find a solution faster by reducing the search space explored during the search process) based on the size of the domain and the number of constraints of the variables to speed up the search process. Rather than choosing benchmark instances for testing, they tested their approach on real data instances and were able to produce some feasible results within a reasonable time.

Zhang and Lau [218] developed a constraint satisfaction problem model for the UCTP. They used a simple case study of the problem. Implementing a constraint satisfaction approach using the ILOG scheduler and ILOG solver and using different goals in ILOG to investigate the performance of their approach, they achieved some good results regarding the time factor.

2.4.2 Sequential Heuristics

The early work on timetabling problems is based on sequential heuristics. According to Reeves [179], a heuristic is:

> "a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is."

Sequential heuristic methods are normally divided into two phases: a construction phase and an improvement phase [59]. The construction phase orders events using domain heuristics and then assigns the events sequentially into the first feasible slot or valid timeslot so that no events are in conflict with each other. The second phase is the improvement phase, which makes modifications and improvements. Usually, backtracking is used to undo some of the previous decisions and the assignment process can be re-applied [59]. This type of method may not, however, always converge toward a feasible solution and care must be taken to avoid cycling [69]. Some work on these methods can be found in [75, 104, 171].

Most of these heuristics stem from heuristics for solving the graph colouring problem due to the close link between the timetabling problem and the graph colouring problem. Graph colouring heuristics are often called sequential heuristics because their main idea is to schedule events to timeslots sequentially or one by one [60].

Timetabling has been widely investigated as a graph colouring problem [207, 208]. De Werra [207, 209] first transferred a course timetabling problem into graph colouring. Subsequently, Selim [192] also employed a graph colouring approach for the faculty timetable problem. Real data from the Faculty of Science of the American University in Cairo was used to test the approach. The events/courses were split in order to reduce the timeslots number.

Burke *et al.* [51] applied a graph-based hyper-heuristics with the tabu search approach to solve optimisation problems. They also tested their approach on the UCTP. The key feature of this approach is that they used tabu search (TS) approach to change the permutations of six graph colouring heuristics before constructing a timetable. The approach works more efficiently when a larger number of low-level heuristics are used. Although this approach gave good solutions on all problem instances, it was unable to give new best solutions, perhaps due to the fact that this approach was designed to deal with general optimisation problems, not specially for the UCTP.

2.4.3 Meta-heuristic Approaches

Meta-heuristics are a class of heuristic techniques. The term “meta-heuristic” was coined by Fred Glover in 1986 [116]. Meta-heuristics have become a leading edge among heuristic approaches for solving a wide range of complex combinatorial optimisation problems. According to Osman and Kelly [167], a meta-heuristic is:

> "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions."

Glover and Kochenberger [118] made the following comments:

> "meta-heuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space."

Meta-heuristics approaches are usually based on local-area based searches and population based searches. A local-area based search explores the solution space by a gradual improvement of the current solution. Some examples of local-area based searching techniques are tabu search and simulating annealing, etc. These techniques iteratively calculate a candidate solution or the neighbourhood from a current solution and replace the current solution by a solution in its neighbourhood. The construction of a neighbouring solution is called move. These techniques differ from each other on the basis of different parameters such as the process to generate moves or the control scheme for selecting a new neighbourhood solution, etc. Population based techniques perform the search by maintaining a population of candidate solutions. These methods explore the neighbourhood of a whole population rather than exploring that of a single solution [175].

In this section, we will review widely used meta-heuristic approaches to course timetabling.

2.4.3.1 Genetic Algorithms (GAs)

Genetic algorithms (GAs) are stochastic search methods which were popularised by Holland [125], who derived GAs from Darwin's theory of evolution, which is based on the theory of survival of the fittest due to natural selection. Before Holland [125], different researchers had developed evolution-inspired algorithms for function optimization based on the principles of natural selection and genetics [39, 100], but their work attracted little follow-up. All the essential elements like recombination and mutation were used by Fraser's simulations [100] . Similarly, Bremermann [39] also adopted a population of solutions to optimization problems, undergoing recombination, mutation, and selection. Holland was the first researcher who explicitly proposed crossover and other recombination operators. Later, GAs were widely used in a broad variety of fields to solve optimization and search problems, such as in astronomy [70], electrical engineering [25], routing and scheduling [54, 122], and many more.

Algorithm 1 shows the framework of a conventional GA. GAs are population based heuristic methods, which start from an initial population of usually random solutions for a given problem. Each solution in the population is called an individual. Each individual is evaluated according to a problem-specific objective function, usually called the fitness function. After evaluation, there is a selection phase in which

Algorithm 1 The Conventional Genetic Algorithm

```
1: Initialise a population of solutions
2: Evaluate the individuals in the population
3: while the termination condition is not reached do
4:     Select parents through a selection scheme
5:     Crossover the parents to create offspring
6:     Apply mutation to offspring
7:     Replace the worst member(s) of the population for the next generation
8: end while
```

possibly good individuals will be chosen by a selection operator to undergo the recombination process. In the recombination phase, crossover and mutation operators are used to create new individuals in order to explore the solution space. The newly created individuals replace old individuals, usually the worst ones, of the population based on their fitness. This process is repeated until a stopping criterion is reached, which may be the maximum number of generations or a time limit.

In 1990, GAs were used for the first time by Colorni *et al.* [73] for the timetabling problem. Since then, there have been a number of researchers who have investigated and applied GAs for the UCTP [69, 205]. Colorni et al. [73, 74] proposed a timetable for an Italian high school with the help of EAs. They encoded a timetable as a matrix where columns represent timeslots and each row represents a teacher involved with the timetable. They used a weighted sum function to distinguish between hard and soft constraints. A novel crossover operator was used to create individuals. After the crossover operation, they used a genetic repair procedure to fix hard constraints if their violation was raised to an above predefined threshold level. The main advantage of this approach is that it is able to produce a large number of different good quality timetables in a single run.

Erben and Keppler [97] applied a GA to deal with a weekly-course timetabling problem to schedule classes, teachers, course modules and rooms to a number of timeslots in a week. A large data sample was used to test the algorithm. The experiments carried out show that the algorithm was able to obtain promising results. Lewis and Paechter [145] produced a grouping GA for the UCTP. They only considered hard constraints to be solved and grouped the events according to timeslots. A feasible timetable is one where all the events have been assigned into feasible timeslots, where feasible timeslots are those that have no conflict events, and all events are in their suitable rooms. They used recombination and grouping mutation operators. Sixty problem instances were tested and their approach was able to produce feasible solutions for 23 instances out of these 60 instances.

Lewis and Paechter [144] proposed a number of different crossover operators for GAs to solve a UCTP. They applied a genetic repair function to regain feasibility during the crossover and mutation process. They used different crossover operators, for example, sector-based, day-based, student-based, and conflict-based crossover operators. Their algorithm was tested on twenty problem instances from the first international timetabling competition [1] and the conflict-based crossover operator gave good results. They concluded that effective crossover operators and efficient GAs are also able to produce a large number of diverse and feasible timetables in a reasonable amount of time.

Many researchers combined GAs with other approaches, called memetic algorithms (MAs), to solve the UCTP. The term of MA was coined by Moscato [159] from

Dawkin's [85] term "meme". The term "meme" is different from the term "gene" in the sense that memes are ideas or concepts that are passed around and can be altered to the environment but genes pass from generation to generation without alteration [85]. A MA [159, 160] is an EA that uses knowledge of the problem in the form of, for example, approximate algorithms, local search techniques, heuristics or specialised recombination operators [132]. It is generally believed that MAs are successful because they combine the exploitive search ability of LS methods and the explorative search ability of recombinative EAs [49, 73]. Many researchers have applied MAs to address timetabling problems by combining GAs and LS techniques. Some well-known examples are, Paechter *et al.* [170], Rossi-Doria *et al.* [181, 182], Alkan and Ozcan [23], Abdullah *et al.* [15], and Chiarandini *et al.* [71].

2.4.3.2 Ant Colony Optimisation (ACO)

Ant colony optimisation is a population-based searching technique. The idea is based on the mechanism of "food collection" used in an ant colony. In this process, once an ant finds a food source, it will lay a chemical substance (called pheromone) during the path. Pheromones help other ants to find the same food source because every ant relies on pheromones trails, that were added every time when an ant passed through the path. The more ants that have passed the same path, the higher the probability that this path will be chosen by future ants. All the pheromone trails laid by the whole ant colony thus optimise the time and effort spent on the food collection of the colony. Dorigo *et al.* [90] were the first to employ this idea in a search meta-heuristic. In an ACO algorithm, an ant constructs a solution for a given problem, and a population of ants is maintained. During the evolving process, useful information is kept as the pheromone, which will be updated and used in the creation of the next generation of individuals [88, 89]. Over the last decade, several ACO algorithms have been proposed in the literature. Stützle and Hoos [198] proposed the MAX-MIN Ant System, whose main characteristics are that only the best ant updates the pheromone trails and that the value of the pheromone is bounded. Several successful applications of ACO on different optimization problems can be found in [7, 8, 36, 91, 155].

Socha *et al.* [196] applied a MIN-MAX ant system [198] for the UCTP. They transformed the problem into an optimal path problem which can be tackled by generating a construction graph. The assignment of courses to timeslots was dependent on the pheromone value within the bounds. They concluded that the MIN-MAX ant system performs better than an iterative local search method.

Rossi-Doria *et al.* [181] presented a comparative study of five different meta-heuristic algorithms applied to a UCTP, of which the ant colony was one algorithm. In the studied ACO approach, an ant constructs a timetable using a sequential strategy. An ant chooses a course from a pre-defined list and assigns it to a timeslot in a probabilistic manner. They concluded that the performance of ACO is slightly worse than simulated annealing and tabu search. However, it is better than a GA for the UCTP.

2.4.3.3 Tabu Search (TS)

TS is a powerful tool for solving difficult optimisation problems. The idea of TS was first proposed by Glover [117]. Glover and Laguna [119] defined TS as follows:

> "Tabu search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality."

TS is a local-area based search method that guides a heuristic search to explore the solution space beyond local optimality by including "intelligent" features. The main "intelligent" feature of TS is the use of adaptive memory. According to Burke *et al.* [47],

> "tabu search is an intelligent search technique that uses a memory function in order to avoid being trapped at a local minimum."

The TS algorithm starts from an initial solution and then iteratively explores a subset of the neighbourhood of the current solution. The next move will be chosen after evaluating all neighbourhood solutions of the current solution. A worse intermediate solution can be accepted as long as its solution quality is the best among all neighbourhood solutions. This process may lead to cycling, i.e., moving repeatedly between small portions of the search space. To avoid cycling, TS uses memory to store a tabu list. If a new solution is found to be in a tabu list, then it is called a "tabu solution" and the next best solution in the neighbourhood of the current solution is considered, and so on. The tabu list allows the algorithm to escape from local optima and globally search the solution space [119]. Moves in the tabu list are prohibited for a predefined number of iterations, called "tabu tenure", and for some tabu restrictions. Tabu restrictions are used to avoid repetition within the search space. TS methods can be classified according to the intensification and/or diversification [119]. "Intensification" means intensive exploration of the search area, where good solutions were previously yielded. "Diversification" means search towards un-explored areas. TS has been successfully used to solve different combinatorial optimization problems. More details and relevant applications can be found in [34, 119, 186].

Extensive research has been done on solving timetabling problems by using the TS heuristics. Hertz [123, 124] first presented a TS approach to solving large-scale course timetabling problems. His algorithm starts from a random initial solution. A move is defined as moving a single random event to a new time slot. As the neighbourhood size using this criterion becomes too large for practical problems, the algorithm only calculates a fraction of the neighbourhood, that is proportional to the number of events. The size of the tabu list is restricted to 7, where a full state of previously visited solutions are not stored. The tabu list stores a move in terms of event and time slot pairs (e, t) rather than solutions, where e represents an event and t represents the previous time slot where the event e was scheduled. An event present in the tabu list is not allowed to be scheduled on the stated time slot until the pair is removed or expired off from the tabu list. The author concluded that this approach produces satisfactory results on both exam and course timetabling problems. The main advantage of this approach is that, by allowing the most promising tabu move, the searching process is speeded up, which is important when solving large-scale problems.

Costa [77] used TS for constructing different real course schedules. Two tabu lists were introduced. The first one is a list of lectures which are moved from one timeslot to another and the second one is a list of pairs (l, t), which represent information about a lecture and the previous timeslot with the aim that lecture l cannot be replaced at timeslot t while the pair remains on the list. The author also introduced a diversification strategy that drives the search to other areas that are not examined. Experimental results showed that the algorithm was able to obtain satisfactory solutions to the timetabling problem. However, there are several parameters that need to be tuned in the algorithm, such as those for the diversification and the lengths of the tabu lists.

Colorni *et al.* [74] presented an investigation of three different meta-heuristics, i.e., simulated annealing (SA), TS, and GA, on the high school timetabling problem instance. A variable size of tabu list where the length of the list is changed after a constant number of iterations is used. A relaxation procedure is also incorporated into the TS algorithm. Their experimental results show that TS was consistently the best performing algorithm when compared against the tested SA and GA.

Bellio *et al.* [185] analysed a dynamic TS algorithm for the UCTP, where the tabu list changes continuously according to the shape of the cost function in an adaptive way. This way, they allow for the possibility of passing infeasible states and visiting states that have different structures from the previously visited ones.

Aladağ and G. Hocaoğlu [21] applied a TS algorithm to solve the UCTP. They argued that, in the literature, the problem formulation does not contain the constraint that there should be no conflicts between lessons in the same section. They gave a new mathematical formulation of the problem and also proposed TS to solve the problem. They used a real data set from the Statistical Department of Hacettepe University to test their TS algorithm. Their experimental results show that TS produced clash free timetables.

2.4.3.4 Simulated Annealing (SA)

SA is based on the idea of the Metropolis algorithm [156] for statistical mechanics. The Metropolis algorithm simulates the change in system energy, subject to the cooling process, until it converges to a steady or frozen state. Annealing is the process of cooling material in a heat bath. First, the material or solid is heated to a high energy (where the state frequently changes) so that its molecules and atoms are set randomly. Then, it is gradually cooled down to a low energy (where the state rarely changes) so that its molecules reach the state of the minimum energy. The decreasing of the temperature is made slowly because if the cooling process is too fast, unstable structures might appear instead of those with the minimum energy. This method of decreasing the temperature is called "the cooling schedule". Kirkpatrick *et al.* [134] first introduced this concept to solve optimization problems. Later, Černý [61] also applied SA to the travelling salesman problem. They suggested that SA could search feasible solutions, where the objective is to converge to an optimum state [134].

An SA algorithm starts by calculating the neighbourhood moves at random. The temperature and the change in the evaluation function determine whether to accept the worse moves. A high temperature gives a high acceptance probability and a low temperature rejects moves with a high probability. The acceptance probability is defined as $exp(-\delta/t)$, where δ is the change in the solution quality and t is the current temperature. An SA algorithm proceeds, the temperature decrease according to the cooling schedule. The performance of SA is dependent on the cooling schedule and the choice of the neighbourhood structure [9].

Elmohamed *et al.* [95] applied an SA algorithm with different cooling schedules (geometric, adaptive and adaptive with reheating) for a UCTP. They tested their approach on real data at Syracuse University. Their experimental results show that the SA with the adaptive cooling and reheating algorithm outperformed other methods.

Frausto-Solis *et al.* [101] proposed extended SA algorithms for the UCTP. An initial solution was created by adding extra timeslots into the timetable in order to make a feasible solution. They used a geometric cooling function as the temperature cooling schedule. Their experimental results show that the extended SA algorithms are able to produce feasible results on many of the problem instances that have not been reported previously in the literature.

2.4.3.5 The Great Deluge (GD)

The GD algorithm is a local search method proposed by Dueck [94]. The GD algorithm explores neighbouring solutions in such a way that these solutions are accepted only if they are better than the best solution so far or if the detriment in quality is not larger than the current water level [195]. The GD method was introduced as an alternative to SA. Apart from accepting a move that improves the solution quality, like SA, GD may accept a degrading move, i.e., a move that decreases the solution quality. In the acceptance criterion of GD, a degrading move is accepted if the fitness of the new solution is less than or equal to some given upper boundary value, referred to as the “water-level”. Its value is initially set to be equal to the penalty of the initial solution and at every iteration it is lowered by a fixed decay rate. The decreasing of the water level could be thought of as a control process, which drives the search towards a desirable solution.

Burke *et al.* [42] were the first to employ the GD approach to the UCTP. This approach involves two parameters, i.e., the estimated search time and the level of the solution quality. They tried to reduce the soft constraint violations while staying in the feasible region of the search space. The method was tested on the UCTP from an international timetabling competition and was confirmed as an effective method among 21 compared algorithms, where 7 out of 20 best-known results were obtained. The main advantage of this search is that it uses an estimated search time parameter, which enables the algorithm to adapt the intensity of the search and will only start to converge on local (or global) optimum when the estimated search time limit is being approached.

Subsequently, Landa-Silva and Obit [195] proposed an extended version of the GD algorithm with a non-linear decay rate. The approach produced 4 best results out of 11 problem instances. Later, McMullan [154] proposed an extended GD algorithm, which allows re-heating, similar to SA, and gives 5 new results out of the 11 problem instances.

2.4.4 Hybrid Meta-heuristics

In recent years, optimization problems have been of great importance for the scientific and industrial world. Many techniques (meta-heuristic and classical techniques such as branch and bound, dynamic programming, and gradient-based methods, etc.) are used to solve different optimization problems. The combination of a meta-heuristic with other optimization techniques (such as, operation research and artificial intelligence techniques) is called "hybrid meta-heuristics". It has become evident that hybrid meta-heuristic approaches show more efficient behaviour and higher flexibility when dealing with real-world and large-scale problems. It combines the complementary strengths of meta-heuristics (directly applicable to complex problems with relatively few modifications) with the strength of more classical optimization techniques [199]. Generally, hybrid meta-heuristic approaches can be classified as collaborative combinations and integrative combinations [65]. Collaborative combinations are based on the exchange of information between a meta-heuristic and another optimization technique that is running in parallel. Integrative combinations utilize another optimization technique as a subordinate part of a meta-heuristic. More details and relevant applications can be found in [46, 65, 78, 199]

Kostuch [139] employed a three-phase approach, which combines graph colouring and SA, for the UCTP. In the first phase, an initial feasible timetable is generated using the graph colouring heuristics. Improvement is made in the second phase using an SA algorithm. The final phase is applied to make further improvement using an LS method guided by SA. The approach was able to produce 13 best solutions out of 20 instances taken from the 2003 International Timetabling Competition.

2.4.5 Hyper-heuristic Approaches

The hyper-heuristic is considered to be an emerging methodology in search and optimisation [45]. Burke *et al.* [45] defined a hyper-heuristic as:

> “The process of using meta-heuristics to choose (meta) heuristics to solve the problem in hand.”

A hyper-heuristic is a powerful approach that can be thought of as a high level heuristic that modifies the solution quality by employing a set of low level heuristics. This is unlike most meta-heuristics that modify solutions directly. We can therefore say that it operates on the search space of heuristics rather than on the search space of candidate solutions. The main difference between hyper-heuristics and meta-heuristics in timetabling is that hyper-heuristics may use meta-heuristic from a variety of different heuristics to solve timetabling problems.

Burke *et al.* [47] investigated a TS hyper-heuristic approach for the UCTP and nurse rostering problems. The approach consists of choice function, tabu list, and a set of low-level heuristics. In this algorithm, a set of six low-level heuristics compete with each other. The change in the cost function value from the previous solution to a new solution is noted when heuristics are needed to be applied. A variable length dynamic tabu list prevents some heuristics from being used for some time during the search. The status of the heuristics in the tabu list will be changed from tabu active to non-tabu active if there is an improvement in the cost function. Experimental results showed that this technique is able to produce good quality solutions for UCTPs as well as different types of optimisation problems.

2.4.6 Other Approaches

Burke *et al.* [52] investigated a case-based reasoning (CBR) approach to solve the UCTP. Leake [141] described CBR as:

> "In CBR, new solutions are generated not by chaining, but by retrieving the most relevant cases from memory and adapting them to fit the new situations."

In CBR, all the problems are represented as cases. A case has two parts: one is the problem itself and the other is the solution to the problem or the lesson it will reach [141].

Burke *et al.* [52] modelled the UCTP as an attribute graph where nodes and edges represent courses and conflicts, respectively. The attributes on both nodes and edges represent information about the problem structure. The most similar case(s) from the target problem and candidate cases are selected for adaptation. The adaptation process is carried out by using a graph heuristic approach that is used to minimise constraint violations. Later, Burke *et al.* [53] investigated a multi-retrieval CBR approach to solve large-scale problem instances of UCTPs. This approach partitions a large problem into small solvable sub-problems by recursively inputting the unsolved part of the graph into the decision tree for retrieval. The adaptation combines the retrieved partial solutions of all the sub-problems, which, after graph heuristic method, construct the whole solution for the new case. They concluded that the proposed approach gives good results on different problem instances.

Gaspero and Schaerf [106] investigated a multi-neighbourhood search approach to solve the course timetabling problem. They used three neighbourhood structures, i.e., neighbourhood union, neighbourhood composition, and token ring search, in the LS technique. They proposed a set of operators that can automatically compose the neighbourhood function to a more complex one. Weighted objective function was used in order to penalise the occurrence of hard constraint violations. They also used a constraint technique in the exploration of a large neighbourhood to prune the list of candidates. To test the proposed approach, they used a real data set from the Department of Mathematics and Information, University of Udine, and concluded that the composition of neighbourhood gave much better results than the traditional LS method. The key aspect of their work is that their neighbourhood union operator chooses a random neighbourhood to perform a random movement within this chosen neighbourhood to produce a new solution in each step of the algorithm. This method explores a large search space and produces good results.

Malim *et al.* [151] proposed three artificial immune algorithms (AISs) for the UCTP. AIS are intelligent methodologies inspired by theoretical immunology and observed immune functions, principles, and models, toward real-world problem solving [82]. Malim *et al.* [151] proposed three immune algorithms to solve this problem. They are the clonal selection algorithm (CSA), the negative selection algorithm (NSA), and the immune network algorithm (INA). CSA uses selection, cloning, and mutation operators; INA uses cloning and mutation; and NSA uses negative deletion, cloning, and mutation. Malim *et al.* tested their approaches on the examination and course timetabling problems and their approaches are competitive in producing good results but on different time scales.

Asmuni *et al.* [29] applied a fuzzy multiple heuristic ordering method for the course timetabling problem. The fuzzy meta-heuristic was introduced by Zadeh in 1965 [216]. Asmuni *et al.* [29] used ordering of events by simultaneously considering three heuristics (large degree, saturation degree, and largest enrolment) using a fuzzy approach. The proposed technique was tested on 11 problem instances and the results showed that this approach is able to produce good results with a low requirement for rescheduling.

Abdullah *et al.* [13] employed a variable neighbourhood search (VNS) method for course timetabling and tested it on standard benchmark problems. The VNS method was proposed by Mladenovic and Hansen [157]. In VNS, more than one neighbourhood structure is systematically changed during the LS process to explore a variety of possible new search areas. Abdullah used twelve neighbourhood structures that explore the search space. VNS was investigated with the exponential Monte Carlo acceptance criterion [31] and tabu list. Their experimental results showed that VNS is able to produce some good results on small and medium problem instances on the cost of significant computational time. Later, Abdullah *et al.* [14] used eleven neighbourhood structures to investigate a randomised iterative improvement approach. In the iterative improvement algorithm, all neighbourhood structures were applied on each iteration on a solution and the best among all neighbourhood solutions was selected. If the selected solution is better than the current best solution, then it is treated as the current solution; otherwise, the Monte Carlo criterion [31] is applied. This approach also accepts a worse solution with a certain probability. The approach was able to produce competitive results on small problem instances.

Al-Betar [22] applied a new population-based meta-heuristics, called "harmony search" (HS), to solve the UCTP. Geem *et al.* [114] introduced the HS heuristic, that is based on the natural phenomenon of the behaviour of musicians when they play their musical instruments together to achieve a harmony. The proposed approach is able to integrate exploitation and exploration in a parallel optimisation environment. The experimental results showed that HS is able to give feasible solutions on small and medium problem instances.

Anh *et al.* [26] presented LS methods to solve course timetabling. Their approach works in two phases: the first phase consists of the creation of an initial solution that satisfies all hard constraints, while the second phase uses the iterative improvement method to minimise the soft constraints violations. The second phase was developed according to the requirements of the real data instances at Ho Chi Minh City University. The approach produced good timetables.

2.4.7 Multi-Objective Approaches

In this section, we will first describe the general concept of multi-objective optimization problems (MOOPs) along with some multi-objective EAs (MOEAs) for solving general MOOPs. These MOEAs will be used later in Chapter 7 to solve the UCTP. We will then see how different researchers have solved the UCTP as a MOOP.

2.4.7.1 MOEAs for General MOOPs

Many real-world problems have multiple conflicting objectives, where improvement of one objective may degrade one or more other objectives. The model of a general MOOP can be simply described as follows:

$$Max/Min\ F(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \cdots, f_M(\vec{x})) \tag{2.6}$$

$$subject\ to : \vec{x} = (x_1, x_2, \cdots, x_n) \in X \tag{2.7}$$

where $\vec{x}$ is called an *n-dimensional decision vector*, representing a solution for the MOOP, X is called the *parameter space*, *solution space*, or *search space*. $F(\vec{x})$ is the image of $\vec{x}$ in the *M-dimensional objective space*, and each function $f_i(\cdot)$, $i = 1, \cdots, M$, represents one objective to be maximized or minimized.

Given two decision vectors $\vec{a}$ and $\vec{b} \in X$, and, without loss of generality, assuming a maximization problem, then $\vec{a}$ is said to *dominate* $\vec{b}$ if and only if the following two conditions hold:

$$\forall i \in \{1, 2, \cdots, M\} : f_i(\vec{a}) \geq f_i(\vec{b}) \tag{2.8}$$

and

$$\exists j \in \{1, 2, \cdots, M\} : f_j(\vec{a}) > f_i(\vec{b}) \tag{2.9}$$

We can write $\vec{a} \succ \vec{b}$ to mean that $\vec{a}$ dominates $\vec{b}$.

Given a set of solutions, all solutions which are not dominated by any other solution in this set are called *non-dominated* regarding this set. The solutions that are non-dominated within the whole search space are called "Pareto optimal". These solutions constitute the *Pareto optimal set* or *Pareto optimal front* for an MOOP. The purpose of a multi-objective optimization algorithm is to find solutions that are as close to the Pareto optimal set as possible and as diverse as possible in the obtained non-dominated front [79].

EAs, as a class of population-based approaches, are well suited to solve these MOOPs because they possess several characteristics that are good for solving them [110, 137]. For example, an EA simultaneously searches different regions of the solution space, making it possible to find a diverse set of solutions; an EA has the ability to optimise multiple objectives simultaneously and requires minimal prior knowledge to solve an MOOP; and the performance of an EA is not affected by the shape and continuity of the search space [188].

The first multi-objective EA (MOEA), called "vector evaluated GA" (VEGA), was proposed by Schaffer [189]. Since then, a number of MOEAs [67, 99, 126, 187, 215, 222, 223], e.g., the elitist non-dominated sorting GA (NSGA-II) [80], the strength Pareto EA (SPEA) [221], and the ϵ-multi-objective EA (ϵ-MOEA) [81], have been developed and successfully applied for MOOPs.

NSGA-II was introduced in [80] based on the concepts of non-dominated sorting and crowding distance. The pseudo-code of NSGA-II is shown in Algorithm 2. Initially, a random population of size N is created and sorted based on the non-dominated

Algorithm 2 Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II)

```
1: **input**: A problem instance **I**
2: set the generation counter $g := 0$
3: initialise a population $P_g$ of $N$ solutions
4: evaluate the individuals in $P_g$
5: assign rank and crowding distance for individuals in $P_g$
6: **while** the termination condition is not reached **do**
7:     create a child population $Q_g$ from $P_g$ using the crowded tournament selection,
       crossover, and mutation
8:     evaluate the child solutions in $Q_g$
9:     merge the child and parent populations into a combined population $R_g := P_g \bigcup Q_g$
10:    assign rank and crowding distance for individuals in $R_g$
11:    create a new population from $R_g$ based on rank and crowding distance
12:    $g := g + 1$
13: **end while**
14: **output**: Non-dominated set of solutions
```

sorting [80]. Then, at each generation g, a child population Q_g is created from the parent population P_g by using the crowded tournament selection (two individuals are randomly selected from P_g and the tournament winner among them is decided according to their ranks and crowding distance values), crossover, and mutation operators. Next, the child population Q_g and P_g are merged together in R_g (so, R_g has $2N$ individuals) and the rank and crowding distance values of individuals in R_g are calculated. Finally, based on the ranks and crowding distances, the best N solutions are picked up from R_g to form a new population P_{g+1} for the next generation. Thus, at the end of each generation, the set of non-dominated solutions so far are obtained.

Knowles and Corne [136] introduced PAES based on a (1+1)-ES, as shown in Algorithm 3. PAES uses an archive A of a fixed size to store the best so far solutions during the solving process. The archive A is initially empty. As the searching

Algorithm 3 Pareto Archived Evolutionary Strategy (PAES)

```
1: input: A problem instance I
2: set the generation counter $g := 0$
3: initialise a solution $P_g$ and evaluate the solution
4: copy $P_g$ to the archive $A_g$
5: while the termination condition is not reached do
6:    apply mutation to $P_g$ to produce a child $C_g$
7:    evaluate $C_g$
8:    if $P_g$ dominates $C_g$ then
9:       discard $C_g$
10:   else if $C_g$ dominates $P_g$ then
11:      add $C_g$ to $A_g$ and replace $P_g$ with $C_g$
12:   else if $C_g$ is dominated by any member of $A_g$ then
13:      discard $C_g$
14:   else if $C_g$ dominates some members of $A_g$ then
15:      remove those members dominated by $C_g$ from $A_g$
16:      add $C_g$ to $A_g$ and replace $P_g$ with $C_g$
17:   else
18:      if $A_g$ is not full then
19:         add $C_g$ to $A_g$
20:         if $C_g$ resides in a less crowded region of $A_g$ than $P_g$ then
21:            replace $P_g$ with $C_g$
22:         end if
23:      else
24:         if $C_g$ resides in the most crowded region of $A_g$ then
25:            discard $C_g$
26:         else
27:            replace a random member from the most crowded region of $A_g$ with $C_g$
28:            if $C_g$ resides in a less crowded region of $A_g$ than $P_g$ then
29:               replace $P_g$ with $C_g$
30:            end if
31:         end if
32:      end if
33:   end if
34:   $g := g + 1$
35: end while
36: output: Non-dominated set of solutions in $A$
```

progresses, good solutions are added to A and updated.

At first, the parent P is created and added to archive A. Then, at each generation g, the parent P_g is mutated to create a child C_g. If the parent P_g dominates child C_g,then child is discarded and if the child C_g dominates parent P_g, the child is replaced as the new parent and added to archive. If C_g and P_g do not dominate each other, then the child C_g is compared with members in A_g. If C_g dominates any member of A_g, then it is replaced with child C_g and the child is also used as a parent. If C_g does not dominate any member in the archive, both parent and offspring belong to the same non-dominated front to which the archive solutions belong. In this case, there are two scenarios. In the first one, if the archive is not full, C_g is copied to A_g, and is accepted as the parent for the next generation if it is in the less crowded region[1] in the parameter space among the members of the archive than the parent. In the second scenario (i.e., the archive is full), if C_g resides in the most crowded region in the parameter space among the members of the archive, it is discarded; otherwise, C_g will replace one random member of A_g from the most crowded region, and is accepted as the parent for the next generation if it is in the less crowded region in the parameter space among the members of the archive than the parent.

Another MOEA that is used to solve different optimization problem is the improved strength pareto evolutionary algorithm (SPEA-II). SPEA-II was introduced by Zitzler *et al.* in [221]. The pseudo-code of SPEA-II is shown in Algorithm 4. SPEA-II uses two populations: one regular population P of size N and one archive A of size $\bar{N}$. First, a population P with N random solutions and an empty archive A are created. After that, each individual in P is first evaluated according to its objective values and then assigned a fitness value. In order to calculate the fitness value of an

[1]See [136] for the definition of regions constructed by the solutions in the archive. A region is more crowded if it has more archive solutions within it.

Algorithm 4 Improved Strength Pareto Evolutionary Algorithm (SPEA-II)

```
1: input: A problem instance I
2: set the generation counter $g := 0$
3: initialise a population $P_g$ of $N$ solutions
4: create an empty archive $A_g$ of size $\bar{N}$
5: while the termination condition is not reached do
6:    calculate the fitness values of individuals in $P_g$ and $A_g$
7:    merge $P_g$ and $A_g$
8:    perform the environmental selection on the merged population to form $A_g$
9:    perform the mating selection on $A_g$ to form a mating pool
10:   create the child population $P_g$ by applying crossover and mutation operators to the
      mating pool
11:   $g := g + 1$
12: end while
13: output: Non-dominated set of solutions
```

individual, it is first assigned a strength value $S(i)$, which is defined as follows:

$$S(i) = \mid \{j \mid j \in P_g + A_g \wedge i \succ j\} \mid \tag{2.10}$$

where "||" denotes the cardinality of a set, "+" stands for multiset union, and "$\succ$" corresponds to the Pareto dominance relation. Based on the strength value, the raw fitness $R(i)$ of individual i is calculated as follows:

$$R(i) = \sum_{j \in P_g + A_g, j \succ i} S(j) \tag{2.11}$$

Hence, this raw fitness value is calculated using the strengths of the dominators in both the archive and population sets.

In the event that individuals have the same raw fitness values, a density estimation technique is used. The density $D(i)$ corresponding to individual i is calculated as follows:

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{2.12}$$

For each individual i, the distances in the objective space to all individuals j in the archive and population are calculated and stored in a list. After sorting the list in increasing order, the k-th element gives the distance sought, denoted as σ_i^k, where k is equal to the square root of the sample size, i.e., $k = (N + \bar{N})^{1/2}$. Finally, the fitness value $F(i)$ of individual i is calculated on the basis of density and raw fitness as:

$$F(i) = R(i) + D(i) \tag{2.13}$$

After finding the fitness of individuals, archive and population are merged for environmental selection [221] to form a new archive for the next generation. In the process of environmental selection, all non-dominated individuals in the merged population are selected to re-fill the archive. If the number of non-dominated individuals is equal to the predetermined archive size $\bar{N}$, we copy them to the archive and stop the archive update operation; otherwise, there can be two situations: If the archive is bigger than the non-dominated set in size, we copy all non-dominated individuals and some dominated individuals from the previous archive and population into the archive up to the size $\bar{N}$; otherwise, if the archive is smaller than the non-dominated set, then a truncation method is used to remove individuals from the non-dominated set one by one as follows. The individual which has the minimum distance to another individual in the remaining non-dominated set is chosen to be removed. If

Algorithm 5 ϵ-Multi-Objective Evolutionary Algorithm (ϵ-MOEA)

```
1: input: A problem instance $\mathbf{I}$
2: set the generation counter $g := 0$
3: initialize a population $P_g$ of $N$ solutions
4: evaluate the individuals in $P_g$
5: copy non-dominated solutions in $P_g$ to the archive $A_g$
6: while the termination condition is not reached do
7:     select two parents: one from $P_g$ and one from $A_g$
8:     mate the two parents to create a set $C$ of children via crossover and mutation
9:     evaluate each child in $C$
10:    update $P_g$ according to $C$ using the dominance measure
11:    update $A_g$ according to $C$ using the $\epsilon$-dominance measure
12:    $g := g + 1$
13: end while
14: output: Non-dominated set of solutions in $A$
```

there is more than one individual with the same minimum distance, the tie is broken by considering the second smallest distances, and so on. The process continues until there are $\bar{N}$ individuals left in the non-dominated set, which are copied to form the archive. After the environmental selection, only members of the archive participate in the mating selection process using the binary tournament selection with replacement to create the mating pool. The child population is then created by applying crossover and mutation to the mating pool. This process is continued until the termination condition is reached.

Another MOEA is ϵ-MOEA, proposed by Deb *et al.* [81] based on [140], as shown in Algorithm 5. In ϵ-MOEA, the search space is divided into a number of grids (or hyper-boxes) and diversity is maintained by ensuring that a hyper-box can be occupied by only one solution [81]. Two co-evolving populations were used: one EA population P and one archive population A. The EA population P is initialized with N random solutions, which are evaluated. All non-dominated individuals in

P are copied into the archive A. In each generation, a child is created through two parents. One parent is chosen from A randomly. Another parent is chosen from P as follows: two individuals are selected randomly from P and compared. If one solution dominates the other, the dominating solution is chosen; otherwise, one is randomly chosen as the parent. After two parents are selected, they are mated to create a set of children $C = \{C_1, C_2, \cdots, C_\lambda\}$ via crossover and mutation operations, where λ is a parameter of ϵ-MOEA. After the set of children are created and evaluated, both P and A are updated according to C as follows.

For updating P, each child C_i, $i = 1, \cdots, \lambda$ is compared with all members in P. If C_i dominates any member in P, then it replaces that member. Otherwise, if any member of P dominates C_i, then C_i is not included in P. If both of the above tests fail, then C_i replaces a random individual from P. For updating A, each child C_i is compared with each member of A using the concept of ϵ-*dominance* [81]. In this process, each individual in A is assigned an identification array $\vec{B} = \{B_1, B_2, \cdots, B_M\}$, where M is the total number of objectives. The identification array $\vec{B}$ is defined as follows:

$$B_j(\vec{f}) = \begin{cases} \lfloor (f_j - f_j^{min})/\epsilon_j \rfloor, & \text{for minimizing } f_j \\ \lceil (f_j - f_j^{min})/\epsilon_j \rceil, & \text{for maximizing } f_j \end{cases} \tag{2.14}$$

where f_j is the j-th objective value of the individual, f_j^{min} is the minimum possible value of the j-th objective, and ϵ_j is the allowable tolerance in the j-th objective, below which two values are not significant to the user [140]. The identification arrays divide the whole objective space into hyper-boxes with the size ϵ_j in the j-th objective. The child C_i enters A according to its position in the hyper-boxes as described in [81]. In the end, A has the solutions obtained.

Comprehensive reviews, references, and recent research directions on MOEAs can be found in [6, 16, 98, 220].

2.4.7.2 Solving the UCTP as a MOOP

In the real world, the UCTP has multiple conflicting objectives. Some researchers have also investigated MOEAs for examination and school timetabling problems [62, 64, 172]. However, so far, only a few researchers have applied multi-objective optimization techniques to solving the multi-objective UCTP (MOUCTP). Burke *et al.* [50] proposed a hyper-heuristic approach for MOOPs and tested their approach on the space allocation problem and UCTP. Carrasco and Pato [58] used a bi-objective GA to solve the class teacher timetabling problem. They minimized the constraints violation of teachers and classes as two separate objectives.

Carrasco and Pato [58] used a bi-objective GA to solve the class teacher timetabling problem. They minimized the constraints violations of teachers related objective and classes oriented objective as two separate objectives. As these two objectives are basically in conflict with each other, the authors decided to apply MOEAs to solve this problem. A timetable is represented as a room and timeslot matrix. The initial population is generated by a constructive heuristic. After that, individuals are evaluated according to both objective values. Carrasco and Pato used their own genetic operators together with the non-dominated sorting approach [197], and an archive population to store the non-dominated solutions. They tested their approach on real data instances and concluded that their approach produces better timetables than manual timetables. The main advantage of their approach is that a user may be provided with a range of trade-off solutions with regard to the two competing objectives.

Datta *et al.* [83] used NSGA-II to solve the UCTP. They developed a bi-objective model to minimize soft-constraint violations, applied one crossover and four mutation operators into their algorithm, and tested the performance of their algorithm on real-world data. They concluded that the performance of their algorithm depends on user defined mutation probabilities and initial solutions.

Badri [32] formulated a multi-objective method for course scheduling for the United Arab Emirates University. This is a two-stage optimisation problem. Firstly, the model tries to maximise faculty course preferences when assigning faculty members to courses. Secondly, when allocating courses to timeslots, the approach tries to maximise faculty time preferences. Experimental results show that the model was able to offer an assignment that fulfils departmental policies, course offerings, and personnel preferences.

Abdullah *et al.* [19] proposed a multi-objective approach for the PECTP. They used NSGA-II with a variable population size. They gave individuals a life time at the time of its birth, the main purpose of assigning a life time being to ensure that only the good quality solutions will always be kept in the population pool. Their experimental results showed that this approach improves the performance of traditional NSGA-II.

2.5 Chapter Summary

In this chapter, we have described various types of timetabling problems, especially the UCTP. We also discussed many approaches that have been applied to solve the timetable problem. Each approach has its own advantages and disadvantages. Heuristic-based methods achieve good timetables quickly, but rely on a good choice of heuristics. Constraint logic programming has the ability to cope with different constraints to the system, but has a limited optimisation capability. It may be the case that some approaches are more suitable than others for certain types of problems/situations and user requirements.

We notice that many researchers tested their approaches on their own institutes' data or real data instances. As a result, it sometimes becomes very difficult to judge or assess how well an algorithm performs in comparison with other algorithms. Therefore, in this field, there is a great need for some standardised UCTP instances that can be used for the effective analysis and comparison of various timetabling algorithms. Hence, we will look at some well-known UCTP instances in the next chapter for this purpose.

Chapter 3

Benchmark Timetabling Problem Instances

3.1 Introduction

In this chapter, we will describe the specification of different versions of university course timetabling problems (UCTPs) that have been tested or used in the experimental study of the proposed approaches. We also present the formulation of these problem instances. In our research, we will use different benchmarks for the experimental studies, which are also mentioned here. These benchmarks are different from each other in terms of constraints or objective functions, etc. Detailed description are given later in this chapter. We will also look at various algorithms that have been proposed for these problem instances.

For the sake of descriptive convenience, throughout this chapter, we will use the acronyms UCTP (University Course Timetabling Problem), PECTP (Post Enrolment Course Timetabling Problem) and MOUCTP (Multi-objective University Course Timetabling Problem), to refer to these problem versions. In addition, unless explicitly stated otherwise, these acronyms will also apply throughout the remainder of the thesis. This chapter is organised as follows: Section 3.2 describes the specification for the UCTP, including the problem definition, formulation, benchmark dataset, and the state-of-the-art results for these benchmark problems. Section 3.3 describes the PECTP, including the problem definition, formulation, benchmark dataset, and the state-of-the-art results for these benchmarks used in the research. In Section 3.4, we first present the basic concepts of general MOOPs, then describe the MOUCTP, including the problem definition and benchmark dataset used in the research. Section 3.5 summarises this chapter.

3.2 The University Course Timetabling Problem (UCTP)

3.2.1 Problem Definition

This version of course timetabling was originally defined in 2001 so that it could be used for various research purposes by the meta-heuristics network [3], and was intended to overcome some of the common ambiguities and inconsistencies that existed in the study of automated course timetabling [143].

In this problem version, we deal with the following hard constraints:

- H1: No student attends more than one event at the same time;
- H2: The room is big enough for all the attending students and satisfies all the features required by the event;
- H3: Only one event is in a room at any time slot.

There are also soft constraints, which are equally penalised by the number of their violations and are described as follows:

- S1: No student should be required to attend an event in the last time slot of a day;
- S2: No student should sit more than two events in a row;
- S3: No student should have a single event in a day.

3.2.2 Problem Formulation

In a UCTP, we assign an event (course, lecture) into a time slot and also assign a number of resources (students and rooms) in such a way that there is no conflict between the rooms, time slots, and events. As mentioned by Rossi-Doria et al. [181], the UCTP consists of a set of n events (classes, subjects) $E = \{e_1, e_2, ..., e_n\}$ to be scheduled into a set of p time slots $T = \{t_1, t_2, ..., t_p\}$, a set of m available rooms $R = \{r_1, r_2, ..., r_m\}$ in which events can take place, a set of k students $S = \{s_1, s_2, ..., s_k\}$ who attend the events, and a set of l available features $F = \{f_1, f_2, ..., f_l\}$ that are satisfied by rooms and required by events.

In addition, interrelationships between these sets are given by five matrices.

- The first matrix $A_{k,n}$, called the *Student-Event* matrix, shows which event is attended by which students. In $A_{k,n}$, the value of $a_{i,j}$ is 1 if student $i \in S$ should attend event $j \in E$; otherwise, the value is 0.

- The second matrix $B_{n,n}$, called the *Event-Conflict* matrix, indicates whether two events can be scheduled in the same time slot or not.

- The third matrix $C_{m,l}$, called the *Room-Features* matrix, gives the features that each room possesses, where the value of a cell $c_{i,j}$ is 1 if $i \in R$ has a feature $j \in F$; otherwise, the value is 0.

- The fourth matrix $D_{n,l}$, called the *Event-Features* matrix, gives the features required by each event. It means that event $i \in E$ needs feature $j \in F$ if and only if $d_{ij} = 1$.

- The last matrix $G_{n,m}$, called the *Event-Room* matrix, lists the possible rooms to which each event can be assigned. Through this matrix, we can quickly identify all rooms that are suitable in size and feature for each event.

Usually, a matrix is used for assigning each event to a room r_i and a time slot t_i. Each pair of (r_i, t_i) is assigned a particular number which corresponds to an event. If a room r_i in a time slot t_i is free or no event is placed, then "-1" is assigned to that pair. In this way, we assure that there will be no more than one event assigned to the same pair so that one of the hard constraints will always been satisfied.

For the room assignment, we use a matching algorithm described by Rossi-Doria [181]. For every time slot, there is a list of events taking place in it and a pre-processed list of possible rooms to which the placement of events can occur. The matching algorithm uses a deterministic network flow algorithm and gives the maximum cardinality matching between rooms and events.

In general, the solution to a UCTP can be represented in the form of an ordered list of pairs (r_i, t_i), of which the index of each pair is the identification number of an event $e_i \in E$ $(i = 1, 2, \cdots, n)$. For example, the time slots and rooms are allocated to events in an ordered list of pairs like:

$$(2, 4),\ (3, 30),\ (1, 12),\ \cdots,\ (2, 7),$$

where room 2 and time slot 4 are allocated to event 1, room 3 and time slot 30 are allocated to event 2, and so on.

The goal of the UCTP is to minimize the soft constraint violations of a feasible solution (a feasible solution means that no hard constraint violation exists in the solution). The objective function $f(s)$ for a timetable s is the weighted sum of the number of hard constraint violations $\#hcv$ and soft constraint violations $\#scv$, which was used in [180], as defined below:

$$f(s) := \#hcv(s) * C + \#scv(s) \tag{3.1}$$

where C is a constant, which is larger than the maximum possible number of soft-constraint violations. In all the experiments carried out in later chapters for the UCTP, we set $C = 10^6$. If the objective function value shown in the experimental results exceeds 10^6, it means that the results contain infeasible solutions.

