Developing Novel Meta-heuristic, Hyper-heuristic and Cooperative Search for Course Timetabling Problems

by

Joe Henry Obit, MSc

A thesis submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computer Science
University of Nottingham

November 2010

Abstract

The research presented in this PhD thesis focuses on the problem of university course timetabling, and examines the various ways in which metaheuristics, hyper-heuristics and cooperative heuristic search techniques might be applied to this sort of problem. The university course timetabling problem is an NP-hard and also highly constrained combinatorial problem. Various techniques have been developed in the literature to tackle this problem. The research work presented in this thesis approaches this problem in two stages. For the first stage, the construction of initial solutions or timetables, we propose four hybrid heuristics that combine graph colouring techniques with a well-known local search method, tabu search, to generate initial feasible solutions. Then, in the second stage of the solution process, we explore different methods to improve upon the initial solutions. We investigate techniques such as single-solution metaheuristics, evolutionary algorithms, hyper-heuristics with reinforcement learning, cooperative low-level heuristics and cooperative hyper-heuristics. In the experiments throughout this thesis, we mainly use a popular set of benchmark instances of the university course timetabling problem, proposed by Socha et al. [152], to assess the performance of the methods proposed in this thesis. Then, this research work proposes algorithms for each of the two stages, construction of initial solutions and solution improvement, and analyses the proposed methods in detail.

For the first stage, we examine the performance of the hybrid heuristics on constructing feasible solutions. In our analysis of these algorithms we discovered that these hybrid approaches are capable of generating good quality feasible solutions in reasonable computation time for the 11 benchmark instances of Socha et al. [152]. Just for this first stage, we conducted a second set of experiments, testing the proposed hybrid heuristics on another set of benchmark instances corresponding to the international timetabling competition 2002 [91]. Our hybrid construction heuristics were also capable of producing feasible solutions for the 20 instances of the competition in reasonable computation time. It should be noted however, that most of the research presented here was focused on the 11 problem instances of Socha et al. [152].

For the second stage, we propose new metaheuristic algorithms and cooperative hyper-heuristics, namely a non-linear great deluge algorithm, an evolutionary non-linear great deluge algorithm (with a number of new specialised evolutionary operators), a hyper-heuristic with a learning mechanism approach, an asynchronous cooperative low-level heuristic and an asynchronous cooperative hyper-heuristic. These two last algorithms were inspired by the particle swarm optimisation technique. Detailed analyses of the proposed algorithms are presented and their relative benefits discussed. Finally, we give our suggestions as to how our best performing algorithms might be modified in order to deal with a wide range of problem domains including more real-world constraints. We also discuss the drawbacks of our algorithms in the final section of this thesis.

Acknowledgements

Thanks God for giving me the strength to start and finish this challenging journey. When I was down and tired you were always there for me no matter it were difficult or wonderful times, you always were holding me.

I would like to take this opportunity to express my sincere gratitude to my exceptional academic supervisor, Dr. Dario Landa-Silva for his trust, guidance, encouragement and constant support throughout this PhD journey. My thanks also go to EPSRC for funding this research project (Next Generation Decision Support- Information Sharing Mechanisms for Hyper-heuristics: Automating the Heuristic Design Process (EP/D061571/1)) and UMS who were responsible for funding this project.

Many thanks to Dr. Djamila Ouelhadj and Dr. Rong Qu for their advice and encouragement. To my mother, sisters and brothers, thanks to all of you for giving me indirect support in finishing this challenging task. To my wife, Sue and my little Dansih who always bring joy and laughter, thanks for being with me.

Last but not least, my thanks go to ASAP research group, especially Dr. Jason Atkin, Dr. Tim Curtois, Dr Peer-Olaf Siebers, Dr Jan Feyereisl,Dr Yuri Bykov, Juan Pedro, Sven, Khoi, Bob, Rupa Jagannathan, Abdullah and his wife Azura and many others for your help.

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Timetabling is a type of assignment problem, where each problem has its own unique characteristics and variations which differ from one organisation to another. In the modern world where it is crucial to avoid time and resources wastage, the timetabling of activities requires that resources are in place at the right time and in the correct quantity in order to operate effectively and efficiently.

An example of a timetabling problem which must be solved effectively is train timetabling. The construction of a train timetable must take into account where and when the train starts and ends its journey every day. The availability and drivers' preferences (based on seniority) also need to be considered, including how many hours the drivers are able to work per shift every week.

Bus timetables are also another example, a problem that shares many features with train timetabling. However, there are other features which do not exist in train timetabling but are important attributes of bus timetabling, such as dead mileage and lay-overs. Dead mileage is the travelling distance from the last point of the service to the depot whilst not in service. Lay-over is the break given to the bus at the end of a trip before it starts operating its reverse route. Without proper bus and train timetables, the entire transportation network system in any city or country would be in turmoil. Hence, the construction of timetables is extremely important, for it usually affects people daily lives.

Examination timetabling is another example of an important timetabling problem. In these problems, each examination activity must be assigned to the right timeslot avoiding clashes between exams that have students in common. Once the clashing problem has been solved, the quality of the timetable is often improved by spreading the exams as much as possible in order to give adequate time for students to revise before sitting the exam, or to have sufficient rest before starting the next exam.

This short overview of some of the different types of timetabling has shown the importance of timetabling in many scenarios. Whilst the focus of this thesis is on the university course timetabling problem, the same objectives are usually involved in many other forms of timetabling problems, where the aim is to ensure that the resources are allocated at the right time and in the right place, avoiding conflicts that can provoke chaos. One objective commonly found in timetabling is to satisfy all people who are directly affected by the timetable, such as examinees and invigilators. Another objective in timetabling is usually to control costs. For example, poor quality course timetabling can cause higher cost for educational institutions, since students will not be able to attend all of their lessons when clashes exist and may need to instead take the course next term, extending their study time. From a lecturer's perspective, they might not be able to teach all their courses because all or some of them have been assigned to the same timeslot. Alternatively, lecturers may be unable to deliver their lectures if their courses have been assigned to rooms with unsuitable teaching equipment. In addition, poor quality timetables can also result in students having to attend more than two consecutive courses without a break. This is known to have a detrimental effect upon student concentration.

The University Course Timetabling Problem (UCTTP) is known to be a very difficult combinatorial problem and it has been extensively studied over the last few decades. The general timetabling problem is known to be NP-hard [42, 75](see the description in section 2.3). Therefore, in general there is no known efficient deterministic algorithm which will solve all instances in polynomial time. In addition most timetabling problem by nature is highly constrained and complicated. For example, feasible, good and acceptable timetable are very different from one university to the others subject to their own interpretation. Moreover, different universities are likely to have their own set of timetable constraints that need to be satisfied. In spite of various techniques have been proposed by researchers in recent years to automatically generate solutions to benchmark and real-world timetabling problems, unfortunately, in some cases if not often that an algorithmic approach that is successful for one particular problem may not perform well for other problems.

The construction of a course timetable is a very complex problem common to a wide range of educational institutions. Many factors contribute to this complexity. Among them we distinguish the requirements to satisfy various constraints. For example: each course should be composed of the correct number of lectures, no student can attend more than one lecture at the same time, and one lecture has to be scheduled in exactly one room, etc. [148, 152]. Furthermore, the introduction in many universities of a modular course structure, where each student is allowed to choose a set of subjects, has made the timetabling process (for both courses and exams) even more complex [35]. Hence, the manual process of preparing the timetable is tedious, time consuming, error prone and not even guaranteed to produce a timetable free of constraint violations. The complex nature of the timetabling problem has attracted the attention of many researchers and practitioners. It has been observed that local search algorithms might be more efficient in solving timetabling problems if the problem is split into smaller sub-problems [33]. Carter proposed a decomposition method to break large instances into smaller sub-problems which then can be successfully handled by a local search algorithm [37]. Recently, some researchers have proposed the use of multi-agent systems for timetabling problems. A Multi-agent system is a network of agents that work together to solve problems that are beyond the agent's individual capability [124]. Multi-agent systems are distributed and autonomous systems made up of autonomous agents that support reactivity, and are robust against failures both locally and globally [133]. Multi-agent systems have been applied for a long time to problems in other domains such as production scheduling, employee timetabling, e-commerce, etc., and they have produced good results [46, 132, 162]. In contrast, not much research has been conducted on the application of multi-agent systems to tackle educational timetabling problems. Kaplansky et al. claimed that the distributed nature of the timetabling problem can be tackled by using the multi-agent paradigm [93]. Each agent in their model has a different set of requirements that lead them to finding high quality solutions. In order to coordinate their own timetable, all agents in the distributed environment have to communicate and negotiate to avoid conflicts in the process of allocating the shared resources. Kaplansky et al. used mainly a nurse rostering problem to investigate their multi-agent model. In the present thesis, we propose an asynchronous hyper-heuristic that also incorporates elements of particle swarm optimisation as one of the contributions of this work. Hyper-heuristics are a relatively new type of search methodology with the aim of developing general domain-independent search methods capable of performing well enough, soon enough, and cheap enough across a wide range of optimisation problems [25]. When given a particular problem instance and a number of low-level heuristics, the hyper-heuristic process manages the selection and acceptance of the low-level heuristics to apply at any given time, until a stopping condition is met. A low-level heuristic is a simple local search operator or domain dependent heuristic.

The work presented by Burke et al. [33] has suggested that advanced algorithms such as evolutionary algorithms will not fully solve the complex timetable problem especially when dealing with large instances. The problem with these algorithms is that to find near-optimal solutions for timetabling is extremely difficult because of the large number of constraints which limit the feasibility of solutions. As a result, Burke et al. [35] later proposed a decomposition method to break a large instance into small sub-problems so that the algorithms were able to handle these smaller problems and find near-optimal solutions. The process of decomposition has also been studied by Carter [37] who used a heuristic method to split large instances into small problems until each sub-problem was small enough to be solved by local search algorithms.

1.2 Summary of Contributions

The focus of this thesis is to investigate meta-heuristic, hyper-heuristic and cooperative search approaches for tackling the university course timetabling problem (UCTTP). In this work, we investigate the Great Deluge algorithm and how this simple yet effective method can improve the quality of the timetable by extending the algorithm with a non-linear decay and floating water level. In recent years, the demand for more general frameworks for decision support systems to aid in the solution of timetabling and other problems has increased. This has driven our work into the development of a general framework for hyper-heuristics. Therefore, it is worth investigating hyper-heuristics and the use of machine learning technique in order to select the low-level heuristics without involving too much parameter tuning. To harvest the benefit of cooperative search, we investigate how beneficial it is when more than one hyper-heuristic cooperate to explore a large region in the search space. We also propose here several approaches for generating feasible solutions to the UCTTP as the first stage of a two-stage optimisation strategy. In this approach, we study existing methods such as local search, tabu search and graph colouring and investigate how beneficial it is to hybridise those methods to compensate for the weaknesses of each other. Therefore, in this research, we try to find simple algorithms (reducing the complexity, increasing the robustness and effective) rather than very complex ones for UCTTP. We outline next the scientific contributions of the present PhD thesis:

1. Development of four methods for the construction of feasible solutions for the UCTTP.

2. An investigation of the Great Deluge algorithm originally proposed by Dueck [68]

    - Extension of the great deluge algorithm with a non-linear decay rate and floating water level. Evaluation of the benefit of modifying the water level decay rate from linear to non-linear and floating in the great deluge algorithm. Therefore, the Non-Linear Great Deluge (NLGD) acceptance criterion always accepts improving solutions and non-improving solutions are only accepted if the objective function value is less than or equal to a certain water level.

3. A hybridisation of the non-linear great deluge approach with elements of an evolutionary strategy.

4. Development of a non-linear great deluge hyper-heuristic approach with several variants:

    - Non-Linear Great Deluge Hyper-heuristic (NLGDHH-SM) with static memory length.
    - Non-Linear Great Deluge Hyper-heuristic (NLGDHH-DM) with dynamic memory length.

- In this chapter we investigate the non-linear great deluge algorithm within a hyper-heuristic framework. The Non-Linear Great Deluge Hyper-Heuristic (NLGDHH) uses a learning mechanism for the selection of low-level heuristics, and a NLGD acceptance criterion. The learning mechanism has knowledge about the performance of each heuristic during the search. This knowledge is used to guide the hyper-heuristic in selecting a low-level heuristic at each decision point of the search.

5. Development of a cooperative hyper-heuristic algorithm.

   - We investigate the effectiveness of asynchronous cooperative search within a hyper-heuristic approach. In this work, particle swarm optimisation, which is inspired by intelligent social and individual behaviour of swarms, was used as inspiration. Based on the hyper-heuristic and particle swarm optimisation perspective, we then proposed a novel search methodology in which the interaction between cooperative low-level heuristics and cooperative hyper-heuristics mimics the particle swarm behaviour.

1.3 Scientific Publications Resulting From This Thesis

The research work described in this thesis has been disseminated in a number of scientific papers already published or under submission. Almost every chapter discusses in this thesis has been published or is in press. The list of papers published or in press is provided bellow in reverse chronological order:

1. Joe Henry Obit, Dario Landa-Silva, Juan P Castro Gutierrez, Djamila Ouelhadj, Rong Qu. A Particle Swarm Optimisation Inspired Asynchronous Cooperative Distributed Hyper-heuristic. Submitted to Journal of Heuristics.

2. Joe Henry Obit, Dario Landa-Silva, Marc Sevaux, Djamila Ouelhadj. Non-Linear Great Deluge with Reinforcement Learning for University Course Timetabling. Post-conference Volume of the 2009 Metaheuristics International Conference (MIC 2009).

3. Joe Henry Obit, Dario Landa-Silva. Computational Study of Non-Linear Great Deluge for University Course Timetabling. Intelligent Systems: From Theory to Practice, Sgurev, Vassil and Hadjiski, Mincho and Kacprzyk, Janusz (eds), Studies in Computational Intelligence, Vol. 299, pp. 309-328, Springer-Verlag, 2010.

4. Joe H. Obit, Dario Landa-Silva, Djamilah Ouelhadj, Marc Sevaux. Non-Linear Great Deluge with Learning Mechanism for Solving the Course Timetabling Problem. Proceedings of the 8^{th} Metaheuristics International Conference (MIC 2009), Hamburg Germany, 2009.