3.2.3 Benchmark Dataset

The experiments for the UCTP to be discussed in some chapters of this thesis were tested on the eleven benchmark UCTP instances proposed by Socha *et al.* [196]. In these problem instances, we need to schedule 100-400 events/courses into 45 time slots, corresponding to 5 days of 9 hours each day. By assigning events we have to satisfy room constraints and student constraints. These eleven problem instances are divided into three groups: five small instances, five medium instances, and one large instance. In the experimental studies to be carried out in the following chapters on this dataset, we use “S1” to “S5” to represent small instance 1 to small instance 5, respectively, “M1” to “M5” to represent medium instance 1 to medium instance 5, respectively, and “L” to represent the large instance. Table 3.1 presents different parameter values of these problem instances.

Table 3.1: Features of UCTP instances [196]

Class	Small	Medium	Large
Number of events	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Per room approximate features	3	3	5
Percentage (%) of features used	70	80	90
Number of students	80	200	400
Maximum events per student	20	20	20
Maximum students per event	20	50	100

3.2.3.1 Best Known Results on the UCTP Benchmark Instances

These problem instances were used in the First International Timetabling Competition (ITC-2002). A number of good papers have been published regarding this set of UCTP benchmark instances. In this sub-section, we will review some of the most notable works on this set of UCTP benchmark instances.

Burke *et al.* [47] proposed a tabu-based hyper-heuristic search (THHS) method and tested it on the UCTP and nurse rostering problems. They introduced a hyper-heuristics where 6 low-level heuristics compete with each other. They used a ranking mechanism to dynamically rank each low-level heuristics. When a heuristic has been applied, the change in the cost function value is recorded. If the low-level heuristic does not lead to any improvement of the current solution quality, then this heuristic will not be allowed to be selected for several following iterations. Burke *et al.* [47] used variable length dynamic tabu list and the status of the heuristics in the tabu list is changed from tabu active to non-tabu active if there is an improvement in the cost function. Their experimental results showed that their approach gives the best or similar results on 7 out of 11 problem instances in comparison with other algorithms.

A noticeable aspect of their approach is that, although it is not especially designed for timetabling problems, it proves to be capable of producing acceptable solutions for this problem, and hence, shows robustness for other optimization problems.

Rossi-Doria *et al.* [181] applied a genetic algorithm (GA) to solve the UCTP. They proposed a steady state GA along with a local search (LS) technique. In their algorithm, a population of random solutions is first generated. Each individual then undergoes an LS process and then is evaluated according to an objective function. Crossover and mutation operators are applied with a probability 0.8 and 0.5, respectively. They compared different heuristics and found that the GA is not a very effective approach in comparison with other heuristics towards solving this problem regarding the objective value.

Socha *et al.* [196] developed the first ant colony optimization (ACO) algorithm (AA) with the help of a construction graph and a pheromone model appropriate for the UCTP. They presented two ant based systems, an ant colony system and a MAX-MIN ant system. In both systems, an ant first constructs a complete assignment of events to time slots using the heuristic and pheromone information. The authors set an upper and lower bound for the value of each pheromone trail. An LS operator is then applied [180] to further improve the solution quality. A qualitative comparison is provided between the two ant colony algorithms. The only difference between the two algorithms lies in the interpretation and updating procedure of pheromones. Socha *et al.* compare their algorithms with a random restart LS scheme that starts from a random solution and keeps trying to find a better solution in its neighbourhood solutions. They generated 11 problem instances (described above) using a problem instance generator. Their experimental results showed that the ant colony algorithm based on the MAX-MIN ant system achieved better results than the LS scheme.

Asmuni *et al.* [29] proposed a fuzzy algorithm (FA) for the UCTP. In their work, they focused on the issue of ordering events by simultaneously considering three different heuristics using fuzzy methods. They used the combinations of two of the three heuristics, i.e., the Largest Degree (the degree of an event is the number of events that have the same students enrolled and conflict with the event), the Largest Enrolment (the number of students enrolled for each event), and the Saturation Degree (the number of time slots available to order the events). They compared their results with five single heuristics. Their experimental results showed that better and more feasible solutions could be obtained by combining more heuristics.

Abdullah *et al.* [13] used a variable neighbourhood search (VNS) approach based on a random-descent LS with an exponential Monte Carlo acceptance criterion for the UCTP. In their approach, the algorithm starts with a solution, initialized by constructive heuristics. The LS operator, which only accepts improved solutions, is then applied, 12 neighbourhood structures for the VNS were used. A worse move is accepted based on the Monte Carlo acceptance scheme [31]. TS was also used to penalize neighbourhood structures that are not performed. Ordered and non-ordered VNSs[1] were investigated and it was concluded that the ordered VNS gives better results on small and medium instances, but is unable to get a feasible solution on the large instance.

Later, Abdullah *et al.* [14] proposed a randomised iterative improvement method (RIIA). They used 11 neighbourhood structures that are applied one by one to an initial solution and the best result among all neighbourhood structures is selected. They also used an acceptance criterion [31] for choosing a worse move. The main advantage of their approach is that different neighbourhood structures help the algorithm explore different areas of the search space. Their experimental results showed that their approach is able to produce good results on small and medium problem instances.

Abdullah *et al.* [15] also investigated a randomised improvement algorithm with EA and proposed a hybrid EA (HEA). They tested a light mutation operator followed by a randomised iterative improvement algorithm on the UCTP. The light mutation operator is applied on some individuals in the population. The algorithm will terminate if the penalty cost is zero or the number of iterations reaches 200,000 and

[1]In a non-ordered VNS, if any improvement is found during the search with the current neighbourhood structure, then the VNS continues its search with the current neighbourhood structure rather than going back to the first (neighbourhood structure) one. In an ordered VNS, the neighbourhood structures are sequenced in the order of increasing size. Each time an improvement is found or a worse solution is accepted by the acceptance criterion, the search will go back to the first neighbourhood structure. If no improvement is found, the next neighbourhood structure is used to search.

takes approximately ten hours per run on each data set. Their experimental results showed that their approach is able to produce good results on all problem instances.

A graph-based hyper-heuristic (GBHH) was proposed by Burke *et al.* [51]. They employed TS with graph-based hyper-heuristics on the UCTP and the examination timetabling problem. TS is employed to search for the list of low-level heuristics. Low-level heuristics consist of permutations of graph heuristics and a random ordering method. A move in TS is to pick a new heuristic list, which is obtained by randomly changing two of the heuristics in the previous heuristic list. The newly visited heuristic lists are added into the tabu fixed length list. Here, TS finds the heuristic list that gives the best quality solution under consideration. Burke *et al.* tested their approach on 11 benchmark problem instances [196]. Their experimental results showed that their approach is able to find good solutions while maintaining the generality of the hyper-heuristic framework. This research shows that effective results on the UCTP can be achieved when a large number of low-level heuristics are used.

Kostuch [138] proposed a simulated annealing (SA) based heuristic. This approach is divided into two stages. First, it finds a feasible timetable via simple graph colouring heuristics. Second, it uses an SA scheme to improve the timetable according to an objective function. An algorithm's time limit was used to calculate a temperature decrement rate. The advantage of this approach is that it allows a cooling that is as slow as possible and spends sufficient time at low temperature for possible movement toward a near-optimal solution in the search space. One interesting feature of the research is that 40 instead of 45 time slots were used to place the events because the last time slot of every day is associated with one soft constraint. These time slots were only used if the feasibility of a timetable was effective. The algorithm was run on 20 problem instances of the ITC-2002 and became a winning entry.

An efficient timetabling solution (ETTS) with TS approach was proposed by Cordeau *et al.* [76]. They developed a tabu heuristic that first finds a feasible solution and then improves the quality of the solution by reducing soft constraints. Their algorithm allows the breaking of some hard constraints while dealing with soft constraints. The weighted sum function is used to penalize the occurrence of hard constraints. A parameter is used to penalize the hard constraints and the value of this parameter varies so that, when the number of hard constraints in the timetable rises above a certain amount, no further in-feasibilities will be allowed further. One advantage of this approach claimed by authors is that it allows free movements about the search space. Their algorithm is able to produce very good results on 20 problem instances of the ITC-2002 [1].

Bykov [56] used a great deluge LS algorithm (GDLS) to solve the problem. His algorithm is based on the approach by Burke *et al.* [42]. Bykov used a modified Brelaz sequential algorithm to initialize an initial solution, where the event with the minimum number of available time slots is scheduled first. The author reported that most of the time, the initial solution is feasible. After the initialization phase, a great deluge LS operator is applied to minimize soft-constraint violations. The neighbourhood randomly chooses an event and a new time slot for it. A move is accepted if it is feasible and satisfies the great deluge acceptance condition. The most noticeable thing about Bykov's algorithm is that subroutines are used to allocate rooms in a timetable. His algorithm produced good results on 20 benchmark instances of the ITC-2002 [1].

Gaspero and Schaerf [113] used a three-stage LS paradigm (TSLS). Their LS method consists of hill climbing, TS, and multi-swap shake stages. In the hill climbing stage, they used a set of three neighbourhood (i.e., change time, change room, and swap time). A move is accepted if it improves or leaves unchanged the value of the objective function. In the TS stage, they used a variable length short-term tabu mechanism. Only one neighbourhood that moves an event in a different time slot and different room is considered here. In the multi-swap shake stage, only one single move, in which all events in a time slot are moved to a different time slot was allowed. Results were reported on the basis of 20 trails for 20 instances of the ICT-2003 [1]. Gaspero and Schaerf's algorithm won fourth place among all competition entries.

Arntzen and Løkketangen [27] described a two-stage TS-based approach (AMLS). In the first stage, their algorithm uses a constructive heuristic procedure to build an initial feasible timetable, which takes a very small amount of time to execute and which operates by taking events one by one and assigning them to feasible places in the timetable, according to some specialised heuristics that also take into account the potential number of soft constraint violations that such assignments might cause. Their constructive heuristic dynamically inserts events into the current partial timetable. Their heuristic finds a feasible solution without restarting the process. In the second stage, they used a TS heuristic with simple neighbourhoods in order improve the soft constraints. In this stage, they maintain the feasibility. When an event is moved to a possible time slot, they mark the event as tabu. Their search is guided by tabu mechanisms based on recency and frequency of certain attributes of previous moves. When a long sequence of moves has been performed without improving the solution, they use an ejection chain process to move the search into another part of the search space. Their approach produced good results on 20 instances of the ITC-2002 [1].

Dubourg *et al.* [93] also proposed a TS approach (DTS) to solve the UCTP. They used a greedy algorithm for the initial configuration. The neighbourhood consists of swapping of two variables. The tabu list size varies according to the number of conflicts in terms of students attending an event. They concluded that lists which are too long can prohibit the visiting of unexplored regions of the search space. Tabu list records the couple of events previously swapped and their position in the configuration. Their approach is able to produce good results on 20 UCTP instances of the ITC-2002 [1].

3.3 The Post Enrolment Course Timetabling Problem (PECTP)

The UCTP can be further divided into two categories as proposed by the organizers of ITC-2007 [2]: the post enrolment course timetabling problem (PECTP) [146] and the curriculum base course timetabling problem (CBCTP). The main difference between them lies in the fact that in the CBCTP, courses are scheduled according to the curricula published by the university while in the PECTP, courses are scheduled according to the student enrolment data [150]. In this thesis, we also deal with the PECTP, which is similar to the ITC-2002[2] UCTP problem, but having additional new hard constraints. According to Lewis [146], these new hard constraints make it very difficult to find a feasible solution in the search space and make the PECTP not dissimilar to a real-world timetabling problem.

3.3.1 Problem Description

The PECTP consists of assigning university courses to time slots and rooms according to student enrolment data, where each assignment has to fulfil various constraints. Among these constraints, some are hard and some are soft. The PECTP deals with the following hard constraints [146]:

- H1: No student attends more than one event at the same time;

[2]For details, see http://www.idsia.ch/Files/ttcomp2002/oldindex.html.

- H2: The room is big enough for all the attending students and satisfies all the features required by the event;
- H3: Only one event is in a room at any time slot;
- H4: Events should only be assigned to time slots that are predefined as available for those events;
- H5: Where specified, events should be scheduled to occur in the correct order.

A solution is feasible if all hard constraints are satisfied by the solution. According to the organisers of the ITC-2007, a timetable is required to satisfy all hard constraints. Due to the fact that it may be very difficult to achieve this hard constraints feasibility, it was suggested that some events may be left unassigned in a timetable if necessary, and they introduced the notion of "distance to feasibility (Df)" [146], which is defined as the number of students that are affected by unassigned events, was introduced. Given a solution, if there is any event that causes any hard-constraint violation, it needs to be removed (i.e., unassigned) from the timetable, and as a consequence, some students who have to take this event will suffer from its removal. The Df of the given solution is calculated by identifying the number of students that are required to attend each of the unassigned events and then simply adding these values together. For example, if a solution has three events that need to be unassigned to prevent any violation of the hard constraints, and the number of students that need to attend each of these events is 2, 3, and

1, then the Df of the solution is $(2+3+1)=6$. With the notion of "distance to feasibility", an infeasible solution can be characterized by its Df.

In the PECTP, there are also soft constraints, which are penalized equally by their occurrences

- S1: No student should be required to attend an event in the last time slot of a day;
- S2: No student should sit more than two events in a row;
- S3: No student should have a single event in a day.

The soft constraint penalty value is denoted as *SCP* when we deal with the PECTP later in this thesis.

3.3.2 Problem Formulation

In a PECTP, we assign an event (courses, lectures) into a time slot and also assign a number of resources (students, rooms) in such a way that there is no conflict between the rooms, time slots, and events. The PECTP consists of a set of n events (classes, subjects) $E = \{e_1, e_2, ..., e_n\}$ to be scheduled in a set of p time slots $T = \{t_1, t_2, ..., t_p\}$, a set of m available rooms $R = \{r_1, r_2, ..., r_m\}$ in which events can take place, a set of k students $S = \{s_1, s_2, ..., s_k\}$ who attend the events and a set of l available features $F = \{f_1, f_2, ..., f_l\}$ that are satisfied by rooms and required by events.

In addition, interrelationships between these sets are given by the following seven matrices:

- The first matrix $A_{k \times n}$, called the *Student-Event* matrix, shows which event is attended by which students. The value of a cell $a_{ij} \in A_{k \times n}$ is 1 if student $s_i \in S$ should attend event $e_j \in E$; otherwise, the value is 0.
- The second matrix $B_{n \times n}$, called the *Event-Conflict* matrix, indicates whether two events can be scheduled in the same time slot or not. It helps to quickly identify events that can be potentially assigned to the same time slot.
- The third matrix $C_{m \times l}$, called the *Room-Feature* matrix, gives the features that each room possesses, where the value of a cell c_{ij} is 1 if $r_i \in R$ has a feature $f_j \in F$; otherwise, the value is 0.
- The fourth matrix $D_{n \times l}$, called the *Feature-Event* matrix, gives the features required by each event. It means that event $e_i \in E$ needs feature $f_j \in F$ if and only if $d_{ij} = 1$.
- The fifth matrix $G_{n \times m}$, called the *Event-Room* matrix, lists the possible rooms to which each event can be assigned. Through this matrix, we can quickly identify all rooms that are suitable in size and feature for each event.
- The sixth matrix $H_{n \times t}$ is called the *Event-Availability* matrix, where the value of a cell H_{ij} is 1 if event $e_i \in E$ should take place at time slot $t_j \in T$; otherwise, the value is 0.

- The last matrix $I_{n\times n}$ is called the *Event-Preference* matrix, where the value of cell I_{ij} is 1 if $e_i \in E$ has to be schedule before $e_j \in E$, or -1 if $e_i \in E$ has to be placed in the timetable after $e_j \in E$; otherwise, the value is 0 if there is no restriction of precedence between $e_i \in E$ and $e_j \in E$.

In addition to the above matrices, we create an array EE of lists. Each element $EE_i \in EE$ is a list of events that have to be scheduled in a timetable after event e_i. This information helps to satisfy the hard constraint H5 in the execution of an algorithm. We also create a list ET of event time slots. Each element $ET_i \in ET$ is a list of possible time slots where event e_i has to be scheduled. There is also a set E'_e of events that are not subject to any time restriction. Similarly, there is a set T'_s of time slots that have no restriction of any event.

Usually, a matrix is used for assigning each event to a room r_i and a time slot t_i. Each pair of (r_i, t_i) is assigned a particular number which corresponds to an event. If a room r_i in a time slot t_i is free or no event is placed, then "-1" is assigned to that pair. This way, we assure that there will be no more than one event assigned to the same pair so that one of the hard constraints will always be satisfied.

For room assignment, we use a matching algorithm described by Rossi-Doria *et al.* [181]. For every time slot, there is a list of events taking place in it and a pre-processed list of possible rooms to which events can be assigned. The matching algorithm uses a deterministic network flow algorithm and gives the maximum cardinality matching between rooms and events.

In general, the solution to a PECTP can be represented in the form of an ordered list of pairs (r_i, t_i) as described in Section 3.2.2 for UCTP.

The goal of the PECTP is to minimise the number of hard- and soft-constraint violations. The objective function $f(s)$ for a timetable s is the weighted sum of the number of hard-constraint violations $\#hcv$ and soft-constraint violations $\#scv$ (we refer to $\#scv$ as SCP later in this chapter), which was also used in [182], as defined below:

$$f(s) := \#hcv(s) * C + \#scv(s) \tag{3.2}$$

where C is a constant larger than the maximum possible number of soft-constraint violations.

3.3.3 Benchmark Dataset

The organisers of the ITC-2007 proposed 24 problem instances for the PECTP [146]. Table 3.2 presents the features of these PECTP instances[3]. These PECTP instances will be used later in the experimental study later in Chapter 6.

3.3.3.1 Best Known Results on the PECTP Benchmark Instances

In this subsection, we will review some of the most notable works on these PECTP benchmark instances, which were used in the ICT-2007.

[3]Details about these PECTP instances can be found at the official ITC-2007 website http://www.cs.qub.ac.uk/itc2007/postenrolcourse/course_post_index.htm.

Table 3.2: Features of the ITC-2007 PECTP instances

ITC-2007 Instance	1	2	3	4	5	6	7	8	9	10	11	12
Number of events	400	400	200	200	400	400	200	200	400	400	200	200
Number of rooms	10	10	20	20	20	20	20	20	10	10	10	10
Number of features	10	10	10	10	20	20	20	20	20	20	10	10
Number of students	500	500	1000	1000	300	300	500	500	500	500	1000	1000
Max students per event	33	32	98	82	19	20	43	39	34	32	88	81
Max events per student	25	24	15	15	23	24	15	15	24	23	15	15
Mean features per room	3	4	3	3	2	3	5	4	3	3	3	4
Mean features per event	1	2	2	2	1	2	3	3	1	2	1	23
ITC-2007 Instance	13	14	15	16	17	18	19	20	21	22	23	24
Number of events	400	400	200	200	100	200	300	400	500	600	400	400
Number of rooms	20	20	10	10	10	10	10	10	20	20	20	20
Number of features	10	10	20	20	10	10	10	10	20	20	30	30
Number of students	300	300	500	500	500	500	1000	1000	300	500	1000	1000
Max students per event	20	20	41	40	195	65	55	40	16	22	69	41
Max events per student	24	24	15	15	23	23	14	15	23	25	24	15
Mean features per room	2	3	2	5	4	4	3	3	3	3	5	5
Mean features per event	1	1	3	3	2	2	1	1	1	2	3	3

Cambazard *et al.* [57] proposed a mixed meta-heuristic approach (MMA), which includes tabu mechanism and simulated annealing in conjunction with various neighbourhood operators. Pure random configuration was used to start the algorithm. A feasible solution was found using LS by considering a unit cost per hard constraint violation. Six neighbourhood structures were used. Moves were run according to the solution quality and computational hardness. A move was included in the neighbourhood at a given iteration depending on a probability decreasing with its complexity. A tabu mechanism was also applied to prevent cycling. After finding a feasible solution. SA was used to minimize the cost of soft constraints. The initial temperature was chosen dynamically as the average of the variation of the objective function when running the SA at a temperature of 1. A standard cooling scheme was applied. A move was only selected if it preserved the feasibility. They also intended to combine their LS with constraints programming but their LS alone gives very good results on the PECTP with 10 optimal solutions out of 24 problem instances. They won first place in the ITC-2007 [2].

Mitsunori *et al.* [30] proposed a technique that is a combination of a general purpose constraint satisfaction solver, TS, and iterated LS techniques (CTI). They treated the PECTP instances as instances of CSP and applied a general purpose CSP solver to find their solutions. As a CSP solver, they used TS and iterative LS. Their solver is based on Nonobe and Ibaraki's work [165], with additional quadratic 0-1 constraints handling capabilities. They assigned different initial weights to soft and hard constraints and the weights were dynamically controlled during the computation to improve the quality of solutions. Their algorithm produced very good results on the PECTP instances and achieved the second place in the ITC-2007 [2].

Chiarandini *et al.* proposed a hybrid algorithm (HA), that combines a constructive procedure for achieving the feasibility, followed by LS and simulated annealing for satisfying the soft constraints. The algorithm consists of several heuristic modules that have been tuned and assembled using an automated algorithm configuration procedure, called ParamILS [127]. The advantage of their solver lies in that it is able to explore a large design space of the hybrid stochastic LS algorithm. Their algorithm consists of two main steps. In the first step, hard constraints are dealt with and in the second step, it tries to minimize the occurrence of soft constraints violations. The first step is a constructive phase followed by a LS phase. The second step is applied to the assignment returned by the first phase. It consists of LS using four neighbourhood structures (1-exchange, 2-exchange, swap-of-time-slots, and Kempe chain[4]). They achieved very good results on the PECTP instances, winning third place in the ITC-2007 [2].

In [166], Nothegger *et al.* proposed an ACO algorithm in conjunction with a local improvement search routine. They proposed ACO with the help of two types of pheromones: one represents the probabilities of assigning an event to time slots and the other represents the probabilities of assigning an event to rooms. These pheromones are the main characteristics of their algorithm. The advantage of this approach is that it avoids the usage of a much larger data structure implied by a more traditional encoding that uses individual pheromone values for all time slot, room and event combinations. It also contains more information than the exclusive use of event-time slot pheromones. By considering the pheromone information, events are assigned to feasible rooms and time slots in a greedy randomized way. An LS procedure was also used to further improve solutions, and an improvement heuristic was employed which tries to move costly events (involved in soft constraint violations) to other time slots until suitable places are found or the search limit is reached. Their approach gained fourth place in the ITC-2007.

[4]The Kempe chain is a neighbourhood structure used in [158, 201]. In the Kempe chain neighbourhood structure, the assignment of events and time slots can be represented as a colouring of the graph. A feasible colouring is a partition of the graph into independent sets. A Kempe chain K is a set of vertices that form a maximal connected component in the sub-graph S of G induced by the vertices that belong to two different independent sets I_i and I_j, $i \neq j$. A Kempe chain interchange produces a new feasible partition by swapping the colours assigned to the vertices belonging to K, i.e. replacing I_i with $(I_i \backslash K) \cup (I_j \cap K)$ and I_j with $(I_j \backslash K) \cup (I_i \cap K)$.

Müller [162] used an LS based algorithm (LSA) with routines taken from the Constraint Solver Library[5]. Various neighbourhood search algorithms were also used to eliminate violations of hard- and soft-constraints. His algorithm consists of two phases: the construction phase and the heuristic phase. In the construction phase, an iterative forward search algorithm is used to create an initial solution. Each event is assigned a room and a time slot. If there is any hard-constraint violation, then the event remains unassigned. A conflict-based statistics is also used to prevent repetitive assignment of the same rooms and time slots to events. In the heuristic phase, different meta-heuristics, such as hill climbing, great deluge, and SA, are used. Firstly, the hill climbing algorithm is applied with a problem-specific neighbourhood to find the local optimum. In this process, a move is only accepted when it does not worsen the overall solution value. This process is finished when no improvement is found after a specified number of iterations. Secondly, the great deluge algorithm is applied, which allows some oscillations of the bound value that is imposed on the overall solution value. The process starts with the bound value $B = GD_{ub} \times S_{best}$, where GD_{ub} is the upper bound coefficient (a problem specific parameter) and S_{best} is the best solution value so far. Moves are only accepted when the value of the solution does not exceed the bound. This bound is decreased at every iteration because of the decrease in the cooling rate. When the cooling rate reaches its minimum level, then the bound is reset to its upper starting limit. The advantage of this process is that it widens the search space and helps the algorithm to get out of local minimum if there is no improvement found. Depending on the

[5]Available at http://cpsolver.sf.net.

solution quality, an optional SA process may be applied between the bound oscillations of the great deluge heuristic. Müller's approach outperformed other algorithms on exam timetabling but was awarded fifth place in the ITC-2007 [2].

3.4 The Multi-Objective UCTP (MOUCTP)

3.4.1 Basic Concepts of General MOOPs

Most real world problems have multiple objectives, where improvement in one objective performance may degrade the performance of one or more other objectives. The MOOP is also known as a multi-criteria optimization, multi-performance, or vector optimization problem. According to Osyczka [169], a MOOP can be described as:

> "a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term 'optimize' means finding such a solution which would give the values of all the objective functions acceptable to the decision maker".

3.4.2 Problem Definition of the MOUCTP

According to Carter and Laporte [69], the UCTP is a multi-dimensional assignment problem, in which students and teachers (or faculty members) are assigned to events (individual meetings between students and teachers, e.g., courses, lectures, or classes) and events are assigned to classrooms and time slots under different constraints. As discussed before, the real-world UCTP consists of different constraints: some are hard constraints and some are soft constraints.

Most researchers have tackled the UCTP as a single-objective optimisation problem. Researchers combine multiple criteria into a single scalar value and minimise the weighted or unweighted sum of constraints violations as the only objective function. But, inherently, the UCTP has many different objectives or constraints, such as minimizing the number of consecutive classes, minimizing the occurrence of classes in the last time slot or maximizing the usage of resources, and many more. All these and many other constraints lead the scheduling of a university timetabling to an optimization problem. On the other hand, it is very difficult, or may even be impossible to satisfy all the hard and soft constraints for complex or large UCTP instances [83]. According to Rudová and Murray [184], this complexity requires the scheduling of timetables to be treated as finding a solution over hard constraints, which is then to be optimized over soft constraints. To this aim, the UCTP should be treated as a MOOP, i.e., we need to study the MOUCTP with different soft constraints taken as different objective functions.

So far, there are no particular published benchmarks for the MOUCTP in the literature. Due to this deficiency, in this thesis, we tackle the UCTP described in Section 3.2 as a MOUCTP. In this thesis, the three soft constraints in the problem (described in Section 3.2.1) are taken as three objectives $f_1(x)$, $f_2(x)$, and $f_3(x)$, respectively, where x is a solution to the MOUCTP. These three objectives are to be minimized in a timetable, while satisfying all hard constraints.

3.4.3 Benchmark Dataset

Since there is no particular published benchmark of the MOUCTP for researchers to test their approaches, most probably due to lack of research in the area, many researchers tested their proposed techniques on real data instances. Some researchers [13, 50, 181] used the benchmark dataset proposed in [196], as described in Section 3.2.3, for testing their approaches for the MOUCTP. We will also use these benchmark instances to test the performance of our proposed MOEAs for the MOUCTP in this thesis. The objective function of the MOUCTP is to minimize all three soft constraints while satisfying hard constraints. The experimental results shown later in this thesis on all these benchmark problem instances are based on average values of 100% feasible solutions runs, unless otherwise stated explicitly.

3.5 Chapter Summary

In this chapter, we described the specification of three versions of the university course timetabling problem (UCTP), namely the general UCTP, the post enrolment course timetabling problem (PECTP), and the multi-objective UCTP (MOUCTP). We gave the definition and formulation of these problems. We also introduced the benchmark problem instances (datasets) for the general UCTP, the PECTP, and the MOUCTP, which are typically used by researchers in the literature, e.g., the ITC-2007 PECTP instances.

In the following chapters of this thesis, we will use these benchmark problem instances to test the performance of investigated approaches for different UCTPs.

Chapter 4

Memetic Algorithms for University Course Timetabling

4.1 Introduction

In [181], Rossi *et al.* compared different metaheuristics that are used to solve the university course timetabling problem (UCTP). They concluded that conventional genetic algorithms (GAs) do not give good results among a number of approaches developed for the UCTP. Hence, conventional GAs need to be enhanced to solve the UCTP. As a starting point of our research, we examine the behaviour of a simple GA for the UCTP. This chapter presents an initial investigation into a steady state GA (SSGA). The SSGA, which is one simple model of GAs, was originally developed by Whitley in [212]. The pseudo-code of the SSGA is shown in Algorithm 6.

Algorithm 6 Steady State Genetic Algorithm (SSGA)

```
1: randomly initialise a population of solutions
2: evaluate the individuals in the population
3: while the termination condition is not reached do
4:     select two parents through a selection scheme
5:     crossover the parents to create a child with a probability $P_c$
6:     apply mutation to the child with a probability $P_m$
7:     replace the worst member of the population by the child
8: end while
```

SSGA is based on the conventional GA in Algorithm 1, and the italicised parts in Algorithm 6 show the changes from the conventional GA.

In SSGA, usually, one offspring is generated via reproduction each iteration (generation). It starts from a random initial population of possible solutions for a problem. Each solution in a population is called an individual of the population. Each individual is evaluated according to a problem-specific objective function, usually called the fitness function. After evaluation, there is a selection phase in which two possibly good individuals will be chosen as parents by a selection operator to undergo the recombination process. In the recombination phase, crossover and mutation operators are used to create a new individual in order to explore the solution space. The newly-created individual replaces an old individual, usually the worst one, of the population based on fitness. This process is repeated until a stopping criterion is reached, which may be the maximum number of generations or a time limit.

In this chapter, we present a memetic algorithm (MA) based on the SSGA. In the proposed MA, two local search (LS) methods are integrated into the SSGA to solve the UCTP. These two LS methods use their exploitive search ability to improve the

Algorithm 7 The Proposed Memetic Algorithm (MA)

```
1: input : A problem instance I
2: for $i = 1$ to population size do
3:     construct random initial solution $s_i$.
4:     apply LS1 to the solution $s_i$
5:     apply LS2 to the solution $s_i$
6: end for
7: sort population by fitness
8: while termination condition not reached do
9:     select two parents through a selection scheme
10:    crossover the parents to create a child $s$ with a probability $P_c$
11:    apply mutation to the child $s$ with a probability $P_m$
12:    apply LS1 to the child $s$
13:    apply LS2 to the child $s$
14:    child solution $s$ replaces the worst member of the population
15:    sort population by fitness
16:    found best solution $s_{best}$ in the population
17: end while
18: output : The best solution $s_{best}$ achieved for I
```

explorative search ability of the SSGA. The evaluation of the MA is undertaken by presenting a series of computational results on the benchmark UCTP instances described in Chapter 3 (Section 3.2).

This chapter is organised as follows: Section 4.2 describes the pseudo-code of the proposed MA, including different components of the MA in detail. The experimental study and experimental results are presented and discussed in Section 4.3. Finally, Section 4.4 presents a summary of the chapter.

4.2 The Proposed Memetic Algorithm

In this section, we present our proposed MA that integrates some new LS approaches into the conventional SSGA to enhance the searching ability of the SSGA for the

UCTP. Algorithm 7 shows the outline of the MA proposed for the UCTP. The proposed MA is based on the SSGA. The changes that have been made in the SSGA are shown in italics fonts in Algorithm 7.

In the proposed MA, we first initialize the population by randomly creating each individual, via assigning a random time slot for each event according to a uniform distribution and applying the matching algorithm to allocate rooms for events. Two LS strategies, LS strategy 1 (LS1) and LS strategy 2 (LS2), which will be described in Section 4.2.1, are then applied in order to each member of the population. LS1, developed by Rossi *et al.* [180], uses three neighbourhood structures, denoted as $N1$, $N2$, and $N3$, which will also be described in Section 4.2.1, to move events to time slots and then uses the matching algorithm to allocate rooms to events and time slots. With LS2, we take the time slot with the worst penalty value from a set of randomly selected time slots and seek to improve it by trying to move each event in that time slot to another one in the neighbourhood structure $N1$ and then using the matching algorithm for room allocations for those involved events.

After the initialization of the population, we use the SSGA model as mentioned in [71], where only one child solution is generated with selection, crossover, and mutation at each generation. The child is then improved by LS1 and LS2. In the end, the worst population member is replaced with the new child individual. The iteration continues until one termination condition is reached, e.g., a present time limit t_{max} is reached.

In the following sub-sections, we describe the components of the MA, including the LS strategies and genetic operators, respectively.

4.2.1 Local Search (LS) Strategies

In the proposed MA, after an initial solution or a new offspring solution is created, two LS operators are applied to the solution one after the other. These two LS strategies are based on three neighbourhood structures, $N1$, $N2$, and $N3$, which are described as follows:

- $N1$: the neighbourhood defined by an operator that moves one event from a time slot to a different one
- $N2$: the neighbourhood defined by an operator that swaps the time slots of two events
- $N3$: the neighbourhood defined by an operator that permutes three events in three distinct time slots in one of the two possible ways other than the existing permutation of the three events.

4.2.1.1 LS Strategy 1 (LS1)

Algorithm 8 shows the pseudo-code of LS1. LS1 works on all events of an individual. Here, we suppose that each event is involved in soft and hard constraint violations. LS1 works in two steps. In the first step (lines 3-13 in Algorithm 8), it checks the

Algorithm 8 Local Search Strategy 1 (LS1)

```
1: input : Individual I selected from the population
2: while Termination condition not reached do
3:   for $i = 1$ to the total number of events do
4:     if event $i$ is infeasible then
5:       if there is untried move left then
6:         calculate the next move (first in $N1$, then in $N2$, and finally in $N3$)
7:         apply the matching algorithm to the time slots affected by the move and
           delta-evaluate the result.
8:         if the move reduces hard constraint violation then
9:           make the move
10:        end if
11:      end if
12:    end if
13:  end for
14:  if any hard constraint violations remain then
15:    terminate LS1
16:  else
17:    for $i = 1$ to total number of events do
18:      if event $i$ has soft constraint violation then
19:        if there is untried move left then
20:          calculate the next move (first in $N1$, then in $N2$, and finally in $N3$)
21:          apply the matching algorithm to the time slots affected by the move and
             delta-evaluate the result
22:          if the move reduces soft constraints violation then
23:            make the move
24:          end if
25:        end if
26:      end if
27:    end for
28:  end if
29: end while
30: output : A possibly improved individual I
```

hard constraint violations of each event while ignoring its soft constraint violations. If there are hard constraint violations for an event, LS1 tries to resolve them by applying moves in the neighbourhood structures $N1$, $N2$, and $N3$ in order[1] until a

[1]For the event being considered, potential moves are calculated in a strict order. First, we try to move the event to the next time slot, then the next, then the next, etc. If this search in $N1$ fails, we then search in $N2$ by trying to swap the event with the next one in the list, then the next one, and so on. If the search in $N2$ also fails, we try a move in $N3$ by using one different permutation formed by the event with the next two events, then with the next two, and so on.

termination condition is reached, e.g., an improvement is reached or the maximum number of steps s_{max} is reached, which is set to different values for different problem instances. After each move, we apply the matching algorithm to the time slots affected by the move and try to resolve the room allocation disturbance and delta-evaluate the result of the move (i.e., calculate the hard and soft constraint violations before and after the move). If there is no untried move left in the neighbourhood for an event, LS1 continues to the next event. After applying all neighbourhood moves on each event, if there is still any hard constraint violation, then LS1 will stop; otherwise, LS1 will perform the second step (line 17-27 in Algorithm 8).

In the second step, after reaching the state of a feasible solution, LS1 then deals with soft constraints and again performs a similar process as in the first step on each event to reduce its soft constraint violations. For each event, LS1 tries to make moves in the neighbourhood $N1$, $N2$ and $N3$ in order without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta-evaluated. When LS1 finishes, we get a possibly improved and feasible individual. We then apply LS2 on this individual.

4.2.1.2 LS Strategy 2 (LS2)

Algorithm 9 shows the pseudo-code of LS2. The basic idea of LS2 is to choose a high penalty time slot that may have a large number of events involving hard and soft constraints. LS2 first randomly selects a *tp* percentage of time slots from the total time slots of T. Rather than choosing a worst time slot out of all the time

Algorithm 9 Local Search Strategy 2 (LS2)

```
1: input : Individual I after LS1 is applied
2: $S$ := randomly select a $tp$ percentage of time slots from the total time slots of $T$
3: for each time slot $t_i \in S$ do
4:     for each event $j$ in time slot $t_i$ do
5:         calculate the penalty value of event $j$
6:     end for
7:     sum the total penalty value of events in time slot $t_i$
8: end for
9: select the time slot $w_t$ with the biggest penalty value from $S$
10: for each event $i$ in $w_t$ do
11:     calculate a move of event $i$ in the neighbourhood structure $N1$
12:     apply the matching algorithm to the time slots affected by the move
13:     compute the penalty of event $i$ and delta-evaluate the result
14: end for
15: if all the moves together reduce hard or soft constraint violations then
16:     apply the moves
17: else
18:     delete the moves
19: end if
20: output : A possibly improved individual I
```

slots, we randomly select a set of time slots and then choose the worst. This is because, for each selected time slot, we need to calculate its penalty value, which costs time. Through selecting a set of time slots instead of all time slots, we try to balance between the computational time and the quality of the algorithm. We then compute the penalty of each selected time slot and choose the time slot w_t that has the biggest penalty value for the LS operation. In this way, LS2 aims to help improve the existing result of LS1.

After taking the worst time slot, LS2 tries a move in the neighbourhood $N1$ for each event of w_t and checks the penalty value of each event before and after applying the move. If all moves in w_t together reduce the hard and/or soft constraint violations, we then apply all the moves; otherwise, we do not make the moves. In this way,

Algorithm 10 *Crossover*()

```
1: input : The current population
2: Select parents $P1$ and $P2$ by the binary tournament selection scheme
3: for each event $e_i$ of the child $Ch$ do
4:     if $rand(0,1) < 0.5$ then
5:         $e_i$ of $Ch \leftarrow$ the time slot allocated to $e_i$ of P1
6:     else
7:         $e_i$ of $Ch \leftarrow$ the time slot allocated to $e_i$ of P2
8:     end if
9: end for
10: allocate rooms to all occupied time slots using the matching algorithm
11: output : A new child generated using $Crossover()$
```

LS2 can not only check the worst time slot but also reduce the penalty value for some events by moving them to other time slots. In general, LS2 can enhance the individuals of the population and increase the quality of the feasible timetable by reducing the number of constraint violations.

4.2.2 Genetic Operators

The proposed MA uses the SSGA model. One offspring is generated from the current population at each generation using following genetic operators and relevant parameters, which were also used in [181].

4.2.2.1 Selection

The binary tournament selection is used, where two parents are randomly selected from the population and the fitter one is used as a parent. At each generation, the tournament selection is applied twice to select two parents for reproduction.

4.2.2.2 Crossover

A uniform crossover operator is used with a probability $P_c = 0.8$. It first randomly assigns a time slot from one of the two parents to each event in the offspring and then allocates rooms to events in each non-empty time slot. The pseudo-code of the crossover operator is as shown in Algorithm 10.

4.2.2.3 Mutation

A mutation operator is used with a probability $P_m = 0.5$. It randomly selects a neighbourhood structure $N1$, $N2$, or $N3$, and makes a move in the selected neighbourhood to mutate an individual.

4.3 Experimental Study

The proposed MA was implemented in GNU C++ with version 4.1 and run on a 3.20 GHz PC. We used the set of benchmark problem instances that were described in Section 3.2 of Chapter 3 to test our algorithm and ran our algorithm on 11 problem instances that were divided into three groups: 5 small, 5 medium, and 1 large instance. Table 3.1 represents the data of the timetabling problem instances of the three different groups. In LS1 of our MA, the maximum number of steps per LS operation s_{max} was set to different values for different problem instances (200 for small instances, 1000 for medium instances, and 2000 for the large instance).

Two sets of experiments were carried out in this study. The first set of experiments are devoted to finding the appropriate value for the parameter tp in the MA and the second set of experiments compares the performance of the MA without LS2. For both sets of experiments, there are 50 runs of each algorithms on each problem instance. For each run, the maximum run time t_{max} was set to 90 seconds for small instances, 900 seconds for medium instances, and 9000 seconds for the large instance. In the end, we compare the experimental results of our MA with current state-of-the-art methods from the literature on the tested 11 timetabling problem instances.

4.3.1 Sensitivity Analysis of the Parameter tp

The performance of LS2 depends upon the tp parameter. This is the percentage of the time slots that are used to find the worst time slot among them. We test the tp value in the set of 10%, 20%, 40%, 60%, 80%, and 100%. Figure 4.1 shows the performance of the MA with different values of tp on the S1 and M3 problem instances.

From Figure 4.1, it can be seen that increasing the value of tp from 10% to 20% improves the performance of the MA on both the S1 and M3 problem instances. However, when we further increase the value of tp from 20% to other larger values, the performance of the MA degrades. The reason may be that there is a larger percentage of time cost and under the predefined time (e.g., for small problem instances

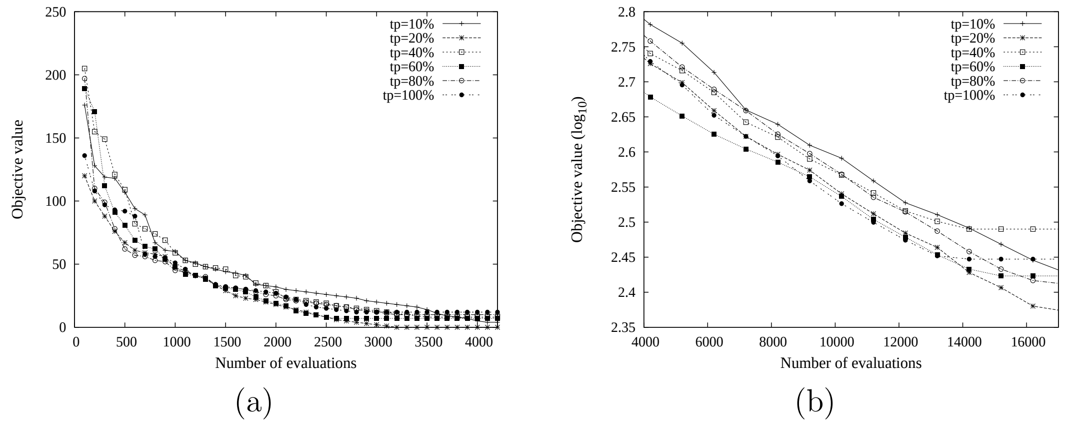

FIGURE 4.1: Comparing the performance of the MA with different *tp* values on the problem instances: (a) S1 and (b) M3.

90 sec) MA is not able to produce a good solution. Figure 4.1 suggests that 20% of time slots is able to make a good balance between the time cost and the quality of the solution.

4.3.2 The Effect of LS2 on the MA

In this experiment, we investigate the effect of LS2 on our proposed MA for the UCTP. We compare the proposed MA with the EA proposed by Rossi-Doria *et al.* [181], which is the same as our MA but with LS2 switched off. Table 4.2 compares MA and EA in terms of the best, average, standard deviation, and worst penalty value over the 50 runs on the problem instances. The results of statistical comparison of our MA against the EA by Rossi-Doria *et al.* [181] using the t-test on the small and medium problem instances are shown in Table 4.1. Details of this t-test are provided in Appendix A. In Table 4.1, the t-test results were based on the average population fitness data of 50 runs of our MA and EA on each instance, and were calculated with

TABLE 4.1: The t-test results of comparing MA against EA

UCTP	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
MA − EA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

TABLE 4.2: Comparison of MA and EA on different problem instances

UCTP	Alg	Best	Ave	Std	Worst
S1	MA	0	3	5.13	15
	EA	0	7	7.10	15
S2	MA	0	8	5.46	19
	EA	3	11	8.67	31
S3	MA	0	5	3.60	19
	EA	0	7	5.41	26
S4	MA	0	1	2.41	7
	EA	0	7	7.01	24
S5	MA	0	3	2.84	7
	EA	0	4	6.80	19
M1	MA	212	240	14.67	271
	EA	267	307	19.68	345
M2	MA	171	208	23.15	267
	EA	175	218	29.97	294
M3	MA	222	253	22.83	357
	EA	254	309	25.90	378
M4	MA	132	165	18.83	206
	EA	237	269	25.04	328
M5	MA	197	227	23.42	287
	EA	230	270	31.17	347

98 degrees of freedom at a 0.05 level of significance. In Table 4.1, the t-test result of comparing two algorithms is shown as "$s+$" when the first algorithm is significantly better than the second algorithm (the data used for these t-test results are shown in Table A.1 and Table A.2, respectively). From Table 4.2 and Table 4.1, it can be seen that our MA performs significantly better than the EA by Rossi-Doria *et al.* [181] on all small and medium problem instances. The reason for this result is that LS2 in the MA works well to further improve the quality of solutions obtained by the MA. This result shows the importance of introducing LS2 into our MA for the UCTP.

4.3.3 Comparison with State-of-the-Art Algorithms

Table 4.3 gives the comparison of the experimental results of our MA with the available results of other algorithms in the literature on the small and medium timetabling problem instances. In Table 4.3, the term "%ln" represents the percentage of runs that failed to obtain a feasible solution, and "Best" represents the best result among a number of runs. We present the best of all the algorithms in bold font. The algorithms compared in the table are described in detail in Chapter 3 Section 3.2. Here we describe them briefly as follows:

- GA: traditional GA without any LS strategy.
- MA: our MA proposed in this study
- RIIA: The randomised iterative improvement method of Abdullah *et al.* [14]. They presented a composite neighbourhood structure with a randomised iterative improvement algorithm.
- VNS: The variable neighbourhood search of Abdullah *et al.* [13]. Abdullah *et al.* used a variable neighbourhood search approach based on a random-descent LS with an exponential Monte Carlo acceptance criteria.
- THHS: The tabu-based hyper-heuristic search of Burke *et al.* [47]. They introduced a TS hyper-heuristics where a set of low-level heuristics compete with each other. This approach was tested on the UCTP and nurse rostering problems.

Table 4.3: Comparison of the proposed MA with state-of-the-art approaches on small and medium problem instances

Datasets	MA		RIIA		HEA	GBHH	VNS	THHS	LS	EA	AA	FA	GA
	Best	Median	Best	Median	Best	Best	Best	Best	Median	Best	Median	Best	Best
S1	**0**	1	**0**	0	**0**	6	**0**	1	8	**0**	1	10	4
S2	**0**	8.5	**0**	0	**0**	7	**0**	2	11	3	3	9	12
S3	**0**	1	**0**	0	**0**	3	**0**	**0**	8	**0**	1	7	3
S4	**0**	0	**0**	0	**0**	3	**0**	1	7	**0**	1	17	2
S5	**0**	0	**0**	0	**0**	4	**0**	**0**	5	**0**	**0**	7	1
M1	212	239	242	245	221	372	317	**146**	199	267	195	243	445
M2	171	208	161	162.6	**147**	419	313	173	202.5	175	184	325	80%In
M3	**222**	243	265	267.8	246	359	357	267	77.5%In	254	248	249	92%In
M4	**132**	162.5	181	183.6	165	348	247	169	177.5	237	164.5	285	467
M5	197	220	151	152.6	**130**	171	292	303	100%In	230	219.5	132	100%In

- EA: The EA with LS1 of Rossi-Doria *et al.* [181] that was applied for the UCTP.

- HEA: The hybrid EA of Abdullah *et al.* [15]. They tested a light mutation operator followed by a randomised iterative improvement algorithm on the UCTP.

- LS: The LS method of Socha *et al.* [196]. They used a random restart LS strategy for the UCTP and compared it with an ant algorithm.

- AA: The ant algorithm of Socha *et al.* [196]. They developed the first ACO algorithm with the help of a construction graph and a pheromone model appropriate for the UCTP.

- FA: The fuzzy algorithm of Asmuni *et al.* [29]. In their paper, they focused on the issue of ordering events by simultaneously considering three different heuristics using fuzzy methods.

- GBHH: The graph-based hyper-heuristic of Burke *et al.* [51]. They employed TS with graph-based hyper-heuristics on the UCTP and examination timetabling problem.

From Table 4.3, it can be seen that our proposed MA is better than the fuzzy algorithm [29] and the graph based approach [51] on 9 out of the 10 small and medium problem instances (except on M5). Our algorithm also obtained better results than VNS [13] and EA [181] on all of the medium problem instances and tied on some or all of the small problem instances. It also gives better results than LS [196] on 9 of the 10 problem instances and is better than the ant algorithm [196] on 7 of the data set (with one tie on S5). When comparing with the result of HEA [15] and RIIA [14], it is interesting to note that our approach is better on 3 of the same medium problem instances (except on M2 and M5) and ties on all small problems. Finally, the results of our approach are better than the tabu-based hyper-heuristic search [47] on most of the problem instances.

The proposed MA did not achieve a feasible result on the large instance within the running time of 9000 seconds within 50 runs. On this large instance, other algorithms from the literature also failed to give a feasible result with the exception of [51], [15], [196], and [29]. This result indicates that the neighbourhood structures may need further improvement to give feasible results for the large instance.

To summarise, the performance of our MA was tested on the benchmark problem instances [4]. When compared with other published works, it can be seen that our proposed MA is capable of producing some of the best results.

4.4 Chapter Summary

This chapter presented a MA for solving the UCTP. The MA combines the SSGA with two LS techniques, LS1 and LS2. With LS1 only, the MA does not perform well in the experiments as mentioned in [181]. However, we have enhanced the power of the MA by introducing a second LS method. Based on the experimental results, it is clear that, with the help of the powerful LS methods, the proposed MA can obtain high quality solutions that satisfy different kinds of timetabling constraints. The proposed MA is capable of finding a near optimal solution for the test problem instances. These results also show that by integrating appropriate neighbourhood moves, GAs can get the best solutions for the UCTP.

In this chapter, we can see that for hard problem instances, the proposed MA does not work well. We are even unable to get feasible solutions. Hence, we need some powerful search algorithms that cope across a much wider range of problem instances. Finding appropriate advanced genetic operators, heuristics, and evaluation routines will be the main aim of the next chapter.

Chapter 5

Genetic Algorithms with Guided and Local Search for UCTPs

5.1 Introduction

In the previous chapter, one of the main observations made in our experimental study was that the performance of the memetic algorithm (MA) with the local search (LS) strategies is generally good for the university course timetabling problem (UCTP) on most problem instances, but is not satisfactory on the large problem instance. The MA cannot even produce feasible solutions for the large problem instance.

We hypothesise that, although LS1 and LS2 try to help a genetic algorithm (GA) escape from local optima, if the individuals are not of good quality, then the GA can quickly get stuck in them. If there is a good population, then the chance of creating a feasible and optimal solution can be increased. To test our hypothesis, we intend to create children through some methods other than crossover operators. One reason for this is that, a good deal of research has been carried out on crossover operators and different crossover operators are useful for different problem constraints. However, we want to develop some method that is useful to create children for UCTPs of different characteristics. To this aim, we propose a guided search (GS) strategy to be used within GAs for solving the UCTP.

In this chapter, we first present the proposed GS strategy. Then, we present several GAs with the GS strategy and the LS strategies described in Chapter 4 for the UCTP. The GS strategy can be used to create offspring of good quality into the population based on a data structure that stores information extracted from previous good individuals. The LS strategies can be used to improve the quality of individuals. Finally, based on a set of benchmark UCTP instances, we experimentally investigate the proposed GAs in comparison with a set of state-of-the-art methods from the literature for the UCTP.

5.2 The Guided Search (GS) Strategy

One of the important concepts of GAs is the notion of population. Unlike traditional search methods, GAs rely on a population of candidate solutions [188]. In the GS strategy, we use an extra memory or data structure, denoted MEM, to store and re-use useful information. This data structure is constructed from the best individuals taken from the population and hence stores useful information that can be used to guide the generation of good offspring into the next populations.

The main advantage of this data structure is that it maintains partial information of good solutions which otherwise may be lost in the selection process. In the following sub-sections, we describe in detail the key components of the GS strategy, including the MEM data structure and its construction and the generation of offspring via the GS strategy, respectively.

5.2.1 The MEM Data Structure

There is a number of research papers in the literature on using extra data structure or memory to store useful information in order to enhance the performance of GAs and other meta-heuristic methods for optimisation and search [11, 12, 148]. Rosin and Belew [183] introduced a *hall of fame* to store elitists in time. In this process, they store the best individual of each generation into the hall of fame. A new individual of the current generation replaces the worst or oldest individual. Louis and McDonnall [149] also incorporated problem specific knowledge into GAs. They used a case-based memory of the past problem for better performance on a set of similar problems. In the GS strategy, we also use a data structure (memory) MEM to guide the generation of offspring. This strategy introduces some parts of the best individuals into the next generation (to maintain the diversity in the population) rather than injecting whole individuals. This MEM is used to provide further

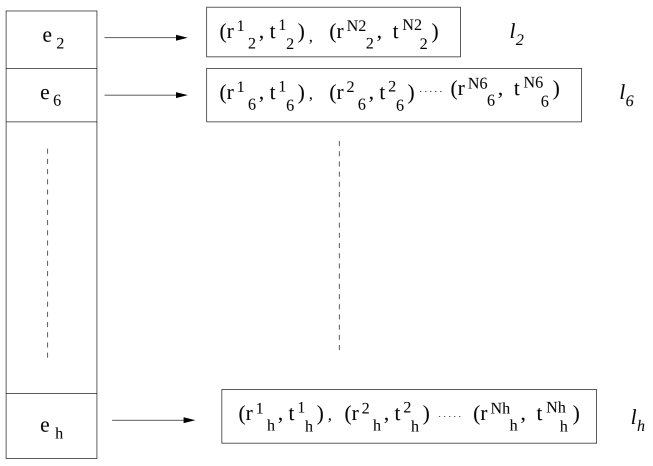
e_2
$(r^1_2, t^1_2), (r^{N2}_2, t^{N2}_2)$
l_2
e_6
$(r^1_6, t^1_6), (r^2_6, t^2_6) \cdots (r^{N6}_6, t^{N6}_6)$
l_6
e_h
$(r^1_h, t^1_h), (r^2_h, t^2_h) \cdots (r^{Nh}_h, t^{Nh}_h)$
l_h

Figure 5.1: Illustration of the data structure MEM.

direction to exploration and exploitation in the search space. It aims to increase the quality of a child solution by re-introducing part of best individuals from previous generations.

Figure 5.1 shows the details of the MEM data structure, which is a two-level list. The first level is a list of events while the second is a list l_i of room and time slot pairs corresponding to each event e_i in the first level list. In Figure 5.1, N_i represents the total number of pairs in the second level list l_i.

The MEM data structure is regularly reconstructed, e.g., every τ generations. Algorithm 11 shows the outline for constructing the MEM. When the MEM is due to be reconstructed, we first select $\alpha \times N$ best individuals from the population P to form a set Q, where N denotes the population size. After that, for each individual $I_j \in Q$, each event is checked by its penalty value, i.e., the hard and soft constraint violations associated with this event. If an event has a zero penalty value, then we store the information corresponding to this event into the MEM.

Algorithm 11 *ConstructMEM*()

```
1:  input : The whole population $P$
2:  sort the population $P$ according to the fitness of individuals
3:  $Q \leftarrow$ select the best $\alpha \times N$ individuals in $P$
4:  for each individual $I_j$ in $Q$ do
5:      for each event $e_i$ in $I_j$ do
6:          calculate the penalty value of event $e_i$ from $I_j$
7:          if $e_i$ is feasible (i.e., $e_i$ has a zero penalty) then
8:              add the room and time slot pair $(r_i, t_i)$ assigned to $e_i$ into the list $l_i$
9:          end if
10:     end for
11: end for
12: output : The data structure $MEM$
```

For example, if the event e_2 of an individual $I_j \in Q$ is assigned room 2 at time slot 13 and has a zero penalty value, then we add the pair $(2, 13)$ into the list l_2. Similarly, the events of the next individual $I_{j+1} \in Q$ are also checked by their penalty values. If the event e_2 in I_{j+1} has a zero penalty, then we add the pair of room and time slot assigned to e_2 in I_{j+1} into the existing list l_2. If for an event e_i, there is no list l_i existing yet, then the list l_i is added into the MEM data structure.

A similar process is carried out for the selected Q individuals and finally the MEM data structure stores pairs of room and time slot corresponding to those events with zero penalty of the best individuals of the current population. This MEM data structure is then used to guide the generation of offspring for the next τ generations. We update MEM every τ generations instead of every generation in order to make a balance between the solution quality and the computational time cost of a GA that uses this GS strategy.

Algorithm 12 *GuidedSearch*()

```
1: input : The $MEM$ data structure
2: $E_s$ := randomly select $\beta * n$ events
3: for each event $e_i$ in $E_s$ do
4:     randomly select a pair of room and time slot from the list $l_i$ in $MEM$
5:     assign the selected pair to event $e_i$ for the child
6: end for
7: for each remaining event $e_i$ not in $E_s$ do
8:     assign a random time slot and room to event $e_i$
9: end for
10: output : A new child generated using $MEM$
```

5.2.2 Generating a Child by the GS Strategy

Using the GS strategy, a child can be created through the MEM data structure, by calling *GuidedSearch*() in Algorithm 12. The procedure is described as follows. We first select a set E_s of $\beta * n$ random events to be generated from MEM. Here, β is a percentage value and n is the total number of events. After that, for each event e_i in E_s, we randomly select a pair of (r_i^j, t_i^j), $j = 1, \cdots, N_i$, from the list l_i that corresponds to the event e_i, and assign the selected pair to e_i for the child. If there is an event e_i in E_s but there is no list l_i in the MEM, then we randomly assign a room and time slot from possible rooms and time slots to e_i for the child. This process is carried out for all the events in E_s. For those remaining events that are not present in E_s, they are assigned random rooms and time slots.

Algorithm 13 Steady State Memetic Algorithm (SSMA)

```
1: randomly initialise a population of solutions
2: evaluate the individuals in the population
3: apply the LS strategy LS1 to each individual of the population
4: while the termination condition is not reached do
5:    select two parents through a selection scheme
6:    crossover the parents to create a child with a probability $P_c$
7:    apply mutation to the child with a probability $P_m$
8:    apply LS1 to the child
9:    replace the worst member of the population by the child
10: end while
```

5.3 GAs with GS and LS Strategies for the UCTP

In this section, we present several GAs that are integrated with the GS and LS strategies for the UCTP. We first introduce the basic framework of these GAs and then describe several GA variants that are investigated in this chapter for the UCTP.

5.3.1 The Basic Framework of Investigated GAs

The basic framework of GAs investigated in this chapter is based on a steady-state memetic algorithm, denoted as *SSMA*. The SSMA combines the SSGA, which has been described in Algorithm 6 in Chapter 4, with the LS strategy LS1, which has been described in Algorithm 8 in Chapter 4. The pseudo-code of SSMA is shown in Algorithm 13, where the italicised lines show the changes that have been made from the SSGA in Algorithm 6. We are not using LS2 in the SSMA because we want to see the effect of LS2 separately as well as with guided operator.

With this basic framework, GAs start from an initial population of solutions that are randomly generated (i.e., events are randomly assigned to rooms and time slots for each solution). It is reasonable to expect that the quality of the initial solutions would affect the quality of the final solutions. However, we start from random initial solutions. Then, in each generation, one individual is generated as follows.

First, two parents are selected using the binary tournament selection. Then, crossover is carried out with a probability P_c to generate one child via exchanging the time slots between the two parents and allocating rooms to events in each non-empty time slot using the matching algorithm. After crossover, the child undergoes the mutation operation with a probability P_m. The mutation operator first randomly selects one of the three neighbourhood structures $N1$, $N2$, and $N3$ (as described in Section 4.2.1 in Chapter 4) and then makes a move within the selected neighbourhood structure. After mutation, LS is performed on the child. Finally, the newly-generated child is used to replace the worst individual from the current population.

5.3.2 The Guided Search Genetic Algorithm (GSGA)

The pseudo-code of GSGA for the UCTP is shown in Algorithm 14. In GSGA, we first initialise the population by randomly creating each individual via assigning a random time slot for each event according to a uniform distribution and applying the matching algorithm to allocate a room for the event. Then, the LS1 method,

Algorithm 14 Guided Search Genetic Algorithm (GSGA)

```
1: input : A problem instance I
2: randomly initialise a population of solutions
3: evaluate the individuals in the population
4: apply the LS strategy LS1 to each individual of the population
5: set the generation counter $g := 0$
6: while the termination condition is not reached do
7:     if ($g \bmod \tau$) == 0 then
8:         apply $ConstructMEM()$ to construct $MEM$
9:     end if
10:    create a child using $GuidedSearch()$ or $Crossover()$ with a probability $\gamma$
11:    apply mutation to the child with a probability $P_m$
12:    evaluate the child solution
13:    apply LS1 to the child
14:    replace the worst member of the population by the child
15:    $g := g + 1$
16: end while
17: output : The best solution $s_{best}$ achieved for I
```

as described in Algorithm 8 in Chapter 4, is applied to each member of the initial population, using the three neighbourhood structures.

After the initialisation of the population, a data structure MEM (described in Section 5.2.1) is constructed, which stores a list of room and time slot pairs (r, t) for each event that has a zero penalty (i.e., no hard and soft violation at this event) of individuals selected from the population. After that, the MEM can be used to guide the generation of offspring for the following generations. The MEM data structure is re-constructed every τ generations. In each generation of GSGA, one child is first generated either by the GS strategy or the crossover operator, depending on a probability γ. That is, when a new child is to be generated, a random number $\rho \in [0.0, 1.0]$ is first generated. If $\rho < \gamma$, Algorithm 12, i.e., *GuidedSearch*(), will be used to generate the new child; otherwise, a crossover operation, by calling

$Crossover()$ (see Algorithm 10 in Chapter 4), is used to generate the new child.

After that, the child will be mutated by a mutation operator with a probability p_m, followed by the LS1 strategy. Finally, the worst member in the population is replaced by the newly generated child individual. This iteration continues until one termination condition is reached, e.g., a present time limit t_{max} is reached.

5.3.3 Extended Guided Search Genetic Algorithm (EGSGA)

We also want to test the effect of the LS strategy LS2 that was described in Algorithm 9 in Chapter 4 on the performance of GSGA for the UCTP. To this purpose, we propose an extended version of GSGA, denoted EGSGA in this chapter, in which LS2 is added into GSGA, for the UCTP. In EGSGA, whenever LS1 is applied to an individual, LS2 is applied to that individual immediately after LS1 is finished.

5.3.4 GA with Both LS Strategies (GALS)

In order to investigate the effect of the GS strategy in GSGA and EGSGA, we also investigate a steady-state GA with the two LS strategies LS1 and LS2 and without the GS strategy. This GA is denoted *GALS* in this chapter. In Chapter 4, we also use both LS strategies in MA. But, the difference between GALS and MA is that the crossover probability P_c is 0.8 in MA and 1.0 in GALS. This probability tells us the behaviour of the algorithm if we do not use the GS strategy.

In each generation of GALS, one child is first generated through selection, crossover (Algorithm 10 in Chapter 4), and mutation. Then, the two LS strategies, LS1 and LS2, are applied in order to the child. In the end, the worst individual in the population is replaced by this new child. That is, GALS is an SSMA with the integration of LS2. This will also allow us to check the effect of LS2 on the performance of SSMA for the UCTP. GALS differs from EGSGA with $\gamma = 0.0$ in that in GALS, crossover is carried out with a probability P_c, while in EGSGA with $\gamma = 0.0$, crossover is carried out with a probability 1.0. In other words, EGSGA with $\gamma = 0.0$ is equivalent to GALS with $P_c = 1.0$.

5.4 Experimental Study

In this section, we experimentally investigate the performance of the proposed methods GALS, GSGA and EGSGA in addition with SSMA and a TS method proposed by Rossi et al. [181] for the UCTP.

All algorithms were coded in GNU C++ under version 4.1 and run on a 3.20 GHz PC. To test the algorithms, We use a set of benchmark UCTP instances which were described in Chapter 3. Table 3.1 in Chapter 3 presents the data of these UCTP instances in three different groups: five small instances, five medium instances, and one large instance. As mentioned before, the basic framework of all GAs we investigate (SSMA, GALS, GSGA and EGSGA) is a steady-state MA. The basic parameters for GAs were set as follows: the population size N was set to 50, the mutation probability P_m was set to 0.5, and the crossover probability P_c was set to 0.8 in SSMA and 1.0 in GALS. The value of the constant C in the objective function was set to 10^6.

Two sets of experiments were carried out in this study. The first set of experiments were devoted to analysing the sensitivity of parameters for the performance of GSGA for the UCTP. The second set compared the performance of investigated GAs with or without the GS strategy on the test UCTP instances. For both sets of experiments, there were 50 runs of each algorithm on each problem instance. Following other works [181, 196], for each run of an algorithm on a problem, the maximum run time t_{max} was set to 90 seconds for small instances, 900 seconds for medium instances, and 9000 seconds for the large instance based on the fact that larger UCTP instances are more complex and have more conflicting constraints and a larger search space as compared to smaller UCTP instances and therefore require more processing time.

In the end, we compared our experimental results of EGSGA with a set of current state-of-the-art methods from the literature on the above set of test UCTP instances and another set of UCTP instances taken from the ITC-2002 [1].

5.4.1 The Sensitivity of Key Parameters of GSGA

The performance of the GS strategy depends on the parameters and operators used. Through our previous work [129], we found that α, β, γ, and τ are key parameters

Table 5.1: Parameter settings in GSGA

Parameter	Settings					
α	0.2	0.4	0.6	0.8		
β	0.1	0.3	0.5	0.7	0.9	
γ	0.0	0.2	0.4	0.6	0.8	1.0
τ	20	40	60	80	100	

that can greatly affect the performance of GSGA for the UCTP, where α is the percentage of best individuals selected from the current population for creating the data structure MEM, β is the percentage value of the total number of events that are used to create a child through the data structure MEM, γ is the probability that indicates whether a child is created through MEM or crossover, and τ decides the frequency of updating MEM (i.e., MEM is updated every τ generations). Hence, we test our algorithm GSGA with different settings of these parameters.

Table 5.1 shows different parameters and their settings that were tested in our experiments. In order to find out which parameter settings have a greater effect on the performance of GSGA, we ran GSGA 50 times for all parameter combinations in Table 5.1. In total, 600 combinations of parameter settings were observed. Here, we only present some of those that seem to have a great effect on the performance of GSGA. We chose two α values 0.2 and 0.6, three β values 0.1, 0.3, and 0.7, three γ values 0.2, 0.4, and 0.8, and two τ values 20 and 60. The experimental results with respect to the average objective value of GSGA with these selected parameter settings are presented in Table 5.2. Table 5.2 shows that different parameter settings give different results (we explain the reasons later), because the performance of the proposed approach depends on these key parameters.

TABLE 5.2: Average best value of 50 runs of GSGA with different parameter settings on the test problem instances

α	β	γ	τ	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L
0.2	0.1	0.2	20	6	9	5	9	4	221	146	211	178	126	726
0.2	0.1	0.2	60	11	10	11	21	5	247	148	217	178	142	803
0.2	0.1	0.4	20	2	3	6	14	3	271	101	192	143	112	716
0.2	0.1	0.4	60	6	10	4	11	3	225	133	201	166	114	806
0.2	0.1	0.8	20	0	2	4	3	1	192	110	203	98	107	645
0.2	0.1	0.8	60	2	3	5	4	0	195	125	221	101	115	669
0.2	0.3	0.2	20	7	6	8	4	4	178	126	183	116	108	722
0.2	0.3	0.2	60	11	11	10	9	4	190	130	187	163	126	812
0.2	0.3	0.4	20	2	6	4	3	4	176	119	178	153	113	647
0.2	0.3	0.4	60	6	4	8	4	5	182	124	185	161	118	654
0.2	0.3	0.8	20	0	0	1	2	0	152	108	121	101	96	637
0.2	0.3	0.8	60	1	2	1	1	2	161	109	155	121	111	675
0.2	0.7	0.2	20	8	6	8	4	5	181	172	162	321	139	856
0.2	0.7	0.2	60	12	11	8	4	5	192	210	175	343	142	881
0.2	0.7	0.4	20	14	7	13	6	4	183	161	154	219	128	778
0.2	0.7	0.4	60	9	2	9	5	5	190	184	162	232	132	811
0.2	0.7	0.8	20	2	0	1	3	5	170	139	218	123	116	664
0.2	0.7	0.8	60	8	5	6	5	6	191	142	225	135	118	679
0.6	0.1	0.2	20	11	8	21	13	7	322	152	223	181	128	834
0.6	0.1	0.2	60	13	15	24	13	8	340	156	231	182	162	888
0.6	0.1	0.4	20	22	17	12	14	3	271	101	192	143	112	716
0.6	0.1	0.4	60	12	10	6	11	3	228	142	211	164	117	764
0.6	0.1	0.8	20	5	3	4	6	5	196	111	220	101	112	652
0.6	0.1	0.8	60	11	12	8	6	3	201	122	247	121	126	689
0.6	0.3	0.2	20	11	10	9	8	5	181	127	198	132	128	722
0.6	0.3	0.2	60	11	11	10	9	4	190	130	200	163	126	812
0.6	0.3	0.4	20	7	7	11	9	5	179	125	181	173	153	647
0.6	0.3	0.4	60	9	11	17	10	5	184	129	185	219	164	712
0.6	0.3	0.8	20	2	4	3	5	0	156	113	132	116	103	645
0.6	0.3	0.8	60	5	3	4	6	3	159	128	161	124	123	663
0.6	0.7	0.2	20	13	21	12	9	7	209	180	167	245	184	912
0.6	0.7	0.2	60	15	21	21	22	9	234	194	197	289	190	934
0.6	0.7	0.4	20	14	9	11	8	7	189	175	161	189	134	781
0.6	0.7	0.4	60	14	11	13	9	10	211	191	186	194	143	792
0.6	0.7	0.8	20	5	4	6	4	5	163	151	192	126	112	670
0.6	0.7	0.8	60	6	4	8	6	10	178	172	212	129	128	705

In order to help understand the experimental results of different parameter settings, Figure 5.2 shows the dynamic performance regarding the average objective value against the number of evaluations over 50 runs of GSGA with one parameter changing while the other parameters are kept constant on different UCTP instances. Figure 5.2(a) shows the effect of changing α on M1 with $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$. Figure 5.2(b) shows the effect of changing β on S2 with $\alpha = 0.2$, $\gamma = 0.8$, and $\tau = 20$. Figure 5.2(c) shows the effect of changing γ on S5 with $\alpha = 0.2$, $\beta = 0.3$, and $\tau = 20$. Figure 5.2(d) shows the effect of changing τ on S4 with $\alpha = 0.2$, $\beta = 0.3$, and $\gamma = 0.8$. From Table 5.2 and Figure 5.2, several results can be observed and are analysed below.

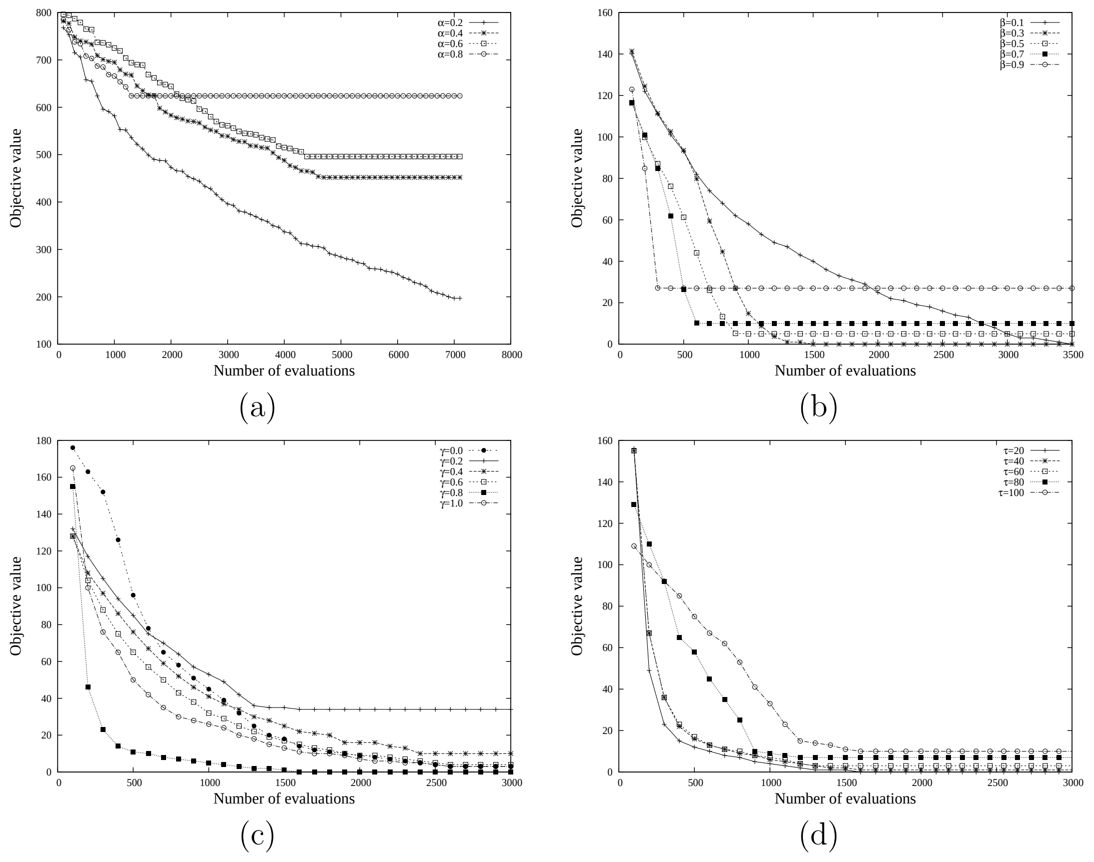

FIGURE 5.2: Comparison on the effect of parameters on the performance of GSGA on different problem instances: (a) M1 with $\beta = 0.3$, $\gamma = 0.8$ and $\tau = 20$, (b) S2 with $\alpha = 0.2$, $\gamma = 0.8$ and $\tau = 20$, (c) S5 with $\alpha = 0.2$, $\beta = 0.3$ and $\tau = 20$, and (d) S4 with $\alpha = 0.2$, $\beta = 0.3$ and $\gamma = 0.8$.

First, the parameter α has a significant effect on the performance of GSGA for the UCTP. The performance of GSGA drops when the value of α increases from 0.2 to 0.8, see Figure 5.2(a) for reference. This occurs because when a small part of the population is chosen to create the MEM data structure, MEM can provide a strong guidance during the genetic operations and help GSGA exploit the area of the search space that corresponds to the best individuals of the population sufficiently.

This sufficient exploitation can ensure that GSGA quickly achieves better solutions. In contrast, when a large part of the population is taken to create or update MEM, then MEM loses its effect of guiding GSGA to exploit promising areas of the search space. In other words, when α is set to large values, GSGA tends to be SSMA and hence the performance will drop or be weak. This can be observed in Figure 5.2(a): when the value of α increases, the best solution of GSGA cannot improve after a certain number of evaluations, e.g., after about 4000 evaluations when $\alpha = 0.6$ and after about 1500 evaluations when $\alpha = 0.8$.

Secondly, regarding the effect of β, it can be seen that setting this parameter to a very small or very large value affects the penalty value. This result can be observed from the interesting behaviour of GSGA on the S2 problem instance with $\alpha = 0.2$, $\gamma = 0.8$, $\tau = 20$, and different β values in Figure 5.2(b). From Figure 5.2(b), it can be seen that when the value of β increases from 0.1 to 0.3, the performance of GSGA improves due to the enhanced effect of the MEM data structure. However, when the value of β is further raised, the performance of GSGA drops. This occurs because if a large portion of individuals is created through MEM, e.g., when $\beta = 0.9$, the chance of creating a similar child may be increased every generation, and after a few generations, GSGA may be trapped in a sub-optimal state and hence cannot obtain the optimal solution. From Figure 5.2(b), it can be seen that setting the value of β to 0.7 or 0.9 leads to an earlier stagnation in the performance of GSGA in the solving process.

Third, regarding the effect of γ on the performance of GSGA, from Table 5.2, it can easily be seen that increasing the value of γ results in near-optimal solutions. The reason lies in the fact that a small value of γ leads to the proposed GSGA algorithm acting as the conventional SSMA. Figure 5.2(c) shows the behaviour of GSGA on the S5 problem instance with $\alpha = 0.2$, $\beta = 0.3$, $\tau = 20$, and different values of γ. It is quite obvious that a large value of γ, e.g., $\gamma = 0.8$, leads to an optimal solution quickly. The effect of γ also shows the importance of the MEM data structure. From Figure 5.2(c), it can also be seen that when $\gamma = 1.0$, the performance of GSGA drops in comparison with when $\gamma = 0.8$. This result shows that the use of crossover helps improve the performance of GSGA for the UCTP.

Fourth, regarding the effect of τ, it can be seen from Table 5.2 and Figure 5.2(d) that setting τ to 20 gives a better objective value than setting τ to other values (i.e., 40, 60, 80, and 100). That is, updating the MEM data structure every small number of generations gives a better performance of GSGA. This is because when MEM is updated more frequently, the information extracted from the best individuals of the population can be more timely used to guide the generation of offspring and hence gives a greater chance to create better individuals. The difference is significant when τ is set to 20 over 100. The effect of τ also shows the importance of the MEM data structure for the performance of GSGA.

Based on the above parameter analyses, in the following experiments, we set the parameters for GSGA and EGSGA as follows: $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$.

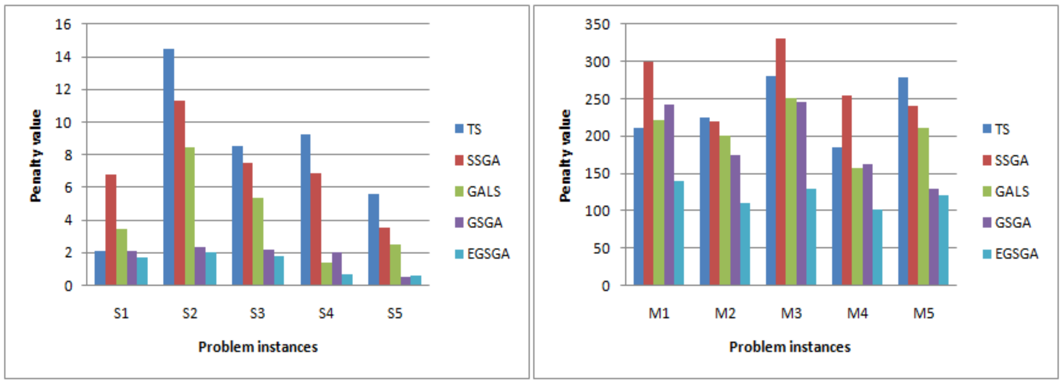

FIGURE 5.3: Comparison of EGSGA with other algorithms regarding the average performance on (a) small instances and (b) medium instances.

5.4.2 Comparative Experiments

This set of experiments compares the performance of EGSGA with other implemented algorithms (TS, SSMA, GALS, and GSGA). The parameter settings identified by the previous experiments were used for all results presented in this section. The same set of parameters was used for GAs in order to have a fair comparison of the performance of algorithms.

Figure 5.3 presents the comparison of EGSGA with other algorithms with respect to the average performance over 50 runs on small and medium UCTP instances. Table 5.3 compares all algorithms in terms of the best, average, standard deviation, and worst penalty value over the 50 runs on the 11 problem instances, where "–" means that no feasible solution was found by the corresponding method.

From Figure 5.3 and Table 5.3, it can be seen that EGSGA produces a lower average and standard deviation of the objective value on most of the UCTP instances and that the worst objective values produced by EGSGA are by far the best of all the

Table 5.3: Comparison of algorithms on different problem instances

UCTP	Alg	Best	Ave	Std	Worst
S1	TS	0	2.06	3.35	9
	SSMA	0	6.75	8.27	15
	GALS	0	3.43	5.12	15
	GSGA	0	2.11	3.33	9
	EGSGA	0	1.71	2.42	4
S2	TS	1	14.5	9.6	27
	SSMA	3	11.3	8.66	32
	GALS	0	8.43	5.21	19
	GSGA	0	2.32	5.59	16
	EGSGA	0	2.01	3.71	11
S3	TS	0	8.53	7.56	22
	SSMA	0	7.26	5.40	26
	GALS	0	5.32	6.60	19
	GSGA	0	2.2	3.21	11
	EGSGA	0	1.8	1.53	2
S4	TS	2	9.26	7.34	22
	SSMA	0	6.81	7.01	24
	GALS	0	1.24	2.41	7
	GSGA	0	1.84	2.20	11
	EGSGA	0	0.63	1.89	5
S5	TS	0	5.58	6.42	16
	SSMA	0	3.49	7.00	19
	GALS	0	2.53	2.89	7
	GSGA	0	0.51	1.86	5
	EGSGA	0	0.55	0.82	3
M1	TS	211	220.5	27.64	267
	SSMA	280	302	36.117	321
	GALS	227	229.5	10.65	256
	GSGA	240	247	9.02	260
	EGSGA	139	142	6.384	202
M2	TS	185	230.5	21.59	273
	SSMA	188	225.2	20.01	290
	GALS	180	203	20.62	256
	GSGA	162	172.4	14.49	209
	EGSGA	92	112	10.96	134
M3	TS	280	286	8.170	301
	SSMA	249	330	23.45	389
	GALS	235	249.2	10.21	300
	GSGA	242	247	6.021	290
	EGSGA	122	128.4	4.832	160
M4	TS	176	187	18.38	241
	SSMA	247	256	21.86	321
	GALS	142	160.1	16.90	203
	GSGA	158	162.7	17.01	212
	EGSGA	98	100.2	5.451	112
M5	TS	255	276	20.45	365
	SSMA	232	245.6	15.32	343
	GALS	200	212.2	24.77	298
	GSGA	124	128.5	23.67	200
	EGSGA	116	121.3	13.29	151
L	TS	–	–	–	–
	SSMA	–	–	–	–
	GALS	–	–	–	–
	GSGA	801	858.2	40.35	921
	EGSGA	615	648.5	19.11	670

algorithms. This is a really good result, and means that EGSGA is much more reliable than the other algorithms. EGSGA produces good solutions due to the usage of the GS and LS strategies. As mentioned earlier, this is due to the fact that we assign to an event a pair of room and time slots extracted from one of the best individuals of a previous population. This means that the pair satisfies different constraints that are suitable to that event. The LS technique further helps EGSGA find the local optimum around an individual. By doing so, we increase the chance of getting better and better solutions during the solving process.

Figure 5.4 shows the dynamic behaviour of algorithms against the number of evaluations on Tsome problem instances, where the x-axis represents the number of evaluations and the y-axis represents the average objective value over 50 runs. Figures 5.4(a) and 5.4(b) show the performance of different algorithms on small UCTP instances S1 and S3, respectively. Figure 5.4(c) represents the performance of algorithms on the medium UCTP instance M5, where the y-axis shows the objective value expressed in the log scale. Figure 5.4(d) shows the performance of GSGA and EGSGA on the large UCTP instance, where the y-axis is also expressed in the log scale.

From Figure 5.4, it can be seen that on the small instances, SSMA and TS reach near-optimal solutions as the number of evaluations increases and that GALS and GSGA perform similarly to each other. We notice that when the two strategies in EGSGA are used independently, GALS and GSGA are not significantly better than each other, but when they are combined in EGSGA, we see a great improvement in performance of EGSGA. The penalty value of EGSGA is reduced at the beginning of the search and gives the optimal solution between 1000 and 1500 evaluations,

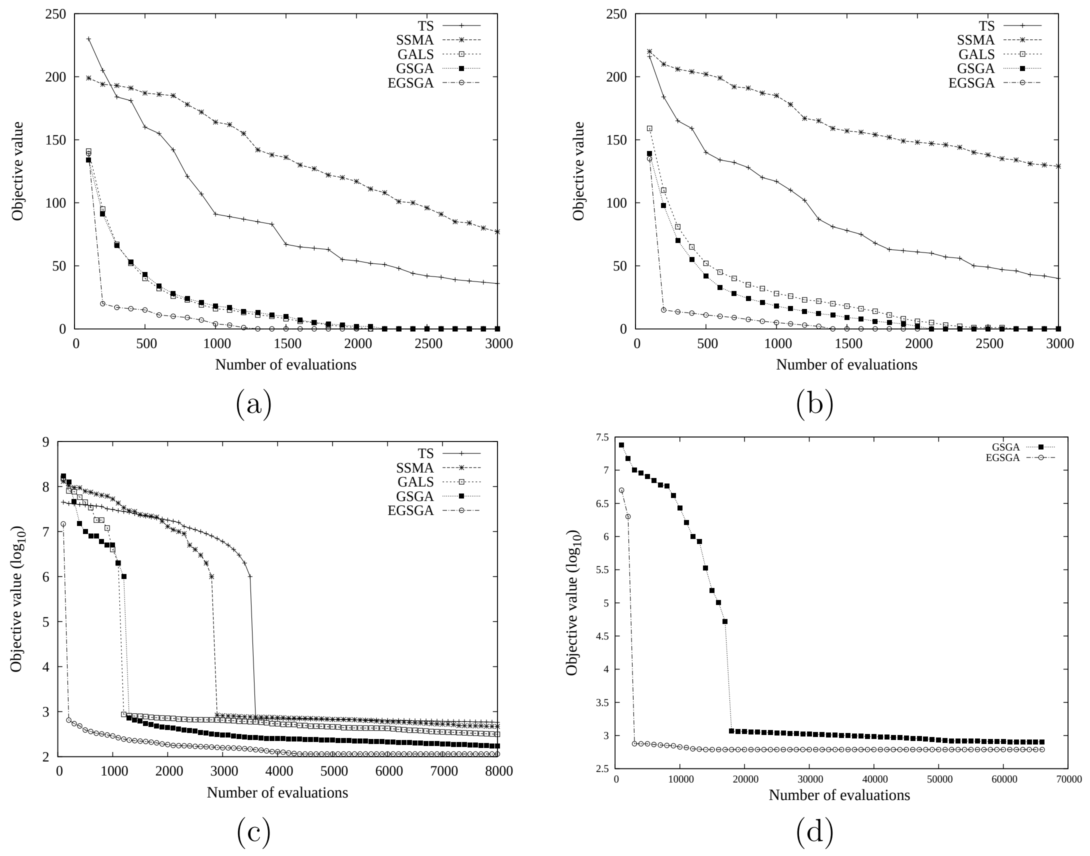

Figure 5.4: Dynamic performance of algorithms on different problem instances: (a) S1, (b) S3, (c) M5, and (d) L.

while GSGA and GALS give near-optimal solution after 2000 evaluations. On the M5 medium problem instance, a considerable fall in the penalty value can be noticed in the performance of EGSGA. EGSGA quickly generates a feasible solution after a few evaluations and makes positive movement towards the near-optimal solution by exploring the search space as the number of evaluations increases. On the other hand, GALS and GSGA achieve a feasible solution over 1000 evaluations. It can be observed from Figure 5.4(d) that the search speed of EGSGA on the large problem instance is better than that of GSGA. We anticipate this result because partial solutions from good individuals provide more efficient solutions when combined with

Table 5.4: The t-test values of comparing algorithms on different problem instances

UCTP	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L
SSMA − TS	$s-$	$s+$	$+$	$s+$	$s+$	$s-$	$+$	$s-$	$s-$	$s+$	In
SSMA − GALS	$s-$	$s-$	$s-$	$s-$	$-$	$s-$	$-$	$s-$	$s-$	$s-$	In
SSMA − GSGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	In
SSMA − EGSGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	In
GALS − GSGA	$-$	$s-$	$s-$	$s+$	$s-$	$s+$	$s-$	$+$	$+$	$s-$	In
EGSGA − GALS	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	In
EGSGA − GSGA	$+$	$+$	$s+$	$s+$	$+$	$s+$	$+$	$s+$	$s+$	$+$	$s+$
TS − GALS	$-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	In
TS − GSGA	$-$	$s-$	$s-$	$s-$	$s-$	$s+$	$-$	$s-$	$-$	$s-$	In
TS − EGSGA	$-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	In

the LS strategy LS2.

The t-test results of statistically comparing investigated algorithms are shown in Table 5.4 (details of the t-test are presented in Appendix A). The average performance values over 50 runs of all algorithms on all problem instances are used for the t-test (the data used for the t-test are presented in Tables A.3, A.4, A.5, and A.6, respectively). The t-test statistical comparison was carried out with 98 degrees of freedom at a 0.05 level of significance and the results are shown in Table 5.4, where the t-test results of comparing two algorithms are shown as "$s+$", "$s-$", "$+$", or "$-$" when the first algorithm is significantly better than, significantly worse than, insignificantly better than but better than (the mean value difference is not statistically significant), or insignificantly worse but worse than (the mean value difference is not statistically significant) the second algorithm, respectively. In Table 5.4, "In" means that one or both of the algorithms being compared failed to find a feasible solution for the corresponding problem instance.

From Table 5.4, it can be seen that the performance of EGSGA is significantly better than that of all other algorithms on all medium and large problem instances and that the performance of EGSGA is significantly better than that of GSGA on most small, medium, and large problem instances. These results show that the integration of proper LS strategies with the GS strategy can greatly improves the performance of GAs for the UCTP.

5.4.3 Comparison with Algorithms from the Literature

In this section, we compare the experimental results of our algorithms with the available results of other algorithms described in Section 3.2. Here we briefly describe those approaches:

- EGSGA: The EGSGA proposed in this chapter. The results reported here were derived from 50 runs with each run lasting for 90 seconds for small UCTP instances, 900 seconds for medium instances, and 9000 seconds for the large instance.
- RIIA: The randomized iterative improvement method of Abdullah *et al.* [14]. They presented a composite neighbourhood structure with a randomized iterative improvement algorithm. The results were reported out of 5 runs with each run lasting for 200,000 evaluations.
- HEA: The hybrid evolutionary approach of Abdullah *et al.* [15] consists of an EA that uses a light mutation operator followed by a randomized iterative improvement algorithm. The results were reported out of 5 runs with 200,000 evaluations per run.

- GBHH: The graph-based hyper-heuristic of Burke *et al.* [51]. They employed TS with graph-based hyper-heuristics for the UCTP and examination timetabling problems. The results were reported out of 5 runs with 12000 evaluations per run for the small problem instances, 1200 evaluations per run for the medium problem instances, and 5400 evaluations per run for the large problem instance, respectively.

- THHS: Burke *et al.* [47] introduced a TS hyper-heuristics for the UCTP, where a set of low level heuristics compete with each other. This approach was tested on the course timetabling and nurse rostering problems. The results were reported out of 5 runs with 12000 evaluations per run for the small problem instances, 1200 evaluations per run for the medium problem instances, and 5400 evaluations per run for the large problem instance, respectively.

- LS: The LS method of Socha *et al.* [196]. They used a random restart LS method for the UCTP and compared it with an ant algorithm. The results were reported out of 50 runs with each run lasting for 90 seconds for small UCTP instances, 40 runs with each run lasting for 900 seconds for medium instances, and 10 runs with each run lasting for 9000 seconds for the large instance.

- GA: The GA of Rossi-Doria *et al.* [181]. They used a LS method with the GA to solve the UCTP and also compared several meta-heuristics methods on the UCTP. The results were reported out of 50 runs with each run lasting for 90 seconds for the small problem instances, 900 seconds for the medium problem instances, and 9000 seconds for the large problem instance, respectively.

- AA: The ant algorithm used of Socha *et al.* [196]. They developed an ant colony optimisation algorithm with the help of a construction graph and a pheromone model appropriate for the UCTP. The results were reported out of 50 runs with each run lasting for 90 seconds for small UCTP instances, 40 runs with each run lasting for 900 seconds for medium instances, and 10 runs with each run lasting for 9000 seconds for the large instance.

- FA: The fuzzy algorithm of Asmuni *et al.* [29]. In [29], Asmuni *et al.* focused on the issue of ordering events by simultaneously considering three different heuristics using fuzzy methods. The results were reported from one run for each problem instance but the stopping criterion for each run on a problem instance was not clearly mentioned in the paper.

All the above compared algorithms have different conditions under which their results were reported. However, these approaches have been frequently used by other researchers to compare the performance of their algorithms. Strictly speaking, it is not entirely fair to use the results reported in the literature since the conditions involved are not the same for all algorithms. However, the results reported can give us a rough understanding of how good or bad an algorithm is in comparison with existing methods. Hence, we also follow the trend in the literature and roughly compare our algorithm EGSGA with the above state-of-the-art methods using the reported results. Table 5.5 gives the comparison results, where the term "%In" represents the percentage of runs that failed to obtain a feasible solution, and "Best" and "Med" indicate the best and median result of 50 runs, respectively. We present the best result from all the algorithms for each UCTP instance in the bold font.