5. Dario Landa-Silva, Joe Henry Obit. Evolutionary Non-Linear Great Deluge for University Course Timetabling. Proceedings of the 2009 International Conference on Hybrid Artificial Intelligence Systems (HAIS 2009), Hybrid Artificial Intelligent Systems, Lecture Notes in Artificial Intelligence, Vol. 5572, Springer, pp. 269-276, 2009.

6. Dario Landa-Silva, Joe Henry Obit. Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), IEEE Press, pp. 8.11-8.18, 2008.

1.4 Thesis Guide

This thesis consist of nine chapters. This first chapter explained the background, motivation and main aim of this research. The rest of this thesis is organised in the following way:

Chapter 2 gives a review of the state-of-the art and an analysis of the works which have been published on the subject of university course timetabling.

Chapter 3 gives a detailed description of the problem domain. This chapter also includes the problem formulation and the description of standard benchmark instances (University Course Timetabling Problem) that we used as the test bed for our proposed algorithms.

Chapter 4 is dedicated to the initialisation of timetables. In solving the course timetabling problem, most meta-heuristic approaches fall into one of the following categories: one-stage, two-stage or algorithms that allow relaxations. Our overall solution approach in this thesis falls into the second category, i.e. a two-stage optimisation strategy. Chapter 4 details how we construct a feasible solution for the UCTTP. In order to examine the effectiveness of our proposed construction methods, we conducted experiments by using standard benchmark problem instances and obtained very good results.

Chapter 5 investigates the application of the great deluge algorithm invented by Dueck [68] to the UCTTP tackled here. The aims of the initial study were to conduct a thorough investigation of this simple but effective algorithm, and study how it could be further extended. We also discuss the proposed modification of the linear decay rate in great deluge to a non-linear decay rate with floating water level. We then tested this modified algorithm on the standard benchmark instances to examine how beneficial it is to modify the decay rate. The experimental results on the UCTTP show that the extended great deluge outperforms some of the best results reported in the literature.

Chapter 6 continues the study of the great deluge algorithm proposed in chapter 5 (non-linear decay rate with floating water level). In that chapter we propose the hybridisation of that algorithm with some evolutionary operators in order to further improve the timetable quality.

Chapter 7 furthers our research of non-linear great deluge by implementing a stochastic selection mechanism as a tool to select low-level heuristics. The result is a proposed non-linear great deluge hyper-heuristic method. The difference between this hyper-heuristic algorithm and the algorithm of chapter 5 is the stochastic selection mechanism that takes into account the success rate history of the low-level heuristics. Therefore, the focus of chapter 7 is more on the learning mechanism and how this mechanism selects the low-level heuristic to apply at every decision point. As in the previous chapters, we also use the standard benchmark problem to examine the performance of the proposed learning mechanism.

Chapter 8 incorporates cooperation strategies in our non-linear great deluge hyper-heuristic approach of chapter 7. Asynchronous cooperative search in complex combinatorial optimisation has been studied in many research papers and such strategies have been demonstrated to produce promising results on other problems. We propose the cooperation of low-level heuristics and cooperation of hyper-heuristics with different strategies and acceptance criteria such as non-linear great deluge and simulated annealing. The proposed algorithm is then a population-based algorithm and was inspired by Particle Swarm Optimisation.

Last but not least, the overall conclusions of the work presented in this thesis and research directions for future work in this area are presented and discussed in Chapter 9.

Chapter 2

Literature Review

2.1 Introduction

This chapter discusses some of the fundamental aspects of the automated timetabling research area and some of the previous work in the literature covering different optimisation techniques to tackle course timetabling problems. This chapter also describes several timetabling problems in addition to the university course timetabling problem which is the focus in this thesis.

The content of this chapter is divided into three sections. Section 2.2 defines various educational timetabling problems. Section 2.3 discusses a course timetabling model based on graph colouring. Section 2.4 provides an overview and discussion of the different approaches used to tackle the university course timetabling problem.

2.2 Timetabling

Timetabling can be defined as the process of allocating, subject to constraints on given resources, a number of events in space and time, in such a way as to satisfy as nearly as possible a set of desirable objectives [163]. As noted by Wren [163], constructing good quality timetables is a difficult task due to the combinatorial and highly constrained nature of most timetabling problems, which are common problems for all institutions of higher education like universities. The construction of a timetable to ensure all activities are in place accordingly is a very challenging task as many factors contribute to its complexity, for example the need to satisfy a considerable number of hard and soft constraints.

A timetable is said to be feasible when it can allocate sufficient resources (room space, time, people, etc.) for every event to take place. As in many other combinatorial problems, the constraints in timetabling can be distinguished into *hard constraints* and *soft constraints.* Hard constraints need to be satisfied in all circumstances, whereas soft constraint violations should be minimised to increase the timetable quality, increasing the satisfaction of the people who are affected by the timetable.

Generally, educational timetabling problems can be classified into three main classes according to Schaerf [148]. These classes are:

- School timetabling: a school timetable usually follows a cycle every week for all classes and the objective is to avoid teachers having to attend two classes at the same time. In school timetabling students are normally pre-assigned, only teachers and rooms need to be assigned in the timetabling problem [95]. According to Santos et al. [146], the basic hard constraints that must be satisfied in school timetabling are: (i) no teacher should be assigned to more than one class in the same timeslot; (ii) take into account the teacher's availability for each timeslot; and (iii) allocate the right number of timeslots for each teacher-class pair.
- University course timetabling: university course timetabling is also the weekly scheduling of all lectures of a set of university courses, avoiding lectures that

have students in common being assigned to the same timeslot. Therefore, university course timetabling is the process of allocating, subject to constraints, limited rooms and timeslots for a set of courses to take place. Usually, in addition to constructing a feasible timetable (all hard constraints satisfied), there are desirable goals like minimising the number of undesirable allocations (e.g. students attend more than two consecutive events on a day). Such desirable goals are usually expressed as soft constraints.

- Examination timetabling: the set of university exams needs to be scheduled into a limited number of timeslots (periods) avoiding cases where students take more than one exam in the same timeslot (the clash free requirement). Each room has a certain capacity which must not be violated when assigning an exam or exams to it. These constraints are known as hard because they are inflexible. Besides the hard constraints there are usually several soft constraints that are considered to be desirable but not compulsory. Obviously, significant differences exist across institutions in what constraints they consider compulsory and which they do not [24]. Examples of commonly occurring soft constraints are where students prefer to spread the exams as much as possible throughout the examination week, or the institution wants to schedule large exams earlier (to give more time for marking), some institution staff-members might also have specific preferences, for example, with respect to invigilation duties.

In reality, these two types of problem (examination and course timetabling) are fairly similar in some characteristics, however, there are some distinct underlying differences between them. For example, in examination timetabling, several examinations can be scheduled into one large room at the same time providing that the seating capacity of the room is not exceeded, whilst, this is not possible for course timetabling, where the assignment is generally only allowed one course per room, per timeslot.

As mentioned earlier, different universities have their own particular set of timetable constraints. A common hard constraint is that no student is assigned to more than one course at the same time. In ideal circumstances, a feasible timetable is only considered feasible if and only if all the hard constraints are not violated. Soft constraints, meanwhile, are those that are required to be satisfied as much as possible, in other words, they are desirable but are not compulsory. The quality of the timetable is usually determined by the violation of the soft constraints. In reality, it would usually be impossible to satisfy all of the soft constraints in a given problem. For example, when certain events should occur before certain others or common students should not attend three or more events in successive timeslots occurring in the same day.

In general, although the above timetabling problems share the same basic characteristics, significant differences among them still exist. Extensive surveys on timetabling can be found in [24, 95, 146, 148].

2.3 Graph Colouring Model for University Course Timetabling

Many researchers in the Artificial Intelligent and the Operational Research communities have formulated real world timetabling problems in educational institutions using graph colouring theory. The UCTTP can be modelled as follows. Nodes represent events or courses and edges joining these nodes represent conflicts between the corresponding events (a conflict occurs when at least one student is registered for two different courses) [55]. Therefore, in a graph coloring formulation, given an undirected graph $G = (V, E)$, $V = \{v_1, v_2, v_3..., v_n\}$ is the set of nodes with size $|V| = n$ and $E = \{e_{ij} | \exists \; \textit{edge between } v_i \textit{ and } v_j\}$ is the set of edges. Therefore, if (v_i, v_j) is an edge in a graph $G = (V, E)$, then vertex v_i is adjacent to vertex v_j. The graph colouring problem is to determine a k-colouring of $V = \{v_1, v_2, v_3..., v_n\}$ in such a way that no two adjacent vertices are given the same colour. That is, this means not assigning the same timeslot to conflicting events in the UCTTP.

To illustrate the connection between the graph colouring problem and the UCTTP, Figure 2.1 shows this relationship between graph colouring and timetabling. Graph (1) in Figure 2.1 shows the undirected graph where nodes represent events and arcs connecting nodes represent conflicts between the events. This means that if two events are in conflict (have students in common), they must not be assigned to the same timeslot. This is similar to the well known problem of graph-colouring [20] in which the vertices of a graph must be assigned a colour such that no two vertices sharing a common edge have the same colour. The problem in Figure 2.1 contains ten courses, labelled A to J together with a set of edges showing the conflicting courses. Graph (2) can easily be seen that no solution can be found using fewer than five timeslots (colours), it also shows a colouring that assigns different colours (timeslots) to the adjacent nodes to avoid conflicts. For example, since courses A, B, C, D, F and G all clash with each other and must therefore all be in different colours (timeslots). while Table(3) in Figure 2.1 illustrates how the graph is translated into a real timetable. In this example courses A and J assigned to green colour (timeslot one), courses B, D and I assigned to purple colour (timeslot two), courses C and E assigned to red colour (timeslot three), course F assigned to black colour (timeslot four) and finally courses G and H assigned to blue colour (timeslot five).

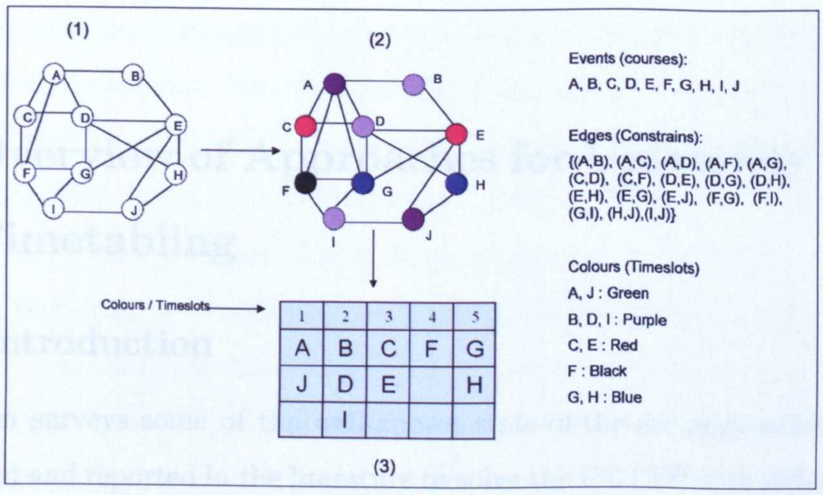


Figure 2.1: Graph Colouring Model for a Simple Course Timetabling Problem.

In the related graph problem, the chromatic number is a term that used to indicate minimum number of colours that are needed to colour a particular problem instances. In simple timetabling problem this is equivalent to a course timetabling problem in which the minimum number of timeslots that are needed to construct feasible timetable which is a clash-free. It is said to be that determining the chromatic number is also an NP-hard problem [111].

The identification of cliques is another important feature of graph problems. In the real world timetabling problem instances usually far more complex, sometimes containing up to 1000 courses and will often contain quite large cliques. A clique is a collection of vertices that are mutually adjacent, such as vertices A, C, D, F and G in Figure 2.1. Therefore, when given a graph colouring instance that has a maximum clique size of x, therefore a minimum x colours will be needed to colour the graph. Finding cliques of a given size within a graph is in itself an NP-hard problem. In the example in Figure 2.1, courses/vertices A, C, D, F and G form a clique and hence the minimum number of colours/timeslots required to solve the problem is five. The task of finding cliques of a given size within a graph is also an NP-hard problem [111].

2.4 Overview of Approaches for University Course Timetabling

2.4.1 Introduction

This section surveys some of the well-known state-of-the-art approaches that have been applied and reported in the literature to solve the UCTTP with different success levels. The aim of this chapter is to provide a brief discussion of some of the standard meta-heuristic, hyper-heuristic and distributed hyper-heuristic methods which are commonly used in many optimisation problems, especially in timetabling optimisation problems. Rather than being exhaustive, the main objective of this survey is to give a consistent background on the state-of-the-art research in this area, which underpins the work in this thesis.

This overview section is divided into three subsections as follows. Subsection 2.4.1.1 gives an overview of meta-heuristic approaches applied to the UCTTP including results reported in the literature. Subsection 2.4.1.2 focuses on more recent approaches where algorithms called hyper-heuristics have been applied to solve the UCTTP. Finally, subsection 2.4.1.3 surveys distributed hyper-heuristics which comprise the more recent research direction in this problem domain.

2.4.1.1 Review of Meta-heuristics

- **Constraint-based Methods**

  Constraint programming is the study of computational systems based on constraints. A constraint can be defined as a restriction on a space of possibility, in other words the restriction of possible values that a variable can take in a given domain [88]. In this approach, a problem is formulated as a set of variables, each with a finite domain. The objective of this method is to find a consistent set of values which can be assigned to the variables so that the predefined constraints are satisfied. Constraint Satisfaction Problems (CSP) can be formulated as $CSP = (X, D, C)$ where X is a finite set of variables $X = x_1, x_2, ..., x_n$ (for example timeslots or rooms for each course), D is a finite set of domain value, $D = d_1$ x d_2 x,...,x d_n, which the variables can take (for example, possible starting times or possible rooms); and C is a finite set of constraints, $C = c_1, ..., c_m$, where the constraints are the relations over subsets of variables (for example classroom requirements or precedence relations between times). Therefore, the final solution of a CSP is an assignment of values to each variable such that all the constraints are satisfied [57, 158].