From Table 5.5, it can be seen that EGSGA performs better than the fuzzy algorithm [29], LS [196], and graph-based approach [51] on all the 11 problem instances. EGSGA outperforms RIIA [14], and GA [181] on all the medium problem instances and ties with them on some or all of the small problem instances. EGSGA also gives better results than the ant algorithm [196] on 10 problem instances and equals it on S5. When compared with the tabu-based hyper heuristic search [47], EGSGA performs better or equally well on all the problem instances. Finally, the result of EGSGA is better than that of HEA [15] on all medium problem instances and ties it on small instances. On the whole, EGSGA beats all algorithms on medium problem instances and gives promising results on the large problem.

We also ran EGSGA according to the ITC-2002 rules [1] on the set of 20 UCTP instances. There were 50 runs of EGSGA on each problem instance. For each run on a problem instance, the maximum run time t_{max} was set to 900 seconds. The results of the algorithms we compared were taken from the ITC-2002 website [1]. A detailed description of these algorithms can be found in Chapter 3, Section 3.2, but we describe them here briefly as follows:

- SA: The simulated annealing based heuristic of Kostuch [138]. This approach

TABLE 5.5: Comparison of algorithms on different problem instances

	EGSGA		RIIA	HEA	GBHH	THHS	LS	GA	AA	FA
UCTP	Best	Med	Best	Best	Best	Best	Med	Best	Med	Best
S1	**0**	0	**0**	0	6	1	8	**0**	1	10
S2	**0**	0	**0**	0	7	2	11	3	3	9
S3	**0**	0	**0**	0	3	**0**	8	**0**	1	7
S4	**0**	0	**0**	0	3	1	7	**0**	1	17
S5	**0**	0	**0**	0	4	**0**	5	**0**	**0**	7
M1	**139**	143	242	221	372	146	199	280	195	243
M2	**92**	96.5	161	147	419	173	202.5	188	184	325
M3	**122**	124	265	246	359	267	77.5%In	249	248	249
M4	**98**	101	181	165	348	169	177.5	247	164.5	285
M5	**116**	119.5	151	135	171	303	100%In	232	219.5	132
L	615	622.5	100%In	**529**	1068	80%In	100%In	100%In	851.5	1138

is divided into two stages. First, it finds a feasible timetable them it uses a simulated annealing scheme to improve the timetable according to an objective function value.

- ETTS: The efficient timetabling solution with TS of Cordeau *et al.* [76]. They developed a tabu heuristic that first finds a feasible solution and then improves the quality of the solution by reducing soft constraints.
- GDLS: Bykov [56] used a great deluge local search algorithm to solve the problem.
- TSLS: Gaspero and Schaerf [113] used a three-stage local search paradigm. Their local search method consists of hill climbing, TS, and multi-swap shake stages.
- AMLS: Arntzen and Løkketangen [27] proposed a simple adaptive memory search to improve the quality of an initial solution. The search is guided by TS mechanisms based on recency and frequency of certain attributes of previous moves.
- DTS: Dubourg *et al.* [93] proposed a TS approach to solve the UCTP.

Table 5.6 shows the comparison of EGSGA with other results from the literature on the ITC-2002 test set, where “comp01” represents the first benchmark instance, “comp02” represents the second benchmark instance, and so on. Table 5.6 shows the best result (soft constraints violation) achieved by each algorithm, where the best

TABLE 5.6: Comparison of algorithms on ITC-2002 problem instances

UCTP	EGSGA	SA	ETTS	GDLS	TSLS	AMLS	DTS
comp01	54	**45**	61	85	63	132	148
comp02	**25**	**25**	39	42	46	92	101
comp03	**44**	65	77	84	96	170	162
comp04	132	**115**	160	119	166	265	350
comp05	97	102	161	**77**	203	257	412
comp06	**3**	13	42	6	92	133	246
comp07	**12**	44	52	**12**	118	177	228
comp08	**23**	29	54	32	66	134	125
comp09	21	**17**	50	184	51	139	126
comp10	**53**	61	72	90	81	148	147
comp11	46	**44**	53	73	65	135	144
comp12	96	107	110	**79**	119	290	182
comp13	**69**	78	109	91	160	251	192
comp14	**13**	52	93	36	197	230	316
comp15	35	**24**	62	27	114	140	209
comp16	**12**	22	34	300	38	114	121
comp17	104	86	114	**79**	212	186	327
comp18	39	**31**	38	39	40	87	98
comp19	63	**44**	128	86	185	256	325
comp20	2	7	26	**0**	17	94	185

result among all algorithms for each UCTP instance is represented in bold. From Table 5.6, it can be seen that EGSGA is able to produce good results on all problem instances. It gives the best result on 9 out of 20 problem instances. However, there is still a room for improvement in the proposed approach to obtaining an optimal solution on hard problem instances.

From the above experimental results, it can be seen that the GS strategy and proper LS strategies used in GSGAs can help minimize the objective function value and give better results for the UCTP compared to other population-based algorithms employed in the literature. The experimental results also shows that due to the good solutions that are created through the GS strategy in GSGAs, the chance of getting feasible and optimal solutions is increased.

5.5 Chapter Summary

This chapter introduced a guided search (GS) strategy to enhance the searching power of GAs for solving the UCTP. The GS strategy uses a memory (data structure) to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and that has a zero penalty value. This data structure is used to guide the generation of offspring into the following populations. The main advantage of the GS strategy lies in that it improves the quality of individuals by storing part of former good solutions, which otherwise would have been lost in the selection process, and reusing the stored information in the following generations. This can enable a GA to quickly retrieve the best solutions corresponding to the previous and new populations.

Based on the above GS strategy and two LS strategies described in Chapter 4, we presented several GA variants based on the steady-state GA model, including two versions of guided search GAs, i.e., GSGA and EGSGA, for solving the UCTP. In order to test the performance of proposed GAs for the UCTP, experiments were carried out to analyse the sensitivity of parameters within the GS strategy and the effect of the GS strategy for the performance of GSGAs based on a set of benchmark UCTP instances. The experimental results of EGSGA were also compared with several state-of-the-art methods from the literature on the tested UCTP instances.

The experimental results show that the proposed EGSGA is competitive and works well across the tested problem instances in comparison with other state-of-the-art approaches taken from the literature. Generally speaking, with the help of the GS and LS strategies, EGSGA is able to efficiently find optimal or near-optimal solutions for the UCTP and hence can act as a powerful tool for the UCTP.

To our knowledge, this study is the first time GAs have been applied with the GS strategy to address timetabling problems. In this chapter, we investigated the performance of GSGAs under simplified benchmark UCTP instances. We also want to test the performance of our approaches on more harder or near to the real-world UCTP instances. Consequently, the next chapter will be based on the question of how the GS strategy works on the post-enrolment course timetabling problem, which is a harder and nearer-to-the real-world UCTP.

Chapter 6

Hybrid Approaches for Post-Enrolment Course Timetabling

6.1 Introduction

In the previous chapter, we presented a guided search (GS) strategy to enhance the performance of genetic algorithms (GAs) for the university course timetabling problem (UCTP) and tested the performance of two versions of guided search GAs (GSGAs) based on the ITC-2002 benchmark UCTP instances. In this chapter, we move forward towards the more difficult or nearer to real-world UCTP instances, the post enrolment course timetabling problem (PECTP). The PECTP is one type of UCTP, in which a set of events has to be scheduled in time slots and located in suitable rooms according to the student enrolment data.

In this chapter, we propose a hybrid two-phase approach to solve the PECTP. In the first phase, a GSGA is applied to solve the PECTP. In the second phase, a tabu search (TS) heuristic is further used on the best solution obtained by the first phase to improve the optimality of the solution, if this is possible. The proposed hybrid approach is tested on a set of benchmark PECTP instances taken from the ITC-2007 compared with a set of state-of-the-art methods from the literature.

This chapter consists of four sections. The next section describes the proposed hybrid approach and its components. The experimental results obtained by comparing the proposed hybrid approach with other algorithms from the literature are reported and discussed in Section 3. Finally, Section 4 concludes this chapter with discussions on the proposed hybrid approach for the PECTP.

6.2 The Proposed Hybrid Approach for the PECTP

A hybrid approach is proposed based on the GSGA (described in Chapter 5) and a TS heuristic to solve the PECTP. The proposed hybrid approach works in two phases. In the first phase, the GSGA for the UCTP is adapted and applied to solve the PECTP. This GSGA integrates the GS strategy and some LS strategies, where the GS strategy uses a data structure that stores useful information extracted from previous good individuals to guide the generation of offspring into the population, and where the LS strategies are used to improve the quality of individuals. In addition to the original LS strategy LS1 [181], some new neighbourhood structures and relevant LS strategies are integrated into the proposed hybrid approach for the PECTP. Given that finding a feasible solution for the PECTP can be a challenging task [146], the hybrid approach employs a second phase, where a TS heuristic is further used on the best solution obtained by GSGA in the first phase to improve the optimality of the solution, if this is possible. In order to investigate the effect of parameters on the performance of the hybrid approach for the PECTP, a sensitivity analysis of key parameters of GSGA is carried out by systematic experiments based on a set of ITC-2007 benchmark PECTP instances.

The pseudo-code of the proposed hybrid GA and TS approach, denoted *HGATS*, for the PECTP is shown in Algorithm 15. HGATS works in two phases. In the first phase, the GSGA which uses genetic operators, a GS strategy, and two powerful LS techniques, is used to evolve a population of candidate solutions towards better and better solutions, ideally finding the optimal solution. Usually, GAs are able to locate promising regions for global optima in the search space, but sometimes, like other meta-heuristics, GAs have difficulty in finding the exact optimum of highly constrained problems [108]. Several examples can be found from the literature where a solution obtained from a GA is improved by another optimisation technique [121]. In this chapter, we also use this technique in HGATS to try to find an optimal solution for the PECTP. Considering the hardness of the PECTP, if only feasible

Algorithm 15 Proposed Hybrid Approach – HGATS

```
1: input : A problem instance I
2: set the generation counter $g := 0$
3: for $i := 1$ to population size do
4:    $s_i \leftarrow InitializeIndividual(i)$
5:    $s_i \leftarrow$ solution $s_i$ after applying LS strategy LS1
6:    $s_i \leftarrow$ solution $s_i$ after applying LS strategy LS3
7: end for
8: while the termination condition is not reached do
9:    if (g mod $\tau$) == 0 then
10:      apply $ConstructMEM2()$ to construct $MEM$
11:   end if
12:   $s \leftarrow$ child by applying $GuidedSearch2()$ or $BiasedCrossover()$ with a probability $\gamma$
13:   $s \leftarrow$ child after mutation with a probability $P_m$
14:   $s \leftarrow$ child after applying LS1
15:   $s \leftarrow$ child after applying LS3
16:   replace the worst individual of the population by $s$
17:   $g := g + 1$
18: end while
19: if $s$ is an optimal solution then
20:   go to line 24
21: else
22:   $s \leftarrow$ Apply $TabuHeuristic()$ on the best solution obtained in the first phase
23: end if
24: output : The best solution $s_{best}$ achieved for the problem instance I
```

solutions are found during the first phase of HGATS, the second phase is executed, which uses a TS heuristic inspired by [181] to improve the feasible solution toward the optimal solution. Below we describe the two phases of the proposed HGATS in detail, respectively.

6.2.1 The Enhanced GSGA – Phase I of HGATS

The first phase of HGATS uses the GSGA, which is adapted and enhanced according to the PECTP, to solve the PECTP (as described in Section 3.3, Chapter 3). The framework of the GSGA is based on a steady state GA, where one child solution is generated per iteration (or per generation) [102, 177]. GSGA starts from an initial population of individuals that are randomly generated, where events are assigned to rooms and time slots for each solution based on the property of each event. Usually, for GAs, the quality of the initial solutions affects the final solutions and researchers have shown that good initial solutions usually produce good or required results within less computational time [83, 147, 188]. Hence, we want to create a good initial population that would help GSGA to evolve quickly towards the optimal solution quickly. For this purpose, two LS strategies are applied to each individual of the initial population. The LS strategies use six neighbourhood structures, which will be described in Section 6.2.1.6, to first move events to time slots and then use the matching algorithm to allocate rooms and time slots to events.

After the initialisation of the population, a data structure MEM is constructed, which stores a list of room and time slot pairs (r, t) for all the events in the set E'_e that have zero penalty (i.e., no hard- and soft-constraint violation at these events) of good individuals selected from the population. After that, MEM can be used to guide the generation of offspring for the following generations. The MEM data structure is re-constructed regularly, e.g., every τ generation. In each generation of GSGA, one child is first generated either by using MEM or by applying the crossover operator, depending on a probability γ. The child will then undergo the mutation operation followed by the LS strategies for potential improvement. Finally, the worst member in the population is replaced with the newly generated child individual. This

Algorithm 16 *InitializeIndividual*(i)

```
1: input : The index $i$ of individual $I_i$
2: for each event $e_j$ of $I_i$ do
3:     if event $e_j \in E'_e$ then
4:         assign a random time slot from $T'_s$ to $e_j$
5:         assign a random room from a list of suitable rooms
6:     else
7:         assign a random time slot from $ET_j$ to $e_j$
8:         assign a random room from a list of suitable rooms
9:     end if
10: end for
11: output : The generated individual $I_i$
```

iteration continues until one termination condition is reached, e.g., a present time limit t_{max} is reached or the best solution found has no soft- and hard-constraint violations.

In the following sub-sections, we will describe in detail the key components of the adapted GSGA in turn, including the initialisation of the population, the MEM data structure and its construction, the guided search strategy, the crossover and mutation operators, and the two LS strategies.

6.2.1.1 Initialisation of the Population

Each individual I_i of the initial population is created by Algorithm 16. We divide the set of events E into two classes: events in E'_e (the set of events that have no time slot restriction, as described in Section 3.3.2 of Chapter 3) and events not in E'_e. If an event has no particular time slot restriction, it is allocated a random time slot t from the set T'_s of time slots that have no restriction of any event and a suitable room; otherwise, the event is allocated a random time slot from the element time

Algorithm 17 *ConstructMEM2()*

```
1: input : The whole population $P$ with the population size $N$
2: sort the population $P$ according to the fitness of individuals
3: $Q \leftarrow$ select the best $\alpha \times N$ individuals in $P$
4: for each individual $I_j$ in $Q$ do
5:    for each event $(e_i \in E'_e)$ in $I_j$ do
6:       calculate the penalty value of event $e_i$ from $I_j$
7:       if $e_i$ is feasible (i.e., $e_i$ has zero penalty) then
8:          add the room and time slot pair $(r_i, t_i)$ assigned to $e_i$ into the list $l_i$
9:       end if
10:   end for
11: end for
12: output : The data structure $MEM$
```

slot list of ET corresponding to the event, and is randomly allocated a room among suitable rooms. This way, each individual generated will satisfy the hard constraints H2 and H4, described in Section 3.3.1 of Chapter 3. However, it is not guaranteed to be feasible. An infeasible individual will be checked by LS strategies (to be described in Section 6.2.1.6), which will try to make it feasible.

6.2.1.2 The MEM Data Structure

In GSGA, we use a data structure MEM to guide the generation of offspring by re-introducing the best part of good individuals from previous generations. This MEM data structure is used to provide further direction of exploration and exploitation in the search space. It is a two-level structure as described in Section 5.2.1 of Chapter 5. The first level is a list of events and the second level is a list l_i of room and time slot pairs corresponding to each event e_i in the first level list. The MEM data structure is regularly re-constructed every τ generations. Algorithm 17 shows the outline of constructing MEM in the GSGA used in the hybrid approach. Here, the construction of MEM is slightly different from what we described in Section 5.2.1, because we consider only those events which are not restricted to be placed in pre-defined time slots.

When MEM is due to be re-constructed, we first select $\alpha \times N$ best individuals from the population P to form a set Q, where N denotes the population size. After that, for each individual $I_j \in Q$, we check each event $e_i \in E'_e$[1] by its penalty value, i.e., the hard- and soft-constraint violations associated with this event. If an event has a zero penalty value, then we store the information corresponding to this event into MEM. For example, if the event e_2 of an individual $I_j \in Q$ is assigned room 2 at time slot 13 and has a zero penalty value, then we add the pair $(2, 13)$ into the list l_2. Similarly, the events of the next individual $I_{j+1} \in Q$ are also checked by their penalty values. If the event e_2 in I_{j+1} has a zero penalty, then we add the pair of room and time slot assigned to e_2 in I_{j+1} into the existing list l_2. If for an event e_i, there is no list l_i existing yet, then the list l_i is added into the MEM data structure. Similar process is carried out for the selected Q individuals and finally MEM stores pairs of room and time slot corresponding to those events with zero penalty of the best individuals of the current population. This newly re-constructed MEM data structure is then used to guide the generation of offspring for the next τ generations.

[1]We only check those events that are not directly involved with H4 because other events must have been assigned in pre-specified time slots. It is worthless to assign and evaluate those events since they do not help to increase the diversity of the GSGA.

Algorithm 18 *GuidedSearch2*()

```
1: input : The $MEM$ data structure
2: $E_s$ := randomly select $\beta * |E'_e|$ events from $E'_e$
3: for each event $e_i$ in $E_s$ do
4:    randomly select a room and time slot pair from the list $l_i$ in $MEM$
5:    assign the selected pair to event $e_i$ for the child
6: end for
7: for each remaining event $e_i$ not in $E_s$ do
8:    if $e_i \in ET$ then
9:       assign a particular time slot and suitable room to $e_i$
10:   else
11:      assign a random time slot and room to $e_i$
12:   end if
13: end for
14: output : A new child generated using $MEM$
```

6.2.1.3 Generating a Child by the GS Strategy

In GSGA, a child is created through the GS strategy or crossover with a probability γ. When a new child is to be generated, a random number $\rho \in [0.0, 1.0]$ is first generated. If $\rho < \gamma$, *GuidedSearch2*() (as shown in Algorithm 18) will be used to generate the new child; otherwise, a crossover operator *BiasedCrossover*() (as shown in Algorithm 19) will be used.

If a child is to be created using the GS strategy, we first select a set E_s of $\beta * |E'_e|$ random events from E'_e to be generated from the MEM data structure. Here, β is a percentage value and $|E'_e|$ is the size of the set E'_e. We randomly select a pair of (r_i^j, t_i^j), $j = 1, \cdots, N_i$, from the list l_i that corresponds to the event e_i and assign the selected pair to e_i for the child. If there is an event e_i in E_s but there is no list l_i in MEM, then we randomly assign a room and time slot from possible rooms and time slots to e_i for the child. This process is carried out for all the events in E_s.

Algorithm 19 *BiasedCrossover*()

```
1:  input : The current population
2:  Select parents $P1$ and $P2$ by the binary tournament selection
3:  for each event $e_i$ of the child $Ch$ do
4:      if penalty value of $e_i$ of $P1 <$ penalty value of $e_i$ of $P2$ then
5:          $e_i$ of $Ch \leftarrow$ the time slot and room allocated to $e_i$ of $P1$
6:      else
7:          $e_i$ of $Ch \leftarrow$ the time slot and room allocated to $e_i$ of $P2$
8:      end if
9:  end for
10: output : A new child generated using crossover
```

For those remaining events that are not present in E_s, we assign time slots and rooms according their particular requirements for the child.

6.2.1.4 Crossover

If a child is to be generated using the crossover operator, we first select two individuals from the current population as the parents via the binary tournament selection scheme. A child is then generated as follows: for each event, we first select the parent that has the smaller penalty value corresponding to that event, and then allocate the corresponding room and time slot pair to the event of the child.

Note that the crossover operator here is different from the one shown previously in Algorithm 10. Here, for each event of a child, the crossover is *biased* toward the parent that does better for that event (assuming that a good parent that has fewer violations related to the chosen event may result in good children in the next generation), while in Algorithm 10 the crossover is uniform or unbiased regarding each parent for each event of a child.

6.2.1.5 Mutation

After a child is generated by using either the GS strategy or crossover, a mutation operator is used on the child solution with a probability P_m. The mutation operator first randomly selects one from four neighbourhood structures N1, N2, N3, and N4, which will be described below in Section 6.2.1.6, and then makes a move within the selected neighbourhood structure.

6.2.1.6 Local Search Strategies

When a child solution is generated after the mutation operation, two LS strategies, denoted *LS3* and *LS4* respectively, are applied to the solution for possible improvement. We allocate the name LS3 and LS4 because LS1 and LS2 were described in Chapter 4. LS3 works on all events of a solution while LS4 works on a set of events of a solution. Here, we suppose that each event is involved with soft- and hard-constraint violations.

The two LS strategies are based on six neighbourhood structures, denoted as N1, N2, N3, N4, N5, and N6, respectively. The first three neighbourhood structures N1, N2, and N3 are the same as defined previously in Section 4.2.1.1 of Chapter 4. The other three neighbourhood structures N4, N5, and N6 are defined as follows:

- N4: the neighbourhood defined by an operator that takes two random events from the set E'_e and replaces their time slots by random ones from T'_s, where E'_e and T'_s are as defined in Section 3.3.2 of Chapter 3.

- N5: the neighbourhood defined by an operator that takes each event e_i from the list of EE and tries to find a place in the timetable before all the events in EE_i, where EE and EE_i are as defined in Section 3.3.2 of Chapter 3.

- N6: the neighbourhood defined by an operator that takes a subset of time slots from all occupied time slots. Among this subset, the worst time slot (that contains events that collectively have the highest penalty value) is selected and its events are moved to another randomly chosen time slot in the subset.

The pseudo-code of LS3 is shown in Algorithm 20, where the italicised parts show the changes that differentiate between LS1 and LS3. Comparing LS3 with LS1 (see Algorithm 8 in Section 4.2.1.1 of Chapter 4), we can see that LS3 is similar to LS1, except that LS3 now uses four neighbourhood structures instead of three as used in LS1. Like LS1, LS3 also works on all events and is based on two steps. In the first step (Lines 3-13), LS3 checks the hard-constraint violations of each event while ignoring its soft-constraint violations. If there are hard-constraint violations for an event, LS1 tries to resolve them by applying moves in the neighbourhood structures $N1$, $N2$, $N3$, and $N4$ in order as follows. First, we try to move the event to the next time slot, then the next, then the next, etc. If this search in $N1$ fails, we then search in $N2$ by trying to swap the event with the next one in the list, then the next

Algorithm 20 Local Search Strategy 3 (LS3)

```
1: input : Individual I selected from the population
2: while  Termination condition not reached do
3:    for $i = 1$ to the total number of events do
4:       if event $i$ is infeasible then
5:          if there is untried move left then
6:             calculate the next move (first in $N1$, then in $N2$, then in $N3$, and finally in
               $N4$)
7:             apply the matching algorithm to the time slots affected by the move and
               delta-evaluate the result.
8:             if the move reduces hard constraint violation then
9:                make the move
10:            end if
11:         end if
12:      end if
13:   end for
14:   if any hard constraint violations remain then
15:      terminate LS3
16:   else
17:      for $i = 1$ to total number of events do
18:         if event $i$ has soft constraint violation then
19:            if there is untried move left then
20:               calculate the next move (first in $N1$, then in $N2$, then in $N3$, and finally
                  in $N4$)
21:               apply the matching algorithm to the time slots affected by the move and
                  delta-evaluate the result
22:               if the move reduces soft constraints violation then
23:                  make the move
24:               end if
25:            end if
26:         end if
27:      end for
28:   end if
29: end while
30: output : A possibly improved individual I
```

one, and so on. If the search in $N2$ also fails, we try a move in $N3$ by using one different permutation formed by the event with the next two events, then with the next two, and so on. If the search in $N3$ also fails, we try a move in $N4$ by replacing the time slots of two random events that are in the set E'_e with random time slots from T'_s, then the next two, and so on, until a termination condition is reached, e.g., an improvement is reached or the maximum number of steps s_{max} is reached, which is set to different values for different problem instances in the experimental study.

After each move, we apply the matching algorithm to the time slots affected by the move and try to resolve the room allocation disturbance and delta-evaluate the result of the move (i.e., calculate the hard- and soft-constraint violations before and after the move). If there is no untried move left in the neighbourhood for an event, LS1 continues to the next event. After applying all neighbourhood moves on each event, if there is still any hard-constraint violation, then LS3 will stop; otherwise, LS3 will perform the second step (lines 17-27 in Algorithm 20).

In the second step, after reaching a feasible solution, LS3 performs a similar process as in the first step on each event to reduce its soft-constraint violations. For each event, LS3 tries to make moves in the neighbourhood $N1$, $N2$, $N3$, and/or $N4$ in order without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta-evaluated.

Algorithm 21 describes the second LS strategy, LS4, used in GSGA. LS4 works on a set of events with $N5$ (corresponding to lines 2-9 in Algorithm 21) and $N6$ (corresponding to lines 10-27 in Algorithm 21). The basic idea of LS4 is that it first tries to place an event e_i (involved in the precedence constraint H5) in a time slot before the corresponding list of events EE_i.

After moving a concerned event into a new time slot in the neighbourhood structures $N1$ and $N2$ every time, the new penalty value of the event is calculated. If the move

Algorithm 21 Local Search Strategy 4 (LS4)

```
1: input : Individual I after LS1 is applied
2: for each event $e_i$ of $I$ in $EE$ do
3:     for each event $e_j$ in $EE_i$ do
4:         try to place event $e_j$ in the timetable after the time slot of $e_i$ by calculating a
           move of $e_i$ in the neighbourhood $N1$ and $N2$
5:         apply the matching algorithm to the time slots affected by the move
6:         compute the penalty of $e_i$ and delta-evaluate the result
7:         apply the move if it reduces hard- or soft-constraint violations
8:     end for
9: end for
10: $S :=$ randomly pick a percentage of occupied time slots from $T$
11: for each time slot $t_i \in S$ do
12:     for each event $e_j$ in the time slot $t_i$ do
13:         calculate the penalty value of event $e_j$
14:     end for
15:     sum the total penalty value of events in the time slot $t_i$
16: end for
17: select the time slot $w_t$ with the biggest penalty value from $S$
18: for each event $e_i$ in $w_t$ do
19:     calculate a move of $e_i$ in the neighbourhood $N1$
20:     apply the matching algorithm to the time slots affected by the move
21:     compute the penalty of $e_i$ and delta-evaluate the result
22: end for
23: if all the moves of events in $w_t$ together reduce hard- or soft-constraint violations
    then
24:     apply the moves
25: else
26:     delete the moves
27: end if
28: output : A possibly improved individual I
```

reduces the penalty value, then it is saved; otherwise, it is not.

After applying $N5$, LS4 applies $N6$. It first randomly selects a percentage of time slots[2] from the total time slots in T. The penalty value of each selected time slot is then calculated and the time slot w_t that has the biggest penalty value is selected for

[2]Rather than choosing the worst time slot out of all time slots, we randomly select a set of time slots and then choose the worst time slot from the set. This is because for each selected time slot we need to calculate its penalty value, which costs time. By selecting a set of time slots instead of all time slots, we try to balance between the computational time and the quality of the algorithm.

LS operation. This way, LS4 aims to help improve the existing result. After taking the worst time slot, LS4 tries a move in the neighbourhood structure $N1$ for each event of w_t and checks the penalty value of each event before and after applying the move. If all moves in w_t together reduce the hard- and/or soft-constraint violations, then we apply all the moves; otherwise, we do not make the moves. This way, LS4 can not only place the events according to their precedence but also check the worst time slot and reduce the penalty value for some events by moving them to other time slots.

In general, LS4 is expected to enhance the individuals of the population and increase the quality of the feasible solution by reducing the number of constraint violations. When LS4 finishes, we get a possibly improved and feasible individual.

At the end of each generation of GSGA, the obtained child solution replaces the worst member of the population to make a better population in the next generation. By the end of Phase I, GSGA may produce several different optimal or near-optimal solutions.

6.2.2 The Tabu Search Heuristics – Phase II of HGATS

We try to find an optimal solution using the above proposed GSGA. However, due to the hardness of the PECTP, after the first phase of HGATS, sometimes an optimal or feasible solution may not be obtained. In order to further improve the quality of the solution obtained by GSGA, a simple TS heuristic, $TabuHeuristics()$, shown

Algorithm 22 *TabuHeuristics*() – Phase II of HGATS

```
1: input : The best solution $\mathbf{s_{best}}$ from Phase I (GSGA)
2: $s \leftarrow s_{best}$
3: if $s$ is not feasible then
4:    remove all events that involve hard-constraint violations
5: end if
6: $TL \leftarrow \emptyset$
7: while the termination condition is not reached do
8:    for $i := 0$ to 10% of the neighbours do
9:       $s_i \leftarrow s$ after the $i$-th move
10:      compute the objective value $f(s_i)$
11:   end for
12:   if $\exists s_j | f(s_j) < f(s)$ and $f(s_j) \leq f(s_i) \forall i$ then
13:      $s \leftarrow s_j$
14:      $TL \leftarrow TL \cup E_i$ where $E_i$ is the set of events moved to get $s_j$
15:   else
16:      $s \leftarrow$ the best non-tabu moves among all $s_i$
17:      $TL \leftarrow TL \cup E_b$ where $E_b$ is the set of events moved by the best non-tabu
         move
18:   end if
19:   $s_{best} \leftarrow$ the best solution so far
20: end while
21: output : The optimised solution $s_{best}$
```

in Algorithm 22, is applied as the second phase of HGATS in the hope of getting an improved and feasible solution from the best solution obtained from Phase I. TS is a kind of heuristic methods, which has the advantage of having internal memory [107]. This internal memory prevents TS from revisiting previously visited areas of the search space. Therefore, it is easier to escape from the local optimum and approach the global or near-global optimum in a short time [66]. TS is usually known to be a powerful tool for all types of timetabling problems [47].

The TS heuristic used in HGATS is similar to the TS scheme described in [181]. We first check the best solution obtained from the first phase. If it is optimal, Phase

II will not be executed. Otherwise, if it is a feasible solution, then we improve the solution by applying the TS heuristic. If a solution is not feasible, we first remove all events that involve hard-constraint violations and re-consider them if and only if they satisfy all hard constraints during the neighbourhood search.

We apply $N1$, $N2$, and $N4$ as neighbourhood structures for moving a solution. A move of a solution is defined as moving one random event of the solution using $N1$, swapping two random events of the solution using $N2$, or swapping two specific events of the solution to time slots using $N4$, in order. The reason for not applying $N5$ and $N6$ in the move is that using them takes time and extra work in removing a hard-constraint violation.

A move is a tabu move if at least one of the events involved has been moved less than l steps before, where l is the length of the tabu list TL. The tabu list length is set to the number of events divided by a constant K ($K = 100$ as described in [181]). In order to decrease the probability of generating cycles of moves and to enhance the exploration, a variable neighbourhood set is applied, as suggested in [181], where every move uses the neighbourhood $N1$, $N2$, or $N4$ with a probability 0.1. In order to explore the search space more efficiently, we accept a tabu move if it improves the best solution ontained so far.

In summary, the TS heuristic considers a variable neighbourhood set and performs the best move that improves the best solution obtained so far; otherwise, it performs the best non-tabu move chosen among those that belong to the current variable set of neighbours. The TS heuristic continues until a time limit is reached or the best solution obtained so far has no soft- and hard-constraint violation (i.e., an optimal solution is obtained).

6.3 Experimental Study

In this section, we experimentally investigate the performance of the proposed hybrid approach for the PECTP compared with several other algorithms. All algorithms were coded in GNU C++ under version 4.1 and run on a 3.20 GHz PC. We used 24 benchmark PECTP instances to test the algorithms, which were proposed in [146] for the ITC-2007. Table 3.2 presents the features of these PECTP instances[3].

Two sets of experiments were carried out in this study. The first set of experiments were devoted to analysing the sensitivity of key parameters for the performance of HGATS for the PECTP. The second set of experiments compared the performance of HGATS with two relevant algorithms on the test PECTP instances. Finally, we compared our experimental results with current state-of-the-art methods from the literature on the tested instances. All the figures in experimental study show objective function values of the problem.

[3]Details about these PECTP instances can be found at the website http://www.cs.qub.ac.uk/itc2007/postenrolcourse/course_post_index.htm.

Table 6.1: Parameter settings in HGATS

Parameter	Settings			
α	0.2	0.4	0.6	0.8
β	0.1	0.3	0.5	0.7
γ	0.2	0.4	0.6	0.8
τ	20	40	60	80

6.3.1 Sensitivity Analysis of Key Parameters of HGATS

The performance of the proposed hybrid approach depends on the parameters and operators used, especially in GSGA. Through our previous work [129], we found that α, β, γ, and τ are key parameters that can greatly affect the performance of GSGA for the UCTP, where α is the percentage of best individuals selected from the current population for creating the data structure MEM, β is the percentage of the total number of events that are used to create a child through the data structure MEM, γ is the probability that indicates whether a child is created through MEM or crossover, and τ decides the frequency of updating MEM (i.e., MEM is updated every τ generations). Hence, we test our algorithm HGATS with different settings of these parameters. Table 6.1 shows the different parameters and their settings that were tested in our experiments. Some other parameters for HGATS were set as follows: the population size N was set to 50 and the mutation probability P_m was set to 0.5.

In order to find out which parameter settings have a greater effect on the performance of HGATS, we ran HGATS 50 times for all parameter combinations in Table 6.1. Here, we report some typical results in Figure 6.1, where the dynamic performance of

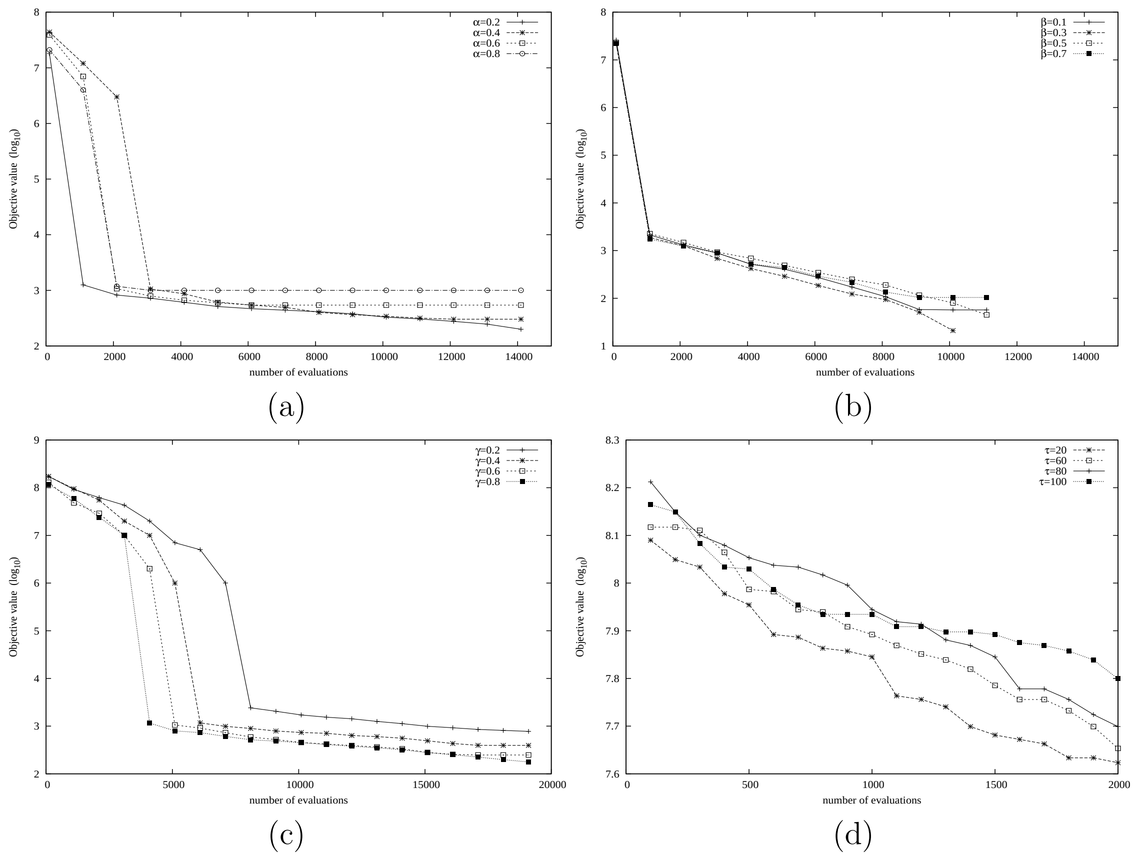

FIGURE 6.1: Comparison on the effect of parameters on the performance of HGATS on different problem instances: (a) 2007-21 with $\beta = 0.3$, $\gamma = 0.8$ and $\tau = 20$, (b) 2007-17 with $\alpha = 0.2$, $\gamma = 0.8$ and $\tau = 20$, (c) 2007-11 with $\alpha = 0.2$, $\beta = 0.3$ and $\tau = 20$, and (d) 2007-03 with $\alpha = 0.2$, $\beta = 0.3$ and $\gamma = 0.8$.

HGATS regarding the average objective value against the number of evaluations over 50 runs with one parameter changing while the other parameters are kept constant on different PECTP instances is shown. Figure 6.1(a) shows the effect of changing α on the 2007-16 problem instance with $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$. Figure 6.1(b) shows the effect of changing β on 2007-17 with $\alpha = 0.2$, $\gamma = 0.8$, and $\tau = 20$. Figure 6.1(c) shows the effect of changing γ on 2007-11 with $\alpha = 0.2$, $\beta = 0.3$, and $\tau = 20$. Figure 6.1(d) shows the effect of changing τ on 2007-3 with $\alpha = 0.2$, $\beta = 0.3$, and $\gamma = 0.8$.

From Figure 6.1, several results can be observed and are analysed below. First, the parameter α has a significant effect on the performance of HGATS for the PECTP. The performance of HGATS drops when the value of α increases from 0.2 to 0.8, see Figure 6.1(a) for reference. This occurs because when we choose a small part of population to create the MEM data structure, MEM can provide strong guidance during the genetic operations and help HGATS exploit the area of the search space that corresponds to the best individuals of the population sufficiently.

This sufficient exploitation can ensure that HGATS achieves better solutions quickly. In contrast, when a large part of the population is taken to create or update MEM, then MEM will lose its effect of guiding HGATS to exploit promising areas of the search space. In other words, when α is set to large values, HGATS tends to be GALS and, hence, the performance will drop or be weak. This can be observed from Figure 6.1(a): when the value of α increases, the best solution of HGATS can not improve after a certain number of evaluations, e.g., after about 4000 evaluations when $\alpha = 0.6$ and after about 2000 evaluations when $\alpha = 0.8$.

Secondly, regarding the effect of β, an interesting behaviour of HGATS can be observed on the 2007-17 problem instance with $\alpha = 0.2$, $\gamma = 0.8$, $\tau = 20$, and different β values in Figure 6.1(b). From Figure 6.1(b), it can be seen that when the value of β increases from 0.1 to 0.3, the performance of HGATS improves due to the enhanced effect of the MEM data structure. However, when the value of β is further raised, the performance of HGATS drops. This occurs because if a large proportion of individuals is created through MEM, e.g., when $\beta = 0.7$, the chance of creating a similar child may be increased every generation, and after a few generations, HGATS may be trapped in a sub-optimal state and hence be unable to obtain the optimal solution. From Figure 6.1(b), it can be seen that setting the value of β to 0.5 or 0.7 leads to an earlier stagnation in the performance of HGATS during the solving process.

Third, regarding the effect of γ, from Figure 6.1(c), it can easily be seen that increasing the value of γ results in better solutions. The reason lies in the fact that the small value of γ leads to the proposed GSGA acting as the conventional GA. The effect of γ also shows the importance of introducing the MEM data structure.

Finally, regarding the effect of τ, it can be seen from Figure 6.1(d) that updating MEM every 20 generations gives a better performance for HGATS than updating it every 80 generations. This is due to the fact that in the former case the search space is explored more than in the latter case, which increases the diversity and offers a greater chance of creating better individuals. The difference is significant when τ is set to 20 over 100. This is because increasing the value of τ slows down the updating of the MEM data structure and hence degrades the efficiency of MEM.

As a result of the above parameter analyses, in the following experiments, we set the parameters for HGATS as follows: $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$.

6.3.2 Comparison with Relevant Algorithms

This set of experiments compares the performance of HGATS with two relevant algorithms. One is the same as the proposed GSGA, except that the GS strategy is switched off i.e. it is the standard steady state GA with LS3 and LS4, denoted *GALS* in this study. For GALS, the crossover operator is applied with a crossover probability $P_c = 0.8$. The second algorithm is the TS algorithm. The basic framework of the TS algorithm tested is inspired by [181] with the same new neighbourhood structures as used by the tabu heuristics in HGATS. The parameter settings identified for HGATS by the previous experiments were used in HGATS and GALS (if relevant) in this section. The $InitializeIndividual()$ procedure is used for initial solutions for all algorithms in order to have a fair comparison of the performance of algorithms. There were 50 runs of each algorithm on each problem instance. The run time for each run of an algorithm on each problem instance was set to $t_{max} = 600$ seconds based on the time allocation used by the ITC-2007. Other parameter settings are as follows: the population size N was set to 50 and the mutation probability P_m was set to 0.5.

Algorithms are evaluated on the basis of two values, Df (distance to feasibility, see Section 3.3.1) and SCP (soft constraint penalty). Table 6.2 presents the results of the algorithms in terms of the best, worst, average, and standard deviation of Df and SCP values over the 50 runs on the 24 problem instances. From Table 6.2, it can be seen that HGATS produces a lower average and standard deviation of the

TABLE 6.2: Comparison of algorithms on different problem instances

		Best		Worse		Average		Std	
PECTP	Alg	*Df*	*SCP*	*Df*	*SCP*	*Df*	*SCP*	*Df*	*SCP*
2007-1	TS	0	1069	51	1732	11.33	1202.6	17.55	257.04
	GALS	0	641	12	976	3.44	704.89	5.13	152.33
	HGATS	**0**	**501**	12	842	0	587	1.84	108.61
2007-2	TS	0	989	72	2213	25.87	1191.22	17.44	386.77
	GALS	0	747	50	2311	8.78	1005.11	16.87	504.79
	HGATS	**0**	**342**	0	695	0	476.2	0	96.94
2007-3	TS	0	756	18	821	5.89	794.33	5.84	21.9
	GALS	**0**	**509**	0	801	0	697.44	0	129.53
	HGATS	0	3770	432	0	0	407.78	0	19.73
2007-4	TS	0	794	76	1130	18.33	910.5	29.44	141.45
	GALS	0	521	11	791	2	669	4.09	46
	HGATS	**0**	**234**	4	524	0	369	0.33	26.72
2007-5	TS	0	496	65	678	22	544.2	9.62	884
	GALS	0	98	20	310	8.62	154	9.62	78
	HGATS	**0**	**0**	0	325	0	118	0	88.05
2007-6	TS	0	218	0	788	0	428	0	272.93
	GALS	0	10	0	430	0	207	0	134.18
	HGATS	**0**	**0**	0	342	0	201	0	139.5
2007-7	TS	0	84	198	508	82	258	66.59	183.74
	GALS	0	275	70	489	25.5	381.75	35.20	89.531
	HGATS	**0**	**0**	2	543	0.53	418	1.62	98.404
2007-8	TS	**0**	**0**	0	751	0	481	0	315.7
	GALS	**0**	**0**	0	424	0	322	0	193.1
	HGATS	**0**	**0**	0	309	0	257.12	0	120.78
2007-9	TS	0	1711	152	2361	40.12	1797	53.04	294.85
	GALS	0	1547	115	2141	28.87	1237	39	412.37
	HGATS	**0**	**989**	42	1183	4.5	1002	20.09	81.12
2007-10	TS	0	763	0	1978	0	999	0	406
	GALS	4	548	26	1040	5	850	9	154
	HGATS	**0**	**499**	0	810	0	614	0	117
2007-11	TS	0	680	0	1980	0	968	0	414
	GALS	0	701	0	984	0	897	0	84
	HGATS	**0**	**246**	0	691	0	452	0	121
2007-12	TS	0	373	56	1563	17	702	23	393
	GALS	0	444	0	984	0	576	0	178
	HGATS	**0**	**172**	13	546	1.625	226	0.59	129
2007-13	TS	0	624	20	1873	6.37	1230	9.1	380
	GALS	0	201	0	1639	0	852	0	392
	HGATS	**0**	**0**	0	717	0	616	0	249
2007-14	TS	0	241	17	416	4.75	287	7.62	76
	GALS	0	61	0	104	0	78.2	0	17
	HGATS	**0**	**0**	0	19	0	4.125	0	7.29
2007-15	TS	0	101	0	164	0	135	0	33
	GALS	0	14	0	97	0	69	0	21
	HGATS	**0**	**0**	0	37	0	26	0	6.54
2007-16	TS	0	109	0	1158	0	563	0	161
	GALS	0	168	0	771	0	377	0	195
	HGATS	**0**	**0**	0	270	0	168	0	115.27
2007-17	TS	**0**	**0**	0	42	0	32	0	10
	GALS	**0**	**0**	0	21	0	5	0	7.4
	HGATS	**0**	**0**	0	11	0	2.5	0	4.65
2007-18	TS	**0**	**0**	0	1241	0	924	0	420.52
	GALS	**0**	**0**	0	842	0	631	0	270
	HGATS	**0**	**0**	0	572	0	446	0	108
2007-19	TS	147	1078	346	1867	138	1372	110	334
	GALS	0	1015	430	2693	174	1612	154	673.65
	HGATS	**0**	**84**	319	1900	133	810	115	513.7
2007-20	TS	40	348	113	1192	71	1100	29	133
	GALS	0	318	138	1942	67	1199	86	439
	HGATS	**0**	**297**	234	2305	75	1274	95	622
2007-21	TS	0	137	261	1162	69.5	805	96	267
	GALS	**0**	**0**	10	621	22.5	305	4.6	241
	HGATS	**0**	**0**	15	1359	2.5	780	2	422
2007-22	TS	91	1742	102	2439	97.37	2051	4.47	260.6
	GALS	42	1579	188	2466	94	1715	42	396
	HGATS	**0**	**1142**	73	1315	33.125	1196	38	243
2007-23	TS	0	2062	34	5556	362	1604	16	8.75
	GALS	11	1001	43	1291	81	1193	13	20
	HGATS	**0**	**963**	16	1896	1.2	1152	3.6	2
2007-24	TS	0	629	0	2309	0	1407	0	541
	GALS	0	368	9	2007	2.25	1112	4.16	463.6
	HGATS	**0**	**274**	0	2142	0	1002	0	519

objective value on most of the PECTP instances. HGATS produces good solutions due to the usage of the MEM data structure and LS schemes. As mentioned earlier, this is due to the fact that we assign to an event a pair of room and time slot that was extracted from one of the best individuals of previous populations. This means that the pair satisfies different constraints that are suitable to that event. The local and tabu search techniques further help find the local optimum of an individual. By doing so, we increase the chance of getting better and better solutions during the solving process.