Constraint propagation is the process of eliminating all values from the domain variables that do not satisfy predefined constraints. Let's say that there are two variables x_1 and x_2 from the given finite domain D, where $x_1 = \{1, 2, 3, 4, 5\}$ and $x_2 = \{1, 2, 3, 4, 5, 6\}$, and there is a constraint $x_1 > x_2 + 1$. The constraint propagation technique works as follows: by reducing the domains of x_1 and x_2 to $x_1 = \{3, 4, 5\}$ and $x_2 = \{1, 2, 3\}$. The elimination of the values 1 and 2 from x_1 domain variable happens as they do not satisfy the predefined constraint $x_1 > x_2 + 1$ and the values 4, 5 and 6 from x_2 also conflict with the constraints. However, when another constraint is added, let's say $x_1 + x_2 = 6$ we will find that none of the values can be eliminated from the domain variables. Therefore, in real-world applications, constraints are not always as simple as shown in this example because constraints are often connected with each other. Thus, it will not be practical for this technique to remove all conflict values from the domain variables. In addition, the performance of the constraints propagation technique can be evaluated by assessing the trade-off between the number of eliminated values from the domain variables and the execution time [105].

CSPs are usually solved by using Constraint Logic Programming (CLP) [58]. This approach combines logic programming and a constraint solving method. Thus, CLP is a logical axiom based programming method, which uses a constraint propagation approach in assigning values from the domain to variables. To reduce the search space, a pruning technique is employed to avoid variable instantiation that is not consistent with the predefined constraints upon the given problem. A network arc-consistency algorithm is used to perform constraint propagation in order to reduce the domain of the variables by eliminating those values which are inconsistent with the constraints [112, 118, 160, 161]. Many search techniques such as branch and bound and backtracking are integrated into CLP to guide the search towards an optimal solution. Problem solving using a CLP approach is also know as Constraint-based reasoning (CBR), a reasoning method that utilises an arc-consistency algorithm for constraint propagation purposes [58].

Constraint based-methods have been extensively studied and applied to course timetabling problems. Zervoudakis and Stamatopolous [165] formulated the course timetabling problem within a constraint programming object-oriented framework and employed the ILOG Solver C++ library for the university course timetabling problem faced by the Department of Informatics and Telecommunications at the University of Athens. They implemented several popular and efficient search methods such as Depth First Search (DFS), Iterative Broadening (IB), Limited Discrepancy Search (LDS) and Depth-Bounded Discrepancy Search (DDS). Their problem instances consisted of 68 lectures with five teaching days and nine timeslots per day. Based on their experimental results, they found that DFS was capable of finding feasible solutions in a short time but it was unable to improve the quality of the solutions even when executed for a very long time. LDS was able to improve the solution quality in the long-run but sometimes this search method struggles to find feasible solutions. DDS case, the acceptance criterion is too rigid resulting poor quality of solution. And finally, the authors found that IB did not improve DFS's performance significantly.

Deris et al. [58], at first modelled the timetabling problem as a CSP and after that, they formulated it using the CBR technique. The constraint-based reasoning algorithm was tested using real data from one of the colleges in Malaysia. The instance problem consisted of an 18-weeks timetable for 1673 subject sections, 10 rooms and 21 lecturers. In addition, the authors represented their timetabling problem using a graph tree representation of the state space in order to assist the search for solutions. They also employed variable orderings based on the size of the domain and the number of the constraints of the variable to speed up the search process. Their results showed that their algorithm was able to solve the timetabling problem in less than 33 minutes as compared to several weeks when the problem was solved manually.

Deris et al. [59] proposed a hybrid algorithm to solve timetabling planning, in which they incorporated constraint propagation into a genetic algorithm to construct feasible solutions as well as finding near optimal solutions. The authors used the constraint-based reasoning method to validate individual solutions, which were generated by the genetic operators. To evaluate the performance of the proposed algorithm, real timetabling problems from the Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia were used as a test bed. The experimental results showed that the algorithm was able to generate faster convergence and most importantly the constraint-based reasoning was able to reduce the search spaces. In addition, the proposed hybrid algorithm is capable of finding near-optimal solutions.

- **Population-based Meta-heuristic Approaches**

  Population-based approaches are also called multiple-point [18] approaches and they consist of a collection of individual solutions which are maintained in a population. At each iteration, an appropriate selection mechanism is used to choose how to update solutions in the population and then new solutions are created which may be included in then new current solution [56, 126]. There are many ways in which we could implement the selection mechanism such as the following [147]:

  - Delete all: Delete all individual solutions in the current population and replace them with the same number of solutions that have been generated.
  - Steady-state: In this approach, n old individual solutions are deleted and replaced with n new individual solutions. Another consideration which needs to be taken into account is which individual solutions need to be replaced. One common approach is to delete the worst solutions and replace them with solutions of highest fitness. A second approach is to replace individual solutions in the current population at random.
  - Steady-state-no-duplicates: This selection mechanism is similar to the steady-state approach, however, this method does not allow duplication of individual solutions in the current population. On the other hand, this technique increases the computational cost, since more search space is explored.

Genetic Algorithms

Genetic Algorithms (GAs) were proposed by Holland [89] and these are search methods, which were derived from Darwin's theory of evolution (survival of the fittest). Bremmermann [21] was the first person who proposed and implemented the idea of evolution and recombination of solutions for optimisation problems. The important feature of GAs is the notion of population. This algorithm generates a set of individual solutions and evolves them over a number of generations (iterations), a process called self-adaptation and recombination. The general GAs procedure is illustrated by Algorithm 1.

Algorithm 1: Genetic Algorithm (cited from [103])

```
Step 1. Generate initial population.
Step 2. Evaluate population.
Step 3. Select individuals that will act as parents.
Step 4. Apply Recombination to create offspring.
Step 5. Apply Mutation to offspring.
Step 6. Select parents and offspring to form the new population for the next generation.
Step 7. If stopping condition is met finish, otherwise go to Step 2.
```

Most metaheuristics employ both intensification and diversification strategies. In order to perform intensification in GAs, a selection mechanism is used to select the parents upon which to impose the Darwinian theory of survival of the fittest. The best quality individuals will be more likely to survive for reproduction in the next generation. In contrast, the basic operators that perform the self-adaptation and recombination are called mutation and crossover respectively. These two operators perform the exploration strategy, where they help to investigate new regions of the search space that could not be reached by the intensification strategy alone. There are many variations of Mutation and Crossover [142, 147]. One of the common ways of implementing mutation is to select randomly one or more traits in the chromosome (individual) and then change them at random with usually low probability. Recombination is the process of combining two or more parents to create new solutions, potentially of better quality than the parents (this new solution is called offspring). There are many ways of achieving this, and the simplest and most frequently used are single-point and two-point crossover. In both of these, one or two points respectively are selected randomly to split the chromosome of the parents into sections. The sections on one side of the split point from one parent are then exchanged with the sections from the other parent. Once the offspring have been created using crossover and mutation, it is essential to decide which individual from the previous generation should be kept in the parental population. The process of replacing the individual solutions with the new offspring in the population is called the replacement scheme. This approach could be implemented in many ways, and schemes include both non-elitist strategies and elitist strategies. Non-elitist approaches replace all individual solutions in the current population, while an elitist strategy maintains the best individuals in the current population with the hope that the good-quality genetic material can be inherited by the offspring in the next generation [113].

Erben and Kepler in [74] presented a prototype for the automated construction of timetables, employing genetic algorithm techniques. The main aim of the system was to generate feasible solutions while satisfying as many constraints as possible. An initial random population of feasible solutions was generated at the first step. The proposed algorithm must start with a feasible timetable and always stay in the feasible region of the search space. The genetic operators such as mutation and crossover were developed to use the knowledge specific to the particular problem. In other words, these specialised genetic operators only produce feasible solutions. The authors claim that the results generated from the experiments were quite promising, however a number of improvements still need to be done, for instance to find the optimal parameter settings.

Ueada et al. [159] presented a two-phase genetic algorithm and introduced two types of populations, one for class scheduling and another for room allocation. Both populations were evolved independently and each individual fitness was then calculated. The individuals with the lowest cost were selected for crossover and then the resulting cost was calculated. These steps were repeated until the stopping criterion was satisfied. The algorithm was tested on a real-world problem instance, derived from the Faculty of Information Science at Hiroshima City University. Based on the experiments conducted, the algorithm was able to generate feasible solutions. However, the algorithm failed to find a feasible solution when the average room utilisation ratio was high. The room utilisation ratio is the proportion of time that a room is used within a given period of time. Thus, the room utilisation ratio multiplied by the number of periods in a week gives the average frequency of the use of a room per week.

Konstantinow and Coakley [96] presented an investigation of genetic algorithms to course timetabling problems. In their procedure, once the initial solution is generated, the genetic algorithms are applied to repair the schedule in reactive scheduling. In this specific problem domain, two types of schedule perturbations were employed. One perturbation called 'surges', consisted in changing the student load that can affect the instructors that need to be assigned. The second perturbation, called 'encroachment', referred to when students have to repeat classes or change their class assignment based on previous year's assignment. The results of the experiment clearly showed that the genetic algorithm was a suitable approach under a variety of changes in the student load.

Lewis and Paechter [106] employed four types of crossover operators in their proposed evolutionary algorithms. These operators were: sector-based crossover, day-based crossover, student-based crossover and conflicts-based crossover. The authors also employed a mutation operator which selects two genes $x(a,b)$ and $x(c,d)$ in the chromosome at random such that $x(a,b) \neq x(c,d)$, and swaps them. For all operators, whether crossover or mutation, a similar approach of genetic repair function was then applied, to avoid unfeasible solutions and make sure the crossover operators always produce legal offspring. The genetic repair function is responsible for moving events in order to recover feasibility. A steady state population was employed, where the offspring with the best quality in terms of their fitness, will replace the lower quality parent in the population. The replacement of a parent by a higher quality offspring is done once in each generation so the population evolves in a steady-state fashion. To measure the performance of the algorithm, experiments were conducted using the ITC 2002 dataset [91] as a test bed. The results showed that the conflict-based crossover operator is the most effective crossover method and was capable of producing the best results within the time limit and always outperform the others. Later, Lewis and Paechter in [107] proposed a grouping genetic algorithm (GGAs) and generated their own sixty test instances. GGAs are a class of algorithms that are specialised for grouping problems. According to De Lit et al. [54] the grouping problem involves partitioning a set of U of items into a collection of mutual disjoint subsets of U_i of U such that $\cup_{U_i} = U$ and $U_i \cap U_j = \emptyset$ for all $i \neq j$. Therefore, in the timetabling case, the items represent the list of events and the groups represent the available timeslots. Hence, a valid timetable means each event must be in the right timeslot, complying with predefined hard constraints. The instances were separated into three classes: small, medium and large. The algorithm only found 23 feasible solutions out of 60 instances.

Ant Colony Optimisation

Ant colony optimisation algorithms (ACO) are a class of population-based metaheuristics. They were initially proposed by Colorni, Dorigo and Maniezzo in early 1990's [40, 65, 64]. This method was inspired by the observation of the foraging behaviour of real ant colonies and in particular, how ants find the shortest path from a food source to their nest [66]. Normally, real ants are able to find their nest as long as the food source is not too far away from their nest. Initially, ants explore the area surrounding their nest at random and drop a pheromone trail on the ground while walking from their nest to food sources and vice versa. When ants choose their paths, they will first examine the pheromone on the path and turn towards the direction where there is a greater level of concentration of pheromone. The pheromone on the path dropped by the ant dissipates as time passes, therefore the highest concentration of pheromone normally occurs in the shortest path and, obviously, it is more attractive to ants. Besides that, the quantity of pheromone on the path is also determined by quality and the quantity of the food the ants carry back to their nest.

Socha et al. [152] presented a *MAX-MIN* ant system for timetabling where they transformed the assignment of each of the events to one of the possible timelots into an optimal path problem which the ants are able to tackle. In order to make the transformation possible, they first selected a suitable construction graph that the ants can follow easily. Second, they designed the most appropriate pheromone matrix and heuristic information that was able to guide the ants to choose the right paths in the graph. Therefore, the fundamental elements of the proposed system required the authors to map their problem into a construction graph. The pheromone level within the predefined bound τ_{min} and τ_{max} will influence the assignment of the courses to the timeslots. Then, rooms are assigned to event-timeslot pairs using the matching algorithm to produce a full timetable, which is then further improved by local search. Socha et al. found that the artificial ants were indeed capable of learning to construct good timetables. Later, Socha et al. [153] compared their *MAX-MIN* ant system to an ant colony system but the former algorithm had a better overall performance. The main difference between these two algorithms is on the strategy to update the pheromone.

Mayer et al. [115] applied Ant Colony Optimisation to tackle the post enrolment course timetabling problem which has been specified in the International Timetabling Competition 2007. In that implementation, ants assigned events to rooms and timeslots based on two types of pheromone T^s_{ij} and T^r_{ik}. These pheromone types represent the probabilities of assigning an event i into slot j and room k respectively. Based on the results obtained, their experiments showed that the proposed algorithm seemed to be very well suited to the timetabling problem. The authors argued that with longer running times better results could be obtained.

Memetic Algorithms

Memetic Algorithms (MAs) include a broad class of metaheuristics that were inspired by models of adaptation in natural systems that combine evolutionary adaptation of populations of individuals with individual's life-span time scale. Hence, MAs are also known as Evolutionary Algorithms (EAs), Hybrid Genetic Algorithms, Genetic Local Search, Baldwinian GAs, and Lamarckian GAs [85, 98]. The central theme of MAs is the hybridisation of a local search approach with crossover and mutation operators, enabling the algorithm to find solutions with the best fitness value among neighbouring solutions. Therefore, MAs are specifically designed to exploit all available knowledge about the problem under study (Moscato [120, 121]). In addition, another mechanism called problem and instance-dependent knowledge was introduced to speed-up the search process [122]. With these new features, MAs have proved to be a type of

EAs that are fast and accurate [98, 150, 129], with the algorithm allowing chromosomes (solutions) to improve throughout their life time [150]. The general procedure of memetic algorithm can be shown in Algorithm 2.

Algorithm 2: Memetic Algorithm (cited from [98])

```
Initialize: Generate an initial population;
while ( not some stopping condition do) do
    Local Search (Parent, $P_{ls}$);
    mating Pool := Select Mating(Parents);
    Offsprings := Cross(mating Pool);
    Mutate(Offspring);
    Parents := Select(Parents, Offsprings);
```

Burke and Landa-Silva in [28] gave an account of memetic algorithms for scheduling and timetabling problems and also gave some design guidelines. In a memetic algorithm, the genetic or evolutionary part of the memetic algorithm simulates the genetic evolution of individuals through generations and the local search simulates the individual learning [28]. Paechter et al. [136] used various types of mutations in their memetic algorithm, such as blind mutation, selfish mutation and co-operative mutation. Real-world problem instances from the Computer Studies Department at Napier University were used to evaluate the effectiveness of the various mutations. Later, Paechter et al. [137] came out with another version of their memetic algorithm as an extension to their previous approach in [136]. This time they made some improvements on the user interface as well as the timetable engine. In addition, the authors defined two concepts: a feature (a property satisfied by some resources or events) and a container (a resource that can hold other similar resources, including other containers). Their extended memetic algorithm was tested using Napier University science faculty problem instances. From the experimental results Paechter et al. found that their version of memetic algorithm worked well. In addition, the system was able to produce a feasible solution quickly, and within short enough time, the system was able to generate better solutions than those produced manually. Rossi-Doria et al. [144] implemented a memetic algorithm and tested it on the 20 instances of the ITC 2002. The approach was considered effective on that problem domain. Ozcan et al. [10] implemented different types of mutation strategies and two type of crossovers operators: one-point crossover and uniform crossover. Ozcan et al. tested their memetic algorithm on real timetabling data from Yeditepe University in Turkey. Experiments carried out demonstrated that genetic search combined with hill climbing achieved the best performance.

Hybrid Harmony Search

Al-Betar et al. [9] proposed a hybridisation strategy search called harmony search algorithm (HHSA). This algorithm is composed of the main procedure called Harmony Search Algorithm (HSA), which was first developed by Geem et al. [78]. Al-Betar et al. introduced Hill Climbing Optimisation (HCO) as a new operator for HSA in order to improve the quality of the new harmony in each run. In HSA, few parameters play an important role to drive the success of the algorithm and they are: harmony consideration rate, harmony memory size, pitch adjustment rate and number of improvisations. The exploration and exploitation will be determined by the memory consideration and random consideration respectively. The pitch adjustment is responsible for local improvement, and the size of the local improvement is normally determined by the number of decision variables for the specific problem. Therefore, the size of the local improvement is formulated by the probability of harmony consideration rate and pitch adjustment rate. The idea behind this hybrid approach is to find the trade-off between local improvement and global improvement, which are performed by HCO and HSA respectively. Their experimental results show that the approach is able to produce high quality solutions. With the promising results shown by the hybridisation strategy and with inspiration from particle swarm optimisation concepts, the authors then introduced the modification of the memory consideration operator. This operator mimics the current best harmony. From their experimental results, the operator was able to bring further improvement and was also able to find the best results in four out of five medium instances proposed by Socha et al. [152].

- **Single-solution Meta-heuristic Approaches**

Single-solution meta-heuristics approaches are also called single-point [18] algorithms and differ from population-based approaches in that they improve and maintain a single solution from the beginning until the end of the search process. These approaches employ different strategies to avoid getting stuck in local optima and explore other areas of the solution space. Example of the strategies such as applying different criteria to accept worst solution or hybridised different techniques from several existing heuristics. A brief account of single-solution metaheuristic approaches is given in the following:

Tabu Search

The tabu search methodology was first introduced by Glover [81] back in 1986, although most of the elements in tabu search had already been introduced earlier by Glover [80]. Since then, many papers have been published presenting tabu search algorithms applied to several problem domains, like course timetabling [72], examination timetabling [79, 62], school timetabling [145, 60], job shop scheduling [139, 90], personnel timetabling [155] and many more. The basic idea of tabu search is to prevent cyclic repetition of recent moves by using memory structures called *tabu lists*, which enable the algorithm to memorise some elements of the search history. The reason for using memory is to avoid the algorithm visiting the same neighborhoods for at least a certain duration. The tabu *list size* identifies the number of recently visited solutions or their attributes which are classified as tabu. In addition, the *tabu tenure* defines the duration (usually determined by the number of iterations) that solutions or attributes remain tabu. The concept of the tabu search is quite powerful indeed, and sometimes some attractive moves might be prohibited to take place because of the tabu list. Therefore, in order to mitigate this problem, *aspiration criteria* are introduced to override the tabu status and allow solutions which are better than the currently-known best solution.

In the course timetabling problem domain, many researchers have proposed tabu search. Costa [44] for example, implemented two tabu lists T_1 and T_2 to prevent cycling. The first tabu list consists of lecture l that has been selected to move from timeslot t_1 to t_2. Therefore, whilst the lecture still remains in the tabu list T_1, any action performed to move the lecture l from its current timeslot is prohibited. In the second tabu list, the author introduced the pair of lecture l and the previous timeslot t, (l, t_1). It means that while the pair remains in the tabu list, lecture l cannot be moved to timeslot t_1. As a diversification strategy, the author designed the system in such a way that allowed drastic reduction in weights of the relaxed constraints. That means that the algorithm forces the search activity to discover solutions which are considered of "decent" quality and try to explore a large area as possible of the search space. By reducing the weights, the algorithm will be able to concentrate on less important relaxed constraints, yet, the diversification is not utterly random. Experimental surveys were conducted to access the effectiveness of the algorithm, where two real world problem instances were used: high school of Porrentruy, a town close to the French border in the north of Switzerland and a secondary school of Sierre, a town in the region Valais in the south of Switzerland. Both experimental results indicated that the algorithm was able to generate good quality of timetable solutions. The author noted that it was necessary to fine-tune the parameters (weights and tabu lists) by preliminary experiments.

Nanobe and Ibaraki [127] developed a tabu search-based algorithm for the constraint satisfaction problem and employed an automatic control mechanism for the tabu tenure with the objective of minimising the total weight of the unsatisfied constraints. The system developed by the authors was mainly proposed as a general problem solver and tested on a wide range of problems, among them the university course timetabling problem. Their experiments showed that the system was able to generate competitive results compared to the results obtained by existing algorithms for the respective problem domains. Colorni et al. [41] investigated three well-known meta-heuristic algorithms namely simulated annealing, tabu search and genetic algorithm and tested them on a high school course timetabling problem. In the tabu search, the authors introduced a variable-sized tabu list where there are minimum and maximum lengths for the tabu list and the actual length is changed at random during the search. Their results showed that the tabu search implementation consistently outperformed the genetic algorithm and simulated annealing implementations used.

Schaerf [149] implemented a tabu search methodology tackling large high school timetabling problems. In their tabu search procedure, each executed move was added to the tabu list and the size of the tabu tenure was determined at random within predefined lengths I_{min} and I_{max}. The size of the tabu tenure was updated when a new move was added to the tabu list and the move was released from the tabu list when the size of its tabu tenure became zero. The standard aspiration criterion function was employed to override the tabu status when an improvement of the cost function was achieved. Results showed that their proposed algorithm was capable of scheduling 90% to 95% of the lectures. The experimental results also showed that the quality of the results outperform manually generated timetables.

Alvarez-Valdes et al. [11] employed tabu search for assigning students to courses sections in order to produce high quality timetables as well as to balance the students enrolment across sections. This assignment process uses two phases. The first phase generates a set of best solutions for every single student. The second phase combines the sets of solutions and employs tabu search with strategic oscillation to further improve the timetable quality without worsening the solution of every single student. The proposed algorithm was tested using real-world problem instances from the Faculty of Mathematics at the University of Valencia. From their experimental results, the authors suggested that the quality of the schedule depends on the structure of the master schedule. Alvarez-Valdes et al. [12] implemented tabu search to solve the course timetabling problem in three phases. Phase one constructs an initial solution. Phase two uses the solution obtained in phase one and employs an improvement procedure. The final phase takes care of room assignment and improves it without changing the original assignment of courses to timeslots. The important part of the algorithm is phase two, where tabu search improves the quality of the initial timetable. The authors employed several moves such as: simple move, swap, and multi swap to build an effective algorithm. The tabu list with length 24 and variable size of tabu list (dynamically changed between 6 to 48) were also implemented. The standard aspiration criterion was applied, where a tabu move is allowed whenever it produced a better solution. Experimental tests were conducted using real-world problem instances from the Business School at the University of Valencia. The results showed that the type of move used influences the performance of the algorithm. The tabu list length also played an important role; the dynamic tabu list length obtained better results compared to the static one.

Lu and Hao [110] proposed the integration of tabu search and iterated local search and called their approach Adaptive Tabu Search. The proposed method employed two neighbourhood structures namely *SimpleSwap* and *KampeSwap*, and a standard tabu list to prevent the cycling of previously visited solutions for both moves *SimpleSwap* and *KampeSwap*. They also implemented the standard aspiration criteria where the tabu status will be revoked if there is no danger of cycling because a better solution is found. In order to guide the search efficiently, they used a special operator called penalty-guided perturbation. The main function of that operator was to disturb the local optimum solution. The proposed algorithm was tested on the curriculum-based course timetabling instances of the International Timetabling Competition 2007. That algorithm was ranked second in the competition.

Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) is basically a local search descent (from a minimisation perspective) method and it was first introduced in [117]. The VNS heuristic does not accept worse solutions and it has several variations. The basic idea of this approach is that a number of neighbourhood structures are used in a systematic order during the search. In particular, a different neighbourhood is explored whenever the local search is stuck in some local optimum. Abdullah et al. [5] implemented a basic VNS (VNS-basic) and also a modification in which they used an exponential monte carlo acceptance criterion (VNS-EMC) at the VNS level. Monte carlo acceptance criterion is similar to simulated annealing, the only difference is no temperature involve in this criterion. The main purpose of applying the Monte Carlo acceptance criterion was to improve the exploration by accepting worsening solutions at certain probability hoping to find more promising neighbourhoods. They employed a number of neighbourhood structures ordered in a certain sequence by increasing the size based on their preliminary experiments. Therefore, in order to obtain the right sequence, preliminary experiments needed to be conducted whenever a new neighbourhood structure is added. This approach was tested on the university course timetabling problem instances proposed by Socha et al. in [152]. A performance comparison of VNS-basic and VNS-EMC variants *with ordering* and *without odering* of neighbourhoods was made and the results showed that *withordering* both variants, VNS-Basic and VNS-EMC, performed better than or equal than *without odering*.

Randomised Iterative Improvement Algorithm with Composite Neighborhood Structure

Later, Abudllah et al. [4] extended their investigation and introduced the hybridisation of VNS and a tabu list. The tabu list was used to penalise the neighbourhood structures that do not perform well or do not lead to promising solutions after a certain number of iterations. The results they obtained were better than or equal to their previous technique in [5] on seven instances. Their algorithm is shown in Algorithm 3 where each neighborhood $i \in 1 \ldots k$ is applied to solution Sol to obtain a new set of temporary solutions $TempSol_j$. Then, the best solution among all $TempSol_j$ is selected to become the new solution $Sol*$. If $Sol*$ is better than the best solution so far Sol_{best} then $Sol*$ also replaces Sol_{best}. Otherwise, the monte carlo acceptance criterion is applied after calculating $\delta = f(Sol*) - f(Sol)$.

Algorithm 3: Randomised Iterative Improvement Algorithm with Composite Neighborhood Structure (cited from [5])

```
Set the initial solution Sol by employing a constructive heuristic;
Calculate initial cost function $f(Sol)$;
Set best solution $Sol_{best} \leftarrow Sol$;
while (no termination criteria) do
    for (i=1 to i=k where k is the total number of neighborhood structures) do
        Apply neighborhood structure on Sol;
        TempSol;
        Calculate cost function $f(TempSol_i)$;
        Find the best solution among the $(TempSol_i)$ where $i$ in $1, ....k$ call new solution $Sol^*$;
        if $f(Sol*) < f(Sol_{best})$ then
            $Sol \leftarrow Sol*$;
            $Sol_{best} \leftarrow Sol*$;
        else
        -
        else
            $\delta = f(Sol*) - f(Sol)$;
            Generate RandNum, a random number in [0.1]; if (RandNum $< e^{-\delta}$) then
                $Sol \leftarrow Sol*$;
        end else;
    end while;
```

Simulated Annealing

Simulated annealing has been broadly studied and it is an extension of Hill-Climbing, in which non-improving candidate solutions are accepted with a certain probability to attempt escaping from local optimum. The probability of accepting worse solutions in simulated annealing is usually expressed as $P = e^{-\delta/T}$, where $\delta = f(S^*) - f(S)$, S^* is the new solution, S is the current solution, f is the cost function and the parameter T denotes the temperature. It is also usually suggested that the search should start with high temperature and to reduce it gradually towards the end of the search process. One way of reducing the temperature is $T_{i+1} = T_i * \beta$ (geometric cooling schedule). The selection of β for a particular problem is typically done empirically. The pseudocode of simulated annealing is given in Algorithm 4.

Algorithm 4: Simulated Annealing Algorithm (cited from [103])

```
Step 1. Generate initial current solution $x$.
Step 2. Temperature = Initial Temperature.
Step 3. Generate candidate solution $x'$ from current solution $x$.
Step 4. If fitness($x'$) ¿ fitness($x$) then $x = x'$.
Step 5. If fitness($x'$) . fitness($x$) then calculate Acceptance Probability.
  Step 5.1 If Acceptance Probability > random[0,1] then $x = x'$.
Step 6.Update Temperature according to Cooling Schedule.
Step 7. If stopping condition met finish, otherwise go to Step 3.
```

Tuning simulated annealing is difficult as noted in an early application of this algorithm to the timetabling problem [53]. Different improvements of the basic simulated annealing algorithm have been suggested, such as adaptive cooling where the temperature is reduced or increased depending on the success of the local search move. Another variation of simulated annealing called multiple-neighbourhoods-based simulated annealing algorithm was implemented by Yan et al. [164]. The authors tested their algorithm on course timetabling problems and found that their approach was effective in finding the optimal solution from an enormous search space thanks to the adaptive cooling method.