Figures 6.2 and 6.3 show the dynamic performance of different algorithms regarding the objective value in the log scale against the number of evaluations. From these figures, it can be seen that on the 2007-14 and 2007-17 problem instances, HGATS and TS reach a solution as the number of evaluations increases. HGATS remarkably decreases in the objective value and gives an optimal solution after 9000 and 4000 evaluations, respectively.

The t-test results of statistically comparing investigated algorithms with 98 degrees of freedom at a 0.05 level of significance are shown in Table 6.3 (details of this t-test are presented in Appendix A). The average performance values over 50 runs of all algorithms on all problem instances are used for the t-test and are shown in Tables A.7, A.8, A.9, A.10, A.11, and A.12, respectively. In Table 6.3, the t-test result is shown as "$s+$", "$s-$", "$+$", "$-$", or "$\sim$" when the first algorithm is significantly better than, significantly worse than, not significantly better but better than, not

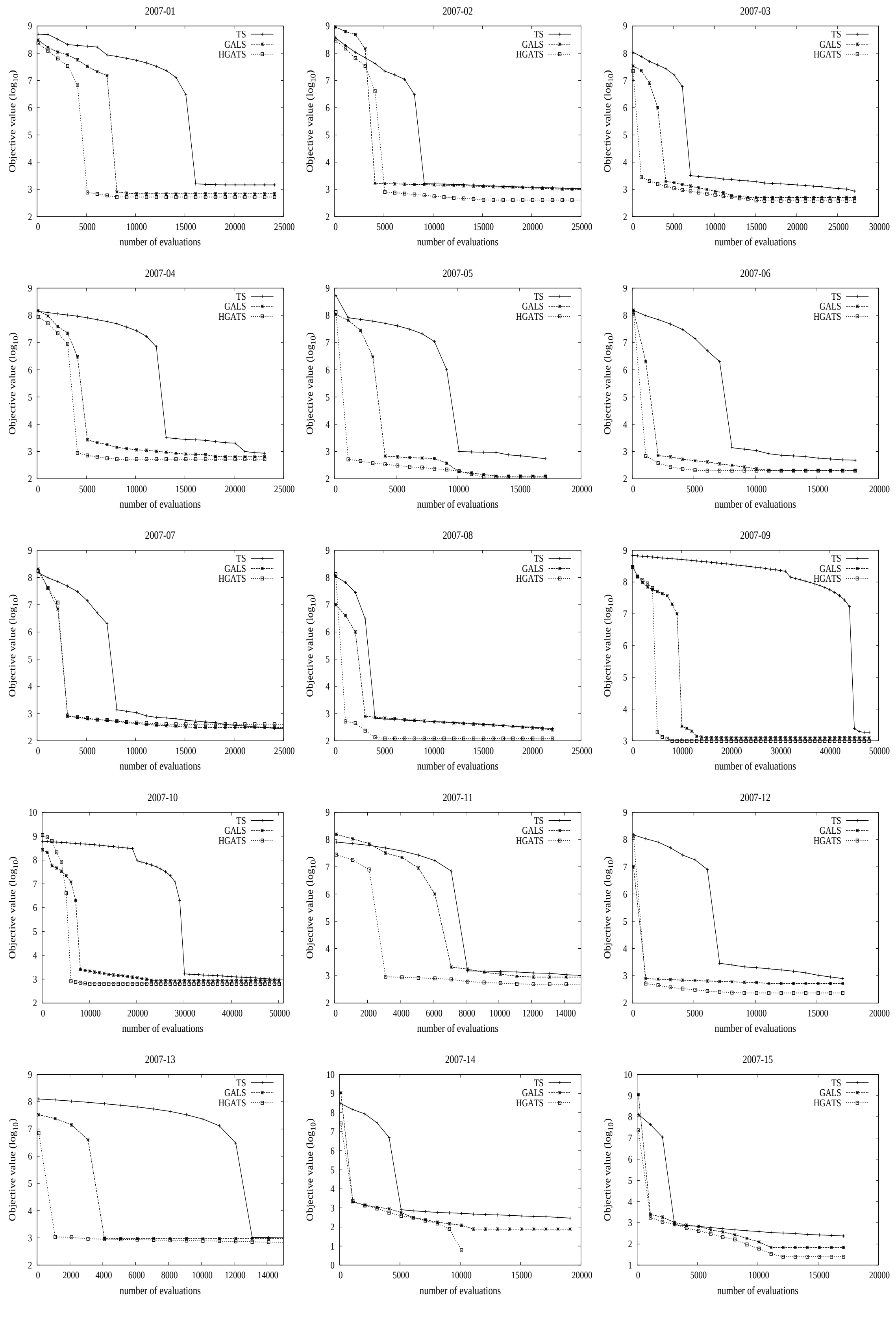

Figure 6.2: Dynamic performance of algorithms on PECTP 2007-01 to 2007-15.

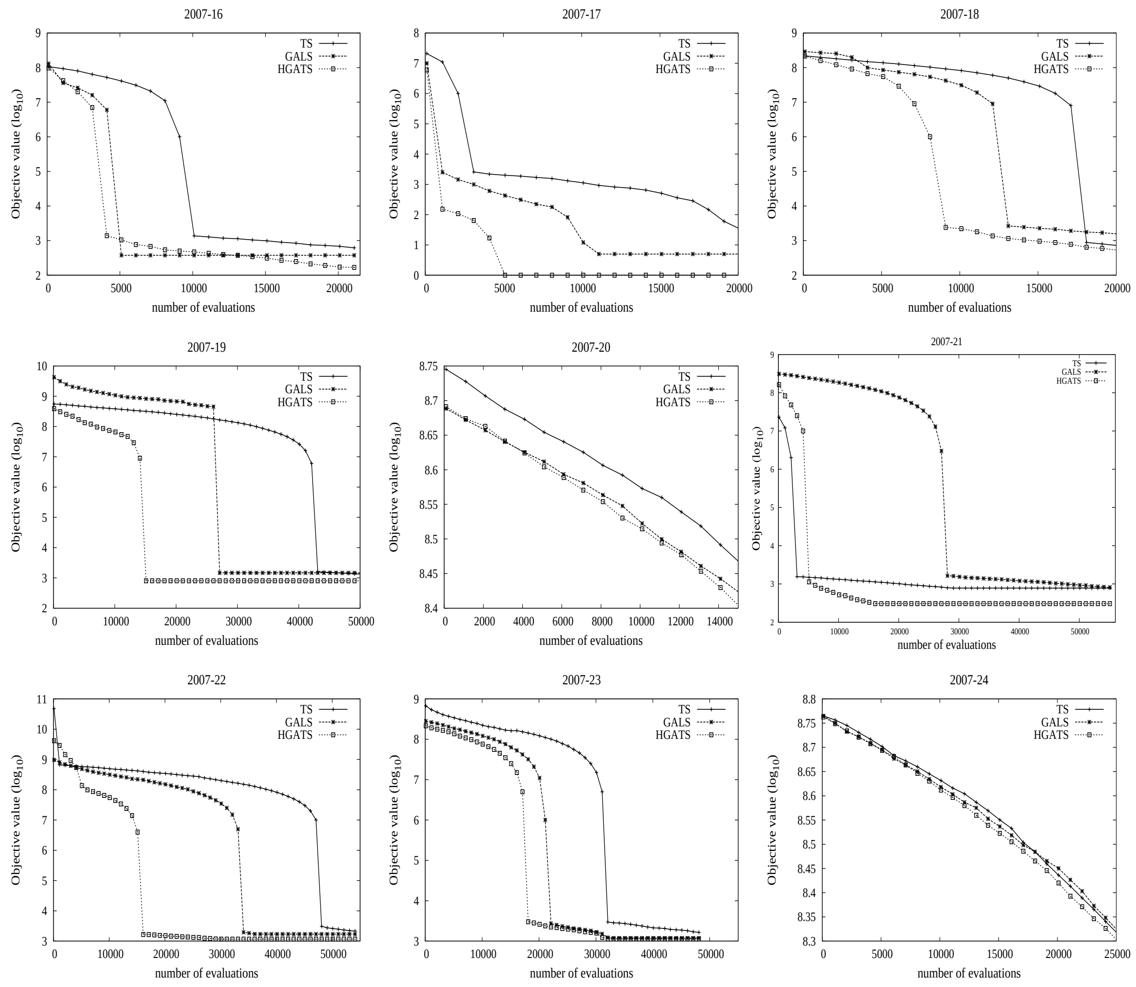

FIGURE 6.3: Dynamic performance of algorithms on PECTP 2007-16 to 2007-24.

significantly worse but worse than, or statistically equivalent to the second algorithm, respectively.

From Table 6.3, it can be seen that the performance of HGATS is significantly better than the performance of the other two algorithms on most problem instances. It can also been observed that the performance of GALS is significantly better than the performance of TS on most problem instances. This result indicates that a single heuristic is not enough for solving a PECTP. It can also be observed that

Table 6.3: The t-test values of comparing algorithms on different ITC-2007 PECTP instances

PECTP	Df			SCP		
	HGATS−GALS	HGATS−TS	GALS−TS	HGATS−GALS	HGATS−TS	GALS-TS
2007-1	$-$	$s+$	$s+$	$+$	$s+$	$s+$
2007-2	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
2007-3	$\sim$	$+$	$+$	$s+$	$s+$	$s+$
2007-4	$+$	$s+$	$s+$	$s+$	$s+$	$+$
2007-5	$\sim$	$\sim$	$\sim$	$+$	$+$	$+$
2007-6	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
2007-7	$\sim$	$\sim$	$\sim$	$+$	$s+$	$s+$
2007-8	$s+$	$s+$	$+$	$+$	$s+$	$+$
2007-9	$+$	$\sim$	$-$	$+$	$s+$	$s+$
2007-10	$\sim$	$\sim$	$\sim$	$s+$	$s+$	$+$
2007-11	$s-$	$s+$	$s+$	$s+$	$s+$	$+$
2007-12	$\sim$	$s+$	$s+$	$s+$	$s+$	$+$
2007-13	$\sim$	$s+$	$s+$	$+$	$s+$	$s+$
2007-14	$\sim$	$s+$	$s+$	$s+$	$s+$	$s+$
2007-15	$\sim$	$\sim$	$\sim$	$s+$	$s+$	$s+$
2007-16	$\sim$	$\sim$	$\sim$	$s+$	$s+$	$s+$
2007-17	$\sim$	$\sim$	$\sim$	$+$	$s+$	$s+$
2007-18	$\sim$	$\sim$	$\sim$	$+$	$s+$	$+$
2007-19	$+$	$+$	$+$	$s+$	$s-$	$s-$
2007-20	$+$	$+$	$+$	$s-$	$s-$	$+$
2007-21	$s+$	$s+$	$s+$	$s-$	$+$	$s+$
2007-22	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$
2007-23	$s+$	$\sim$	$s+$	$s+$	$s+$	$s+$
2007-24	$s+$	$\sim$	$s-$	$+$	$s+$	$+$

Table 6.4: Percentage of feasible solutions achieved by HGATS after Phase I and Phase II over 50 runs on each ITC-2007 PECTP instance

	Problem Instances											
HGATS Phase	1	2	3	4	5	6	7	8	9	10	11	12
Phase I (GSGA)	52%	64%	92%	96%	82%	78%	78%	100%	48%	52%	100%	80%
Phase II (TS)	92%	100%	100%	98%	100%	100%	94%	100%	82%	100%	100%	96%
	Problem Instances											
HGATS Phase	13	14	15	16	17	18	19	20	21	22	23	24
Phase I (GSGA)	86%	86%	100%	90%	100%	100%	20%	26%	30%	48%	48%	50%
Phase II (TS)	100%	100%	100%	100%	100%	100%	54%	68%	94%	70%	96%	100%

the integration of proper LS with guided search techniques can greatly improve the performance of GAs for the PECTP.

In order to show the benefit of introducing the second phase (i.e., the TS heuristics) in HGATS, we also recorded the percentage of feasible solutions obtained by HGATS after Phase I and Phase II over 50 runs on each PECTP instance. The results are shown in Table 6.4. From Table 6.4, it can be seen that the TS heuristics greatly improves the percentage of feasible solutions achieved on top of Phase I of HGATS on most PECTP instances. Hence, the TS heuristics is beneficial to the performance of HGATS, which justifies the two-phase hybrid approach for the PECTP.

6.3.3 Comparison with Algorithms from the Literature

In this section, in order to justify the performance of our proposed algorithm, we compare the experimental results of HGATS with the available results of other algorithms on the ITC-2007 PECTP instances. Another reason for comparing our results to the available results is that we are interested in seeing the behaviour of GAs for highly constraint PECTPs among different heuristic and optimisation methods, since this has not yet been investigated yet in the literature. The algorithms compared are described in detail in chapter 3 section 3.3. Here, we briefly described them as follows:

- HGATS: The hybrid approach proposed in this chapter.
- The mixed meta-heuristic approach (MMA): In their paper [57], Hadrien *et al.* proposed the MMA, which includes TS and simulated annealing used in conjunction with various neighbourhood operators.

- CTI: Mitsunori *et al.* [30] proposed this technique, which is the combination of a general purpose constraint satisfaction solver, TS, and iterated LS techniques.

- The hybrid algorithm (HA): In their paper [63], Chiarandini *et al.* proposed a HA that combines a constructive procedure for achieving the feasibility, followed by LS and simulated annealing for satisfying the soft constraints.

- ACO: In their paper [166], Nothegger *et al.* proposed an ACO algorithm in conjunction with a local improvement search routine.

- the LS based algorithm (LSA): Müller [162] used an LSA with routines taken from the Constraint Solver Library. Various neighbourhood search algorithms are also used to eliminate violations of hard and soft constraints.

Table 6.5 gives the comparison results, where the term "Df" represents the distance to feasibility and "$BSCP$" means the best SCP value over 10 runs. One thing to note is that the ITC-2007 competition results of other algorithms were based on 10 runs per instance. For fair comparison, we also show our results based on 10 runs per instance here.

From Table 6.5, it can be seen that our proposed HGATS achieved feasibility on all of the problem instances over 10 runs. It can also be seen that the chance of HGATS achieving optimal solutions is higher than other algorithms. HGATS achieved the optimal solution on 10 out of 24 problem instances. It gives the best result on problem instances 2007-4, 2007-5, 2007-16, and 2007-20 over all the compared algorithms.

Table 6.5: Comparison of algorithms from the literature on different ITC-2007 PECTP instances

	HGATS		CTI		MMA		HA		ACO		LSA	
PECTP	*Df*	*BSCP*	*Df*	*BSCP*	*Df*	*BSCP*	*Df*	*BSCP*	*Df*	*BSCP*	*Df*	*BSCP*
2007-1	0	523	0	61	0	571	0	1482	0	15	0	1861
2007-2	0	342	0	547	0	993	0	1635	0	0	39	2174
2007-3	0	379	0	382	0	164	0	288	0	391	0	272
2007-4	0	234	0	529	0	310	0	385	0	239	0	425
2007-5	0	0	0	5	0	5	0	559	0	34	0	8
2007-6	0	0	0	0	0	0	0	851	0	87	0	28
2007-7	0	0	0	0	0	6	0	10	0	0	0	13
2007-8	0	0	0	0	0	0	0	0	0	4	0	6
2007-9	0	1102	0	0	0	1560	0	1947	0	0	162	2733
2007-10	0	515	0	0	0	2163	0	1741	0	0	161	2697
2007-11	0	246	0	548	0	178	0	240	0	547	0	263
2007-12	0	241	0	869	0	146	0	475	0	32	0	804
2007-13	0	0	0	0	0	0	0	675	0	166	0	285
2007-14	0	0	0	0	0	1	0	864	0	0	0	110
2007-15	0	0	0	379	0	0	0	0	0	0	0	5
2007-16	0	0	1	91	0	2	0	1	0	41	0	132
2007-17	0	0	0	1	0	0	0	5	0	68	0	72
2007-18	0	0	0	0	0	0	0	3	0	26	0	70
2007-19	0	121	267	1862	0	1824	0	1868	0	22	197	2268
2007-20	0	304	0	1215	0	445	0	596	665	2735	0	878
2007-21	0	36	0	0	0	0	0	602	0	33	0	40
2007-22	0	1154	0	0	0	29	0	1364	0	0	0	889
2007-23	0	963	0	430	0	238	0	688	11	1275	0	436
2007-24	0	274	0	720	0	21	0	822	0	30	0	372

From the results, we can see that the GS strategy and appropriate combination of local and tabu search approaches can help HGATS to optimise the objective values and that HGATS gives better results for the ITC-2007 PECTP instances in comparison with other population-based and heuristics-based algorithms taken from the literature.

6.4 Chapter Summary

This chapter presented a two-phase hybrid approach, which combines a guided search genetic algorithm (GSGA) and a tabu search heuristic, to solve the post-enrolment course timetabling problem (PECTP). In the GSGA, a GS strategy uses a data structure to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and that has a zero penalty value. This data structure is used to guide the generation of offspring into the next population. The main advantage of this data structure lies in the fact that it provides parts of former good solutions, which otherwise would have been lost in the selection process, and reuses the stored information in the following generations. This can enable the algorithm to quickly retrieve the best solutions corresponding to the previous and new populations. In the proposed HGATS algorithm, two LS techniques are used to improve the quality of individuals through searching six neighbourhood structures. At the second phase of HGATS, a TS heuristic is used to further improve the best solution obtained by GSGA in the first phase.

In order to test the performance of HGATS for the PECTP, experiments were carried out to analyse the sensitivity of parameters and the effect of the guided search strategy for the performance of HGATS based on a set of benchmark ITC-2007 PECTP instances. The experimental results of HGATS were also compared with several state-of-the-art methods from the literature on these benchmark ITC-2007

PECTP instances. The experimental results show that the proposed hybrid approach is competitive and work well across all test PECTP instances in comparison with other approaches studied in the literature.

In this chapter, we also analysed the Df values of solutions obtained by algorithms on different PECTP instances. If the Df value of a solution is zero, it means the corresponding algorithm is able to give a feasible solution while trying to minimise the soft constraints. However, if we want to minimise the soft constraints while satisfying all hard constraints, why cannot a UCTP be effectively solved as a MOOP? What would be the better way to solve the problem? As a single-objective or multi-objective optimisation problem? The next chapter of this thesis aims to address the above questions.

Chapter 7

Multi-Objective Approaches to University Course Timetabling

7.1 Introduction

Up until this point in the thesis, we have tackled the university course timetabling problem (UCTP) as a single-objective optimisation problem. As discussed in Chapter 2, for more than forty years, the UCTP has been studied as a single-objective optimisation problem and yet there is no formulated general solving technique for the problem due to the complex and highly-constrained nature of the problem. It is very difficult to find a general and effective solution for timetabling owing the diversity of the problem, the variance of constraints, and special objectives from university to university according to their own particular characteristics.

In this part of our research, we will now switch our effort over to solve the problem as a multi-objective optimisation problem (MOOP) in the hope of dealing with multiple and conflicting objectives simultaneously for the UCTP. In this chapter, we first develop a framework of multi-objective evolutionary algorithms (MOEAs) to solve the multi-objective university course timetabling problem (MOUCTP), as defined in Section 3.4 of Chapter 3, based on the guided search (GS) and local search (LS) strategies developed in previous chapters. we then present several new MOEAs instantiated from the proposed framework using some typical MOEAs to solve the MOUCTP. The proposed MOEAs are validated using several problem instances and performance metrics taken from the literature on MOEAs for general MOOPs.

The chapter is organised as follows. Section 7.2 presents the proposed framework of MOEAs for the MOUCTP, where the basic structure as well as the detailed components are described. Section 7.3 describes in detail several MOEAs that are instantiated from our framework to solve the MOUCTP. Experimental results of comparing the proposed MOEAs and corresponding algorithms on a set of benchmark problem instances are reported and discussed in Section 7.4. Section 7.5 concludes this chapter.

7.2 The Framework of MOEAs for the MOUCTP

As mention earlier in the thesis, most of the research carried out on the UCTP was based on single-objective optimisation problems. Researchers combined multiple criteria into a single scalar value, and then minimised the weighted or unweighted sum of constraints violations as the only objective function. However, the UCTP is inherently based on many different objectives or constraints, such as minimising the number of consecutive classes, minimising the occurrence of classes in the last time slot, maximising the usage of resources, and many more [83]. Consequently, it is very difficult to satisfy all the constraints as these vary from university to university. Hence, this complexity requires that a UCTP must be treated in such a way, that by removing/adding constraints should not effect the solution methods, hence all these problems lead the UCTP to being solved as an MOOP.

In order to solve the UCTP as an MOOP, this chapter investigates the integration of the GS strategy and two LS strategies (as described in Chapter 4 and Chapter 5) into MOEAs to enhance their performance for the MOUCTP. A framework for integrating the GS and LS strategies into MOEAs for the MOUCTP is introduced. The framework is then instantiated onto several widely-used MOEAs, including the elitist non-dominated sorting GA (NSGA-II) [80], the ϵ-multi-objective evolutionary algorithm (ϵ-MOEA) [81], the improved SPEA (SPEA-II) [221], and the Pareto archived evolution strategy (PAES) [136], to construct corresponding new MOEAs to solve the MOUCTP.

In the following sub-sections, we first introduce the basic structure of the proposed framework of MOEAs for the MOUCTP. We will then describe the key components of the proposed framework, including the involved LS strategies involved, the GS strategy, and common genetic operators, respectively.

7.2.1 Basic Structure

The framework of proposed approaches is based on the integration of the GS strategy developed in Chapter 5 and LS strategies into general MOEAs to solve the MOUCTP. Hence, the structure of an instantiated MOEA is similar to the MOEA used to instantiate the framework. In the framework, the GS strategy is used to create offspring to increase the rate of highly-fit individuals in the population while the LS strategies are used to further improve individuals to find non-dominated solutions. Here, we describe the common structure that is used in all proposed MOEAs (if MOEAs use them).

All proposed algorithms are based on GAs. First, we initialise a population of solutions randomly. Each random solution is created via assigning a random time slot for each event according to a uniform distribution and applying the matching algorithm to allocate a room for the event. As random initial solutions have minimal chances of producing a feasible solution, two LS strategies are used to convert these individuals into feasible or near feasible solutions. The LS strategies use four neighbourhood structures, which will be described in Section 7.2.2, to move events among time slots, and then uses the matching algorithm to allocate rooms to events and time slots. After the initialisation of the population, individuals are evaluated by their objective values and evolve according to the particular MOEA used.

The proposed framework uses three data structures, denoted as MEM_i $(i = 1, 2, 3)$, which will be described in Section 7.2.3.1. These data structures are re-constructed regularly and are used to guide the generation of offspring for the following generations. In each generation, children are first generated either by using the data structures (see Section 7.2.3.2) or by crossover, depending on a probability γ. For the crossover operation, parents are first selected as per MOEA, and then a child is created by exchanging the time slots between parents and allocating rooms to events in each non-empty time slot. After crossover, mutation is performed on each child with a probability P_m. After that, the child population is treated as per MOEA. The iteration continues until a termination condition holds, e.g., a time limit t_{max} is reached.

7.2.2 The LS Strategies

In the EA and MOEA literature, LS strategies are widely used to enable solutions to search around their local areas in the search space and have shown promising effects in many cases. In our proposed framework of MOEAs for the MOUCTP, we also include two LS strategies. The first LS strategy, denoted *LS5*, is based on four neighbourhood structures. The first three are the same as $N1$, $N2$, and $N3$, which have been defined previously in Section 4.2.1.1 in Chapter 4. The fourth one is denoted $N7$ (for denotation consistency with the three neighbourhood structures $N4$, $N5$, and $N6$ defined in Section 6.2.1.6 in Chapter 6), and is defined as follows:

- $N7$: the neighbourhood defined by an operator that swaps the time slots of two consecutive events with the time slots of another two consecutive events.

Algorithm 23 Local Search Strategy 5 (LS5)

```
1:  input : Individual I selected from the population
2:  while Termination condition not reached do
3:    for $i = 1$ to the total number of events do
4:      if event $i$ is infeasible then
5:        if there is untried move left then
6:          calculate the next move (first in $N1$, then in $N2$, then in $N3$, and finally in
            $N7$)
7:          apply the matching algorithm to the time slots affected by the move and
            delta-evaluate the result.
8:          if the move reduces hard constraint violation then
9:            make the move
10:         end if
11:       end if
12:     end if
13:   end for
14:   if any hard constraint violations remain then
15:     terminate LS3
16:   else
17:     for $i = 1$ to total number of events do
18:       if event $i$ has soft constraint violation then
19:         if there is untried move left then
20:           calculate the next move (first in $N1$, then in $N2$, then in $N3$, and finally
              in $N7$)
21:           apply the matching algorithm to the time slots affected by the move and
              delta-evaluate the result
22:           if the move reduces soft constraints violation then
23:             make the move
24:           end if
25:         end if
26:       end if
27:     end for
28:   end if
29: end while
30: output : A possibly improved individual I
```

The pseudo-code of LS5 is shown in Algorithm 23. After a child solution is generated once the mutation operation or an initial solution is randomly created, LS5 is applied to the solution for possible improvement. LS5 works on all events of a solution in a similar way to LS1 as described in Section 4.2.1.1 in Chapter 4. The only difference (shown in italics in Algorithm 23) is that LS5 uses an additional neighbourhood structure $N7$ after $N1$, $N2$, and $N3$ in the LS operation. This neighbourhood structure tries to explore the search space quickly. The rest of the LS5 works the same as mentioned in Algorithm 8.

As LS1, LS5 also works on all events of a solution in two steps. In the first step (Lines 3-13), LS5 checks the hard-constraint violations of each event while ignoring its soft-constraint violations. If there are hard-constraint violations for an event, LS1 tries to resolve them by applying moves in the neighbourhood structures $N1$, $N2$, $N3$, and $N7$ consecutively as follows. First, we try to move the event to the next time slot, then the next, then the next, etc. If this search in $N1$ fails, we then search in $N2$ by trying to swap the event with the next one in the list, then the next one, and so on. If the search in $N2$ also fails, we try a move in $N3$ by using a different permutation formed by the event with the next two events, then with the next two, and so on. If the search in $N3$ also fails, we try a move in $N7$ by replacing the time slots of two consecutive events with the time slots of another two consecutive events, then the next two, and so on, until a termination condition is reached, e.g., an improvement is reached or the maximum number of steps s_{max} is reached.

After each move, we apply the matching algorithm to the time slots affected by the move and try to resolve the room allocation disturbance and delta-evaluate the result of the move (i.e., calculate the hard- and soft-constraint violations before and after the move). If there is no untried move left in the neighbourhood for an event, LS5 continues to the next event. After applying all neighbourhood moves on each event, if there is still any hard-constraint violation, then LS5 will stop; otherwise, LS5 will perform the second step (lines 17-27 in Algorithm 23).

In the second step, after reaching a feasible solution, LS5 performs a similar process as in the first step on each event to reduce its soft-constraint violations. For each event, LS5 tries to make moves in the neighbourhood $N1$, $N2$, $N3$, and/or $N7$ consecutively without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta-evaluated.

The second LS strategy used in our framework is the same as the LS strategy LS2, as described previously in Section 4.2.1.2 of Chapter 4. LS2 is used immediately after LS5 on a solution. It chooses a high penalty time slot that may have a large number of events involving hard and soft constraint violations and tries to reduce the penalty values of involved events.

7.2.3 The GS Strategy

7.2.3.1 Data Structures MEM_i $(i = 1, 2, 3)$

All comparative studies on MOEAs agree that elitism and diversity preservation mechanisms improve the performance of MOEAs for MOOPs [16]. In the proposed framework of MOEAs for the MOUCTP, we also create extra memories (data structures) to store the best parts of individuals in the current population in order to

Algorithm 24 *ConstructMEM3*(P) – Constructing data structures

```
1: input: The input population $P$ of size $N$
2: $Q \leftarrow$ select the best $\alpha \times N$ individuals in $P$
3: for each individual $I_j$ in $Q$ do
4:    for each objective $i$ do
5:       if $f_i(I_j) = 0$ then
6:          for each event $e_k$ in $I_j$ do
7:             calculate the penalty value of event $e_k$ from $I_j$
8:             if $e_k$ is feasible (i.e., $e_k$ has zero constraint violation) then
9:                add the pair of room and time slot $(r_{e_k}, t_{e_k})$ assigned to $e_k$ into the list $l_{e_k}$
                   in $MEM_i$
10:            end if
11:         end for
12:      end if
13:   end for
14: end for
15: output: Updated data structures $MEM_i$ $(i = 1, 2, 3)$
```

guide the generation of offspring in the subsequent populations. We create three data structures MEM_i $(i = 1, 2, 3)$. Each data structure is the same as the data structure shown in Figure 5.1 in Section 5.2.1 of Chapter 5, but stores useful information according to one of the three objectives, i.e., MEM_i $(i = 1, 2, 3)$ is associated with the i-th objective. In MEM_i, there is a list of events and each event e_k again has a list l_{e_k} of room and time slot pairs. The data structures are regularly re-constructed, e.g., every τ generations.

Algorithm 24 shows the outline of the construction or re-construction of the data structures. When the data structures are due to be constructed or re-constructed, we first select $\alpha \times N$ best individuals in terms of dominance rankings from the population P (the unique parameter given to *ConstructMEM3*(P)) to form a set Q, where N is the population size of P. After that, for each individual $I_j \in Q$, we check its objective values. If any of its objectives, say $f_i(I_j)$, has a zero value, then

Algorithm 25 *GuidedSearch*3() – Generating a child by GS

```
1: input: The data structures $MEM_i(i = 1, 2, 3)$
2: randomly select one data structure $MEM_j$
3: $E_s :=$ randomly select $\beta * n$ events
4: for each event $e_k$ in $E_s$ do
5:     randomly select a pair of room and time slot from the list $l_{e_k}$ in $MEM_j$
6:     assign the selected pair to event $e_k$ for the child
7: end for
8: for each remaining event $e_k$ not in $E_s$ do
9:     assign a random time slot and room to event $e_k$
10: end for
11: output: A child generated using $MEM_i(i = 1, 2, 3)$
```

each event of I_j is checked by its penalty value (hard and soft constraints associated with this event). If an event has a zero penalty value, then we store the information corresponding to this event into corresponding data structure MEM_i.

7.2.3.2 Generating a Child by the GS Strategy

In the proposed framework, a child is created by the GS strategy or crossover with a probability γ. That is, when a child is to be generated, a random number $\rho \in [0.0, 1.0]$ is first generated if $\rho < \gamma$, the GS strategy is used to generate the child; otherwise, a crossover operation is used. If a child is to be created by the GS strategy, then we apply Algorithm 25.

In Algorithm 25, we first randomly select one data structure, say, MEM_j, as the base. We then select a set E_s of $\beta * n$ random events to be generated from MEM_j. Here, β is the percentage of the total number of events. After that, for each event e_k in E_s, we randomly select a pair (r_{e_k}, t_{e_k}) from the list l_{e_k} in MEM_j that corresponds to the event e_k and assign the selected pair to e_k for the child. If an event e_k in

E_s has no list l_{e_k} in MEM_j, then we randomly assign a room and a time slot from possible rooms and time slots to e_k for the child. This process is carried out for all events in E_s. For those remaining events not present in E_s, available rooms and time slots are randomly assigned to them to get a complete child.

7.2.4 Genetic Operators

7.2.4.1 Objective Functions and Constraints Handling

As we described in Section 3.4 of Chapter 3, our aim is to minimise the three kinds of soft constraints, defined as the three objective functions in the MOUCTP studied in this chapter, subject to resolving all hard constraints.

The MOUCTP is a highly-constrained optimisation problem. All proposed algorithms adopt the constraint handling mechanism used by NSGA-II [80] based on the concepts of feasibility and non-dominance when comparing solutions. A solution $\vec{a}$ is said to constrained-dominate another solution $\vec{b}$ if any of the following conditions holds.

- Solution $\vec{a}$ is feasible and $\vec{b}$ is infeasible;
- Both solutions are feasible and $\vec{a}$ dominates $\vec{b}$;
- Both solutions are infeasible, but $\vec{a}$ dominates $\vec{b}$.

7.2.4.2 Selection Mechanism

The selection scheme depends on the base MOEA that is used for instantiating the framework and there may be different choices. A typical choice is the binary tournament selection without replacement.

7.2.4.3 Crossover

After two parents are selected from the population by the selection scheme defined in the corresponding MOEA, we apply the crossover to create an offspring directly (i.e., with a crossover probability $P_c = 1$). Here, a uniform crossover operator, as shown in Algorithm 10 of Chapter 4, is used. It first randomly assigns to each event in the offspring a time slot from one of the two parents randomly and then allocates rooms to events in each non-empty time slot in the offspring, using the matching algorithm.

7.2.4.4 Mutation

A mutation operation is applied on a newly created child with a probability P_m. It applies a randomly selected neighbourhood structure $N1$, $N2$, $N3$, or $N7$ to make a move.

Algorithm 26 Guided Search NSGA-II (GSNSGA)

```
1: input: A problem instance I
2: set the generation counter $g := 0$
3: initialise a population $P_g$ of $N$ solutions
4: apply LS1 and LS2 to individuals in $P_g$
5: evaluate the individuals in $P_g$
6: assign rank and crowing distance for individuals in $P_g$
7: while the termination condition is not reached do
8:     if ($g$ mod $\tau$) == 0 then
9:         apply $ConstructMEM3(P_g)$ to construct $MEM_i$ ($i = 1, 2, 3$)
10:    end if
11:    create a child population $Q$: use $GuidedSearch3()$ or crossover with a probability $\gamma$
       to generate each child independently, followed by mutation with a probability $P_m$
12:    apply LS1 and LS2 to individuals in $Q_g$
13:    evaluate the child solutions in $Q_g$
14:    merge the child and parent populations into a combined population $R_g := P_g \bigcup Q_g$
15:    assign rank and crowding distance for individuals in $R_g$
16:    create a new population from $R_g$ based on rank and crowding distance
17:    $g := g + 1$
18: end while
19: output: Non-dominated set of solutions
```

7.3 Instantiated MOEAs for the MOUCTP

The framework proposed in the above section can be easily be instantiated onto MOEAs for general MOOPs to construct new MOEAs for solving the MOUCTP. In this section, we present four new MOEAs for the MOUCTP. These are instantiated from the framework, based on four state-of-the-art MOEAs from the literature, i.e., NSGA-II [80], PAES [136], ϵ-MOEA [81], and SPEA-II [221], respectively. In the following subsections, all proposed algorithms show our contribution in italics.

7.3.1 Guided Search NSGA-II (GSNSGA)

GSNSGA is constructed based on NSGA-II, which was introduced in [80] based on the concepts of non-dominated sorting and crowding distance. The pseudo-code of GSNSGA is shown in Algorithm 26. Initially, a random population of size N is created. After the initial or child population is generated, each solution in the initial or child population undergoes the LS strategies for potential improvement. This population is then sorted, based on the non-dominated sorting.

In the non-dominated sorting, for each individual P_i in a population, we calculate two values: the domination count n_{P_i} (the number of solutions which dominate the solution I_i) and S_{P_i} (a set of solutions that the solution P_i dominates). After that, we identify the non-dominated fronts in the population according to the domination counts of individuals. All solutions with their domination count as zero belong to the first non-dominated front and are assigned the rank value of 1 (1 is the best rank). For each solution P_i with $n_{P_i} = 0$, we visit each member r in its set S_{P_i} and reduce the domination count n_{P_r} by one. For any member r, if its domination count becomes zero, it is put into a list L, and all the members in this list belong to the second front and are assigned the rank value of 2. The above procedure is continued in the list L to find the third front, and so on, until all fronts are identified. After ranking, the crowding distance of each front of the population can be calculated. The crowding distance is used for density estimation for each individual. The crowding distance of a solution P_i within a front is the average distance of two solutions from the same front on either side of the solution along each of the objectives [80].

In GSNSGA, every τ generations, including the initial generation, the MEM_i ($i = 1, 2, 3$) data structures are constructed/reconstructed to store parts of solutions corresponding to the best individuals in the current population P_g, which will be used for the GS strategy for the next τ generations.

Then, at each generation g, a child population Q_g is created. Each child in the child population Q_g is now created either by the GS strategy or by crossover, depending on the probability γ. If the GS strategy is used, the $GuidedSearch3()$ procedure in Algorithm 25 is called; otherwise, two individuals are first selected from P_g as the parents by using the crowded tournament selection scheme (for each parent, two individuals are randomly selected and the tournament winner among them is decided according to their ranks and crowding distance values), which are then crossovered to generate a child. In both cases, the created child then undergoes a mutation operation with a probability P_m, as described in Section 7.2.4.4. Next, the child population Q_g and P_g are merged together in R_g (so, R_g has $2N$ individuals) and the rank and crowding distance values of individuals in R_g are calculated. Finally, based on the ranks and crowding distances, the best N solutions are picked up from R_g to form a new population P_{g+1} for the next generation. So, at the end of each generation, the set of non-dominated solutions so far are obtained.

This process is carried out until a termination condition e.g., a certain time limit is reached.

7.3.2 Guided Search PAES (GSPAES)

In GSPAES, we use the basic structure of PAES developed by Knowles and Corne [136]. They introduced PAES based on a (1+1)-ES. PAES uses an archive A of a fixed size to store the best solution so far during the solving process. The archive A is initially empty. As the searching progresses, good solutions are added to A and updated (due to the fixed size of the archive). The pseudo-code of GSPAES is shown in Algorithm 27.

At first, we randomly create a solution, called the parent P, apply LS strategies to it, and add it to the archive A after evaluation. Since GSPAES is basically a (1+1)-MOEA (as is PAES [136]), we use the archive A in the $ConstructMEM3()$ procedure to update MEM_i data structures. Then, at each generation g, a child C_g is created either by the GS strategy or the mutation operator with a probability γ. If the child C_g dominates P_g, it is accepted as the next parent (i.e., $P_g := C_g$) and added to A_g, and the iteration continues. If P_g dominates C_g, then C_g is discarded and the iteration continues by creating a new child by mutating P_g.

If C_g and P_g do not dominate each other, then the child C_g is compared with members in A_g. If C_g dominates any member of A_g, then the dominated solutions are eliminated from A_g, and C_g is copied to A_g and is also accepted as the new parent (i.e., $P_g := C_g$). If C_g does not dominate any member in the archive, both parent and offspring belong to the same non-dominated front to which the archive solutions belong. In this case, there are two scenarios. In the first one, if the archive

Algorithm 27 Guided Search PAES (GSPAES)

```
1: input: A problem instance I
2: set the generation counter $g := 0$
3: initialise a solution $P_g$
4: apply LS1 and LS2 on $P_g$
5: evaluate the individual $P_g$
6: copy $P_g$ to the archive $A_g$
7: while the termination condition is not reached do
8:    if $(g \ mod \ \tau) == 0$ then
9:       apply $ConstructMEM3(A_g)$ to construct $MEM_i$ $(i = 1, 2, 3)$
10:   end if
11:   apply $GuidedSearch3()$ or mutation to $P_g$ with a probability $\gamma$ to produce a child $C_g$
12:   apply LS1 and LS2 on $C_g$
13:   evaluate $C_g$
14:   if $P_g$ dominates $C_g$ then
15:      discard $C_g$
16:   else if $C_g$ dominates $P_g$ then
17:      add $C_g$ to $A_g$ and replace $P_g$ with $C_g$
18:   else if $C_g$ is dominated by any member of $A_g$ then
19:      discard $C_g$
20:   else if $C_g$ dominates some members of $A_g$ then
21:      remove those members dominated by $C_g$ from $A_g$
22:      add $C_g$ to $A_g$ and replace $P_g$ with $C_g$
23:   else
24:      if $A_g$ is not full then
25:         add $C_g$ to $A_g$
26:         if $C_g$ resides in a less crowded region of $A_g$ than $P_g$ then
27:            replace $P_g$ with $C_g$
28:         end if
29:      else
30:         if $C_g$ resides in the most crowded region of $A_g$ then
31:            discard $C_g$
32:         else
33:            replace a random member from the most crowded region of $A_g$ with $C_g$
34:            if $C_g$ resides in a less crowded region of $A_g$ than $P_g$ then
35:               replace $P_g$ with $C_g$
36:            end if
37:         end if
38:      end if
39:   end if
40:   $g := g + 1$
41: end while
42: output: Non-dominated set of solutions in $A$
```

Algorithm 28 ϵ-Guided Search MOEA (ϵ-GSMOEA)

```
1: input: A problem instance I
2: set the generation counter $g := 0$
3: initialise a population $P_g$ of $N$ solutions
4: apply LS1 and LS2 on solutions in $P_g$
5: evaluate the individuals in $P_g$
6: copy non-dominated solutions in $P_g$ to the archive $A_g$
7: while the termination condition is not reached do
8:    if $(g \ mod \ \tau) == 0$ then
9:       apply $ConstructMEM3(P_g)$ to construct $MEM_i$ $(i = 1, 2, 3)$
10:   end if
11:   create a set $C$ of children by using $GuidedSearch3()$ or crossover with a probability $\gamma$
12:   apply mutation to each child in $C$ with a probability $P_m$
13:   apply LS1 and LS2 on each child in $C$
14:   evaluate each child in $C$
15:   update $P_g$ according to $C$ using the dominance measure
16:   update $A_g$ according to $C$ using the $\epsilon$-dominance measure
17:   $g := g + 1$
18: end while
19: output: Non-dominated set of solutions in $A$
```

is not full, C_g is copied to A_g, and is accepted as the parent for the next generation if it is in the less crowded region[1] than the parent in the parameter space among the members of the archive. In the second scenario (i.e., the archive is full), if C_g resides in the most crowded region in the parameter space among the members of the archive, it is discarded; otherwise, C_g will replace one random member of A_g from the most crowded region, and is accepted as the parent for the next generation if it is in the less crowded region than the parent in the parameter space among the members of the archive.

[1]See [136] for the definition of regions constructed by the solutions in the archive. A region is more crowded if it has more archive solutions within it.

7.3.3 ϵ-Guided Search MOEA (ϵ-GSMOEA)

The ϵ-MOEA by Deb *et al.* [81] is also used to instantiate our framework to construct the ϵ-GSMOEA for solving the MOUCTP. The pseudo-code of ϵ-GSMOEA is shown in Algorithm 28. In ϵ-MOEA and ϵ-GSMOEA, the search space is divided into a number of grids (or hyper-boxes) and diversity is maintained by ensuring that a hyper-box can be occupied by only one solution [81]. There are two co-evolving populations: one EA population P and one archive population A. The EA population P is initialised with N random solutions, followed by LS strategies and then evaluated. All non-dominated individuals in P are copied into the archive A. In ϵ-GSMOEA, the MEM_i $(i = 1, 2, 3)$ data structures are created from P for the initial generation and thereafter for every τ generations.

In each generation, a set of children $C = \{C_1, C_2, \cdots, C_\lambda\}$ is created via either the GS strategy or crossover depending on the probability γ. If the set of children are to be created by crossover, a child is created as follows. One parent is chosen from A randomly. Another parent is chosen from P as follows: two individuals are randomly selected from P and compared. If one solution dominates the other, the dominating solution is chosen; otherwise, we choose one randomly as the parent. Once two parents are selected, they are mated to create a set of children $C = \{C_1, C_2, \cdots, C_\lambda\}$ via crossover operator. After a child is created, we apply mutation on it, followed by LS1 and LS2 operations.

Next, both P and A are updated according to C as follows. To update P, we compare each child C_i, $i = 1, \cdots, \lambda$, with all members in P. If C_i dominates any member in P, then it replaces that member. Otherwise, if any member of P dominates C_i, then C_i is not included in P. If both of the above tests fail, then C_i replaces a random individual from P.

For updating A, each child C_i is compared with each member of A using the concept of ϵ-*dominance* [81]. In this process, each individual in A is assigned an identification array $\vec{B} = \{B_1, B_2, \cdots, B_M\}$, where M is the total number of objectives. The identification array $\vec{B}$ is defined as follows:

$$B_j(\vec{f}) = \lfloor (f_j - f_j^{min})/\epsilon_j \rfloor \tag{7.1}$$

where f_j is the j-th objective value of the individual, f_j^{min} is the minimum possible value of the j-th objective, and ϵ_j is the allowable tolerance in the j-th objective, below which two values are not significant to the user [140]. The identification arrays divide the whole objective space into hyper-boxes with the size ϵ_j in the j-th objective. The child C_i enters A according to its position in the hyper-boxes as described in [81]. The above procedure is continued until a termination condition is met (e.g., the maximal allowable time is reached). Finally, we arrive at the solutions obtained in A.

Algorithm 29 Guided Search SPEA-II (GSSPEA)

```
1: input: A problem instance $\mathbf{I}$
2: set the generation counter $g := 0$
3: initialise a population $P_g$ of $N$ solutions
4: apply LS1 and LS2 on individuals in $P_g$
5: create an empty archive $A_g$ of size $\bar{N}$
6: while the termination condition is not reached do
7:     if ($g$ mod $\tau$) == 0 then
8:         apply $ConstructMEM3(P_g)$ to construct $MEM_i$ ($i = 1, 2, 3$)
9:     end if
10:    calculate the fitness values of individuals in $P_g$ and $A_g$
11:    merge $P_g$ and $A_g$
12:    perform the environmental selection on the merged population to form $A_g$
13:    perform the mating selection on $A_g$ to form a mating pool
14:    create the child population $P_g$: use $GuidedSearch3()$ or crossover of individuals from
       the mating pool with a probability $\gamma$ to generate each child independently, followed
       by mutation with a probability $P_m$
15:    apply LS1 and LS2 on each child in $P_g$
16:    $g := g + 1$
17: end while
18: output: Non dominated set of solution
```

7.3.4 Guided Search SPEA-II (GSSPEA)

GSSPEA is the fourth MOEA that is instantiated in this chapter from our proposed framework based on SPEA-II introduced by Ziztler *et al.* [221]. In SPEA-II [221] and GSSPEA, we used two populations: one regular population P of size N and one archive A of size $\bar{N}$. A represents the external or archive set that will contain the non-dominated solutions, and some dominated solution if the number of non-dominated solutions is less than $\bar{N}$, during the searching process. Algorithm 29 shows the pseudo-code of GSSPEA. As in the above three instantiated MOEAs, GSSPEA differs from SPEA-II [221] only in the GS and LS strategies.

In Algorithm 29, we first initialize P with N random solutions and create an empty archive A. After applying LS strategies on each individual in P, we then construct (and thereafter re-constructed every τ generations) the MEM_i ($i = 1, 2, 3$) data structures to guide the creation of offspring into next generations. Each individual in P is evaluated according to its objective values and then assigned a fitness value as follows. In order to calculate the fitness value of an individual i in P, it is first assigned a strength value $S(i)$, which is defined as follows:

$$S(i) = \mid \{ j \mid j \in P_g + A_g \wedge i \succ j \} \mid \tag{7.2}$$

where "$\mid \; \mid$" denotes the cardinality of a set, "+" stands for multiset union, and "$\succ$" corresponds to the Pareto dominance relation. Based on the strength value, the raw fitness $R(i)$ of individual i is calculated as follows:

$$R(i) = \sum_{j \in P_g + A_g, j \succ i} S(j) \tag{7.3}$$

Hence, this raw fitness value is calculated using the strengths of the dominators in both the archive and population sets.

In the event that individuals have the same raw fitness values, a density estimation technique is used. The specific estimation technique is a simple inverse distance of the k-th nearest neighbour [221]. For each individual i, the distances in the objective space to all individuals j in the archive and population are calculated and stored in a list. After sorting the list in the increasing order, the k-th element gives the distance sought, denoted as σ_i^k, where k is equal to the square root of the sample size, i.e., $k = \sqrt{N + \bar{N}}$. The density $D(i)$ corresponding to individual i is then calculated as follows:

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{7.4}$$

Finally, the fitness value $F(i)$ of individual i is calculated as:

$$F(i) = R(i) + D(i) \tag{7.5}$$

After finding the fitness of individuals, we merge the archive and population and perform environmental selection to form a new archive for the next generation from the merged population. In the process of environmental selection, all non-dominated individuals in the merged population are selected to re-fill the archive. If the number of non-dominated individuals is equal to the predetermined archive size $\bar{N}$, we copy them to the archive and stop the archive update operation; otherwise, there can be two situations: If the archive is bigger than the non-dominated set in size, we copy all non-dominated individuals and some dominated individuals from the previous archive and population into the archive up to the size $\bar{N}$; otherwise, if the archive is smaller than the non-dominated set, then a truncation method is used to remove individuals from the non-dominated set one by one as follows. The individual which has the minimum distance to another individual in the remaining non-dominated set is chosen for removel. If there is more than one individual with the same minimum distance, the tie is broken by considering the second smallest distances, and so on. The process continues until there are $\bar{N}$ individuals left in the non-dominated set, which are then copied to form the archive.

After the environmental selection, only members of the archive participate in the mating selection process using the binary tournament selection with replacement to create the mating pool. Child population is created by using the GS approach or the crossover operator. In both cases, the created population then undergoes a mutation operation with a probability P_m followed by local search techniques. This process is continued until the termination condition is reached.

7.4 Experimental Study

In this section, we experimentally investigate the performance of the MOEAs instantiated in this chapter from our proposed framework and their corresponding original versions for the MOUCTP. The programs are coded in GNU C++ with version 4.1 and run on a 3.20 GHz PC. Since there is no formulation or benchmark of the MOUCTP, many researchers have tested their proposed techniques on real data instances. Due to this deficiency, we study multi-objective instances of the MOUCTP that are adapted from well-known single-objective instances that were proposed in [4] (described in Section 3.2 of Chapter 3) and which have received a lot of attention from researchers [13, 181].

7.4.1 Parameter Setting

In Section 3.2.2 in Chapter 3, we discussed the problem instances of three different groups for the UCTP. We run our proposed approaches on them. After some preliminary results, the key parameters for the proposed MOEAs were set as follows: $\alpha = 0.2$, $\beta = 0.4$, $\gamma = 0.6$, $\tau = 20$, and $P_m = 0.6$. We tried to use the same values for other parameters that have already been used for the state-of-the-art MOEAs in the literature. The population size N was set to 48 for NSGA-II, GSNSGA, SPEA-II, and GSSPEA, 1 for PAES and GSPAES, and 40 for ϵ-MOEA and ϵ-GSMOEA. The archive size was set to 10 for PAES and GSPAES, 40 for SPEA-II and GSSPEA, and was non-fixed for ϵ-MOEA and ϵ-GSMOEA. For PAES and GSPAES, we used the depth value 5, which means the number of recursive sub-divisions of the objective space carried out in order to divide the objective space into a grid (this value is the best for 3 objectives, according to [136]). For ϵ-MOEA and ϵ-GSMOEA, the minimum possible value of the j-th objective f_j^{min} is 0, the parameter λ was set to 2, and the minimum allowable tolerance ϵ was randomly set to 5, 10, 15 for the small, medium, and large problem instances, respectively, according to the search space, a larger ϵ value for a larger search space.

In the initialisation of the population, the maximum number of steps per LS operation s_{max} was set to different values for different problem instances, namely 300 for small instances, 1500 for medium instances, and 2500 for the large instance, respectively. There were 20 runs of an algorithm for each problem instance. For each run, the maximum run time t_{max} was set to 100 seconds for small instances, 1000 seconds for medium instances, and 10000 seconds for the large instance based on the fact that a larger dataset contains more conflicting constraints as compared to smaller datasets and therefore requires more processing time.

As in previous chapters, in all tables and figures reported in this chapter, "S1" represents small problem instance 1, "S2" represents small problem instance 2, and so on, "M1" represents medium problem instance 1, "M2" represents medium problem instance 2, and so on, and "L" represents the large problem instance. In all figures, the scale was set according to objective function values and is not fixed.

7.4.2 Performance Measures

The performance measure for MOEAs may include many aspects, such as the quality of the outcome or scalability, etc. Here, the quality of solutions is investigated. As the true Pareto front of the problems is unknown, we use two performance measures, the hypervolume or S matric [219] and the D metric [219], which are not based on the true Pareto front. The first measure concerns the size of the objective space which is covered by a set of non-dominated solutions. The higher the value, the larger the dominated volume in the objective space and hence the better an algorithm's performance. The hypervolume measure or S metric was originally proposed by Zitzler and Thiele [219], who called it the size of the space covered by a set of non-dominated solutions or the size of the dominated space. Zitzler and Thiele noted that this measure prefers convex regions to non-convex ones [219]. Coello *et al.* [68] described it as the Lebesgue measure Λ of the union of hypercubes a_i defined by a non-dominated point m_i and a reference point x_{ref} as follows:

$$S(M) := \Lambda(\{\bigcup_i a_i \mid m_i \in M\}) = \Lambda(\bigcup_{m \in M} \{x \mid m \prec x \prec x_{ref}\}) \tag{7.6}$$

Hypervolume has been used in several comparative studies of MOEAs to measure the population covered by the Pareto front, e.g., see [81, 219, 221]. The D metric measure between two non-dominated sets A and B gives the relative size of the region in the objective space that is dominated by A but not by B, and vice versa. It also gives information about whether either set totally dominates the other set, e.g., $D(A, B) = 0$ and $D(B, A) > 0$ means that A is totally dominated by B. Since in this chapter the focus is on finding the Pareto optimal set rather than obtaining a uniform distribution over a trade-off surface, we do not consider the online performance of MOEAs but consider the offline performance. Hence, the Pareto optimal set regarding all individuals generated over all generations is taken as the output of an MOEA. The performance of a particular algorithm on a test problem was calculated by averaging over all 20 runs.

Table 7.1: Feasibility ratio achieved by algorithms

Prob.	NSGA-II	GSNSGA	PAES	GSPAES	ϵ-MOEA	ϵ-GSMOEA	SPEA-II	GSSPEA
S1	100%	100%	100%	100%	100%	100%	100%	100%
S2	100%	100%	100%	100%	100%	100%	100%	100%
S3	100%	100%	100%	100%	100%	100%	100%	100%
S4	100%	100%	100%	100%	100%	100%	100%	100%
S5	100%	100%	100%	100%	100%	100%	100%	100%
M1	75%	100%	100%	100%	60%	100%	100%	100%
M2	85%	100%	100%	100%	75%	100%	100%	100%
M3	90%	100%	100%	100%	80%	100%	100%	100%
M4	100%	100%	100%	100%	80%	100%	100%	100%
M5	80%	100%	100%	100%	65%	100%	100%	100%
L	30%	100%	85%	100%	20%	95%	82%	100%

7.4.3 The Effect of Different Components of Proposed Algorithms

We carried out experiments with the four MOEAs instantiated from our proposed framework and the corresponding base MOEAs, i.e., NSGA-II and GSNSGA, PAES and GSPAES, ϵ-MOEA and ϵ-GSMOEA, SPEA-II and GSSPEA. Table 7.1 presents the percentage of feasible solutions obtained by all algorithms over 20 runs. Table 7.2 shows the best objective value, average objective value, and standard deviation values of all algorithms on the small, medium, and large problem instances. Figure 7.1 shows the performance of all algorithms in box-plots.

From the figure and tables, we can see the effect of the GS and LS strategies integrated within different MOEAs on the objective function values. In the following sub-sections, we discuss these results and analyse the effect of GS and LS strategies on MOEAs. We also discuss the behaviour of different MOEAs.

Table 7.2: Performance of algorithms on different problem instances

Alg.	UCTP	Best			Average			Stdv		
		f1	f2	f3	f1	f2	f3	f1	f2	f3
NSGA-II	S1	4	5	59	9.43	23.22	87.61	2.23	11.82	11.72
	S2	5	3	72	9.17	21.83	90.94	2.09	9.01	10.01
	S3	5	3	72	8.52	24.42	85.79	2.48	12.31	8.75
	S4	4	1	91	8.88	7.54	111.74	2.07	4.6	13.11
	S5	6	22	52	9.71	48.54	72.06	1.96	15.01	10.82
	M1	38	249	61	44.24	312.78	81.94	2.82	28.29	8.27
	M2	38	277	61	44.67	317.25	82.46	2.93	19.93	8.36
	M3	38	277	58	44.06	346.92	80.94	3.5	26.95	8.47
	M4	38	234	66	44.23	309.79	80.73	2.87	30.09	8.54
	M5	38	234	66	44.23	309.79	80.73	2.87	30.09	8.54
	L	39	497	131	44.48	601.88	160.54	3.52	49.55	12.65
GSNSGA	S1	0	0	0	1.33	6.74	9.91	0.94	3.9	4.66
	S2	0	0	0	1.63	5.75	5.9	1.35	4.12	3.51
	S3	0	0	0	0.65	2.06	7.38	0.76	2.17	5.51
	S4	0	0	0	1.02	1.14	20.46	0.93	1.48	8.86
	S5	0	0	0	1.52	2.04	15.1	1.52	2.22	6.84
	M1	3	95	15	8.74	138.76	32.52	3.75	23.25	11.55
	M2	7	104	15	13	176.6	21	2.73	24.68	2.73
	M3	1	95	5	6.9	145.81	16.69	2.33	20.29	2.88
	M4	0	38	2	7.15	88.81	22.38	5.36	20.29	15.44
	M5	5	94	25	23.15	150.4	43.15	9.72	25.71	9.72
	L	30	221	89	39.72	345.94	124.64	5.46	73.62	21.1
PAES	S1	0	0	3	0	0	8.2	0	0	3.43
	S2	0	0	5	0	0	16.8	0	0	8.04
	S3	0	0	3	0	0	9.5	0	0	5.35
	S4	0	0	0	0	1.2	8.4	0	1.69	3.86
	S5	0	0	0	0.1	1.2	3.3	0.32	1.87	2.67
	M1	23	139	10	31.4	147.1	14.2	8.6	5.69	3.55
	M2	29	167	12	35.09	192.64	17.36	4.11	22.17	4.15
	M3	35	225	15	37.83	283.42	31.33	2.59	39.66	8.82
	M4	37	182	7	38.17	213.67	17.25	1.34	19.54	6.36
	M5	26	151	17	33.9	219.7	35.3	7.53	72.29	18.67
	L	42	546	84	44.65	716.24	135.71	1.58	85.02	28.87
GSPAES	S1	0	0	0	0	0	0.4	0	0	0.7
	S2	0	0	0	0.1	0.2	0	0.32	0.63	0
	S3	0	0	0	0	0	0	0	0	0
	S4	0	0	0	0	0	1.8	0	0	2.49
	S5	0	0	0	0	0	0	0	0	0
	M1	19	60	0	23.2	90.4	3	4.1	22.84	2.21
	M2	18	82	3	17.09	80.91	4.45	0.79	8.65	1.91
	M3	9	65	2	3	8.3	1	8.61	10.92	4.36
	M4	20	50	3	27.4	66.2	5.6	4.6	14.23	1.51
	M5	14	42	2	22.8	58.6	4.5	4.66	12.89	1.51
	L	21	234	5	30.36	275.18	10.45	4.01	24.24	2.98
ϵ-MOEA	S1	5	3	72	9.17	21.83	90.94	2.09	9.01	10.01
	S2	5	9	70	9.23	32.42	86.06	1.94	11.84	8.88
	S3	4	2	75	8.73	14.98	96.19	2.61	8.65	10.23
	S4	3	0	71	7.3	5.02	94.86	2.57	3.61	14.21
	S5	4	14	37	4.4	15.34	28.86	2.31	12.83	8.84
	M1	38	247	60	44.35	312.73	79.03	2.96	24.72	8.89
	M2	39	241	61	44.35	312.73	79.03	2.97	22.6	9.85
	M3	37	245	67	44	298.89	83.67	2.87	19.82	8.72
	M4	37	263	67	44	313.56	81.28	2.98	27.83	9.55
	M5	36	213	55	44.67	275.52	85.19	2.87	19.82	8.72
	L	39	522	121	43.57	601.66	163.34	2.78	35.23	14.9
ϵ-GSMOEA	S1	4	3	52	8.7	11.2	68.2	2.87	5.27	11.56
	S2	6	10	43	9.34	20.01	65	1.96	11.57	7.8
	S3	3	2	56	9.39	11.61	76.5	2.78	9.68	13.26
	S4	3	0	40	6.45	4.45	71.73	2.3	2.54	19.32
	S5	1	8	14	6.67	30.11	35.56	3.01	14.19	12.01
	M1	38	176	29	45.23	210.9	53.03	3.44	24.62	9.77
	M2	40	236	39	43.63	271.21	60.26	2.85	26.56	8.03
	M3	21	195	37	42.9	266.54	58.05	5.78	38.54	12.65
	M4	37	146	63	2.87	26.52	8.35	44.45	199.1	78.94
	M5	22	140	55	42.35	196	70.05	5.89	34.47	8.19
	L	11	446	71	41.39	573.21	101.39	18.02	52.99	18.02
SPEA-II	S1	0	0	37	1.26	2.82	55.3	1.08	3.4	13.39
	S2	0	0	0	4.38	12.86	62.72	2.01	6.45	20.49
	S3	1	0	38	2.6	1.54	51.82	1.25	1.62	8.72
	S4	0	0	38	0.44	1.24	41.64	0.5	1.29	2.2
	S5	2	16	43	5.56	33.1	55.36	2.15	13.55	8.26
	M1	10	71	22	24.46	128.08	42.62	8.84	27.83	12.86
	M2	5	105	35	18.12	152.46	56.46	10.45	28.65	14.7
	M3	7	94	46	26.64	192.14	60.16	14.5	72.48	12
	M4	4	80	31	32	209.12	50.6	14.23	57.98	13.5
	M5	5	57	42	31.92	172.3	60.12	14.65	92.91	17.93
	L	28	370	125	33.42	494.46	152.74	4.45	71.25	17.86
GSSPEA	S1	0	0	0	0.52	1.66	13.9	0.76	2.09	4.28
	S2	0	0	0	1.94	2.28	29.3	1.78	2	15.62
	S3	0	0	0	1.24	1.24	13.08	1.3	2.33	4.76
	S4	0	0	9	0.36	1.28	10.46	0.48	2.3	0.71
	S5	0	0	9	1.42	2.02	13.88	1.28	2.61	4.63
	M1	3	68	35	7.36	105.08	53.48	2.21	36.48	12.73
	M2	8	70	32	11.08	84.82	43.88	2.35	10.04	7.84
	M3	6	80	24	20.94	116.32	37.98	8.53	36.79	16.07
	M4	3	67	35	20.26	153.94	47.3	18.47	98.7	9.73
	M5	2	43	38	17.02	93.04	50.16	10.76	32.44	10.96
	L	35	217	55	39.56	285.7	82.74	3.54	34.72	17.86

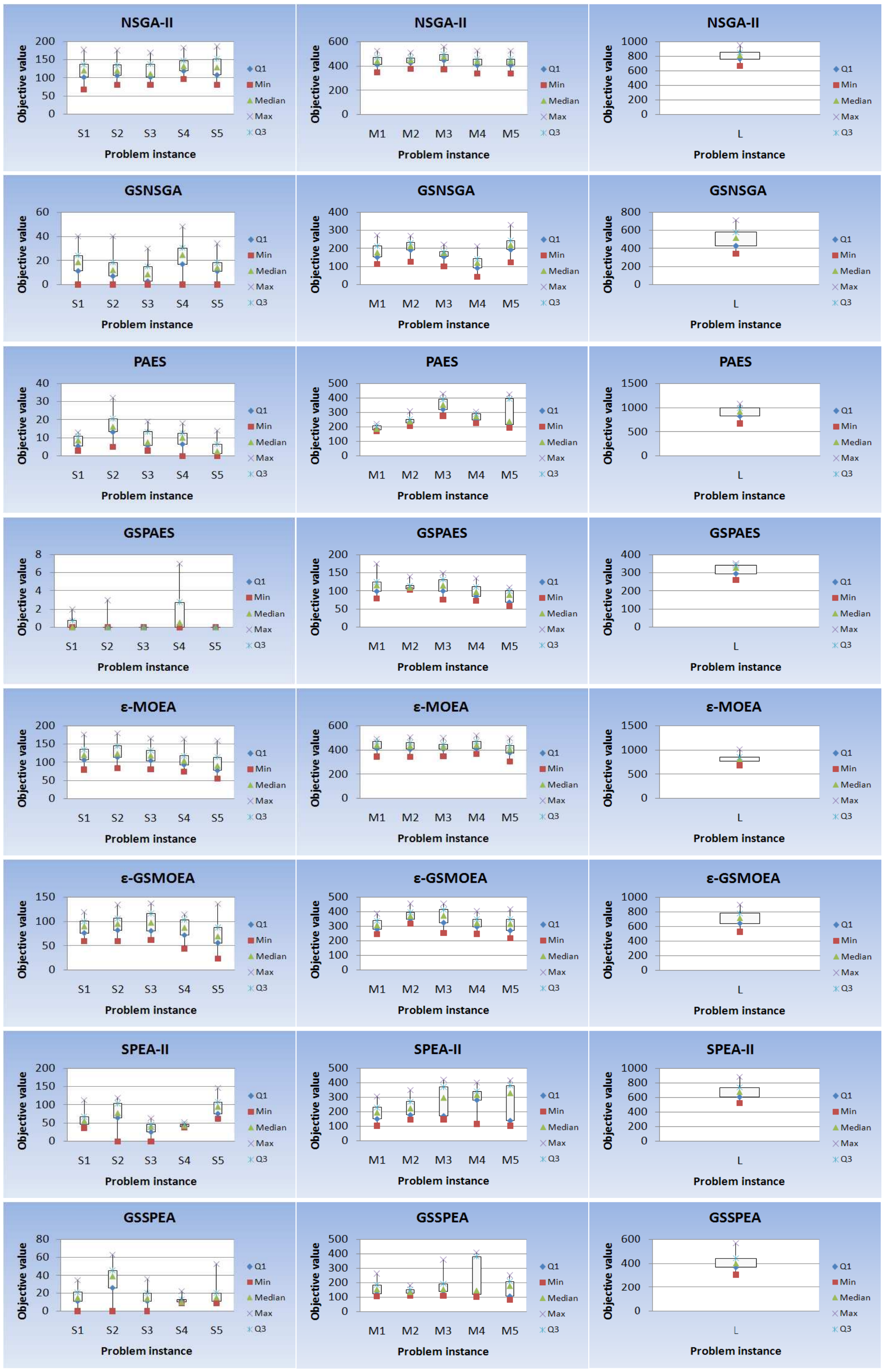

FIGURE 7.1: Dynamic performance of algorithms on different problem instances.

7.4.3.1 Key Parameters of the GS Strategy

The parameters of the GS strategy were determined experimentally for each MOEA over all problem instances. First, we checked the effect of parameters on all proposed algorithms over all problem instances. Then, the parameter setting which yielded the best results regarding the size of the space covered was selected.

The performance of the GS strategy depends on the parameters and operators used. We found that α, β, γ, and τ are key parameters that can greatly affect the performance of MOEAs for the MOUCTP, where α is the percentage of best individuals selected from the current population for constructing the data structures MEM_i, β is the percentage value of the total number of events that are used to create a child through the data structures MEM_i, γ is the probability that indicates whether a child is created through the GS strategy or crossover, and τ decides the frequency of updating MEM_i ($i = 1, 2, 3$).

Table 7.3 shows different parameters and their settings that were tested in our experiments. In order to find out which parameter settings have the greatest effect on the performance of proposed algorithms, we ran all algorithms 10 times for all parameter combinations in Table 7.3. Here, we only present some of parameter combinations that seem to have a substantial effect on the performance of proposed algorithms. We chose two α values 0.2 and 0.6, three β values 0.1, 0.3, and 0.5, three γ values 0.4, 0.6, and 0.8, and two τ values 20 and 80. Table 7.4 shows the objective values achieved by GSSPEA according to different parameter settings for the GS

TABLE 7.3: Parameter settings in GSGA

Parameter	Settings				
α	0.2	0.4	0.6	0.8	
β	0.1	0.3	0.5	0.7	0.9
γ	0.2	0.4	0.6	0.8	
τ	20	40	60	80	100

strategy. All other proposed algorithms give more or less similar values on these parameters. In Table 7.4, one parameter changes while the other parameters are kept constant on different MOUCTP instances. From Table 7.4, several significant results can be observed and are analysed below.

First, the parameter α has a significant effect on the performance of GSSPEA for the MOUCTP. The performance of GSSPEA drops when the value of α increases from 0.2 to 0.8. This occurs because when we choose a small part of the population to create MEM_i ($i = 1, 2, 3$), the data structures can provide a strong guidance during the genetic operations and help GSSPEA sufficiently exploit the area of the search space that corresponds to the best individuals of the population sufficiently. This sufficient exploitation can ensure that GSSPEA achieves better solutions more quickly. In contrast, if a large part of the population is taken to create or update MEM_i ($i = 1, 2, 3$), then the data structures will lose its effect of guiding GSSPEA to exploit promising areas of the search space.

Second, regarding the effect of β, it can be seen that setting this parameter to a very small or very large value affects the penalty values. When the value of β increases from 0.1 to 0.3, the performance of GSSPEA improves due to the enhanced effect of MEM_i ($i = 1, 2, 3$). However, when the value of β is further raised, the performance

Table 7.4: Average best value of 10 runs of GSSPEA with different parameter settings on the test problem instances S1, M1, and L

Parameters				S1			M1			L		
α	β	γ	τ	f1	f2	f3	f1	f2	f3	f1	f2	f3
0.2	0.1	0.4	20	3	5	26	12	98	31	38	321	81
0.2	0.1	0.4	80	9	16	70	7	111	37	45	301	108
0.2	0.1	0.6	20	8	8	69	14	78	34	37	294	78
0.2	0.1	0.6	80	7	26	80	10	101	31	38	312	72
0.2	0.1	0.8	20	10	13	78	9	90	28	38	347	78
0.2	0.1	0.8	80	6	28	101	13	114	31	41	341	92
0.2	0.3	0.4	20	1	2	10	17	78	33	32	281	72
0.2	0.3	0.4	80	7	3	12	11	92	30	30	293	70
0.2	0.3	0.6	20	1	0	0	8	82	28	30	231	69
0.2	0.3	0.6	80	1	0	3	7	87	31	30	242	73
0.2	0.3	0.8	20	1	0	4	18	93	29	32	232	69
0.2	0.3	0.8	80	2	3	12	11	101	29	36	281	76
0.2	0.5	0.4	20	5	0	56	14	195	44	44	403	97
0.2	0.5	0.4	80	7	2	60	26	247	46	45	478	90
0.2	0.5	0.6	20	2	1	12	14	209	40	44	372	82
0.2	0.5	0.6	80	3	4	12	13	222	41	42	437	85
0.2	0.5	0.8	20	7	0	30	28	198	44	40	501	103
0.2	0.5	0.8	80	1	20	60	29	175	48	39	511	90
0.6	0.1	0.4	20	2	13	72	68	198	55	44	403	72
0.6	0.1	0.4	80	11	8	110	88	289	60	46	791	91
0.6	0.1	0.6	20	8	0	14	10	104	28	43	354	69
0.6	0.1	0.6	80	11	26	85	10	210	39	47	322	124
0.6	0.1	0.8	20	3	9	48	8	101	39	43	359	76
0.6	0.1	0.8	80	4	1	52	13	117	52	44	419	74
0.6	0.3	0.4	20	2	8	67	11	145	29	31	295	96
0.6	0.3	0.4	80	3	1	80	9	196	38	43	314	72
0.6	0.3	0.6	20	4	0	12	6	160	28	34	322	71
0.6	0.3	0.6	80	11	0	52	15	165	42	47	360	74
0.6	0.3	0.8	20	11	2	14	15	170	24	34	354	103
0.6	0.3	0.8	80	16	1	23	11	208	47	33	460	81
0.6	0.5	0.4	20	5	18	60	43	200	49	40	846	93
0.6	0.5	0.4	80	2	26	63	81	313	53	48	823	103
0.6	0.5	0.6	20	3	11	42	49	180	42	46	791	91
0.6	0.5	0.6	80	7	0	52	30	215	47	42	807	83
0.6	0.5	0.8	20	4	7	63	48	212	47	43	789	82
0.6	0.5	0.8	80	5	11	90	64	243	45	44	855	75

of GSGA drops. This occurs because if a large portion of individuals is created through MEM_i ($i = 1, 2, 3$), e.g., when $\beta = 0.9$, the chance of creating a similar child may be increased every generation and after a few generations GSGA may be trapped in a sub-optimal state and hence be unable to obtain the optimal solution.

Third, regarding the effect of γ, from Table 7.4, it can easily be seen that increasing the value of γ results in near-optimal solutions. The reason lies in the fact that a small value of γ leads to the proposed algorithms creating children with the crossover operator only while a very large value reduces the diversity due to the creation of children over certain generation through the same data structures. The effect of γ also shows the importance of the data structures.

Fourth, regarding the effect of τ, it can be seen from Table 7.4 that updating MEM_i $(i = 1, 2, 3)$ every 20 generations gives a better penalty value than updating them every 80 generations. This is due to the fact that in the former case, the search space is explored more than in the latter case, which increases the quality and gives a greater chance of creating better individuals. This is because increasing the value of τ decreases the exploration ability of GSSPEA because the pairs of room and time slot obtained from potentially promising solutions additionally increases the diversification of GSSPEA.

7.4.3.2 Effect of the GS and LS Strategies

Usually, crossover leads the population to converge by making the chromosome alike, and mutation re-introduces genetic diversity back into the population and assists the algorithm to escape from local optima [16]. The ability of GAs to simultaneously search different regions of the search space makes it possible to find a diverse set of solutions for difficult and highly constrained problems.

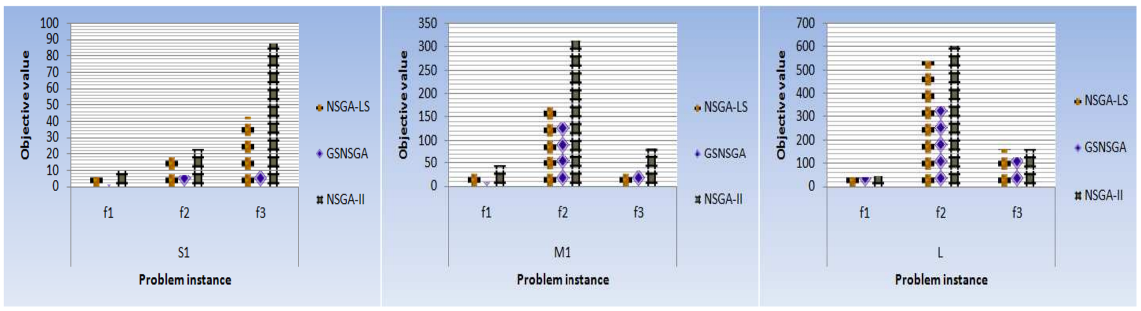

FIGURE 7.2: Comparison of NSGA-II, NSGA-LS, and GSNSGA regarding the achieved objective values on problem instances S1, M1, and L.

In GSNSGA, we create a population not only by crossover but also with the GS strategy, which re-introduces the best parts of previous good solutions along with new parts (random assignments), thus preventing the population from getting stuck in local optima. The GS strategy uses the best part of individuals with respect to different objectives to create new non-dominated solutions in un-explored parts of the Pareto front. Keeping individuals from many non-dominated fronts in the population help the GS strategy to create solutions with diversity. Figure 7.2 shows the effect of the GS strategy. We tested the performance of GSNSGA and NSGA-II with the LS strategies but without the GS strategy (denoted NSGA-LS). Figure 7.2 shows the performance of NSGA-II, NSGA-LS, and GSNSGA on S1, M1 and L problem instances. GSNSGA seems to be effective since new individuals with better genetic information are re-introduced into the new population.

7.4.3.3 Performance of Investigated MOEAs

The main objective of MOEAs is to provide a Pareto optimal set of solutions from which the timetable maker can make a decision. We can see the performance of proposed MOEAs from different aspects. Table 7.5 and Table 7.6 show the hypervolume values and D metrics of different MOEAs on the MOUCTP instances. Figure 7.3 to 7.8 show the obtained non-dominated solutions of MOEAs obtained on problem instances S1, M2, and M5, respectively. Each point in the plots is a point in the objective domain. The scales of the plots are not fixed and differ according to the objective values of MOEAs. To study the trends clearly, objective functions have also been projected in two-dimensional planes regarding $f1 - f2$, $f1 - f3$, and $f2 - f3$, respectively.

Generally speaking, Figure 7.3 to 7.8, and Table 7.2 show that all MOEAs with the GS and LS strategies do better than MOEAs without the GS and LS strategies. We can see that the inclusion of the GS and LS strategies in MOEAs is vital to the success of the algorithms. If we consider the performance of different MOEAs we can see that GSPAES mostly covers the other MOEAs and created trade-off front by giving greater hypervolume. GSNSGA and GSSPEA also perform well on all problem instances if we compare them to ϵ-GSMOEA.

We also statistically tested the performance of MOEAs using the Corne and Knowler approach [136]. Table 7.7 shows the statistical test values of different MOEAs with two rows for each problem instance. The first "unbeaten" row shows the percentage of the objective space on which the performance of a corresponding algorithm is UNBEATEN by any of the other algorithms compared, i.e, the percentage of the objective space (based on a non-parametric test – The Mann-Whitney Rank test [136]) for which we cannot be 95% confident that any other algorithm beats it. On

Table 7.5: Hypervolume of different MOEAs on the MOUCTP instances

MOUCTP	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L
GSNSGA	8.0×10^6	7.9×10^6	8.0×10^6	7.9×10^6	7.9×10^6	9.7×10^7	9.4×10^7	9.9×10^7	1.1×10^8	9.4×10^7	3.1×10^8
GSPAES	8.0×10^6	8.0×10^6	8.0×10^6	8.0×10^6	8.0×10^6	1.0×10^8	1.0×10^8	1.0×10^8	1.0×10^8	1.1×10^8	3.5×10^8
ϵ-GSMOEA	5.1×10^6	4.5×10^6	4.8×10^6	6.2×10^6	6.9×10^6	7.0×10^7	5.5×10^7	6.6×10^7	6.7×10^7	6.1×10^7	1.9×10^8
GSSPEA	8.0×10^6	8.0×10^6	7.2×10^6	7.6×10^6	7.6×10^6	9.9×10^7	9.8×10^7	9.8×10^7	9.9×10^7	1.0×10^8	3.3×10^8

Table 7.6: D matrics of comparing MOEAs on the MOUCTP instances

Compared Algo	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L
GSNSGA – ϵ-GSMOEA	2.8×10^6	3.4×10^6	3.1×10^6	1.7×10^6	1.0×10^6	2.6×10^7	3.8×10^7	3.2×10^7	4.7×10^7	2.2×10^7	1.2×10^8
GSNSGA – GSPAES	0	0	0	0	0	3.0×10^6	1.9×10^6	1.4×10^6	7.5×10^6	1.4×10^6	7.0×10^6
GSNSGA – GSSPEA	0	0	7.6×10^5	3.6×10^5	3.5×10^5	4.0×10^6	3.3×10^6	4.5×10^6	1.5×10^6	2.4×10^6	2.1×10^6
GSPAES – ϵ-GSMOEA	2.8×10^6	3.4×10^6	3.1×10^6	1.7×10^6	1.0×10^6	3.5×10^7	4.4×10^7	3.9×10^7	4.0×10^7	4.9×10^7	1.5×10^8
GSPAES – GSNSGA	0	639	0	2199	7174	1.1×10^7	7.7×10^6	8.1×10^6	6.4×10^4	1.8×10^7	4.0×10^6
GSPAES – GSSPEA	0	0	7.6×10^5	3.6×10^5	3.6×10^5	9.7×10^6	5.8×10^6	8.4×10^6	1.1×10^7	9.3×10^6	2.8×10^7
ϵ-GSMOEA – SPEA	0	0	0	0	0	8.1×10^5	0	0	0	0	4.8×10^6
ϵ-GSMOEA – GSNSGA	0	0	0	2199	0	0	0	0	0	0	7.5×10^6
ϵ-GSMOEA – GSPAES	0	0	0	0	0	0	0	0	0	0	1.8×10^6
GSSPEA – ϵ-GSMOEA	2.8×10^6	3.4×10^6	2.4×10^6	1.3×10^6	6.6×10^5	2.9×10^7	4.3×10^7	3.1×10^7	3.2×10^7	4.2×10^7	1.3×10^7
GSSPEA – GSNSGA	0	639	0	0	0	5.9×10^6	7.4×10^6	3.3×10^6	0	1.2×10^7	1.6×10^7
GSSPEA – GSPAES	0	0	0	0	0	3.1×10^6	4.6×10^6	5.2×10^5	3.3×10^6	2.4×10^6	9.3×10^6

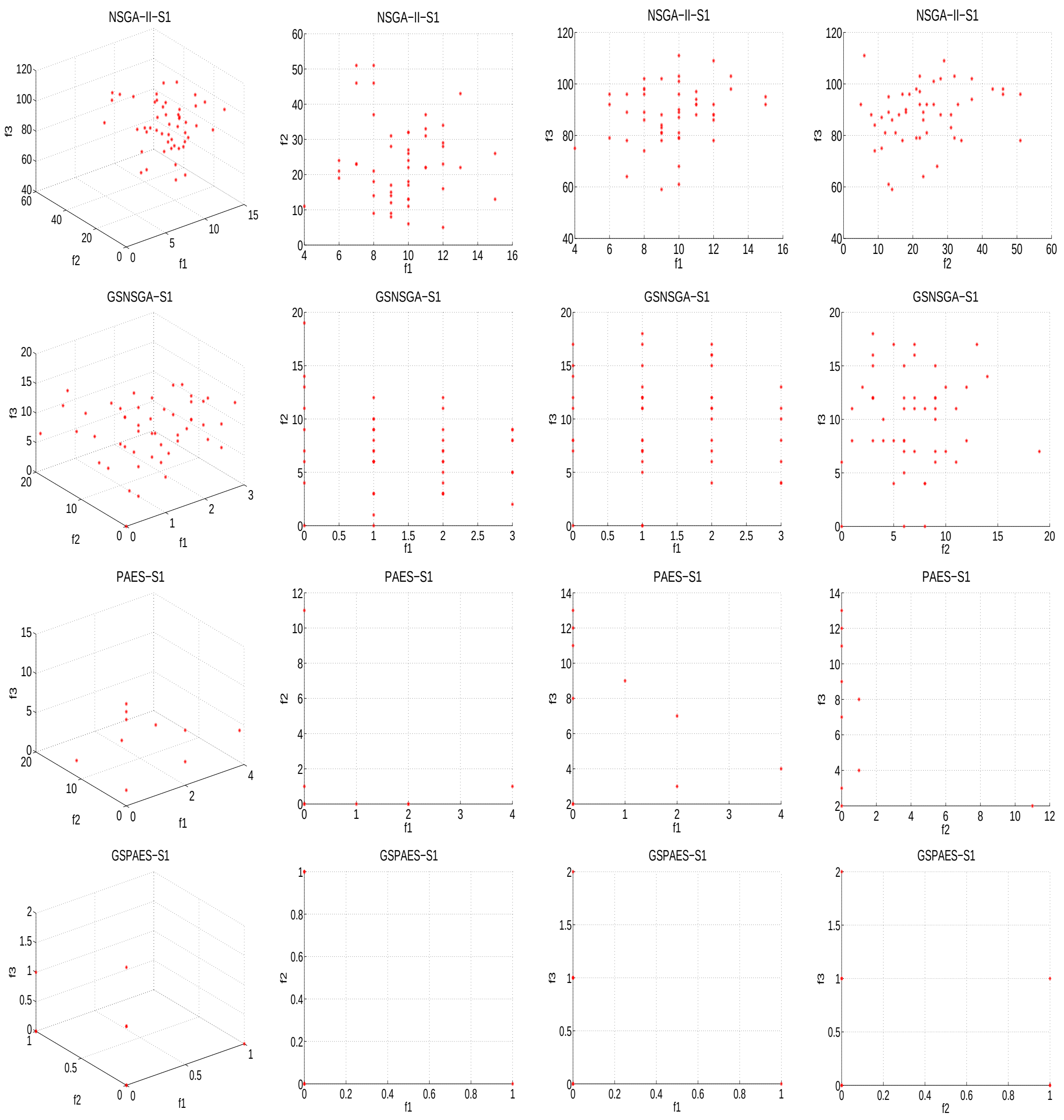

FIGURE 7.3: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance S1.

the other hand, the second "beaten all" row gives the percentage of the objective space on which we can be 95% confident that the corresponding algorithm BEATS ALL of the other algorithms compared. For example, for the S1 problem instance, GSPAES was not beaten anywhere in the objective space, and it crushed the other three algorithms for sure on 23.45% of the space. Similarly, for the M5 problem

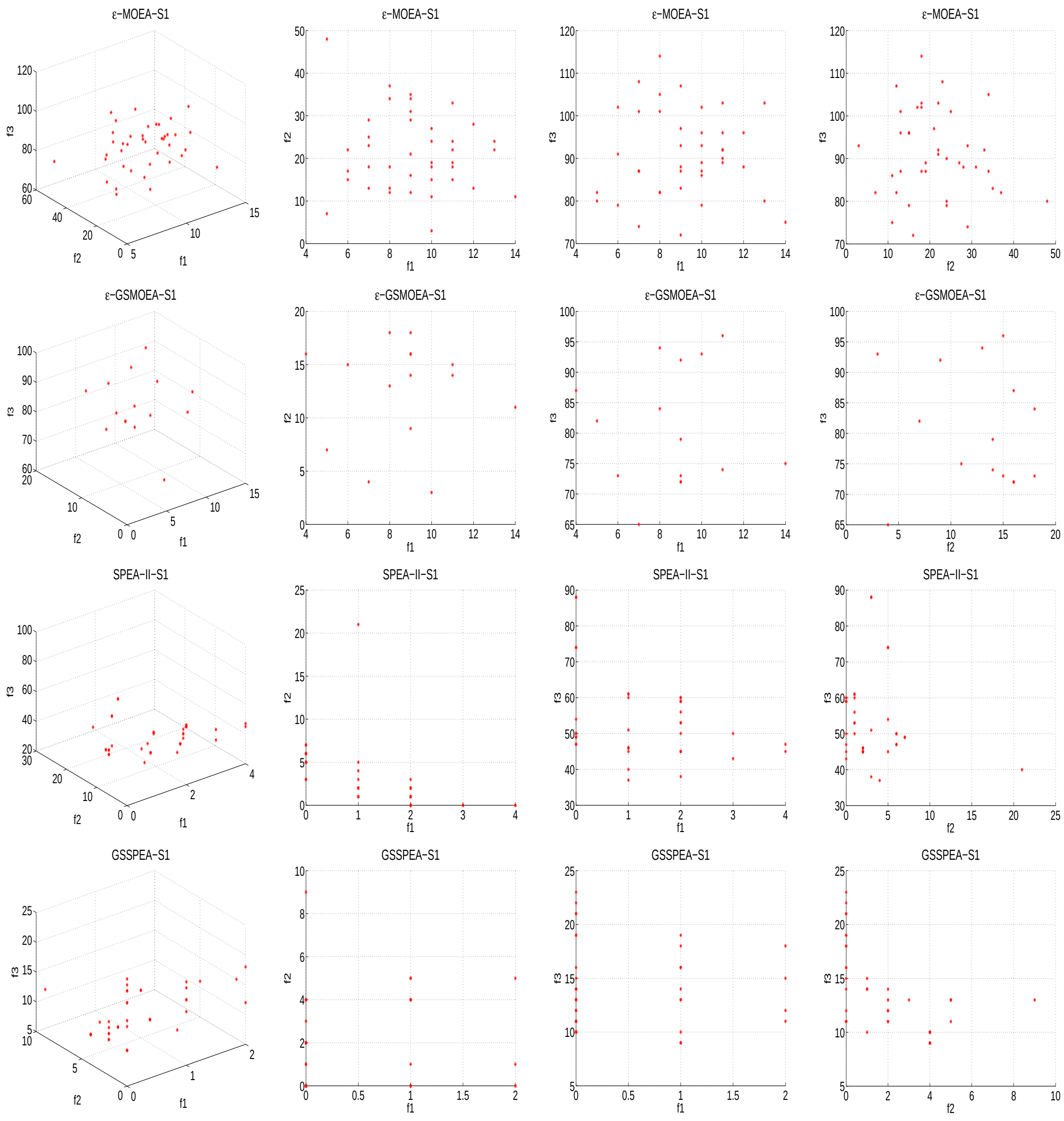

FIGURE 7.4: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance S1.

instance, all algorithms have unbeatable performance except that ϵ-GSMOEA has a 0% unbeatable space. All algorithms, e.g., GSSPEA and GSNSGA, had regions in which they perform well, but GSPAES won by having 55% of the space, which is better than all other algorithms.

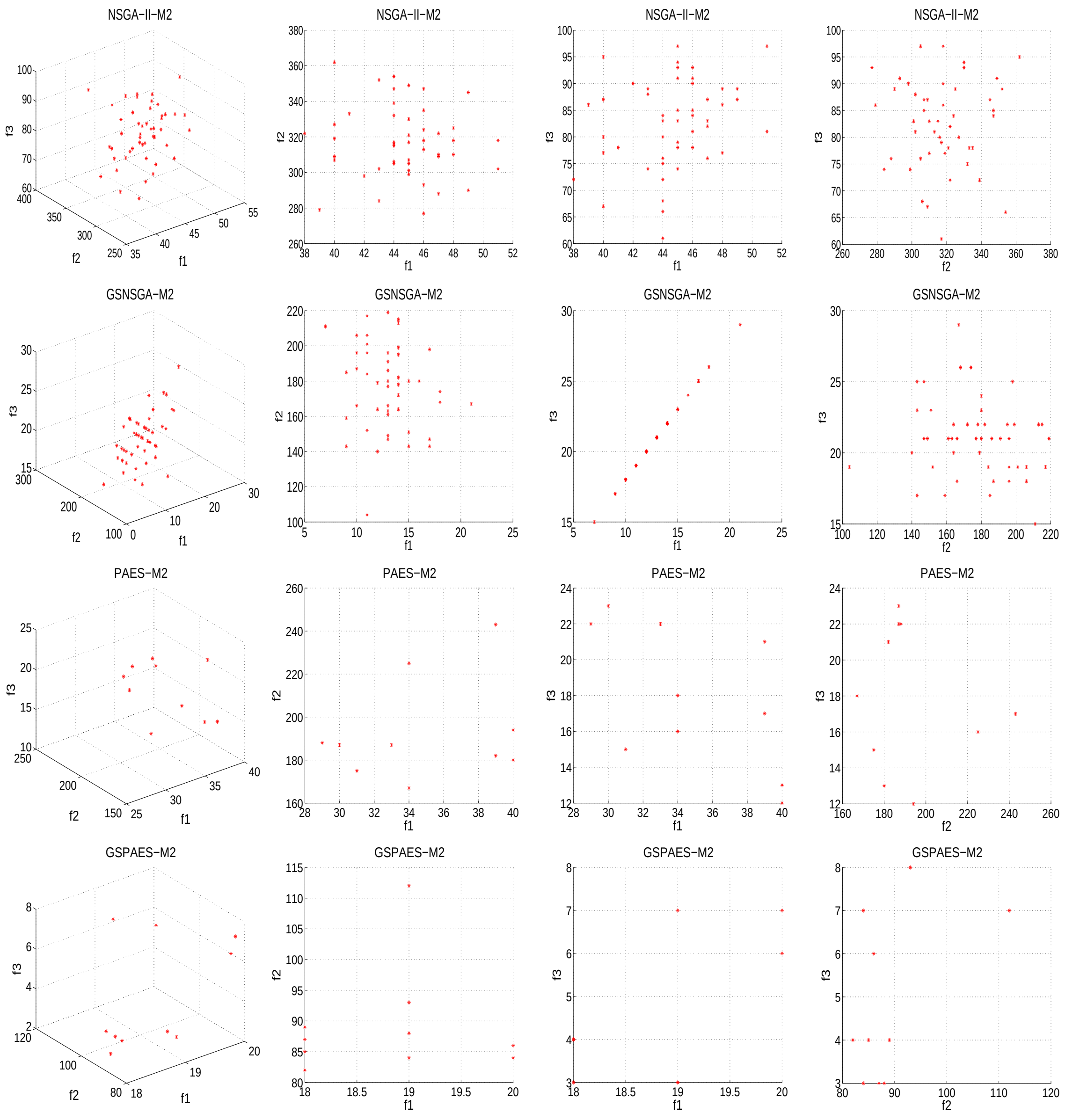

FIGURE 7.5: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance M2.