Elmohamde et al [73] proposed simulated annealing and employed several cooling schedules: geometric, adaptive, and adaptive with reheating. The proposed algorithm with different cooling schedules were tested on real data from Syracuse University. Based on their experimental results they found that simulated annealing with adaptive cooling with reheating outperforms other approaches.

Cambazard et al. [36] applied simulated annealing to minimise the violation of soft constraints in timetabling. The authors employed one single type of move, which changes the position of an event into a conflict free timeslot and reassigns the events within a given timeslot in order to minimise the room conflicts. Therefore, only feasible moves are allowed. Improving and sideways (same quality different solutions) moves are always accepted and the worse moves are accepted depending on the simulated annealing acceptance probability given by $P_{acceptance}(\triangle, \tau) = e^{-\frac{\triangle}{\tau}}$. The initial temperature τ was chosen dynamically and decreased as the search progresses. A standard geometric cooling is employed at each step $\tau_{n+1} = 0.95$ x τ_n. The algorithm was tested on the post enrolment-based course timetabling instances of the international timetabling competition 2007, and this algorithm was officially announced as a winner.

Great Deluge

The great deluge algorithm was first introduced by Dueck in [68] and the basic idea of this algorithm is similar to simulated annealing. However, great deluge is said to be less dependent upon parameter tuning compared to simulated annealing. In fact, great deluge needs only two parameters. These parameters are: 1) the amount of computational time that the user wishes to spend on the search and 2) the expected quality of the final solution. During the search, a new candidate solution is accepted if it is better or equal than the current solution. A candidate solution worse than the current solution will only be accepted if the detriment in quality of the candidate solution is less than or equal to a pre-defined upper limit (called water level). The pseudocode of the great deluge is given in Algorithm 5.

Algorithm 5: Great Deluge Algorithm (cited from [22])

```
Set the initial solution $s$;
Calculate initial cost function $f(s)$;
Initial level $B_0 = f(s)$;
Specify input parameter $B = ?$;
while ( not some stopping condition do) do
    Define neighborhood $N(s)$;
    Randomly select the candidate solution $S* \in N(s)$;
    if $(f(s*) \leq f(s))$ or $(f(s*)(\leq B))$ then
        accept $s*$;
    Lower the level $B = B - \Delta B$;
```

The great deluge algorithm was applied to course timetabling problems in [22]. The authors used the ITC 2002 datasets. In that paper, the authors claimed that their algorithm showed similar behaviour on all problem instances. Based on their experiments, they observed that the fluctuations of the penalty values were very obvious at the beginning of the search, but, later, intermediate solutions moved closer to the current cost line. Overall, their experimental results showed superiority of the Great Deluge algorithm when compared to their implementation of simulated annealing.

Petrovic and Burke reviewed approaches to university course timetabling in [138]. They argued that a major drawback of many metaheuristic approaches for this problem and other scheduling problems was that these approaches are in general very dependent upon a range of parameters. The effectiveness of a given metaheuristic for a given problem is very much dependent on the success of parameter settings. For example, in the cooling schedule in simulated annealing, choosing the wrong starting and ending temperature will not result in good final solutions. On the other hand, choosing the right parameter settings is usually difficult especially for non-expert users of metaheuristics. Therefore, this difficulty led Petrovic and Burke to suggest the investigation of metaheuristic approaches that are not as dependent upon parameter settings. They developed a timetabling method based on the great deluge algorithm. In their approach, the acceptance of a local search move in great deluge is guided by two criteria. A better solution is always accepted while a worse solution is accepted only if the evaluation function value is less than or equal to the upper limit (level) as it is usual in great deluge. The upper limit is lowered during the run by the decay, and it is always fixed in every iteration. The decay is a function of two parameters, the amount of computational time available and the desired quality of the final solution. The decay can be computed as the difference in quality between the initial solution and the estimated desired final solution and this difference is divided by the desired number of local search moves.

Hybrid Meta-heuristics

Kostuch [97] combined a sequential heuristic and simulated annealing to solve course timetabling problems and tested that approach on the ITC 2002 datasets.

In fact, Kostuch was announced as the overall competition winer. That hybrid approach consisted of three phases. In the first phase, a sequential heuristic is used to generate the initial feasible timetable. Then, the second phase employs simulated annealing to minimise the violation of the other soft constraints (no students should have only one class on a day and no students should attend more than two consecutive classes on a day). In this second stage, the algorithm does not allow the move of events that lead to infeasible timetables neither does the algorithm accept moves that could violate the soft constraint related to placing events in the last slot of the day. In the third and final phase, simulated annealing is used to further improve the quality of the timetable. According to Kostuch, this third phase played a major part in the hybrid algorithm. In this stage, only local search moves that do not violate hard constraints and moves of events that can be assigned to a different room within the same timeslot were accepted.

Another hybrid approach is the one proposed by Chiarandini et al. in [39]. At the top level, that algorithm first creates several assignments and only the best assignments are selected to be improved by simulated annealing. This hybrid approach consisted of two steps. First they applied *buildAssignment* to construct a group of initial solutions and then each of the initial solutions is made feasible by *HardConstraintsSolver*. For further processing, *FastLocalSearch* is selected to improve the quality of the timetable. In the assignment representation they used the work in [151]. Chiarandini et al. used different neighborhood schemes for the two sub-procedures to tackle hard and soft constraints. Two neighbourhoods are used to generate an initial feasible solution. N1 moves a single event to a timeslot selected at random. N2 swaps the timeslot and room between two events. For the soft constraints solver, two more neighbourhoods were used. N3 swaps events assigned to two different timeslots and N4 is defined by Kempe chain interchanges [156]. The search strategy uses a list of events randomly ordered and goes through the list trying the moves available in the given neighbourhood until an improvement is found. The authors applied constructive heuristic to create the initial solutions. This approach constructs the initial solutions by assigning the list of events into limited timeslot and rooms one at a time.

Abdullah et al. [6] implemented great deluge and tabu search to tackle the university course timetabling problem. The proposed algorithm consisted of two parts: construction and improvement algorithms where four neighbourhood structures were employed. Move N1: choose a single course at random and move it to a feasible timeslot that can generate the lowest penalty cost. Move N2: selects two courses at random from the same room (the room is randomly selected) and swaps their timeslots. Move N3: move the highest penalty course from a random 10% selection of the courses to a new feasible timeslot, which can generate the lowest penalty cost. Move N4: move the highest penalty course to a random feasible timeslot (both courses are in the same room). In part one, in order to construct feasible timetables they employed a saturation degree strategy which starts with an empty timetable. In this approach the events with fewer possible rooms available will be scheduled first and the process of allocation stops when a feasible solution is found. In the case that feasibility is not achieved, the execution of phase 2 will be carried out. In this phase only neighbourhood moves N1 and N2 are applied. Procedure phase two runs as follows: apply N1 for a certain number of iterations, stop if feasible solution is found, otherwise apply N2 for a certain number of iterations. The construction of feasible solutions based on this procedure gave no proof that this constructive heuristic guarantees to find a feasible solution for a given instance. Therefore, solutions were made feasible before the minimisation of soft constraint violations takes place. The improvement part will be executed when a feasible solution is available, and the search never goes back to the infeasible region. Three steps are involved in the improvement part. Step one employs great deluge followed by tabu search. Step two and step three compare the solution obtained in step one and step two. The best solution between those two algorithms is chosen. If the quality of the best solution between step one and two is less than the quality of the current solution, then the current solution will be updated. The proposed algorithm was tested on 11 standard benchmark instances of the university course timetabling problem. Each instance was solved five times and results were compared to those reported in the literature. Based on their experimental results, their approach produced better results at the time on all datasets except the large one. The pseudo-code for the algorithm implemented by Abdullah et al. [6] is shown in Figure 6.

Algorithm 6: Great Deluge and Tabu Search Algorithm (cited from [6])

```
Set the initial solution Sol by employing a constructive heuristic;
Calculate initial cost function $f(Sol)$;
Set best solution $Sol_{best} \leftarrow Sol$;
while ( not some stopping condition do) do
    Step 1: Great Deluge
    Step 2: Tabu Search
    Step 3: Accept Solution
    Choose the best between $SolbestGD^*$ and $SolbestTS^{a*}$;
    called $Sol^*$
    if $(f(Sol^*) < f(Solbest))$ then
        Sol $\leftarrow$ Sol*;
        $Sol_{best}$ $\leftarrow$ Sol*;
```

2.4.1.2 Review of Hyper-heuristic

The course timetabling problem has been solved using a wide range of heuristics and metaheuristics. However, the main drawback of metaheuristics is that they are domain specific. In recent years, hyper-heuristics have emerged as a new search methodology that is motivated by the goal of increasing the level of generality of metaheuristics. The aim of hyper-heuristics is to develop general domain independent search methodologies that are capable of performing well enough, soon enough, and cheap enough across a wide range of optimisation problems [25]. Besides that, another goal of hyper-heuristic is to produce a generic method, able to generate acceptable quality of solutions based on a set of easy-to-implement low-level heuristics [26].

The term hyper-heuristic has been defined to describe the process of using (meta) heuristics to choose (meta) heuristics in [27]. It is a process which, when given a particular problem instance and a number of low-level heuristics, manages the selection and acceptance of the low-level heuristic to apply at any given time, until a stopping condition is met. A low-level heuristic is a simple local search operator or domain dependent heuristic. A hyper-heuristic operates at a higher level of abstraction without knowledge of the domain under which it operates. The hyper-heuristic searches in the space of low-level heuristics instead of operating on the solution space directly. One of the main challenges in designing hyper-heuristics is to be as general as possible on how to manage the low-level heuristic with minimum parameter tuning.

Early research work on hyper-heuristics emphasised on the development of advanced selection strategies. Soubeiga [154] proposed random, greedy, and choice function hyper-heuristics with two acceptance criteria namely AM (All Moves) and IO (Improving Only). The random hyper-heuristic selects randomly the next low level heuristic to apply at each decision point of the search. The greedy hyper-heuristic selects always the best low level heuristic. The choice function hyper-heuristic uses reinforcement learning to guide the choice of low level heuristics. Another learning mechanism based on tabu search was proposed by Burke et al. [27] to solve the nurse rostering problem. In the tabu search hyper-heuristic, a tabu list was incorporated to prevent the acceptance of low level heuristics with poor performance for a certain number of iterations. Ross et al. [143] used a learning classifier system to learn which heuristics were more useful than others in a bin packing problem. Several Genetic Algorithm (GA) based hyper-heuristics have also been developed. Cowling et al. [45] proposed a GA based hyper-heuristic to solve a trainer scheduling problem. Other high level strategies have also been investigated within the framework of hyper-heuristics. Burke et al. [31] developed a case-based hyper-heuristic for timetabling problems which selects low-level heuristics based on their performance in previous similar situations. Burke et al. [32] proposed an ant-based hyper-heuristic for solving a presentation scheduling problem.

Recently, hyper-heuristics have also been successfully used to solve the university course timetabling problem. Burke et al. [27] proposed a choice function hyper-heuristic which uses a tabu list to guide the iterative application of a set of simple local search heuristics. They used the same six local search heuristics proposed by Socha et al. [152]. The choice function assigns a fitness value to each heuristic according to their success during the search. Also, Burke et al. [29] applied a graph-based hyper-heuristic in which a tabu search procedure is used to change the permutations of six graph colouring heuristics before applying them to construct a timetable. The key feature of this approach is to find good orderings of constructive heuristics to schedule the events. Bai et al. [17] developed a simulated annealing hyper-heuristic for solving the university course timetabling problem; their method selects low-level heuristics based on a stochastic ranking mechanism.

2.4.1.3 Review of Distributed Hyper-heuristics

Recently, researchers have proposed the use of multi-agent systems for tackling timetabling problems. A multi-agent system (MAS) is a network of agents that work together to solve problems that are beyond the agents' individual capabilities [124]. Multi-agent systems are distributed and autonomous systems made up of autonomous agents that support reactivity, and are robust against failures locally and globally [131]. Multi-agent systems have been applied for a long time to other problem domains such as e-commerce, production scheduling, etc., and they have produced impressive results. In contrast, little research work has been done in applying these systems to educational timetabling. Kaplansky et al. claimed that the distributed nature of the timetabling problem can be tackled by using the multi-agent paradigm [93]. Each agent in their model has a different set of requirements that lead them to the different quality of solutions. In order to coordinate their timetables, all agents in the distributed environment have to communicate and negotiate to avoid conflicts in the process of allocating the shared resources. In the real world, education institutions are composed of departments that need to construct their timetables independently, while trying to minimise the shared courses conflicts. For example, students from the department of business and students from the department of economics might be interested to attend courses offered by the department of computer science and vice versa. In order to avoid shared courses conflicts (i.e. global conflicts) the timetables from all departments must be constructed to yield a coherent compatible solution. Besides that, hard and soft constraints need to be considered. According to Di Gaspero et al. [61], a department is usually not willing to share their timetabling information with other departments. Therefore, they assumed that all constraints are unknown to each department. By formulating their model, they proposed an architecture for a multi-agent system to tackle university course timetabling problem. In their approach, no global objectives need to be satisfied. Therefore, their designed agents which represent every department, negotiate with selfish behaviour to fulfil their own interest, but will tolerate other agents provided that it will bring benefit to them, in other words not worsening their own objective. Oprea [130] adopted a multi-agent approach for solving university course timetabling problems. In this approach, the autonomous agents work together in a distributed environment to coordinate their work in order to achieve the global objective. At the same time, every single agent needs to fulfil their own objectives.

Rattadilok et al. [141] implemented a distributed choice function hyper-heuristic and employed seven low-level heuristics. The low-level heuristics were divided into two groups: intensification and diversification groups. Low-level heuristics one to six were mainly for intensification, accepting new generated solutions only if their quality is better than the current solution. Low-level heuristic seven was employed mainly for diversification, accepting new generated solutions anyway if no better solution can be found (if the search stuck in local optimum). To rank the low-level heuristics, the authors developed the choice function-based hyper-heuristic proposed by Soubeiga [154]. The choice function evaluates the performance of the low-level heuristics at every decision point. The low-level heuristic with the highest score will be selected in the next iteration to change the search space landscape. This system employed two parallel architectures: hierarchical and hybrid-agent. The main idea of the hierarchical architecture is branching the search sequence and the hybrid-agent is as a communication mechanism, where it enables the agents to share their good solution among them in the distributed environment. The agents in the distributed environment are composed by a number of hierarchical groups. The distributed choice function is placed in one of the processors called controller in every hierarchal group. Only the controllers have the right to communicate to each other in the environment. Each low-level heuristic has given time and failure limits as stopping condition mechanism. The authors used the 11 problems instances by Socha et al. [152] and compared their results to MMAS and RRLS. The results showed that their distributed choice function outperformed on all the small instances when compared to MMAS and RRLS. For medium instances, the distributed choice function only managed to improve two out of five problem instances when compared to MMAS. The authors did not report their result for the large instance.