7.4.4 Comparison with Other Algorithms

As mentioned before, most researchers have dealt with the single-objective UCTP. In this chapter, we tackle the UCTP as a MOOP. It is interesting to see whether the solutions produced by our algorithms are competitive with those obtained with other

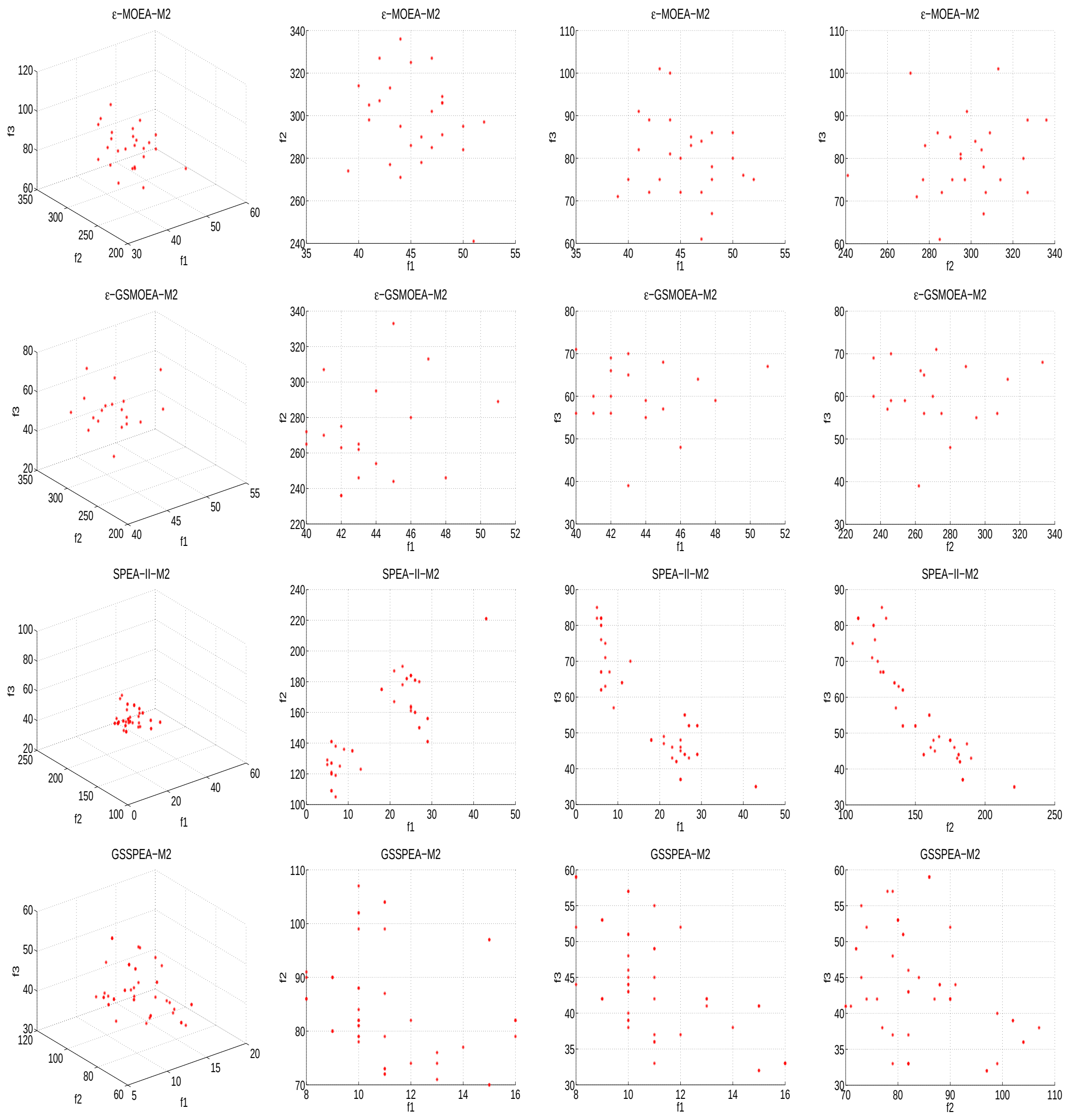

FIGURE 7.6: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance M2.

methods that tackle the UCTP as a single-objective optimisation problem. For this purpose, we transfer the results of proposed MOEAs into single-objective results by weighted aggregation of the three objective values into one objective value. Note that this measure is widely used in the literature for the single-objective UCTP. Table 7.8 presents the comparison of our results with those published results obtained by

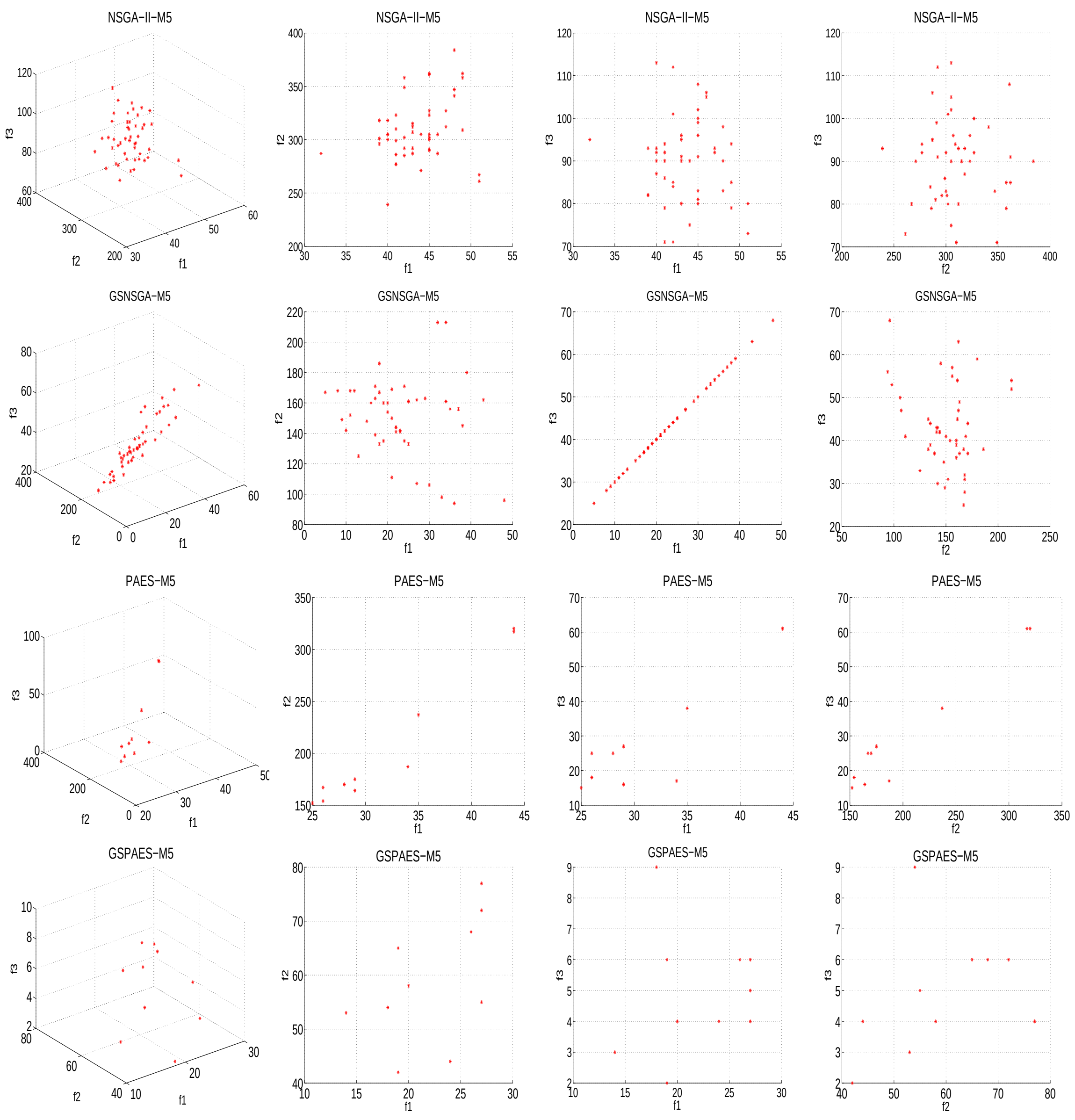

Figure 7.7: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance M5.

the following algorithms.

- The guided search GA (GSGA): The guided search GA proposed in [129], where a guided search technique is used to solve the UCTP.

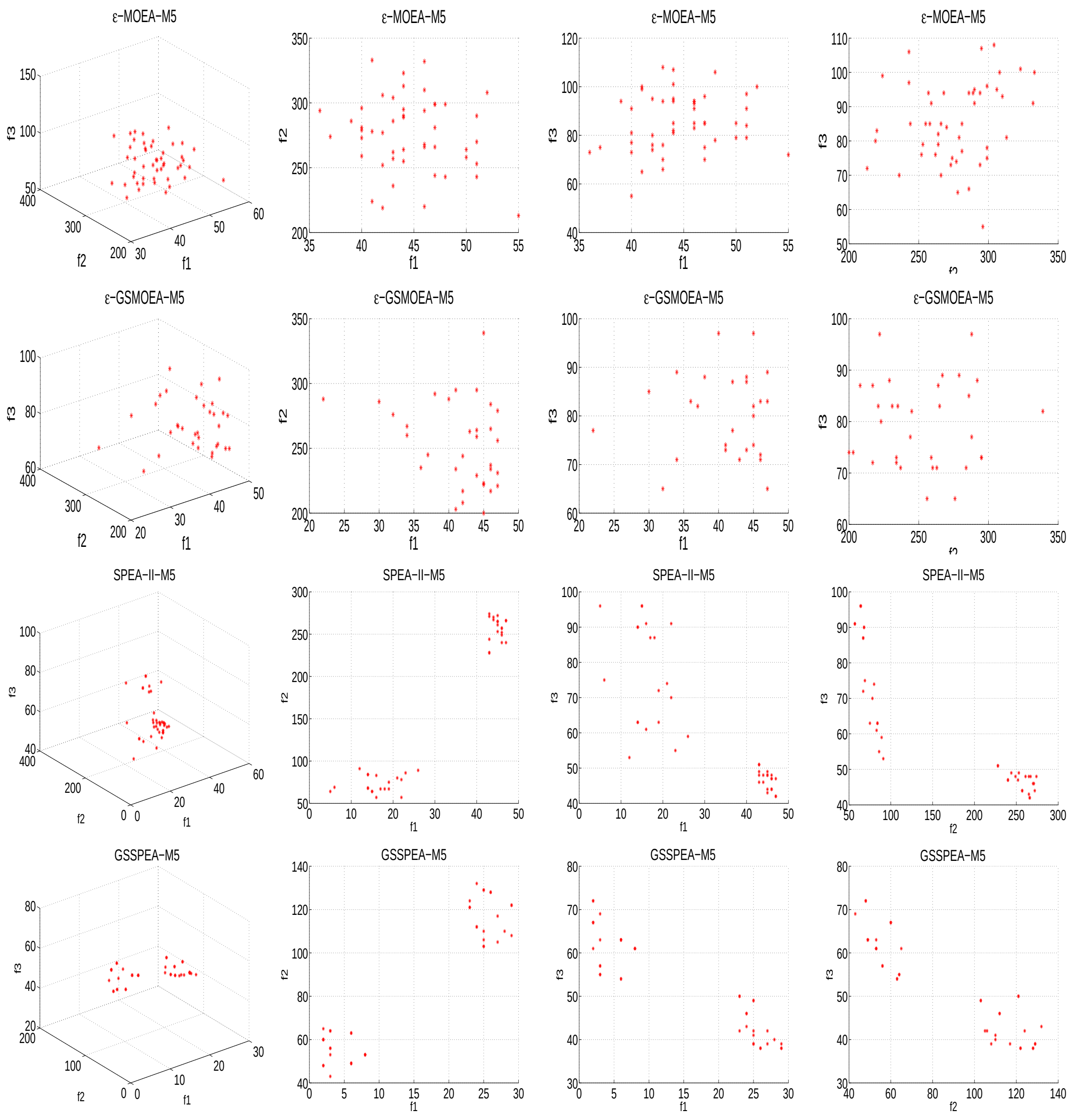

Figure 7.8: Comparison of algorithms regarding obtained non-dominated solutions on the MOUCTP instance M5.

- The variable neighbourhood search (VNS): In [13], Abdullah *et al.* used a VNS approach based on the random-descent LS with an exponential Monte Carlo acceptance criteria.

- *The fuzzy algorithm (FA)*: Asmuni *et al.* proposed a FA in [29]. They focused on the issue of ordering events by simultaneously considering three different

Table 7.7: Statistical comparison of algorithms

MOUCTP	Performance	GSNSGA	GSPAES	ϵ-GSMOEA	GSSPEA
S1	unbeaten	0	100	0	0
	beaten all	0	23.45	0	0
S2	unbeaten	59	100	0	95
	beaten all	0	3.88	0	0
S3	unbeaten	96.03	100	0	51.25
	beaten all	0	0	0	3.97
S4	unbeaten	54.11	100	0	59.65
	beaten all	0	40.43	0	0
S5	unbeaten	0	100	0	50.6
	beaten all	0	49.4	0	0
M1	unbeaten	36.84	61.03	0	31.4
	beaten all	19.4	44.14	0	7.85
M2	unbeaten	30.19	52.17	0	41.64
	beaten all	11.73	42.7	0	22.35
M3	unbeaten	58.08	71.93	0	16.25
	beaten all	26.78	35.27	0	0
M4	unbeaten	100	100	100	100
	beaten all	0	0	0	0
M5	unbeaten	17.73	64.27	0	29.82
	beaten all	9.88	55.5	0	23.27
L	unbeaten	13.76	100	21.8	18.93
	beaten all	0	57.99	0	0

heuristics using fuzzy methods.

- *The graph-based hyper-heuristic (GBHH)*: Burke *et al.* [51] proposed the GBHH. They employed tabu search with graph-based hyper-heuristics for the UCTP and examination timetabling problems.

- The tabu search roulette wheel (TSRW) hyper-heuristic: Burke *et al.* [50] proposed the multi-objective TSRW hyper-heuristic to solve a UCTP as a MOOP. They used three objectives to be minimised. They chose one objective at a time during the search by the roulette wheel selection.

- *The tabu search hyper-heuristic (TSHH)*: Burke *et al.* [47] introduced a TSHH for the UCTP, where a set of low-level heuristics compete with each other. This

Table 7.8: Comparison of algorithms on problem instances

UCTP	GSNSGA Best	GSPAES Best	ϵ-GSMOEA Best	GSSPEA Best	GSGA Best	TSRW Best	VNS Best	GBHH Best	FA Best	LS Best	TSHH Best
S1	**0**	**0**	72	**0**	**0**	–	**0**	6	10	8	1
S2	**0**	**0**	95	**0**	**0**	–	**0**	7	9	11	2
S3	**0**	**0**	81	**0**	**0**	–	**0**	3	7	8	**0**
S4	**0**	**0**	43	9	**0**	–	**0**	3	17	7	1
S5	**0**	**0**	23	9	**0**	–	**0**	4	7	5	**0**
M1	113	**79**	243	106	129	–	317	372	243	199	146
M2	126	**103**	315	110	160	173	313	419	325	202.5	173
M3	101	**76**	253	110	242	224	357	359	249	77.5%ln	267
M4	**42**	73	246	105	158	160	247	348	285	177.5	169
M5	124	**58**	287	83	124	–	292	171	132	100%ln	303
L	340	**260**	528	307	801	–	100%ln	1068	1138	100%ln	80%ln

approach was tested on the course timetabling and nurse rostering problems.

- The LS: Socha *et al.* [196] introduced an LS method. They used a random restart LS method for the UCTP and compared it with an ant algorithm.

All the above algorithms except TSRW tackled the single objective UCTP. In Table 7.8, "–" means that there is no result available in the literature for a corresponding algorithm, and "%ln" indicates the percentage of infeasible solutions for a corresponding problem instance.

From Table 7.8, it can be seen that GSPAES, GSSPEA, and GSNSGA outperform other algorithms on all medium and large problem instances. GSPAES and GSNSGA also perform well on small problems. ϵ-GSMOEA also give good results on most medium problems in comparison with FA [29], GBHH [51], and VNS [13].

From Table 7.8, it can also be seen that Pareto optimal techniques give good results on all of the problem instances because they provide a better coverage of the objective space and is more effective in minimising the total number of constraint violations during the running [172].

7.5 Chapter Summary

In this chapter, we tackled the UCTP as a MOOP. To solve the MOUCTP, we proposed a framework of combining a guided search technique and local search strategies with general MOEAs to solve the MOUCTP. The GS strategy uses memories (data structures) to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and which has a zero penalty value. These data structures are used to guide the generation of offspring into the following populations. The main advantage of these data structures lies in the fact that they help improve the quality of individuals by storing part of former good solutions, which otherwise would have been lost in the selection process, and reusing the stored information in the following generations. This can enable the algorithm to quickly retrieve the best solutions corresponding to the previous and new populations.

In order to test the performance of proposed MOEAs for the MOUCTP, experiments were carried out to analyse the performance of MOEAs based on a set of well-known benchmark UCTP instances [196] (described in Section 3.4 of Chapter 3) and the performance metrics that are taken from the literature on MOEAs for general MOOPs. The experimental results of proposed algorithms were also compared with several state-of-the-art methods from the literature on the tested UCTP instances. The experimental results show that the proposed methods are competitive and work reasonably well across all problem instances in comparison with other approaches studied in the literature. It was found that all proposed algorithms obtained good results and produced a population of non-dominated solutions. Hence, the user is able to choose the most appropriate solutions from the final population, rather than being restricted to a single solution. We can say that the proposed MOEAs are efficient because they take considerably less time to produce a course timetable than real cases, which often take months to produce one before a new semester starts. The proposed MOEAs are scalable because they have effectively tackled the large scale problem instance as well as small scale problem instances.

Chapter 8

Conclusions and Future Work

In this chapter, we summarise the major developments achieved by the work presented in this thesis, including the main technical contributions and major conclusions that can be drawn from the experimental studies carried out in the thesis. We also discuss further research directions that may be undertaken in the future.

8.1 Technical Contributions

Timetabling is one of the common challenging scheduling problem, in which we need to maximise the allocation of resources and minimise the violation of constraints. Timetabling problems are often made complicated by the details of a particular timetabling task, and are often considered to be NP-hard problems. The university course timetabling problem (UCTP) is one type of timetabling problems. This problem has attracted significant research interest over the past decades and is considered to be very difficult to solve due to its highly-constrained nature and the exponential growth of the size of the search space with the problem size.

The UCTP does not have a single agreed definition and varies in its structure according to the particular requirements of different universities, which contributes to making UCTPs a difficult class of problems, offering some serious research challenges. From a practical point of view, they are also very important problems since the quality of solutions to these problems often has a significant impact on the institutions concerned. Hence, it is of great value to investigate the UCTP and develop efficient solvers.

In this thesis, we address the UCTP with the tool of GAs. As mentioned in the introduction to the thesis (Section 1.3 in Chapter 1), in order to develop efficient GAs to tackle the UCTP, in this thesis, we have followed a spiral procedure: first we studied traditional GAs for the simple UCTP, and then developed and added LS and GS strategies to enhance the performance of traditional GAs for solving the UCTP and the more challenging PECTP. Finally, we addressed the more real-world oriented MOUCTP with MOEAs that are enhanced with the LS and GS strategies developed in this thesis. Hence, in this thesis, we investigated various versions of GAs for the UCTP.

Chapter 1 outlined the motivation for the work for this thesis by introducing the need for combining population-based and local area-based algorithms for the UCTP.

It also highlighted the need for the development of new techniques to enhance the solution quality for the UCTP. Chapter 2 described the complicated nature of university course timetabling and also described different methodologies used in the existing literature to solve this problem. Chapter 3 presented the specification of data sets that were used to test the proposed GAs in this thesis. Then, in the remaining chapters, we proposed several techniques and developed a number of GAs for solving the UCTP, the PECTP, and the MOUCTP.

The major technical contributions achieved in this thesis toward the research domain of UCTPs are summarised as follows:

- **A new memetic algorithm (MA) for the UCTP**: A new LS strategy (LS2) was proposed to remove or minimise the penalty cost related to time slots. Based on this LS strategy, we presented a new MA for solving the UCTP. The MA combines the steady state GA with two LS strategies, LS1 and LS2. We established in Chapter 4 that GAs do not work very well even with a traditional LS strategy (LS1) within a limited time. With LS1 only, the MA does not perform well for the UCTP benchmark instances in the experiments, as also observed in [181]. However, we have enhanced the search power of the MA for solving the UCTP by introducing a second LS strategy, LS2, into its structure. LS2 tries to improve the individuals by minimising violation of constraints on a particular time slot and enables the MA to escape from local optima.

In order to investigate the effect of the new LS strategy, i.e., LS2, experiments were carried out to test the performance of a simple GA and the proposed MA in two aspects: one is the comparison of performance on the 11 benchmark UCTP instances in terms of the *t*-test results; the other is the comparison of the performance with other approaches in the literature within the same time scale. Based on the experimental results, it is clear that, with the help of the powerful LS strategy, the proposed MA can obtain high quality solutions that satisfy different constraints. The proposed MA is capable of finding near-optimal solutions for the test problem instances.

- **A novel guided search (GS) strategy**: A novel GS strategy was proposed to enhance the searching power of GAs for solving the UCTP. The GS strategy uses a memory (data structure) to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and has a zero penalty value. This data structure is used to guide the generation of offspring into the following populations. The main advantage of the GS strategy is that it improves the quality of individuals by storing parts of former good solutions, which otherwise would have been lost in the selection process, and reusing the stored information in the following generations. This can enable a GA to quickly retrieve useful information from the best solutions of previous populations to generate new good solutions into the current population.

In Chapters 5, 6 and 7, we have proposed different GAs with the help of the GS strategy, which produced some of the best known solutions for the UCTP and PECTP benchmark instances used in this thesis.

- **Several guided search GAs for the UCTP**: Hybridisations of LS and GS strategies were applied to GAs to solve the UCTP. We integrated LS strategies with the GS strategy and presented several GA variants based on the steady-state GA model, including two versions of guided search GAs (i.e., GSGA and EGSGA) for solving the UCTP. In the proposed GSGAs, two LS strategies were used to improve the individuals in the population around the local areas in which they resided, and the GS strategy is used to guide the generation of good quality solutions into the population. The experimental results of GSGAs on the benchmark UCTP instances show that the proposed EGSGA is competitive and works well across the tested problem instances in comparison with other state-of-the-art approaches taken from the literature and hence can act as a powerful tool for the UCTP.

  In order to test the performance of the proposed GSGAs for the UCTP, experiments were carried out to analyse the sensitivity of parameters within the GS strategy and the effect of the GS strategy on the performance of GSGAs based on a set of benchmark UCTP instances. The experimental results of EGSGA were also compared with several state-of-the-art methods from the literature on the tested UCTP instances. The results show that the proposed EGSGA is competitive and works well across the tested problem instances in comparison with other state-of-the-art approaches taken from the literature. Generally speaking, with the help of the GS and LS strategies, EGSGA is able to efficiently find optimal or near-optimal solutions for the UCTP and hence can act as a powerful tool for the UCTP.

To our knowledge, this study is the first time GAs have been applied together with the GS strategy to address timetabling problems.

- **An efficient two-phase hybrid approach for the PECTP**: A two-phase hybrid approach, denoted HGATS, was presented for solving the PECTP, which is a specialised but more challenging UCTP. This hybrid approach combines the GSGA and tabu search, and tries to strike a balance between their exploitation and exploration abilities. In the first phase of HGATS, the GSGA uses the GS and LS strategies to try to find feasible solutions for the PECTP. Due to the challenge of the PECTP, after the first phase, HGATS may not be able to find feasible or good enough solutions. Hence, the second phase is introduced to HGATS, where a TS scheme is used to further improve the best solution obtained by the GSGA in the first phase.

  In order to test the performance of the proposed HGATS for the PECTP, experiments were carried out to analyse the sensitivity of parameters and the effect of the GS strategy on the performance of HGATS based on a set of benchmark ITC-2007 PECTP instances. The experimental results of HGATS were also compared with several state-of-the-art methods from the literature on these benchmark instances. The experimental results showed that the proposed hybrid approach is competitive and works well across all test PECTP instances in comparison with other approaches studied in the literature. Generally speaking, with the help of the guided, local, and tabu search strategies, HGATS is able to efficiently find optimal or near-optimal solutions for the PECTP and hence can act as a powerful tool for the PECTP.

- **Multi-objective approaches to the UCTP**: In this thesis, the UCTP has also been tackled as a MOOP, i.e., the MOUCTP. To solve the MOUCTP, we presented a framework for combining the above GS and LS strategies with general MOEAs. The GS strategy uses memories (data structures) to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and which has a zero penalty value. These memories help to improve the quality of individuals by reusing the stored information in the following generations, which otherwise would have been lost in the selection process.

  From the proposed framework, it is easy to construct new MOEAs for the MOUCTP using MOEAs for general MOOPs. In the thesis, we have instantiated this framework onto several widely-used MOEAs, including NSGA-II [80], ϵ-MOEA [81], SPEA-II [221], and PAES [136], to construct corresponding new MOEAs to solve the MOUCTP.

  The experimental results of proposed MOEAs on benchmark MOUCTP instances show that they obtained good results and produced a population of non-dominated solutions for the benchmark MOUCTP instances. Hence, the user is able to choose the most appropriate solutions from the final population, rather than being restricted to a single solution. Our proposed MOEAs also gave a non-dominated set of solutions that are better than the corresponding state-of-the-art MOEAs in three aspects: one is the comparison of the performance in terms of the statistical test results; the second is the comparison of the performance regarding the hyper-volume of the obtained non-dominated solutions; and the third is the measurement of D-metric between two non-dominated sets obtained by the MOEAs.

To our knowledge, this study was the first time guided search GAs have been applied to address the MOUCTP.

8.2 Conclusions

In order to justify the algorithms we developed in this thesis, we have carried out several sets of experiments systematically and analysed the experimental results regarding the performance of algorithms in comparison with other state-of-the-art approaches for the UCTP taken from the literature. Here, we summarise the major conclusions based on the experimental results and relevant analyses carried out in this thesis as follows.

- The new LS strategy (i.e., LS2, proposed in Chapter 4) enables the MA to escape from local optima and hence greatly enhance the performance of the

MA for the UCTP. The results show that by integrating LS with appropriate neighbourhood moves, GAs can get the best solutions for the UCTP.

- We observed that the GS strategy is able to produce solutions of good quality into the population and significantly enhances the performance of GAs for the UCTP and the PECTP. The GS strategy also tries to balance between the exploration and exploitation abilities of GAs. The process of randomly creating children through the extra data structure(s) achieves a good balance between the exploitation and exploration properties of GAs. Reusing the best part of previous populations' individuals, for creating some part of a child, enhances the exploitation ability of GAs, and randomly creating some part of the child explores the search space more widely and hence enhances the explorative ability of GAs.

- From the experimental results in Chapter 5, LS strategies with different properties, when integrated with the GS strategy, give significantly improved results. The GS strategy can be used to create good quality solutions on the basis of previous good solutions and to explore the search space as much as possible. The LS operators can enhance the quality of an individual by finding the local optimum of that individual. Consequently, the cooperation of GS and LS strategies with each other can guide a GA to locate optimal or near-optimal solutions.

- We also noticed that the GS strategy works well with another meta-heuristic (TS) in the two-phase hybrid approach, which is also very effective towards solving the PECTP. Experimental results in Chapter 6 show that although the TS and GS approaches work well on all problem instances separately, when combined together, the results are then remarkably improved. We also noticed that the biased crossover operator and improved neighbourhood structures have an ability to explore a huge search space.

- We found that the UCTP can be well handled as an MOOP. The proposed framework of integrating the GS and LS strategies with MOEAs works well. The instantiated MOEAs from state-of-the-art MOEAs are able to solve the MOUCTP and give good quality solutions. We also observed that the GS and LS framework works well with Pareto Optimal techniques, provides a better coverage of the objective space and is more effective in minimising the total number of constraint violations. From this observation, we can conclude that MOEAs can effectively solve the UCTP.

Generally speaking, from the experimental results, we can conclude that the GS and LS strategies proposed in this thesis greatly improve the performance of GAs for both the UCTP and the MOUCTP.

8.3 Future Work

As mentioned before, we followed a spiral procedure to develop efficient GAs for solving the UCTP. Whilst this thesis presented some new heuristics and searching approaches for the improvement of the state-of-the-art of the UCTP research domain, and the results show that these approaches are able to produce some of the best results, there are nevertheless some general further research questions that arise from this research at each step of the spiral procedure. Here, we discuss them along with some suggestions for future work.

- As we have seen in Chapter 4, LS2 alone is not enough to help the traditional GA obtain a feasible solution for hard problem instances. LS2 could be improved if we introduce more neighbourhood structures that emphasise the placement of time slots or rooms in a more efficient way.

- The observation made in Chapter 5 and 6 regarding the GS strategy showed that the performance of the GS strategy is strongly dependent on the parameter settings. This effect can be slightly different for different UCTPs. It is worthwhile to investigate the performance on different optimisation problems in order to find the limitations of this approach. In the future, we might also be able to create new ways of finding the appropriate parameter values for different problems. For example, using adaptive techniques to adjust the key parameters of GSGAs during the searching progress may further improve the performance of GSGAs for the UCTP.

- We discovered that the GS strategy works well in the two-phase hybrid approach for the PECTP along with tabu search. The resultant two-phase hybrid approach is able to produce 100% feasible solutions on some problem instances in the first phase. We can, however, improve its performance in the first phase by using new genetic operators or by developing new neighbourhood techniques, based on different problem constraints. We believe that the GS strategy will further help GAs to improve their performance by applying advanced genetic operators, heuristics, and evaluation routines. Better understanding of the inter-relationships between these techniques and proper placement of these techniques in an algorithm may lead to even better results.

- In Chapter 7, we created three data structures for the minimisation of the three objectives for solving the MOUCTP. It is noted that, in real-world situations, there may be many objectives and creating a data structure for each one of them may not be a good idea. A worthwhile future endeavour could be to investigate the best ways of solving UCTPs with more than three objectives using the memory based GS strategy. It might be that grouping objectives according to their properties or creating a data structure according to time slot, event and room properties gives better solutions for MOUCTPs with many objectives. However, it would be interesting to investigate the limitations of the memory data structures in algorithms for solving the UCTP.

  We also intend to test our developed MOEAs on the ICT-2007 benchmark instances in particular and other problem instances that are available in the literature in general.

- It would also be instructive to see how our proposed approaches for the single-objective UCTP and the MOUCTP cope with various other scheduling problems, such as exam timetabling and sports timetabling.

In summary, the GS and LS strategies are effective in enhancing traditional GAs for solving the UCTP. Nevertheless, more work is greatly needed to develop a powerful general technique or a general framework of techniques to solve the UCTP. We believe that UCTPs can be effectively solved as MOOPs. Hybridisation of different MOEAs with other meta-heuristics and some domain knowledge might be able to give some general intelligent framework for solving the UCTP, along with the advantage of obtaining a set of non-dominated solutions rather than a single solution. We also believe that the proposed methods are open to different extensions and hybridisation. More advanced search mechanisms, appropriate genetic operators, and heuristics and knowledge-based evaluation routines, are greatly needed to develop efficient intelligent algorithms for UCTPs in particular and timetabling problems in general.

Appendix A

The t-Test Results

The t-test evaluates the differences in means between two populations in relation to the variation in the data (expressed as the standard deviation of the difference between the means). The t-test determines a p-value that shows how likely we could have gotten these results by chance. By convention, if there is less than 0.05 chance of getting the observed difference by chance, we reject the null hypothesis and say we found a statistically significant difference between the two groups [5].

In this thesis, for all our t-test in the experimental studies in Chapters 4, 5 and 6, we have used the 2-sample independent t-test. The null hypothesis is that "the means of two compared algorithms are equal". We carried out t-test to "reject the null hypothesis", which shows whether the difference between two compared algorithms is statistically significant.

Following tables show the data of different algorithms on different problem instances used for the t-test in the experimental studies.

Table A.1: The performance data of 50 runs of the Evolutionary Algorithm (EA) for the t-test results shown in Table 4.1 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
0	11	12	1	0	311	198	329	328	327
0	9	26	20	0	319	224	324	309	237
0	7	5	4	0	275	192	365	259	298
0	17	0	3	11	318	194	294	285	265
14	9	7	1	0	300	214	314	248	249
14	19	2	3	15	289	215	315	254	235
0	6	16	3	0	308	241	341	250	248
15	28	2	3	0	320	291	271	253	254
0	3	9	0	0	327	198	378	256	319
15	23	8	0	0	323	213	313	321	301
13	10	5	24	0	315	224	324	257	238
0	3	6	7	0	290	229	329	256	259
0	4	0	4	0	285	245	345	273	316
0	13	22	2	17	313	224	324	306	266
0	16	6	10	0	292	180	280	319	301
15	29	14	6	0	304	205	305	267	340
0	3	9	10	0	305	234	334	254	258
15	10	7	8	0	338	202	302	278	279
0	30	4	9	0	334	223	323	246	235
15	4	18	9	1	339	198	268	278	303
14	8	0	10	0	321	243	343	323	262
14	26	7	7	1	275	200	300	245	295
0	16	6	7	0	304	199	279	256	250
0	11	10	17	0	298	192	292	257	244
0	23	6	9	0	309	187	287	237	244
0	3	10	7	17	277	194	294	256	300
14	8	4	0	0	341	189	289	322	240
15	7	10	0	17	289	212	312	259	256
0	10	9	18	17	338	207	307	307	300
15	31	3	3	0	295	227	327	290	285
12	10	4	0	1	328	192	292	268	239
0	5	9	16	0	267	191	291	282	268
0	14	9	10	1	321	197	275	270	245
13	3	13	2	0	324	234	334	265	256
11	3	7	23	0	320	189	271	258	249
0	24	3	3	0	285	206	306	267	271
0	5	13	0	18	303	239	339	246	260
15	23	9	0	1	285	208	308	249	247
0	4	0	0	0	290	175	275	254	256
13	3	4	20	19	299	196	296	254	347
15	5	3	24	1	290	195	295	265	341
13	10	8	0	0	321	277	277	262	246
0	26	1	1	18	299	289	347	249	302
0	8	6	9	0	311	276	326	256	256
15	3	4	3	0	345	208	308	248	254
0	4	8	11	0	324	212	308	245	257
15	3	5	2	19	324	210	310	275	245
1	5	5	0	0	295	219	319	313	281
11	4	0	11	0	278	278	254	249	230
15	5	9	0	1	291	294	309	246	243

Table A.2: The performance data of 50 runs of the Memetic Algorithm (MA) for the *t*-test results shown in Table 4.1 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
0	0	12	0	0	224	229	237	149	287
14	15	0	0	6	245	223	243	165	278
0	0	5	6	0	238	182	232	155	258
0	8	10	0	0	232	194	248	164	221
3	0	3	0	0	225	214	243	162	227
1	0	17	5	5	248	215	263	156	207
1	10	0	0	7	229	241	300	206	205
0	12	0	0	0	217	171	286	179	209
12	14	17	0	6	233	171	294	145	205
14	0	0	0	0	256	213	234	184	223
1	9	1	6	3	227	224	245	165	209
6	8	0	0	6	231	229	233	143	200
4	10	19	0	0	243	245	264	179	199
0	6	0	0	0	249	224	275	168	212
0	6	5	7	4	245	180	265	156	221
0	7	0	0	0	223	205	263	156	249
4	6	14	0	0	254	234	275	168	258
0	7	0	0	6	240	202	263	156	215
4	14	9	0	4	269	223	290	183	233
3	11	0	5	6	228	185	301	200	224
1	11	0	0	0	225	243	228	185	232
1	8	14	0	2	234	200	235	183	238
0	9	0	0	0	255	179	291	143	276
1	10	0	0	0	257	192	235	153	262
13	6	0	0	7	243	187	243	184	256
0	7	1	7	0	256	194	357	154	204
0	12	17	6	0	227	189	301	204	199
0	6	0	0	0	243	212	232	163	225
0	6	0	0	0	260	207	237	163	232
0	10	1	0	4	248	227	257	152	254
15	8	11	0	0	238	192	239	132	254
0	15	1	5	0	225	191	234	206	222
1	19	0	0	6	227	175	233	191	221
0	12	0	0	2	217	234	232	169	197
15	0	0	6	0	231	171	252	185	227
10	18	0	0	0	257	206	222	201	219
1	10	5	0	0	249	239	253	153	211
13	12	19	0	6	260	208	243	154	218
0	8	15	0	6	254	175	234	154	208
1	0	10	3	7	253	196	236	143	207
0	14	3	6	6	227	195	243	165	219
15	0	9	0	7	262	177	230	162	212
0	2	9	0	0	271	247	242	143	215
0	13	5	0	7	226	226	234	143	254
4	17	16	0	0	230	208	278	156	274
0	2	1	0	0	224	208	234	168	219
9	5	0	0	4	228	210	242	174	210
0	12	16	0	2	254	219	238	141	214
0	9	0	0	0	243	267	228	154	213
3	16	1	0	7	212	209	230	145	202

Table A.3: The performance data of 50 runs of the Tabu Search (TS) for the t-test results shown in Table 5.4 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
0	2	15	8	1	212	234	281	254	279
5	25	0	3	0	222	256	278	260	292
0	7	3	18	15	227	240	227	256	294
6	21	3	2	14	211	213	262	242	281
0	24	5	2	0	287	222	280	238	289
0	1	3	2	15	263	216	288	236	284
0	26	4	2	15	218	255	281	230	303
0	20	22	6	0	225	212	265	216	308
8	1	2	20	6	213	243	268	214	311
9	1	0	2	1	218	265	298	187	309
2	24	17	9	0	214	266	281	176	275
0	22	0	2	0	220	242	279	171	236
9	2	16	20	16	219	245	279	166	265
0	1	0	2	0	218	198	298	231	300
0	20	1	2	0	261	188	271	187	273
0	8	19	14	6	262	234	272	212	234
0	17	6	2	5	216	198	286	202	303
0	22	0	2	0	217	274	287	234	307
5	1	14	12	1	211	243	285	212	301
0	7	22	22	11	230	246	266	198	300
0	20	1	2	0	212	234	263	176	288
0	20	1	14	0	214	202	276	175	308
0	5	16	3	9	214	205	274	189	301
0	27	10	20	2	234	218	279	197	282
4	20	6	6	0	216	276	283	167	290
1	9	7	7	0	216	234	267	188	311
0	18	12	12	16	213	243	298	256	326
0	20	9	6	9	224	277	276	217	353
4	22	3	6	1	212	234	266	235	302
0	8	10	20	12	257	275	278	233	247
0	25	8	9	15	221	212	281	251	353
0	22	15	21	1	214	263	289	232	273
2	20	20	8	6	214	272	292	189	284
0	1	21	20	0	218	272	281	256	281
0	2	0	2	0	212	199	284	186	296
9	4	1	18	10	212	204	280	197	282
9	27	7	7	0	244	185	291	241	300
0	11	15	17	12	222	189	265	245	307
1	20	13	9	3	214	198	276	186	320
0	22	0	2	16	211	198	267	175	275
0	1	14	18	0	218	189	278	187	308
9	1	11	3	2	212	187	278	189	251
0	25	0	2	0	245	256	264	188	296
0	22	20	3	0	211	257	286	190	238
9	22	21	21	13	212	186	283	187	285
0	3	14	7	0	212	215	272	176	246
9	23	1	5	16	219	254	276	178	302
0	27	0	20	14	240	277	287	185	246
0	1	0	2	0	214	221	280	183	256
2	26	18	21	16	219	212	297	182	312

Table A.4: The performance data of 50 runs of the Steady-State Memetic Algorithm (SSMA) for the t-test results shown in Table 5.4 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
0	11	12	1	0	311	198	329	328	327
0	9	26	20	0	319	224	324	309	237
0	7	5	4	0	275	192	365	259	298
0	17	0	3	11	318	194	294	285	265
14	9	7	1	0	300	214	314	248	249
14	19	2	3	15	289	215	315	254	235
0	6	16	3	0	308	241	341	250	248
15	28	2	3	0	320	291	271	253	254
0	3	9	0	0	327	198	378	256	319
15	23	8	0	0	323	213	313	321	301
13	10	5	24	0	315	224	324	257	238
0	3	6	7	0	290	229	329	256	259
0	4	0	4	0	285	245	345	273	316
0	13	22	2	17	313	224	324	306	266
0	16	6	10	0	292	180	280	319	301
15	29	14	6	0	304	205	305	267	340
0	3	9	10	0	305	234	334	254	258
15	10	7	8	0	338	202	302	278	279
0	30	4	9	0	334	223	323	246	235
15	4	18	9	1	339	198	268	278	303
14	8	0	10	0	321	243	343	323	262
14	26	7	7	1	275	200	300	245	295
0	16	6	7	0	304	199	279	256	250
0	11	10	17	0	298	192	292	257	244
0	23	6	9	0	309	187	287	237	244
0	3	10	7	17	277	194	294	256	300
14	8	4	0	0	341	189	289	322	240
15	7	10	0	17	289	212	312	259	256
0	10	9	18	17	338	207	307	307	300
15	31	3	3	0	295	227	327	290	285
12	10	4	0	1	328	192	292	268	239
0	5	9	16	0	267	191	291	282	268
0	14	9	10	1	321	197	275	270	245
13	3	13	2	0	324	234	334	265	256
11	3	7	23	0	320	189	271	258	249
0	24	3	3	0	285	206	306	267	271
0	5	13	0	18	303	239	339	246	260
15	23	9	0	1	285	208	308	249	247
0	4	0	0	0	290	175	275	254	256
13	3	4	20	19	299	196	296	254	347
15	5	3	24	1	290	195	295	265	341
13	10	8	0	0	321	277	277	262	246
0	26	1	1	18	299	289	347	249	302
0	8	6	9	0	311	276	326	256	256
15	3	4	3	0	345	208	308	248	254
0	4	8	11	0	324	212	308	245	257
15	3	5	2	19	324	210	310	275	245
1	5	5	0	0	295	219	319	313	281
11	4	0	11	0	278	278	254	249	230
15	5	9	0	1	291	294	309	246	243

Table A.5: The performance data of 50 runs of the Guided Search Genetic Algorithm (GSGA) for the *t*-test results shown in Table 5.4 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
0	0	2	0	0	262	212	230	189	124
0	0	4	6	1	269	214	254	155	145
0	0	0	2	4	245	205	241	183	138
6	0	0	2	0	267	194	253	166	123
0	2	0	0	0	240	156	264	172	125
0	12	0	3	0	249	205	255	200	148
0	1	0	2	0	248	204	241	206	199
0	0	2	0	0	249	171	272	165	167
0	0	0	3	0	259	207	278	157	151
9	0	1	3	0	243	213	243	167	193
0	0	1	3	0	265	204	264	165	128
6	0	0	2	0	260	163	259	208	131
7	0	7	0	0	245	166	253	156	131
0	15	8	2	0	243	156	244	164	129
0	0	0	4	0	269	180	281	190	145
0	0	0	0	0	256	205	285	156	123
8	0	6	0	5	267	185	265	166	193
0	0	7	0	0	248	178	252	166	140
7	0	0	4	0	242	165	245	192	163
0	0	0	4	0	267	168	268	178	128
9	16	1	0	5	241	204	253	162	125
0	0	0	1	0	254	200	278	163	187
0	0	0	0	0	243	179	254	154	166
0	0	2	0	0	245	192	254	165	172
1	0	1	0	0	270	187	278	155	143
0	15	0	4	0	254	194	244	155	142
0	0	9	4	0	253	189	289	204	127
8	0	10	2	0	254	202	265	183	134
0	0	6	0	5	247	207	267	155	155
0	0	9	4	0	265	204	256	157	148
8	0	10	0	0	245	192	256	154	141
0	16	0	1	0	267	190	291	206	154
0	0	0	3	0	253	175	276	156	127
0	0	0	0	0	242	203	272	169	120
0	0	0	1	0	240	171	246	158	129
8	10	7	0	0	266	206	246	201	172
0	15	4	0	0	267	204	239	154	194
6	0	0	0	0	248	165	278	156	126
0	0	0	0	0	250	175	275	156	142
1	0	0	1	0	255	169	294	206	153
3	0	0	5	0	250	156	295	155	190
1	0	0	5	0	262	166	254	163	164
0	0	2	4	0	266	201	246	172	149
7	0	1	0	0	244	212	252	162	126
0	0	0	2	2	250	167	253	165	162
0	0	5	0	0	245	201	254	166	150
9	14	0	1	3	270	210	256	162	153
0	0	5	3	0	255	190	265	172	154
1	0	0	11	0	245	187	267	172	143
0	0	0	0	0	240	156	265	158	212

Table A.6: The performance data of 50 runs of the Extended Guided Search Genetic Algorithm (EGSGA) for the t-test results shown in Table 5.4 on different problem instances

S1	S2	S3	S4	S5	M1	M2	M3	M4	M5
2	0	2	0	1	156	102	129	108	126
0	2	0	0	0	132	96	125	102	161
3	4	2	0	0	145	98	163	128	117
1	1	0	0	1	142	103	135	103	116
0	3	0	0	0	146	100	127	109	119
4	1	0	5	0	172	113	138	99	152
3	1	2	0	0	139	109	142	112	119
3	2	2	0	1	156	130	161	123	112
0	8	0	0	0	194	98	128	109	124
0	0	0	1	0	152	97	135	102	157
0	1	2	0	0	146	99	125	102	123
0	6	2	0	0	139	107	129	99	129
0	4	0	0	0	160	126	145	102	127
0	0	2	0	1	134	111	124	103	127
0	0	2	0	3	157	102	148	102	119
0	0	2	5	0	163	103	135	112	117
4	4	2	0	0	163	101	134	112	129
0	0	0	0	1	157	98	129	112	117
0	4	2	4	0	182	99	123	128	117
0	4	0	0	3	132	111	168	105	134
4	0	1	0	0	145	145	136	121	120
4	0	2	0	0	152	112	200	102	117
0	4	0	0	1	152	98	145	112	120
4	0	0	0	0	143	106	157	112	124
4	0	2	0	0	139	98	128	121	141
4	2	0	4	0	161	116	124	109	134
0	0	0	0	0	160	116	189	109	137
4	4	1	0	0	178	115	174	105	128
3	4	0	0	1	175	125	132	102	116
3	0	2	4	0	152	128	127	102	117
3	1	2	0	0	134	98	122	112	128
3	9	0	4	0	134	102	132	112	124
4	1	2	0	1	145	114	127	98	117
0	1	0	0	0	143	126	136	108	117
2	0	2	0	0	134	121	121	112	145
2	0	0	0	0	156	109	126	105	119
0	11	2	0	2	134	98	138	106	124
3	0	0	0	0	165	142	127	123	117
0	1	2	5	0	145	109	125	121	117
3	0	0	4	2	134	112	126	108	126
3	4	0	5	0	142	112	155	124	125
0	11	2	0	0	142	99	156	124	123
4	0	2	4	0	189	112	147	102	128
4	2	0	5	2	152	139	145	135	127
0	4	0	3	0	145	116	128	102	158
3	0	0	0	1	167	112	128	128	162
0	0	2	0	2	162	115	120	102	158
0	0	0	0	1	143	107	129	112	145
5	1	0	0	1	143	102	136	106	152
4	2	0	0	0	140	112	178	106	142

TABLE A.7: The Df values of 50 runs of the TS for the t-test results shown in Table 6.3 on different problem instances (2007-1 – 2007-12)

2007-1	2007-2	2007-3	2007-4	2007-5	2007-6	2007-7	2007-8	2007-9	2007-10	2007-11	2007-12
0	10	5	0	15	0	24	0	50	0	0	0
51	50	18	34	20	0	45	0	10	0	0	40
0	40	0	0	0	0	67	0	0	0	0	10
0	53	0	76	20	0	65	0	100	0	0	10
20	0	0	76	17	0	66	0	50	0	0	0
0	40	17	0	20	0	12	0	0	0	0	0
33	0	0	0	30	0	12	0	0	0	0	0
32	72	18	0	20	0	23	0	100	0	0	47
0	25	17	76	10	0	34	0	0	0	0	56
10	20	0	0	20	0	24	0	10	0	0	6
0	40	10	0	32	0	23	0	100	0	0	22
0	16	8	0	0	0	28	0	149	0	0	44
0	40	8	0	30	0	19	0	0	0	0	56
51	7	0	0	0	0	9	0	0	0	0	0
0	50	0	0	20	0	3	0	100	0	0	53
29	0	5	0	0	0	2	0	0	0	0	0
0	10	0	24	25	0	16	0	0	0	0	0
0	30	10	60	20	0	70	0	50	0	0	45
40	15	0	0	36	0	170	0	10	0	0	56
0	15	0	10	32	0	180	0	0	0	0	0
50	21	8	0	29	0	197	0	50	0	0	0
0	10	6	69	22	0	70	0	50	0	0	0
0	10	0	0	31	0	189	0	1	0	0	45
0	10	2	0	0	0	70	0	50	0	0	0
0	20	9	0	0	0	56	0	0	0	0	45
0	17	5	0	26	0	12	0	0	0	0	0
0	20	9	0	0	0	23	0	0	0	0	0
0	20	9	3	22	0	198	0	152	0	0	50
49	20	0	10	30	0	189	0	0	0	0	0
0	20	8	0	34	0	70	0	0	0	0	0
0	17	4	0	35	0	82	0	149	0	0	0
45	70	10	55	33	0	82	0	0	0	0	50
0	15	3	0	22	0	160	0	152	0	0	0
15	20	8	0	26	0	70	0	100	0	0	0
29	45	7	69	30	0	178	0	0	0	0	0
0	32	0	76	32	0	70	0	149	0	0	56
0	13	0	0	30	0	189	0	50	0	0	0
0	13	0	76	26	0	70	0	152	0	0	0
0	16	0	0	30	0	189	0	100	0	0	33
0	15	10	70	20	0	160	0	0	0	0	0
0	50	18	0	15	0	160	0	0	0	0	49
0	24	7	59	30	0	0	0	50	0	0	0
0	28	0	61	65	0	10	0	40	0	0	0
5	28	15	9	15	0	140	0	10	0	0	0
24	56	10	0	24	0	156	0	0	0	0	56
40	37	12	0	30	0	145	0	12	0	0	40
0	20	0	0	15	0	110	0	10	0	0	0
8	44	7	0	20	0	83	0	0	0	0	0
30	5	12	0	21	0	80	0	0	0	0	0
4	41	0	0	20	0	0	0	0	0	0	0

Table A.8: The Df values of 50 runs of the TS for the t-test results shown in Table 6.3 on different problem instances (2007-13 – 2007-24)

2007-13	2007-14	2007-15	2007-16	2007-17	2007-18	2007-19	2007-20	2007-21	2007-22	2007-23	2007-24
20	17	0	0	0	0	147	40	0	102	21	0
0	0	0	0	0	0	338	50	40	91	23	0
0	16	0	0	0	0	147	78	0	101	0	0
20	0	0	0	0	0	322	40	10	102	0	0
20	0	0	0	0	0	189	89	100	101	30	0
0	0	0	0	0	0	163	40	0	102	0	0
0	0	0	0	0	0	294	40	0	91	15	0
0	17	0	0	0	0	331	40	0	102	34	0
13	0	0	0	0	0	159	40	0	101	33	0
0	16	0	0	0	0	152	78	187	100	0	0
0	0	0	0	0	0	147	60	0	91	33	0
17	17	0	0	0	0	147	67	10	91	33	0
0	0	0	0	0	0	155	44	256	96	0	0
0	17	0	0	0	0	147	56	0	91	0	0
20	0	0	0	0	0	290	78	190	102	34	0
0	0	0	0	0	0	239	40	259	91	0	0
0	17	0	0	0	0	312	89	0	102	0	0
0	0	0	0	0	0	285	40	200	98	0	0
18	0	0	0	0	0	148	40	244	96	33	0
0	16	0	0	0	0	152	40	0	100	31	0
0	0	0	0	0	0	148	40	100	99	0	0
20	0	0	0	0	0	169	78	0	91	33	0
0	0	0	0	0	0	289	67	0	102	5	0
0	0	0	0	0	0	152	67	0	100	0	0
0	0	0	0	0	0	338	50	137	100	0	0
0	0	0	0	0	0	331	109	0	102	22	0
20	0	0	0	0	0	171	112	0	91	0	0
18	17	0	0	0	0	346	40	261	100	34	0
0	0	0	0	0	0	147	46	238	91	34	0
0	0	0	0	0	0	200	100	0	100	34	0
0	0	0	0	0	0	340	73	0	99	0	0
0	17	0	0	0	0	147	40	200	96	0	0
20	0	0	0	0	0	300	100	0	91	29	0
13	0	0	0	0	0	331	109	0	100	12	0
20	0	0	0	0	0	300	100	260	102	0	0
0	0	0	0	0	0	147	100	0	97	34	0
0	17	0	0	0	0	167	112	220	100	34	0
20	17	0	0	0	0	346	78	0	95	0	0
0	0	0	0	0	0	325	40	0	102	33	0
0	0	0	0	0	0	300	89	0	92	0	0
0	17	0	0	0	0	147	102	200	102	32	0
19	0	0	0	0	0	300	112	0	90	0	0
0	0	0	0	0	0	147	113	100	91	0	0
0	0	0	0	0	0	338	100	85	98	34	0
20	0	0	0	0	0	300	78	8	102	0	0
0	17	0	0	0	0	198	67	89	95	0	0
20	0	0	0	0	0	321	67	0	100	17	0
0	0	0	0	0	0	300	78	0	98	34	0
0	0	0	0	0	0	341	95	0	102	33	0
0	0	0	0	0	0	270	99	80	91	34	0

Table A.9: The Df values of 50 runs of the GALS for the t-test results shown in Table 6.3 on different problem instances (2007-1 – 2007-12)

2007-1	2007-2	2007-3	2007-4	2007-5	2007-6	2007-7	2007-8	2007-9	2007-10	2007-11	2007-12
0	40	0	0	20	0	70	0	115	26	0	0
12	0	0	0	4	0	0	0	0	0	0	0
0	10	0	0	0	0	60	0	0	0	0	0
10	0	0	0	20	0	0	0	34	0	0	0
0	0	0	0	0	0	70	0	0	0	0	0
12	3	0	0	15	0	0	0	0	22	0	0
1	0	0	0	0	0	70	0	109	20	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	20	0	0	0	0	0	0	0
0	0	0	0	0	0	70	0	78	20	0	0
12	34	0	0	20	0	70	0	78	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	18	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	25	0	0
11	0	0	0	18	0	0	0	112	0	0	0
0	0	0	0	19	0	0	0	0	0	0	0
9	27	0	0	0	0	0	0	0	0	0	0
0	0	0	0	20	0	0	0	56	13	0	0
12	46	0	0	0	0	0	0	78	0	0	0
0	0	0	0	20	0	0	0	0	5	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	20	0	0	0
0	0	0	0	20	0	0	0	0	0	0	0
0	0	0	0	0	0	70	0	0	0	0	0
12	0	0	0	20	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	67	0	0	0
0	0	0	0	0	0	70	0	0	0	0	0
0	0	0	0	20	0	70	0	0	0	0	0
12	0	0	11	18	0	0	0	0	0	0	0
0	46	0	11	20	0	0	0	0	25	0	0
0	48	0	11	20	0	0	0	0	0	0	0
0	0	0	11	0	0	0	0	100	26	0	0
0	10	0	11	0	0	0	0	0	0	0	0
2	0	0	11	19	0	70	0	23	0	0	0
0	0	0	9	0	0	0	0	0	10	0	0
0	0	0	9	0	0	28	0	56	0	0	0
12	51	0	9	20	0	70	0	78	0	0	0
5	0	0	7	0	0	70	0	100	25	0	0
0	0	0	0	20	0	0	0	0	0	0	0
11	35	0	0	0	0	70	0	67	0	0	0
0	47	0	0	20	0	0	0	34	0	0	0
0	3	0	0	0	0	0	0	56	13	0	0
0	0	0	0	0	0	70	0	0	0	0	0
0	0	0	0	18	0	69	0	0	0	0	0
11	0	0	0	0	0	70	0	98	0	0	0
0	39	0	0	0	0	0	0	60	3	0	0
0	0	0	0	0	0	70	0	0	17	0	0
0	0	0	0	20	0	0	0	22	0	0	0
12	0	0	0	0	0	69	0	0	0	0	0
12	0	0	0	2	0	0	0	0	0	0	0

Table A.10: The Df values of 50 runs of the GALS for the t-test results shown in Table 6.3 on different problem instances (2007-13 – 2007-24)

2007-13	2007-14	2007-15	2007-16	2007-17	2007-18	2007-19	2007-20	2007-21	2007-22	2007-23	2007-24
0	0	0	0	0	0	172	138	10	57	11	0
0	0	0	0	0	0	179	0	10	101	11	0
0	0	0	0	0	0	123	129	0	97	39	3
0	0	0	0	0	0	143	138	0	78	16	0
0	0	0	0	0	0	136	0	10	42	15	0
0	0	0	0	0	0	0	0	0	130	28	1
0	0	0	0	0	0	430	0	0	42	11	2
0	0	0	0	0	0	0	110	0	70	16	6
0	0	0	0	0	0	289	0	10	100	39	0
0	0	0	0	0	0	234	0	0	56	41	0
0	0	0	0	0	0	0	0	0	42	23	0
0	0	0	0	0	0	278	130	10	110	11	0
0	0	0	0	0	0	378	0	10	67	38	0
0	0	0	0	0	0	0	0	0	85	27	4
0	0	0	0	0	0	0	135	0	50	11	5
0	0	0	0	0	0	400	132	0	100	30	0
0	0	0	0	0	0	0	129	0	50	41	3
0	0	0	0	0	0	0	0	0	96	42	1
0	0	0	0	0	0	179	83	0	88	36	2
0	0	0	0	0	0	256	0	0	42	22	5
0	0	0	0	0	0	0	0	0	85	29	0
0	0	0	0	0	0	172	0	0	90	40	0
0	0	0	0	0	0	0	0	0	67	41	0
0	0	0	0	0	0	123	138	0	50	39	0
0	0	0	0	0	0	143	0	0	62	21	4
0	0	0	0	0	0	136	0	0	73	39	0
0	0	0	0	0	0	0	138	0	42	39	0
0	0	0	0	0	0	430	129	10	90	11	1
0	0	0	0	0	0	400	137	10	74	12	2
0	0	0	0	0	0	289	0	10	60	12	8
0	0	0	0	0	0	234	134	10	90	12	1
0	0	0	0	0	0	256	0	0	90	40	9
0	0	0	0	0	0	278	138	0	86	40	1
0	0	0	0	0	0	378	0	0	80	40	9
0	0	0	0	0	0	354	130	0	90	11	9
0	0	0	0	0	0	0	0	0	80	40	0
0	0	0	0	0	0	345	0	0	100	11	0
0	0	0	0	0	0	0	134	10	89	11	9
0	0	0	0	0	0	378	0	0	178	41	0
0	0	0	0	0	0	0	132	0	187	11	9
0	0	0	0	0	0	0	138	0	150	39	7
0	0	0	0	0	0	273	0	0	187	43	2
0	0	0	0	0	0	410	123	0	186	39	0
0	0	0	0	0	0	378	136	0	187	11	0
0	0	0	0	0	0	0	138	0	179	40	4
0	0	0	0	0	0	0	119	0	135	39	8
0	0	0	0	0	0	0	137	0	112	43	0
0	0	0	0	0	0	0	137	0	107	11	0
0	0	0	0	0	0	256	110	0	115	43	0
0	0	0	0	0	0	279	0	0	89	12	0

Table A.11: The Df values of 50 runs of the HGATS for the t-test results shown in Table 6.3 on different problem instances (2007-1 – 2007-12)

2007-1	2007-2	2007-3	2007-4	2007-5	2007-6	2007-7	2007-8	2007-9	2007-10	2007-11	2007-12
0	0	0	0	0	0	2	0	0	0	0	1
0	0	0	0	0	0	2	0	0	0	0	1
0	0	0	0	0	0	2	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	2	0	0	0	0	2
0	0	0	0	0	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	2	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
12	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	3
0	0	0	0	0	0	0	0	0	0	0	13
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	4	0	0	0	0	0	0	0	2
0	0	0	0	0	0	2	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	2	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	4	0	0	0	0	40	0	0	2
0	0	0	1	0	0	0	0	40	0	0	2
0	0	0	2	0	0	2	0	0	0	0	2
0	0	0	1	0	0	0	0	0	0	0	1
5	0	0	0	0	0	2	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	40	0	0	0
0	0	0	0	0	0	0	0	20	0	0	2
0	0	0	0	0	0	1	0	42	0	0	2
2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	42	0	0	2
0	0	0	0	0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0	3	0	0	1
0	0	0	4	0	0	2	0	0	0	0	2
0	0	0	0	0	0	2	0	0	0	0	2
0	0	0	0	0	0	0	0	0	0	0	0

TABLE A.12: The Df values of 50 runs of the HGATS for the t-test results shown in Table 6.3 on different problem instances (2007-13 – 2007-24)

2007-13	2007-14	2007-15	2007-16	2007-17	2007-18	2007-19	2007-20	2007-21	2007-22	2007-23	2007-24
0	0	0	0	0	0	23	189	6	69	0	0
0	0	0	0	0	0	67	199	5	0	0	0
0	0	0	0	0	0	54	190	5	58	0	0
0	0	0	0	0	0	98	0	6	73	0	0
0	0	0	0	0	0	0	0	7	0	0	0
0	0	0	0	0	0	46	213	5	0	0	0
0	0	0	0	0	0	112	0	6	70	0	0
0	0	0	0	0	0	23	213	8	0	0	0
0	0	0	0	0	0	55	90	6	56	0	0
0	0	0	0	0	0	12	0	5	0	0	0
0	0	0	0	0	0	0	0	5	70	0	0
0	0	0	0	0	0	211	0	6	53	0	0
0	0	0	0	0	0	209	200	5	0	0	0
0	0	0	0	0	0	267	0	5	0	0	0
0	0	0	0	0	0	245	0	5	0	0	0
0	0	0	0	0	0	297	100	0	63	0	0
0	0	0	0	0	0	26	0	0	0	0	0
0	0	0	0	0	0	275	0	0	0	0	0
0	0	0	0	0	0	0	220	0	72	0	0
0	0	0	0	0	0	112	190	0	70	0	0
0	0	0	0	0	0	145	0	0	69	0	0
0	0	0	0	0	0	132	202	0	0	0	0
0	0	0	0	0	0	111	0	0	0	0	0
0	0	0	0	0	0	0	43	0	0	0	0
0	0	0	0	0	0	24	0	0	73	0	0
0	0	0	0	0	0	17	0	0	73	0	0
0	0	0	0	0	0	312	14	0	73	0	0
0	0	0	0	0	0	276	0	5	0	16	0
0	0	0	0	0	0	25	0	2	67	0	0
0	0	0	0	0	0	277	0	2	0	0	0
0	0	0	0	0	0	0	213	3	0	0	0
0	0	0	0	0	0	112	0	2	69	5	0
0	0	0	0	0	0	11	202	2	0	5	0
0	0	0	0	0	0	223	0	2	0	5	0
0	0	0	0	0	0	267	0	2	65	11	0
0	0	0	0	0	0	0	213	0	69	1	0
0	0	0	0	0	0	0	0	2	72	3	0
0	0	0	0	0	0	299	213	2	0	16	0
0	0	0	0	0	0	287	220	2	0	0	0
0	0	0	0	0	0	312	0	2	61	0	0
0	0	0	0	0	0	100	100	0	67	0	0
0	0	0	0	0	0	301	0	2	0	0	0
0	0	0	0	0	0	309	0	1	69	0	0
0	0	0	0	0	0	300	200	1	0	0	0
0	0	0	0	0	0	155	234	1	47	0	0
0	0	0	0	0	0	200	0	1	55	0	0
0	0	0	0	0	0	190	0	1	0	0	0
0	0	0	0	0	0	40	100	2	73	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	100	10	0	0	0	0

Bibliography

[1] http://www.idsia.ch/Files/ttcomp2002 (Website of the 2002 International Timetabling Competition ITC-2002).

[2] http://www.cs.qub.ac.uk/itc2007 (Website of the 2007 International Timetabling Competition ITC-2007).

[3] http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm (Scholl's library of bin packing problem instances).

[4] http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html

[5] http://www.graphpad.com/quickcalcs/ttest1.cfm

[6] http://www.lania.mx/∼ccoello/EMOO/EMOObib.html (References of multi-objective evolutionary algorithms)

[7] http://www.aco-metaheuristic.org (The official website of the ant colony metaheuristic)

[8] http://www.metaheuristics.net (Website of the metaheuristics network)

[9] E. Aarts and J. Korst. *Simulated Annealing and Boltzman Machines*, Wiley, 1998.

[10] S. Abdennadher and M. Marte. University course timetabling using constraints handling rules. *Applied Artificial Intelligence*, 14(4): 311–326, 2000.

[11] A. Acan and Y. Tekol. Chromosome reuse in genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pp. 695–705, 2003.

[12] A. Acan. An external memory implementation in ant colony optimization. *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, pp. 73–82, 2004.

[13] S. Abdullah, E. K. Burke, and B. McCollum. An investigation of variable neighbourhood search for university course timetabling. *Proceedings of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, pp. 413–427, 2005.

[14] S. Abdullah, E. K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures. *Proceedings of the 6th International Conference on Meta-heuristic*, pp. 153–169, 2007.

[15] S. Abdullah, E. K. Burke, and B. McCollum. A hybrid evolutionary approach to the university course timetabling problem. *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pp. 1764–1768, 2007.

[16] K. Abdullah, D. W. Coit, and A. E. Smith. Multi-objective optimisation using genetic algorithms: a tutorial. *Reliability Engineering and System Safety*, 91: 992–1007, 2006.

[17] S. Abdullah and H. Turabieh. Generating university course timetable using genetic algorithm and local search. *Proceedings of the 3rd International Conference on Hybrid Information Technology*, pp. 254–260, 2008.

[18] S. Abdullah, K. Shaker, B. McCollum, and P. McMullan. Incorporating great deluge with Kempe chain neighbourhood structure for the enrolment-based course timetabling problem. *Proceedings of the 5th International Conference on Rough Set and Knowledge Technology*, LNAI 6401, pp. 70–77, 2010.

[19] S. Abdullah, H. Turabieh, B. McCollum, and P. McMullan. A multi-objective post enrolment course timetabling problems: a new case study. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pp. 1–7, 2010.

[20] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37(1): 98–113, 1991.

[21] Ç. H. Aladağ and G. Hocaoğlu. A tabu search algorithm to solve a course timetabling problem. *Hacettepe Journal of Mathematics and Statistics*, 36(1): 53–64, 2007.

[22] M. A. Al-Betar, A. T. Khader, and A. T. Gani, A harmony search algorithm for university course timetabling. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, pp. 18–22, 2008.

[23] A. Alkan and E. Ozcan. Memetic algorithms for timetabling evolutionary computation. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 1796–1802, 2003.

[24] M. Almond. An algorithm for constructing university timetables. *The Computer Journal*, 8: 331–340, 1965.

[25] E. Altshuler and D. Linden. Design of a wire antenna using a genetic algorithm. *Journal of Electronic Defence*, 20(7): 50–52, 1997.

[26] D. T. Anh, V. H. Tam, and N. Q. V. Hung. Generating complete university course timetables by using local search method. *Conference Internationale Associant Chercheurs Vietnamiens et Francophones en Informatique*, pp. 67–74, 2006.

[27] H. Arntzen and A. Løkketangen, A local search heuristic for a university timetabling problem. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[28] J. A. Atkin, E. K. Burke, J. Greenwood, and D. Reeson. Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. *Transportation Science*, 41(1): 90–106, 2007.

[29] H. Asmuni, E. K. Burke, and J. M. Garibaldi. Fuzzy multiple heuristic ordering for course timetabling. *Proceedings of the 5th UK Workshop on Computational Intelligence*, pp. 302–309, 2005.

[30] M. Atsuta, K. Nonobe, and T. Ibaraki. ITC2007 Track 2, An approach using general CSP solver. *www.cs.qub.ac.uk/itc2007*

[31] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. *Proceedings of the International Conference on Intelligent Technologies*, pp. 132–141, 2003

[32] M. A. Badr. A two-stage multiobjective scheduling model for faculty-course-time assignments. *European Journal of Operational Research*, 94(1): 16–28, 1996.

[33] P. Baptiste, C. Le Pape, and W. Nuijten, Incorporating efficient operations research algorithms in constraint based scheduling. *First International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[34] R. Battiti and G. Tecchiolli. The reactive tabu search. *Journal on Computing*, 6(2): 126–140, 1994.

[35] J. R. Blakesley. Automation in college management. *College and University Business*, 27: 39–44, 1959.

[36] C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search an application to open shop scheduling. *Computers and Operations Research*, 32(6): 1565–1591, 2005.

[37] P. P. Bonissone, R. Subbu, N. Eklund, and T. R. Kiehl, Evolutionary algorithms + domain knowledge = real-world evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 10(3): 256–280, 2006.

[38] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint Satisfaction Problems: Algorithms and Applications. *European Journal of Operational Research*, 119: 557–581, 1999.

[39] H. J. Bremermann. The evolution of intelligence. The nervous system as a model of its environment. *Technical Report No. 1*, Department of Mathematics, University of Washington, 1958.

[40] J. A. Breslaw. A linear programming solution to the faculty assignment problem. *Socio-Economic Planning Science*, 10: 227–230, 1976.

[41] S. Broder. Final examination scheduling. *Communications of the ACM*, 7(8): 494–498, 1964.

[42] E. K. Burke, Y. Bykov, J. Newall, and S. Petrovic. A time-predefined approach to course timetabling. *Yugoslav Journal of Operations Research (YUJOR)*, 13(2): 139–151, 2003.

[43] E. K. Burke, P. D. Causmaecker, G. V. Berghe, and H. V. Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6): 441–499, 2004.

[44] E. K. Burke, D. G. Elliman, and R. F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 605–610, 1995.

[45] E. K. Burke, E. Hart, G. Kendall, J. P. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. *Handbook of Meta-Heuristics*, Chapter 16, Kluwer, pp. 457–474, 2003.

[46] E. k. Burke and G. Kendall. *Search Methodologies*, Springer, 2005.

[47] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(6): 451–470, 2003.

[48] E. K. Burke, J. Kingston, K. Jackson, and R. Weare, Automated university timetabling: the State of the art. *The Computer Journal*, 40(9): 565–571, 1997.

[49] E. K. Burke and D. J. Landa-Silva. The design of memetic algorithms for scheduling and timetabling problems. *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, 166: 289–312, 2004.

[50] E. K. Burke, J. D. Landa-Silva, and E. Soubeiga. Multi-objective hyper-heuristic approaches for space allocation and timetabling. *Meta-heuristics: Progress as Real Problem Solvers*, pp. 129, 2003.

[51] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper heuristic for timetabling problems. *European Journal of Operational Research*, 176(1): 177–192, 2007.

[52] E. K. Burke, B. MacCarthy, S. Petrovic, and R. Qu. Structured cases in CBR-re-using and adapting cases for timetabling problems. *Knowledge-based System*, 13(2-3): 159–165, 2000.

[53] E. K. Burke, B. acCarthy, S. Petrovic, and R. Qu. Multiple-retrieval case-based reasoning for course timetabling problems. *Journal of the Operational Research Society*, 57(2): 148–262, 2006.

[54] E. K. Burke and J. P. Newall. A multistage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1): 63–74, 1999.

[55] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2): 266–280, 2002.

[56] Y. Bykov. Algorithm description. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[57] H. Cambazard, E. Hebrard, B. Ośullivan, and A. Papadopoulos. Local search and constraint programming for the post enrolment-based course timetabling problem. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[58] M. P. Carrasco and M. V. Pato. A multiobjective genetic algorithm for the class/teacher timetabling problem. *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, pp. 3–17, 2001.

[59] M. W. Carter A survey of practical applications of examination timetabling algorithms. *Operations Research Society of America*, 34(2): 193–202.

[60] M. W. Carter and G. Laporte. Recent developments in practical examination timetabling. *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1153, pp. 3–21, 1996.

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

[62] C. Y. Cheong, K. C. Tan, and B. Veeravalli. A multi-objective evolutionary algorithm for examination timetabling. *Journal of Scheduling*, 12(2): 121–146, 2009.

[63] M. Chiarandini, C. Fawcett, and H. H. Hoos. A modular multiphase heuristic solver for post enrollment course timetabling. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[64] L. M. Christine. A multiobjective framework for heavily constrained examination timetabling problems. *Annals of Operations Research*, 180(1): 3–31, 2008.

[65] B. Christian, P. Jakob, R. Günther, and R. Andrea. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6): 4135–4151, 2011.

[66] S. C. Chu and H. L. Frang. Genetic algorithm vs. tabu search in timetabling scheduling. *Proceedings of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering System*, 1999.

[67] C. A. Coello and G. Toscano. A Micro-Genetic Algorithm for Multiobjective Optimization. *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization*, pp. 126–140, 2001.

[68] C. A. Coello, D. A. Van Veldhuizen, and G. B. Lamont. Evolutionary Algorithms for Solving Multi-Objective Problems. *Kluwer Academic Publishers*, New York, 2002.

[69] M. W. Carter and G. Laporte. Recent developments in practical course timetabling. *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1408, pp. 3–19, 1998.

[70] P. Charbonneau. Genetic algorithms in astronomy and astrophysics. *The Astrophysical Journal Supplement Series*, 101: 309–334, 1995.

[71] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5): 403–432, 2006.

[72] A. J. Cole. The preparation of examination timetables using a small store computer. *The Computer Journal*, 7: 117–121, 1964.

[73] A. Colorni, M. Dorigo, and V. Maniezzo. Genetic algorithms - A new approach to the timetable problem. In: Akgul et al. (eds.), *NATO ASI Series, Combinatorial Optimization*, Lecture Notes in Computer Science 82, pp. 235–239, 1990.

[74] A. Colorni, M. Dorigo, and V. Maniezzo. Meta-heuristics for high school timetabling. *Computational Optimisation and Applications*, 9: 275–298, 1997.

[75] T. B. Cooper and J. H. Kingston. The solution of real instances of the timetabling Problem. *The Computer Journal*, 36(7): 645–653, 1993.

[76] J. F. Cordeau, B. Jaumard, and R. Morales, Efficient timetabling solution with tabu search. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[77] D. Costa. A tabu search for computing an operational timetable. *European Journal of Operational Research*, 76: 98–110, 1994.

[78] C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications*, 11(34): 223–224, 1998.

[79] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, New York, 2001.

[80] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pp. 849–858, 2000.

[81] K. Deb, M. Mohan, and S. Mishra. Towards a quick computation of well-spread Pareto-optimal solutions. *Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science 2632, pp. 222–236, 2003.

[82] D. Dasgupta. *Artificial Immune Systems and Their Applications*, Springer-Verlag, 1999.

[83] D. Datta, K. Deb, and C. M. Fonseca. Multi-objective evolutionary algorithm for university class timetabling problem. In K. P. Dahal, K. C. Tan, and P. I. Cowling (eds.), *Evolutionary Scheduling*, pp. 197–236, 2007.

[84] L. Davis. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[85] R. Dawkins. *The Selfish Gene*, Oxford University Press, 1976.

[86] R. Dawkins. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*, W. W. Norton, 1996.

[87] S. Deris, S. Omatu, and H. Ohta. Timetable planning using the constraint-based reasoning. *Computers and Operations Research*, 27(9), pp. 819–840, 2000.

[88] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. *New Ideas in Optimization*, McGraw Hill, pp. 11–32, 1999.

[89] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2): 137-172, 1999.

[90] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. *Technical Report No. 91-016*, Politecnico di Milano, Italy, 1991.

[91] M. Dorigo and T. Stützle. Ant Colony Optimization. MIT Press, Cambridge, 2004.

[92] K. A. Dowsland. Simulated Annealing. In Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, 1995.

[93] A. Dubourg, B. Laurent, E. Long, and B. Salotti, Algorithm description. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[94] G. Dueck. New optimization heuristics. *Journal of Computational Physics*, 104: 86–92, 1993.

[95] M. A. S. Elmohamed, P. Coddington, and G. Fox. A comparison of annealing techniques for academic course scheduling. *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1408, pp. 92–112, 1998.

[96] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4): 691–703, 1976.

[97] W. Erben and J. Keppler. A genetic algorithm solving a weekly course timetabling problem. *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1153, pp. 198–211, 1995.

[98] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. *Proceedings of the 5th international Conference on Genetic Algorithms*, pp. 416–423, 1993.

[99] C. M. Fonseca and P. J. Fleming. Multiobjective genetic algorithms. *Proceedings of the IEEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pp. 61–65, 1993.

[100] A. S. Fraser. Simulation of genetic systems by automatic digital computers. II: Effects of linkage on rates under selection. *Australian Journal of Biological Science*, 10: 492–499, 1957.

[101] J. Frausto-Solís, F. Alonso-Pecina, and J. Mora-Vargas. An efficient simulated annealing algorithm for feasible solutions of course timetabling. *Proceedings of*

the 7th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence (MICAI'08), pp. 675–685, 2008.

[102] B. Freisleben and P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 616–621, 1996.

[103] M. P. J. Fromherz. Constraint-based scheduling. *American Control Conference*, vol. 4, pp. 3231–3244, 2001.

[104] O. B. de Gans. A computer timetabling system for secondary schools in the Netherlands. *European Journal of Operation Research*, 7: 175–182, 1981.

[105] L. D. Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 2079, pp. 104–117, 2001.

[106] L. D. Gaspero and A. Schaerf. Multi-neighbourhood local search for course timetabling. *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, pp. 128–132, 2002.

[107] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[108] D. Goldberg. *Genetic algorithms in search, optimisation and machine learning.* Reading, MA: Addison-Wesley, 1989.

[109] H. Gunadhi, V. J. Anand, and Y. W. Yong. Automated timetabling using an object-oriented scheduler. *Expert Systems with Applications*, 10(2): 243–256, 1996.

[110] Y. Guo, E. C. Keedwell, G. A. Walters, and S. T. Khu. Hybridizing cellular automata principle and NSGAII for multi-objective design of urban water networks. *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science 4403, pp. 546–559, 2007.

[111] M. R. Garey and D. S. Johnson. *Computers and Intractability – A guide to NP-Completeness*, First Edition, San Francisco: W. H. Freeman and Company, 1979.

[112] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science.*, 1: 237–267, 1976.

[113] L. D. Gaspero and A. Schaerf, Algorithm description. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[114] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2): 60–68, 2001.

[115] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, Inc., 1997.

[116] M. Gendreau. An introduction to tabu search. *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp. 37–54, 2003.

[117] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13: 533–549, 1986.

[118] F. Glover, and G. A. Kochenberger. *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2003.

[119] F. Glover and M. Laguna. *Tabu search*, Kluwer Academic Publishers, 19977.

[120] C. C. Gotlieb, The construction of class-teacher timetables. *Proceedings of the International Federation of Information Processing (IFIP) Congress*, pp. 73–77, 1962.

[121] J. A. Hageman, R. Wehrens, H. A. Sprang, and L. M. C. Buydens. Hybrid genetic algorithm tabu search approach for optimizing multilayer optical coatings. *Analytica Chimica Acta*, 490: 211–222, 2003.

[122] L. He and N. Mort. Hybrid genetic algorithms for telecommunications network back-up routeing. *BT Technology Journal*, 18(4): 42–50, 2000.

[123] A. Hertz. Tabu Search for Large Scale Timetabling Problems. *European Journal of Operational Research*, 54: 39–47, 1991.

[124] A. Hertz. Finding a Feasible Course Schedule Using Tabu Search. *Discrete Applied Mathematics*, 35: 255–270, 1992.

[125] J. H. Holland. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[126] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, vol. 1, pp. 82-87, 1994.

[127] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. *Proceeding of the Twenty-Second Conference on Artificial Intelligence*, pp. 1152–1157, 2007.

[128] S. N. Jat and S. Yang. A memetic algorithm for the university course timetabling problem. *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 427–433, 2008.

[129] S. N. Jat and S. Yang. A guided search genetic algorithm for the university course timetabling problem. *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications*, pp. 180–191, 2009.

[130] G. Jones. Genetic and evolutionary algorithms. *In Encyclopaedia of Computational Chemistry*, John Wiley & Sons, Inc., pp. 1127–1136, 1998.

[131] S. F. H. Irene, S. Deris, and S. Z. M. Hashim, A study on PSO-based university course timetabling problem. *Proceedings of the 2009 International Conference on Advanced Computer Control*, pp. 648–651, 2009.

[132] G. Kendall. *Applying meta-heuristic algorithms to the nesting problem utilising the no fit polygon*. Ph.D. Thesis, School of Computer Science, University of Nottingham, UK, 2001.

[133] G. Kendall, S. Knust, C. C. Ribeiro, and S. Urrutia. Scheduling in sports: an annotated bibliography. *Computers and Operations Research*, 37(1): 1–19, 2010.

[134] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983.

[135] B. A. Knauer. Solutions of a timetable problem. *Computers and Operations Research*, 1(3-4): 363–375, 1974.

[136] J. D. Knowles and D. W. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pp. 98–105, 1999.

[137] O. Komolafe and J. Sventek. RSVP performance evaluation using multi-objective evolutionary optimisation. *Proceeding of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4, pp. 2447–2457, 2005.

[138] P. A. Kostuch. SA-based heuristic. *The 2002 International Timetabling Competition (TTC 2002)*, 2003.

[139] P. Kostuch. The university course timetabling problem with a three-phase approach. *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT V)*, Lecture Notes in Computer Science 3616, pp. 109–125, 2005.

[140] M. Laumanns, L. Thieleb, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, 10(3): 263–282, 2002.

[141] D. Leake. Case-Based Reasoning: Experiences, Lessons and Future Directions, AAAI Press, 1996.

[142] R. Lewis. A survey of metaheuristic based techniques for university timetabling problems. *Operation Research Spectrum*, 30(1): 167–190, 2008.

[143] R. Lewis. *Metaheuristics for university course timetabling.* Ph.D. Thesis, School of Computing, Napier University, Edinburgh, 2006.

[144] R. Lewis and B. Paechter. New crossover operators for timetabling with evolutionary algorithms. *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*, pp. 189–195, 2004.

[145] R. Lewis and B. Paechter. Application of the grouping genetic algorithm to university course timetabling. *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, Lecture Notes in Computer Science 3448, pp. 144–153, 2005.

[146] R. Lewis, B. Paechter, and B. McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. *Technical Report*, Cardiff University, 2007.

[147] Y. H. Liu. Different initial solution generators in genetic algorithms for solving the probabilistic traveling salesman problem. *Applied Mathematics and Computation*, 216(1): 125–137, 2010.

[148] S.J. Louis and G. Li. Augmenting genetic algorithms with memory to solve travelling salesman problem. *Proceedings of the Joint Conference on Information Sciences*, pp. 108–111, 1997.

[149] S. J. Louis and J. McDonnell. New methods for competitive evolution. *IEEE Transaction on Evolutionary Computation*, 8(4): 316–328, 2004.

[150] Z. Lü, and J. K. Hao, Adaptive tabu search for course timetabling. *European Journal of Operation Research*, 200(1): 235–244, 2010.

[151] M. R. Malim, A. T. Khader, and A. Mustafa, Artificial immune algorithms for university timetabling. *Proceedings of 6th International Conference on the Practice and Theory of Automated Timetabling*, pp. 234–245, 2006.

[152] A. Masri and J. Ghaith, Hybrid ant colony systems for course timetabling problems. *Proceedings of the 2nd International Conference on Data Mining and Optimization*, pp. 120–126, 2009.

[153] B. McCollum. University timetabling: bridging the gap between research and practice. *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pp. 15–35, 2006.

[154] P. McMullan. An extended implementation of the great deluge algorithm for course timetabling. *Proceedings of the International Conference on Computational Science*, pp. 538–545, 2007.

[155] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4): 333–346, 2002.

[156] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6): 1087–1092, 1953.

[157] N. Mladenovi and P. Hansen. Variable neighbourhood search. *Computers and Operations Research*, 24(11): 1097–1100.

[158] C. Morgenstern and H. Shapiro. Coloration neighborhood structures for general graph coloring. *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 226–235, 1990.

[159] P. Moscato. On evolution, search, optimization, genetic algorithms and martial art: towards memetic algorithms. *Caltech Concurrent Computation Program*, Technical Report, 1989.

[160] P. Moscato. Memetic algorithms: A short introduction. *New Ideas in Optimisation*, Mcgraw-Hill'S *Advanced Topics in Computer Science Series*, pp. 219–234, 2005

[161] T. Müller. *Constraint-based Timetabling* Ph.D. Thesis. Prague, 2005.

[162] T. Müller. ITC2007 Solver Description: A Hybrid Approach. *Proceedings of the 7th International Conference on the Practise and Theory of Automated Timetabling*, 2008.

[163] J. M. Mulvey. A classroom/time assignment model. *European Journal of Operational Research*, 9(1): 64–70, 1982.

[164] A. N. Nagar, J. Haddock, and S. Heragu. Multiple and bi-criteria scheduling: a literature survey. *European Journal of Operational Research*, 81: 88–104, 1995.

[165] K. Nonobe and T. Ibaraki. An improved tabu search method for the weighted constraint satisfaction problem. *INFOR*, 39(2): 131–151, 2001.

[166] C. Nothegger, A. Mayer, A. Chwatal, and G. R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[167] I. H. Osman and J. P. Kelly. *Meta-heuristics: Theory and Applications*, Kluwer Academic Publishers, 1996.

[168] I. H. Osman and G. Laporte. Metaheuristics: a bibliography. *Annals of Operations Research*, 63: 513–623, 1996.

[169] A. Osyczka. Multicriteria optimization for engineering design. In J. S. Gero (editor), *Design Optimization*, pp. 193–227, 1985.

[170] B. Paechter, A. Cumming, M. G. Norman, and H. Luchian. Extensions to a memetic timetabling system. *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 1153, pp. 251–265, 1996.

[171] D. B. Papoulias. The assignment-to-days problem in a school time-table, a heuristic approach. *European Journal of Operation Research*, 14: 31–41, 1980.

[172] L. F. Paquete, C. M. Fortseca, and E. L. Pt. A study of examination timetabling with multiobjective evolutionary algorithms. *Proceedings of the 4th Metaheuristics International Conference*, pp. 149–154, 2001.

[173] S. Petrovic and E. K. Burke. University timetabling. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapter 45, (Editor: J.Leung), CRC Press, 2004.

[174] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice Hall, 2002.

[175] M. Pirlot. General local search methods. *European Journal of Operational Research*, 92: 493–511, 1996.

[176] P. Pongcharoen, W. Promtet, P. Yenradee, and C. Hicks. Schotastic optimization timetabling tool for university course scheduling. *International Journal of Production Economics*, 112(2): 903–918, 2008.

[177] S. Prestwich, A. Tarim, R. Rossi, and B. Hnich. A steady-state genetic algorithm with re sampling for noisy inventory control. *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 5199, pp. 559–568, 2008.

[178] R. Qu, E. K. Burke, B. McCollum, and L. T. G. Merlot, A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1): 55–89, 2009.

[179] C. Reeves. *Modern heuristics techniques for combinatorial problems*, McGraw Hill, 1995.

[180] O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, K. Socha, and B. Paechter, local search for the timetabling problem. *Proceedings 4th International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science 2740, pp. 124–127, 2003.

[181] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle, A comparison of the performance of different metaheuristics on the timetabling problem. *Proceedings of the 4th International Conference on the*

Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science 2740, pp. 329–351, 2003.

[182] O. Rossi-Doria and B. Paechter. A memetic algorithm for university course timetabling. *Proceedings of Combinatorial Optimization* , pp. 56, 2004.

[183] C. D. Rosin and R. K. Belew. New methods for competitive evolution. *Evolutionary Computation*, 8(1): 1–29, 1997.

[184] H. Rudová and K. Murray. University course timetabling with soft constraints. *Proceedings of the 7th International Conference on the Practice and Theory of Automatic Timetabling*, Lecture Notes in Computer Science 2740, pp. 310–328, 2003.

[185] B. Ruggero, L. D. Gaspero, and A. Schaerf. A statistical analysis of the features of a dynamic tabu search algorithm for course timetabling problems. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, pp. 1–3, 2008.

[186] S. Salhi. Defining tabu list size and aspiration criterion within tabu search methods. *Computers and Operations Research*, 29: 67–86, 2002.

[187] R. Sarker, K. Liang, and C. Newton. A new multiobjective evolutionary algorithm. *European Journal of Operational Research*, 140(1): 12–23, 2002.

[188] K. Sastry, D. Goldberg, and G. Kendall. Genetic algorithms. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Chapter 4, pp. 97–125, 2005.

[189] J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. Ph.D. Thesis, Vanderbilt University, 1984.

[190] A. Schearf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2): 87–127, 1999.

[191] G. Schmidt and T. Ströhlein. Timetable construction an annotated bibliography. *The Computer Journal*, 23(4): 307–316, 1980.

[192] S. M. Selim. Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. *The Computer Journal*, 30(1): 76–82, 1988.

[193] W. Shin and J. A. Sullivan. Dynamic course scheduling for college faculty via zero-one programming. *Decision Science*, 8: 711–721, 1977.

[194] B. Sigl, M. Golub, and V. Mornar. Solving timetable scheduling problem using genetic algorithms. *Proceedings of the 25th International Conference on Information Technology Interfaces*, pp. 519–524, 2003.

[195] D. L. Silva and J. H. Obit, Great deluge with non-linear decay rate for solving course timetabling problems. *Proceedings of the 4th IEEE International Conference on Intelligent Systems*, pp. 811–818, 2008.

[196] K. Socha, J. Knowles, and M. Samples. A max-min ant system for the university course timetabling problem. *Proceedings of the 3rd International Workshop on Ant Algorithms*, Lecture Notes in Computer Science 2463, pp. 1–13, 2002.

[197] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2: 221–248, 1994.

[198] T. Stützle and H. H. Hoos. MAX–MIN Ant System. *Future Generation Computer Systems*, 16(8): 889–914, 2000.

[199] E. G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5): 541–565, 2002.

[200] N. D. Thanh. Solving timetabling problem using genetic and heuristics algorithms. *Journal of Scheduling*, 9(5): 403–432, 2006.

[201] J. Thompson and K. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25(7-8): 637–648, 1998.

[202] A. Tripathy. School timetabling – a case in large binary integer linear programming. *Management Science*, 30: 1473–1489, 1984.

[203] A. Tripathy. Computerised decision aid for timetabling – A case analysis. *Discrete Applied Mathematics*, 35(3): 313–323, 1992.

[204] M. Tuga, R. Berretta, and A. Mendes, A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. *Proceedings of 6th IEEE/ACIS International Conference on Computer and Information Science*, pp. 400–405, 2007.

[205] H. Turabieh and S. Abdullah. Incorporating tabu search into memetic approach for enrolment-based course timetabling problems. *Proceedings of the 2nd Data Mining and Optimisation Conference*, pp. 122–126, 2009.

[206] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1): 85–86, 1967.

[207] D. Werra. Graphs, hypergraphs and timetabling. *Methods of Operations Research*, 49: 201–213, 1985.

[208] D. Werra. An introduction to timetabling. *European Journal of Operations Research*, 19: 151–162, 1985.

[209] D. Werra. Some combinatorial models for course scheduling. *Lecture Notes in Computer Science 1153*, pp. 296–308, 1996.

[210] D. Werra. Restricted coloring models for timetabling. *DMATH: Discrete Mathematics*, 165/166: 161–170. 1997.

[211] L. D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4: 65–85, 1994.

[212] D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pp. 118–130, 1998.

[213] D. C. Wood. A technique for coloring a graph applicable to large-scale timetabling problems. *Computer Journal*, 12: 317–322, 1969.

[214] A. Wren. Scheduling, timetabling and rostering – a special relationship? *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT I)*, Lecture Notes in Computer Science, vol. 1153, pp. 46–75, 1995.

[215] G. G. Yen and H. Lu. Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. *IEEE Transactions on Evolutionary Computation*, 7(3): 253–274, 2003.

[216] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8: 338–353, 1965.

[217] K. Zervoudakis and P. Stamatopoulos. A generic object-oriented constraint-based model for university course timetabling. *Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT III)*, Lecture Notes in Computer Science 2079, pp. 28–47, 2001.

[218] L. Zhang and S. Lau. Constructing university timetable using constraint satisfaction programming approach. *Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation*, pp. 1–5, 2005.

[219] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Ph.D. Thesis, ETH Zurich, Switzerland, 1999.

[220] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization. *Metaheuristics for Multiobjective Optimisation*, Lecture Notes in Economics and Mathematical Systems 535, pp. 3-38, 2004.

[221] E. Zitzler, M. Laumanns, and L. Thiele. SPEA 2: Improving the strength Pareto evolutionary algorithm. *Proceeding of EUROGEN. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pp. 1–21, 2001.

[222] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4): 257–271, 1999.

[223] E. Zitzler, L. Thiele, and J. Bader. SPAM: set preference algorithm for multiobjective optimization. *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN X)*, Lecture Notes in Computer Science 5199, pp. 847–858, 2008.