2.5 Important Papers

In our review of the literature, we found a survey paper and some early approaches to solving the course timetabling problem to be important and to underpin the research presented in this thesis. They are discussed below:

In [148] Schaerf presented an extensive survey on automated timetabling, and it is a very suitable introduction for new researchers in the area of timetabling problems. This paper provides excellent discussion by classifying the timetabling problems into school, course or examination timetabling problems. This paper also gives a mathematical description of the basic search and optimisation problems. In addition, this paper made the important statement that university timetabling problems are NP-complete in almost all variants of the problems. Finally, it gives a summary of the solution techniques published in the literature.

One of the interesting aspects of the approach presented in [97] is that in the stage of finding the feasible solution the author used only 40 out of the 45 available timeslots. This means that, at this stage, the author tries to satisfy the soft constraint by not allowing the assignment of events to the final slots of the day. Obviously, one immediate advantage of this approach is that it always takes the soft constraint into account to some extent, and therefore, when this algorithm finishes phase one and the feasibility is found the quality of the solution can be considered good, since the violation of the soft constraint by not encouraging students to attend the final slot of the day is solved or nearly solved. Correspondingly, the method for reaching the feasibility is based upon methods for solving graph colouring problems. However, in different particular problems this strategy might be unwise, because there would, of course, be no guarantee that feasibility could be achieved when tested on a different set of problems instances. In addition, this paper also gives a detailed analysis of the given problem, including statistical analysis that presents the degree of difficulty each problem instance.

In [22] Burke et al. presented the great deluge algorithm, and shows the effectiveness of the algorithm. This approach is far less dependent upon tuning a range of parameters than many metaheuristic approaches, such as simulated annealing. Indeed, it only needs two parameters: The amount of computational time that the user wishes to spend and an estimate of the quality of the solution that a user requires. In addition, an implementation is provided in this paper to investigate the properties of both the great deluge and simulated annealing techniques. The great deluge is a simple yet effective method which is able to produce good quality timetables and provides results that are consistently good across the all of the benchmark problems. The good thing about this paper is, it shows that a simple algorithm is able to generate high quality of timetable solutions.

Different metaheuristics have different trade-offs between them. It is therefore not surprising that efforts have been made to develop cooperative search methods which draw on the advantages of different techniques and make them cooperate with each other. Papers in [19, 134, 51, 49, 48, 104] present an extensive discussion and present the main fundamental objectives of cooperative search, for example to provide capabilities of integration, robustness, flexibility, and autonomy. The papers also reveal different ways to exploit the strengths and weaknesses of different metaheuristics by combining them in a cooperative search framework, where the aim is obviously to solve a given problem by using the same or different strategies. Therefore, the uses of cooperative search or parallel search in scheduling provide new efficient search techniques for integration and robustness in complex combinatorial problems. The work presented in these papers has demonstrated that the distribution, local autonomy, and cooperation capabilities of parallel search have lead to a remarkable improvement in the design and performance of complex combinatorial problems, especially in scheduling. The use of cooperative search in timetabling, as discussed in [19, 134, 51, 49, 48, 104] provides answers to how to efficiently integrate the meta-heuristics in a sense of cooperative search and interactive systems in such a complex problem with the use of multi-agent systems.

2.6 Conclusions

This chapter has described a range of educational timetabling problems. One of them was the University Course Timetabling Problem (UCTTP) which is the main focus on this thesis. The chapter provided a glance of the constraints usually associated with this problem, but a more detailed discussion is given in chapter 3. This chapter also reviewed the range of techniques that have been applied to tackle the UCTTP in the literature. Very little research works have reported using cooperative search for this problem. Most of the work published so far concentrates on sequential heuristics. Then, in the rest of this thesis we contribute by investigating a range of metaheuristics, hyper-heuristics and cooperative search approaches for the UCTTP.

Chapter 3

University Course Timetabling Benchmark Problems

3.1 Introduction

This chapter focuses on the specific benchmark instances of the university course timetabling problem that were used in this thesis. These benchmark problems proposed by Meta-heuristics Network [1] have been artificially generated and was intended to represent a simplification of "real-world" problems. The motivation behind using these benchmark datasets was to overcome some of the common ambiguities and inconsistencies that existed in previous course timetabling instances. This chapter is organised as follows: Section 3.2 presents the description of the university course timetabling problem (UCTTP) and its formulation. Section 3.3 introduces the hard and soft constraints in these problem instances. Section 3.4 gives the problem formulation and Section 3.5 describes the data structure used to represent this problem. This is followed by some brief concluding comments in Section 3.6.

3.2 The University Course Timetabling Problem

Several formulations of the university course timetabling problem have been proposed in the literature. We adopt the one by Socha et al. [152] and the corresponding benchmark data sets in order to test the algorithms proposed in this thesis.

More formally defined, the university course timetabling problem tackled in this paper consists of the following:

- n events $E = \{e_1, e_2, \ldots, e_n\}$
- k timeslots $T = \{t_1, t_2, \ldots, t_k\}$
- m rooms $R = \{r_1, r_2, \ldots, r_m\}$ in which events can take place
- a set F of room features satisfied by rooms and required by events
- a set S of students

Each room has a limited capacity and each student attends a number of events which is a subset of E. The problem is to assign the n events to the k timeslots and m rooms in such a way that all hard constraints are satisfied and the violation of soft constraints is minimised. These benchmark data sets are split according to their size into five small, five medium and one large, i.e. 11 instances in total. The detail of the parameter values of the 11 instances and its categories are given in Table 3.1. The instances in each category are different with respect to the number of conflicts or density matrix, even though they have the same number of events, rooms, features and students. In the rest of this thesis, we refer to these 11 problems instances as the *Socha et al. instances.*

Table 3.1: The parameter values for the course timetabling problem categories in the Socha et al. instances.

Category	Small	Medium	Large
Number of courses	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Number of students	80	200	400
Maximum courses per student	20	20	20
Maximum students per course	20	50	100
Approximation features per room	3	3	5
Percent feature use	70	80	90

Table 3.2: The 20 instances in the set of International Timetabling Competition 2002. The last three columns give some indication about the structure of the instances. Details of the competition algorithms are available at: http://www.idsia.ch/Files/ttcomp2002/results.htm.

Instance	No. events n	No. students $\|S\|$	No. rooms m	Rooms/event	Events/student	Students/event
com01	400	200	10	1.96	17.75	8.88
com02	400	200	10	1.92	17.23	8.62
com03	400	200	10	3.42	17.70	8.85
com04	400	300	10	2.45	17.43	13.07
com05	350	300	10	1.78	17.78	15.24
com06	350	300	10	3.59	17.77	15.23
com07	350	350	10	2.87	17.48	17.48
com08	400	250	10	2.93	17.58	10.99
com09	440	220	11	2.58	17.36	8.68
com10	400	200	10	3.49	17.78	8.89
com11	400	220	10	2.06	17.41	9.58
com12	400	200	10	1.96	17.57	8.79
com13	400	250	10	2.43	17.69	11.05
com14	350	350	10	3.08	17.42	17.42
com15	350	300	10	2.19	17.58	15.07
com16	440	220	11	3.17	17.75	8.88
com17	350	300	10	1.11	17.67	15.15
com18	400	200	10	1.75	17.56	8.78
com19	400	300	10	3.94	17.71	13.28
com20	350	300	10	3.43	17.49	14.99

The second set of instances used in this thesis were proposed for the International Timetabling Competition 2002. These data sets also consist of a fixed number (45) of timeslots and the number of events ranges between 350 to 440 while the number of students lies between 200 to 300. Table 3.2 gives a summary of the characteristics of the International Timetabling Competition 2002 problem instances. In addition, the number of possible rooms for each event (room/event) is relatively very low, between 1.11 and 3.94. This relation room/event indicates that a deterministic algorithm can be employed to identify a suitable room for the events. The last two columns (event/student and student/event) are an indication of the difficulty to minimise the violation of the predefined constraints.

For both set of the benchmark instances (11 instances by Socha et al. [152] and International Timetabling Competition 2002 instances), it is known by construction that for each instance optimal value of zero cost assignment exists.

3.3 Constraints in University Course Timetabling

In this problem, as discussed before, *hard constraints* must be satisfied in any circumstances. It means that if hard constraints are violated then the timetable is infeasible, so useless and will be discarded. On the other hand, *soft constraints* may be violated, however we try to minimise such violation in order to improve the solution quality.

3.3.1 Hard Constraints

There are four hard constraints in this problem:

- h1: A student cannot attend two events simultaneously, i.e. events with students in common must be timetabled in different timeslots.
- h2: Only one event is allowed to be assigned per timeslot in each room.
- h3: The room capacity must be equal to or greater than the number of students attending the event in each timeslot.
- h4: The room assigned to an event must satisfy the features required by the event.

3.3.2 Soft Constraints

There are three soft constraints in this problem:

- s1: Students should not have only one event timetabled on a day.
- s2: Students should not attend more that two consecutive events on a day.
- s3: Students should not attend an event in the last timeslot of a day.

3.4 Problem Formulation

The objective in this problem is to find a feasible solution that minimises the violation of soft constraints. Both of the problems data sets described above (11 Socha et al. instances and 20 ITTC 2002 instances) can be formalised as follows. Let X be the set of all possible solutions, where each event has been assigned a pair timeslot-room. Let $\mathcal{A} = \{h1, h2, h3, h4\}$ be the set of all hard constraints. Let $\mathcal{B} = \{s1, s2, s3\}$ be the set of all soft constraints for which violation should be minimised. Let $\tilde{X} \subseteq X$ be the set of all feasible solutions that satisfy the hard constraints in $\mathcal{A}$. The cost function $f(x)$ for both problem data sets can be represented by this formulation. Each solution $x \in \tilde{X}$ is associated with a cost function measuring the total violation of soft constraints in $\mathcal{B}$. The main objective of this problem is to search for an optimal solution $x^* \in \tilde{X}$, in this case an optimal solution is, if $f(x^*) \leq f(x), \forall x \in X$. The cost function $f(x)$ measures the quality of the feasible solution $x \in \tilde{X}$ by measuring the violation of the total soft constraints given by:

$$f(x) = \sum_{s \in S}(f_1(x,s) + f_2(x,s) + f_3(x,s))$$

- $f_1(x,s)$: number of times a student s in timetable x is assigned to the last timeslot of the day.

- $f_2(x,s)$: number of times a student s in timetable x is assigned more than two consecutive classes. Every extra consecutive class will add 1 penalty point, for example $f_2(x,s) = 1$ if a student s has three consecutive classes and $f_2(x,s) = 2$ if the student s has four consecutive classes, and so on.

- $f_3(x,s)$: number of times a student s in timetable x is assigned a single class on a day. $f_3(x,s) = 1$ if student s has only 1 class in a day and if student s has two days with only one class $f_3(x,s) = 2$.

3.5 Data Input

The data for each problem instance includes the size and features for each room, the number of students attending each event and information about conflicting events (those with students in common). The information from each problem instance is stored into five matrices to be used by the heuristic algorithms described in this thesis. These matrices are named: *StudentEvent*, *EventFeatures*, *RoomFeatures*, *SuitableRoom*, *EventConflict* and finally *StudentAvailablity*.

The *StudentEvent* matrix of size $|S| \times n$ has a value of 1 in cell (i,j) if student $i \in S$ should attend event $j \in E$, 0 otherwise. The *EventFeatures* matrix of size $n \times |F|$ has a value of 1 in cell (i,j) if event $i \in E$ requires room feature $j \in F$, 0 otherwise. The *RoomFeatures* matrix of size $m \times |F|$ has a value of 1 in cell (i,j) if room $i \in R$ has feature $j \in F$, 0 otherwise. The *SuitableRoom* matrix of size $n \times m$ is used to quickly identify all rooms that are suitable (in terms of size and features) for each event, a value of 1 in cell (i,j) indicates that room $j \in R$ has the capacity and features required for event $i \in E$. The *EventConflict* matrix of size $n \times n$ has a value of 1 in cell (i,j) if events $i,j \in E$ have students in common, 0 otherwise. The *EventConflict* matrix helps to quickly identify events that can potentially be assigned to the same timeslot.

Table 3.3: *StudentEvent* matrix.

$e_j, j \in \{e_1, ..., e_{12}\}$/ Student	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	0	0	0	1	1	0	1	1	0	1
2	1	1	0	0	1	0	1	1	0	0	1	0
3	0	0	1	0	0	1	1	0	0	1	1	0
4	1	0	1	1	0	1	1	1	1	1	1	1
5	0	1	0	0	1	0	0	1	0	1	1	1
6	1	0	1	1	0	1	1	1	1	1	0	0
7	1	1	1	1	0	1	1	0	0	1	1	1
8	0	1	0	1	1	1	0	1	1	0	1	0
9	1	0	0	1	0	1	0	1	1	0	1	1
10	1	0	1	1	1	1	1	0	0	1	1	0
11	0	1	1	1	1	0	1	1	1	1	1	0
12	1	0	0	1	1	0	1	0	1	0	0	1

Table 3.3 shows an example of the *StudentEvent* matrix where:

- First column: indicates the number of students $|S| = 12$.
- Remaining columns: indicate which students take the event corresponding to that column, here the number of events is E=12.

In the above *StudentEvent* matrix example, we see that the first student attends events e_2, e_6, e_7, e_9, e_{10} and e_{12}. The second student attends events e_1, e_2, e_5, e_7, e_8 and e_{11} and so on until the last column which indicates that student 12 attends events e_1, e_4, e_5, e_7, e_9 and e_{12}.

Table 3.4: *EventsConflict* matrix.

i/j	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	0	1	0	1	1	0	1	1	0	1
2	1	0	1	0	1	0	1	1	0	0	1	0
3	0	1	0	1	0	1	1	0	0	1	1	0
4	1	0	1	0	0	1	1	1	1	1	1	1
5	0	1	0	0	0	0	0	1	0	1	1	1
6	1	0	1	1	0	0	1	1	1	1	0	0
7	1	1	1	1	0	1	0	0	0	1	1	1
8	0	1	0	1	1	1	0	0	1	0	1	0
9	1	0	0	1	0	1	0	1	0	0	1	1
10	1	0	1	1	1	1	1	0	0	0	1	0
11	0	1	1	1	1	0	1	1	1	1	0	0
12	1	0	0	1	1	0	1	0	1	0	0	0

An *EventConflict* matrix is generated using the input of *StudentEvent* matrix that provides information of students attending specific events. Table 3.4 shows an example of the corresponding *StudentEvent* matrix where:

- First column: indicates the number of events $|E|$=12.
- Remaining columns: indicate the existing conflicts among events with a value of 1 in the corresponding cell.

In the above *EventConflict* example, the first event (e_1) should not be assigned in the same timeslot as any of events e_2, e_4, e_6, e_7, e_9, e_{10} and e_{12}. The second event (e_2) should not be assigned in the same timeslot as any of events e_1, e_3, e_5, e_7, e_8 and e_{11} and so on until the last column which indicates that event 12 e_{12} should no be assigned in the same timeslot as any of events e_1, e_4, e_5, e_7 and e_9.

An example of the *RoomFeatures* matrix is shown in Table 3.5 where:

Table 3.5: *RoomFeatures* matrix.

$f_j, j \in \{f_1, ..., f_8\}$/ Room	1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	1	0
2	1	1	0	0	1	0	1	1
3	0	0	1	0	0	1	1	0
4	1	0	1	1	0	1	1	1
5	0	1	0	0	1	0	0	1
6	1	0	1	1	0	1	1	1
7	1	1	1	1	0	1	1	0
8	0	1	0	1	1	1	0	1
9	1	0	0	1	0	1	0	1
10	1	0	0	1	1	0	1	0

- First column: indicates the number of rooms $|R|$=10.
- Columns two to eight: indicate whether room r_j satisfies the features F_j, the number of features is $F = 8$.

In the above *RoomFeatures* example, the first room r_1 satisfies features f_1, f_6 and f_7. The second room r_2 satisfies features f_1, f_2, f_5, f_7 and f_8 and so on until room r_{10} satisfies features f_1, f_4, f_5 and f_7.

Table 3.6 shows an example of the *EventFeatures* matrix where:

- First column: indicates the number of events $|E|$=12.
- Columns two to eight: indicate whether event e_j requires feature F_j, the number of features is $F = 8$.

Table 3.6: *EventsFeatures* matrix.

$f_j, j \in \{f_1, ..., f_8\}$/ Event	1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	1	0
2	1	1	0	0	1	0	1	1
3	0	0	1	0	0	1	1	0
4	1	0	1	1	0	1	1	1
5	0	1	0	0	1	0	0	1
6	1	0	1	1	0	1	1	1
7	1	1	1	1	0	1	1	0
8	0	1	0	1	1	1	0	1
9	1	0	0	1	0	1	0	1
10	1	0	1	1	1	1	1	0
11	0	1	1	1	1	0	1	1
12	1	0	0	1	1	0	1	0

In the above *EventFeatures* matrix example, event e_1 requires features f_2, f_6 and f_7, event e_2 requires features f_1, f_2, f_5, f_7 and f_8 and so on until event e_{12} requires features f_1, f_4, f_5 and f_7.

Table 3.7 shows an example of *SuitableRooms* matrix where:

- First column: indicates the number of events $|E| = 12$.
- Columns two to eight: indicate whether event e_j can be assigned to room R_j, there are $R = 8$ rooms.

In the above *SuitableRooms* matrix example, event e_1 can be assigned to any

Table 3.7: *SuitableRoom* matrix.

$r_j, j \in \{f_1, ..., f_8\}$/ Event	1	2	3	4	5	6	7	8
e_1	1	1	0	1	0	1	1	0
e_2	1	1	1	0	1	0	1	1
e_3	0	1	1	1	0	1	1	0
e_4	1	0	1	1	0	1	1	1
e_5	0	1	0	0	1	0	0	1
e_6	1	0	1	1	0	1	1	1
e_7	1	1	1	1	0	1	1	0
e_8	0	1	0	1	1	1	0	1
e_9	1	0	0	1	0	1	0	1
e_{10}	1	0	1	1	1	1	1	0
e_{11}	0	1	1	1	1	0	1	1
e_{12}	1	0	0	1	1	0	1	0

of rooms r_1, r_2, r_4, r_6 and r_7, event e_2 can be assigned to any of rooms r_1, r_2, r_3, r_5, r_7 and r_8 and so on until event e_{12} can be assigned to any of rooms r_1, r_4, r_5, and r_7.

A feasible solution (timetable) for these problem instances can be presented by two vectors as illustrated in Figure 3.8 and Figure 3.9. In this example we see that event e_1 is assigned to timeslot 37 (the first element in the vector of Figure 3.8) and room 1 (the first element in the vector if Figure 3.9). Likewise, event e_2 is assigned to timeslot 41 and room 3. Finally, course e_{100} is assigned to timeslot 31 and room 1 (last element in both vectors).

Table 3.8: An Example of a vector of timeslots.

(37, 41, 13, 12, 16, 9, 24, 43, 20, 28, 39, 16, 39, 9, 19, 2, 31, 23, 29,
43, 25, 31, 21, 12, 10, 43, 18, 23, 30, 1, 42, 27, 38, 42, 34, 34, 25, 15,
24, 21, 37, 0, 25, 4, 30, 9, 4, 3, 43, 41, 9, 40, 29, 21, 10, 32, 31, 5, 29,
22, 36, 24, 13, 13, 43, 21 42, 0, 34, 36, 3, 10, 27, 34, 27, 4, 41, 33, 14,
19, 19, 14, 0, 40, 4, 25, 30, 1, 6, 22, 28, 33, 1, 22, 40, 5, 11, 7, 5, 31)

Table 3.9: An Example of a vector of rooms.

(1, 3, 1, 4, 3, 3, 3, 0, 1, 1, 1, 1, 3, 1, 1, 4, 0, 4, 1, 1,
1, 4, 0, 3, 0, 4, 3, 3, 3, 4, 3, 3, 4, 2, 3, 1, 4, 1, 4, 4,
0, 1, 3, 2, 4, 2, 3, 0, 3, 0, 0, 3, 3, 3, 4, 1, 3, 3, 4, 0,
3, 0, 0, 3, 2, 2, 0, 2, 0, 1, 4, 3, 0, 4, 1, 1, 4, 1, 1, 3,
4, 3, 0, 4, 0, 0, 0, 1, 1, 1, 0, 3, 3, 4, 0, 4, 3, 3, 1, 1)

3.6 Conclusions

This chapter has provided the description and formulation of the standard benchmark problem instances used in this thesis and corresponding to the university course timetabling problem. There are two objectives to be achieved in solving this problem. The first objective is to avoid the violation of the hard constraints: 1) avoid students attending two events at the same time, 2) avoid more than one event occupying a room at one time, 3) avoid the room capacity to be exceeded, and 4) ensure that the assigned room has the features required by the events timetabled in the room. The second objective is to minimise the violation of soft constraints: 1) students not to attend only one event on a day, 2) spread the events to avoid students attending more than two consecutive events on a day, and 3) avoid the assignment of events into the last timeslot of a day.

Chapter 4

Constructing Feasible Solutions for UCTTP Using Hybrid Heuristics

4.1 Introduction

In this chapter we focus on initialisation approaches for the university course timetabling problem as discussed in chapter three. In addition to describing the hybrid heuristics that we use in this thesis to create initial feasible timetables, we also give an overview of other approaches from the literature. Not many published articles have been devoted solely to initialisation techniques for the UCTTP and therefore, the contribution of this chapter can be outlined as follows:

1. Development of four hybrid heuristics for initialising feasible solutions. These heuristics are then used as part of the two-stage overall solution approach in this thesis.

2. Hybridisation of existing techniques (constructive heuristics) from the literature. Previously, such heuristics were not able to generate feasible solutions on their own. Here, by applying them in the right sequence, the hybrid methods are now able to produce feasible timetables.

3. Evaluation of the proposed hybrid algorithms by presenting and discussing results from a series of experiments on standard benchmark problems (Socha et al. instances and international competition 2002 instances).

Although the work in this thesis concentrates mainly on solving the 11 Socha et al. [152] instances, we used the ITC 2002 datasets to evaluate our algorithms for constructing initial solutions. This is because we want to assess how effective are the proposed methods in generating feasible solutions for a range of instances with different characteristics.

The content of this chapter is organized as follows. Section 4.2 discusses previous works from the literature on initialisation approaches for the UCTTP. Sections 4.3.1, 4.3.2, 4.3.3 and 4.3.4 discuss each of the four methods that we proposed for constructing feasible solutions for the subject problem. Section 4.4 give the analysis and results for each proposed approach and finally Section 4.5 presents the summary of this chapter.

4.2 Literature Review of Initialisation Methods

Lewis [109] indicates that most of the metaheuristic algorithms for course timetabling problems fall into one of three categories which are:

1. One-Stage Optimisation Algorithms: both hard and soft constraints are tackled simultaneously with the aim to satisfy all of them.
2. Two-Stage Optimisation Algorithms: soft constraints are tackled only after a feasible timetable has been constructed by satisfying all hard constraints first.
3. Algorithms that allow Relaxations: no hard constraints violations are allowed from the outset by relaxing some other feature of the problem. Attempts are then made to try and satisfy soft constraints, whilst also giving consideration

to the task of eliminating these relaxations. In other words, some aspect of the problem has been relaxed to allow the algorithm to tackle the soft constraints. To achieve this, events that cannot find the feasible assignment will be left unassigned. The algorithm will then try to minimise the soft constraints violations expecting that the unassigned events will find a feasible assignment at a later stage. Then, extra timeslots are introduced and in a later stage the algorithm tries to reduce the number of timeslots down to the allowed number of timeslots.

It has been shown in the literature that a *sequential heuristic* method can be very efficient for generating initial solutions [23, 29, 97]. Sequential heuristic is a technique to assign events or course one by one, starting from the event which is measured as the most difficult to assign. Therefore, the degree of difficulty is measured by different heuristic in a different way. (i.e Largest Degree and Saturation Degree, see section 4.3.1 and 4.3.2). However, a sequential heuristic alone does not guarantee that feasible solutions will be found even with combination of more than one heuristic. Abdullah et al. [5, 4] proposed their method to construct initial solutions to generate feasible timetables and in particular for the large instance of the Socha et al. benchmark dataset. However, Abdullah et al. did not report the specifics of their method to generate initial solutions and did not mention how long their approach takes to generate an initial solution in terms of computation time. Their method failed to generate a feasible solution for the large instance of the Socha et al. dataset [152].

Kostuch [97] implemented a two-stage algorithm for the UCTTP. The first stage consists of five steps for constructing the initial feasible solution. These steps are: 1) Initial attempt, 2) Improvement attempt, 3) Shuffling, 4) Blow-ups and 5) Open the last timeslots. Each step of Kostuch's method works as follows:

- The first step is *initial attempt*: it uses the sequential colouring algorithm for the graph colouring where each selected event is based on degeneracy order and more importantly only 40 timeslots out of the 45 available timeslots are considered. The event with the least degeneracy order is that event with the minimum degree (with minimum available resources such as timeslots and rooms). So, this process identifies the event with the least degeneracy order and attempts to assign it to a timeslot. This process continues with all events in the problem. Events that fail to get a timeslot are placed in a pool of unassigned events. As soon as the initial assignment is completed, the maximum matching algorithm for bipartite graphs is applied to every single timeslot for the purpose of assigning rooms. In this phase, events that cannot get a room are removed from their assigned timeslot and placed into the pool of unassigned events.

- The second step is *improvement attempt*: selects every unassigned event and checks the 40 available timeslots. First, the algorithm examines whether the unassigned event fits into any of these 40 available timeslots. If an assignment is possible with no conflicts between assigned events including room assignment, the event leaves the pool of unassigned events and is timetabled.

- The third step is *shuffling*: for a number of iterations, select every event in the pool of unassigned events and assign it into a random selected timeslot. If the selected timeslot does not produce conflicts the room assignment then takes place. After the room assignment takes place there are two possibilities. First, all events in the selected timeslot have been assigned to feasible rooms, therefore, the newly added event finally found the feasible assignment. Second, one of the events in the timeslot might not have a feasible room. If this is the case, the event without room is removed from the timeslot and placed into the pool of unassigned events. If there was a change in the assignments in this timeslot, the improvement attempt (second step) takes place again within this shuffling step. This is because before placing the removed event in the pool of unassigned events, the method tries to fit it into a different timeslot.

- The fourth step is *blows-up*: the algorithm tries to assign the unassigned events into an available timeslot by force, hence the name of this stage. Before the unassigned event can be forced into one of the 40 available timeslots, all events from the timeslot are removed and placed the previously unassigned event is placed into the now empty timeslot. Next, one by one try to assign the removed events into the same timeslot provided that no conflicts are created with the events already in the timeslot. The maximum matching algorithm is used to assign a suitable room to each of the events in the timeslot. Any event that cannot get room at this stage will be removed from the timeslot and put back into the pool of unassigned events. It is known that this procedure may lead to an increased number of unassigned events. After the change, the algorithm once again applies the improvement attempt step to all unassigned events. In this stage, a number of repetitions of the shuffling step are necessary. The whole procedure will be re-started with the assignment that was the best so far in terms of number of unassigned events, once a certain number of blow-up combined with improvement attempt and shuffling steps is reached.

- Finally, step five is *open the last timeslots*: assign events into the last timeslots. In this stage, events that failed to be assigned are now placed into the last five timeslots distributing the events into these five timeslots using the method described above. In case that this step cannot find a feasible assignment using the new five timeslots, the shuffling and blow-up steps will take place but now considering all 45 available timeslots.

Frausto-Solis et al. [77] implemented simulated annealing to find initial feasible solutions for timetabling. In that approach, they introduced extra *dummy* timeslots to satisfy the hard constraints violations. The search for a feasible solution is conducted by decreasing the number of events allocated to the extra *dummy* timeslot while increasing the number of events allocated to the valid timeslots. The whole process stops when no events are left in the extra timeslot and then the number of extra *dummy* timeslot becomes zero. Since the success of simulated annealing is quite dependent on the cooling scheme, the authors employed three phases. In the first phase the temperature grows slowly from T_0 to T_b, where T_0 is the initial temperature, set to $T_0 = 470n + e$ (n is the number of students and e is the number of events), and $T_b = -8m/log(0.95)$ (m is maximum room capacity). In the second phase, the temperature grows from T_b to T_t, where $T_t = 1000$ and this parameter value is obtained from experimental observation. In the third phase, the temperature remains constant from T_t to T_f, the value of $T_f = 0.01$ is also obtained from experimental observation. The 20 ITC 2002 instances and some new *harder* instances were used to evaluate the performance of the algorithm proposed by Frausto-Solis et al. From their experimental results, simulated annealing was able to obtain feasible solutions for the 20 ITC 2002 instances. For the second set of *harder* instances, their results also showed that simulated annealing was capable of generating feasible solutions for all small and medium problems. Moreover, that algorithm was able to generate feasible solutions for 7 out of the 20 ITC instances for which no previous algorithm had been successful before in generating feasible timetables.

Mayer et al. [115] employed Ant Colony Optimisation to construct feasible solutions for post-enrolment based Course Timetabling (ITC 2007 competition dataset). The algorithm procedure can be described as follows. In constructing a feasible solution, each event is selected uniformly at random and assigned into a feasible room and timeslot in a greedy randomised way by considering the pheromone information. The available timeslot and room with higher pheromone values are placed first in the order and are more likely to be selected. Hence, the selected event will be assigned to a pair timeslot-room based on the timeslot-room order. The assignment of the event to a pair timeslot-room that does not violate the partial initial solution, will be accepted. Ejaz and Younus [71] implemented a hybrid approach inspired by ants behaviour. In their approach, they employed a heuristic selector function, where the main role of the heuristic selector is to develop a set of solutions by employing one heuristic per solution. All generated solutions are then compared among them and the heuristic which gives the best solution is then selected. During the construction of the initial solution, a quick but less powerful ant begins to assign courses into feasible timeslots based on the available simple heuristics. When the ant finds itself stuck and unable to go any further, it calls for help. A set of diverse helper ant functions are invoked to help the ant to get out from the trap. Once the ant escapes from the trap it then starts to construct the initial solution again until it manages to generate a feasible solution. This method of Ejaz and Younus has been tested on the 11 Socha et al. instances and produced impressive results managing to generate feasible solutions for all instances.

Arntzen and Løkketangen [14] implemented a sequential assignment of events into timeslots. Their procedure works as follows:

- First step: Create a list of unassigned events L.
- Second Step: Select event E from the list L with minimum feasible timeslots. If there are ties between the events then perform the random selection to break ties.
- Third step: The selected event E is then assigned into a timeslot which gives the minimum weight. The weight of the possible timeslots can be calculated as follows: let K be the feasible timeslot for event E. For each $P \in K$ let $q = (q_1, q_2, ..., q_5)$. The formulation can be defined as: q_1 is the number of available timeslot in P, q_2 is the number of unavailable rooms within the same timeslot as P, q_3 is the violation of the last timeslot by placing event E to P, q_4

change the violation of consecutive events on s day and q_5 is the change of the violation one event timetabled on a day. A vector $w = (w_1, w2, ..., w_5)$ is also defined. Therefore, the timeslot that gives the minimum weight $w_q = (w_1 q_1 + w_2 q_2 +, ..., w_5 q_5)$ will be selected for event E. In case of ties between the weights, a random selection is performed.

- Fourth step: event E leaves the list of unassigned events when the algorithm manages to find a feasible solution for the event. The information about available timeslots for the events left in the list of L is updated. The assignment process ends when the list of events in L becomes empty, otherwise go back to step two. This process stops only after all events have been assigned into feasible timeslots.

Lewis and Paechter [108] generated an initial feasible population of timetables using the Grouping Genetic Algorithm (GGA). The algorithm first selects an event with the lowest saturation degree. In case of ties, random selection then takes place. Then the algorithm selects a timeslot with the least number of unplaced events that could be feasibly assigned. The ties are broken by choosing the timeslot with the most events already assigned to it. If ties continue with this criterion, they are broken at random. The whole procedure works as follows: all events are selected one by one from the set of U and added into the selected timeslot. Events that are unable to find a feasible assignment are ignored. If all events managed to find the feasible solution then U will become empty. However, when some of the events cannot find a feasible assignment, a number of timeslots are added to assign the unplaced events. The number of extra timeslots added is determined by $|U|/r$ where $|U|$ is the number of unplaced events and r is the number of rooms. In order to reach feasibility of the timetable, the algorithm then tries to reduce the number of timeslots down to the allowed number of timeslots by placing all events from the extra timeslots into the valid timeslots.

4.3 Hybrid Initialisation Heuristics

This section gives the description of several effective hybrid algorithms for producing feasible timetables; these algorithms fall into one of the categories listed above. The algorithms presented below are usually considered the first stage within a two-stage optimisation strategy. Usually, a first stage of optimisation is only concerned with solving the hard constraints without paying attention to the soft constraints violations. Therefore, satisfying the hard constraints are the main priority of the algorithms proposed here in order to produce timetables that are at least usable without regard to their quality in term of soft constraints violations.

In order to develop effective algorithms for tackling hard constraints, we conducted an investigation of few techniques such as graph colouring, local search and tabu search and proposed their hybridisation to supplement the weaknesses of each of these search techniques. From the experimental observations, we found that each search component in the hybrid methods are interdependent on their ability to produce a feasible timetable. In other words, when one of these components is disabled or removed, the remaining components are not able to produce feasible solutions and in particular for medium and large instances. Therefore, the hybridisation of the search components is an effective mechanism to improve the performance of the whole search process.

4.3.1 Largest Degree, Local Search and Tabu Search (IH1)

In this approach, we adopted the heuristic proposed by Chiarandini et al. [39] and added the Largest Degree (LD) Heuristic to Step 1 as described next. Largest Degree (LD) refers to the event with the largest number of conflicting events. In course timetabling problem, the conflicting events refer to events that have at least one student registered in common. This modification of the proposed heuristic was necessary because otherwise we were unable to generate feasible solutions for the large problem instance. This hybrid initialisation heuristic works as follows.

Step one - Largest Degree Heuristic. In each iteration, the unassigned event with the largest number of conflicts (other events with students in common) is assigned to a timeslot selected at random without respecting the conflict between the events. Once all events have been assigned into a timeslot, we use the maximum matching algorithm for bipartite graph (see Chiarandini et al. [39]) to assign each event to a room. At the end of this step, there is no guarantee for the timetable to be feasible. The description of maximum matching algorithm for bipartite graph is as follows:

A list of events assigned to a given timeslot but without solved room availability, capacity and features is called a bipartite matching problem (assign events to rooms). As described in [36], room assignment of an event in a specific timeslot can be represented in a bipartite graph $G = (V_1, V_2, E)$, where $V_1 = \{1, ..., n\}$, is the set of events, and $V_2 = \{(t_1, r_1), ..., (t_i, r_j)\}$ is the set of all pairs (timeslot t_i, room r_j). An edge E $(a_n, (t_i, r_j))$ is present if event a_k can be assigned to timeslot t_i in room r_j. Therefore, a set of events is assigned to a specific timeslot and there is a set of possible rooms to which these events can be assigned and that satisfy the room-related hard constraints (capacity and features). Bipartite matching can be treated as a maximum flow problem, and maximum matching algorithm grants a maximum cardinality between these two sets of events and rooms by using a deterministic network flow algorithm.

Step two - Local Search. We employ two neighbourhood moves in this step.

Move one (M1) selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random. Move two (M2) selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained. Therefore we use these neighbourhood moves M1 and M2 to improve the timetable generated in step one. A move is only accepted if it improves the satisfaction of hard constraints (because the moves seek feasibility). This step terminates if after ten iterations no move has produced a better (closer to feasibility) solution. We terminate this step after ten iterations as we do not want to run it too long as it will extend the running time in finding the feasible solution. Moreover, the local search is meant to disturb the solution before we run the tabu search.

Step three - Tabu Search. We apply tabu search using only move M2b (select one event at random and assigns it to a feasible pair timeslot-room also chosen at random). However, the M2b is bit different than M1 in step two, where the algorithm only selects an event that violates the hard constraints. In this step, it is necessary for the algorithm to select an event that violates the hard constraints only. Otherwise, it will find difficulty to construct feasible solutions for the medium5 and large instances in the Socha et al. datasets. Moreover, as a result, it will take a longer time to construct the feasible solution for all instances. The motivation for selecting only events that violate hard constraints is that in this stage, usually, the violation of the hard constraints is very low. Then, the algorithm only targets events that violate hard constraints instead of randomly rescheduling all events with the hope of selecting the appropriate timeslot for the right events. Therefore, we concentrate on events that violate hard constraints and try to minimise the time taken to find feasibility. The tabu list contains events that were assigned less than tl iterations before calculated as $tl = rand(10) + \delta \times n_c$, where ≤ 0 $rand(10) \leq 10$, n_c is the number of events involved in hard constraint violations in the current timetable, and $\delta = 0.6$. The rationale behind the selection of $\delta = 0.6$, is that this value was also used by Chiarandini et al. [39] in their experiments and also we found that this parameter value works very well in our experiments. In order to mitigate the power of tabu search, aspiration criterion is applied to accept when the best known assignment is found. This step terminates if after 500 iterations no move has produced a better (closer to feasibility) solution.

Steps two and three above are executed iteratively until a feasible solution is found. This three-step heuristic is capable of finding feasible timetables for most of the Socha et al [152]. problem instances in reasonable time as shown in Table 4.1. The exception is the large instance for which it takes a minimum of 300 seconds to find a feasible timetable. The large problem instance is the most difficult to tackle and therefore, it takes a much longer time to find a feasible timetable. The reason is that the density matrix for this instance indicates a large number of conflicting events (students in common), in addition there are limited rooms per timeslot, and some of the rooms have small capacity. Thus, these attributes make this instance even harder to find feasible solution in short time. We also tested our approach on the 20 ITC 2002 data sets. The results show that this approach produces feasible solutions in reasonable time for all 20 instances. From the experimental results and observations, we can say that this approach is efficient and effective also for these 20 instances and it demonstrates that the hybridisation of the proposed methods and their cooperation supplements the weakness on each method. The pseudo-code for this initialisation hybrid heuristic is shown in Algorithm 7.

Algorithm 7: Initialisation Heuristic 1 (IH1)

```
Input: set of events in the poolOfUnscheduled events list $E$;
Sort the events in $E$ by using Largest Degree (LD) heuristic;
while ( poolOfUnscheduled events list $E$ is not empty ) do
    Select any timeslot $t$ at random;
    Assign event $e$ from $E$ with largest degree (LD) first into $t$ (tie break at random);
$S$ = current solution;
loop = 0;
while ($S$ is not feasible ) do
    if (loop < 10) then
        if ( coinflip() ) then
            $S^*$ = M1(S); // apply M1 to S
        else
            $S^*$ = M2(S); // apply M2 to S
        if ( $f(S^*) \leq f(s)$) then
            $S \leftarrow S^*$ //accept new solution;
    else
        EHC = set of events that violate hard constraints;
        e = randomly selected member of EHC;
        $S^*$ = M2b(S, $e$); //Perform one iteration tabu search with move M2b using $e$;
        if ( $f(S^*) < f(S)$ then
            $S \leftarrow S^*$; //accept new solution
        if (loop == $ts_{max}$ + 10 ) then
            loop = 0;
    loop++;
Output: S feasible solution (timetable)
```

4.3.2 Saturation Degree, Local Search and Tabu Search (IH2)

In this method, we first, start by choosing a random event from the pool of unscheduled events and then we calculate its Saturation Degree (SD) which refers to the number of available resources (timeslots and rooms) to timetable that event without conflicts in the current partial solution. If there is still at least one available resource, assign a timeslot at random to the event and then apply maximum matching algorithm to assign a room. If there is no conflict, we have managed to schedule the unassigned event.

In the case of no resources left for the selected event, the algorithm selects any timeslot at random. Then, it moves all the events from that timeslot into the pool of rescheduled events and assign the selected event into the now empty timeslot. Events in the pool of rescheduled events need to be rescheduled in any available timeslot, as long as there is available resource for the event. If there is no available resource, the algorithm removes the event to the pool of unscheduled events.

After a maximum trial of 10000 iterations, all events left in the pool of unscheduled events which could not get a feasible assignment will be selected at random and assigned into any timeslots without checking hard constraints violations. The process ends when all events from the pool of unassigned events and rescheduled events becomes empty meaning that we managed to assign them to timeslots.

From the experiments we observed that when the whole process ends, the violation of hard constraints is usually very low already. To ensure feasibility, we then implement local search and tabu search in step two and three as explained in the Initialisation Heuristic 1 above.

The difference between IH1 and IH2 is that in IH1, the assignment of events is done without checking conflicts. Whereas in IH2, we first check conflicts between the unassigned events and then select a timeslot. If there are no conflicts the unassigned event leaves the pool of unscheduled events. If there is a conflict to newly added event into the selected timeslot, the added event will be removed and back to the pool of unscheduled events. Therefore, IH1 might find a feasible solution until the final step is completed or for small instances, until step two is completed. Whereas, in IH2, we managed to construct feasible solutions for small instances by using graph colouring alone, without going through to the next step (local search and tabu search). For medium and large instances, before the local search and tabu search run, the penalty due to hard constrains violation is lower with IH2 than with IH1. The detail of the IH2 is given in pseudo-code Algorithm 8.

Algorithm 8: Initialisation Heuristic 2 (IH2)

```
Input: set of events in the poolOfUnscheduled events list $E$;
while ( poolOfUnscheduled events list $E$ is not empty ) do
    Choose event $e$ from $E$ at random;
    Calculate event $e$ saturation degree (SD);
    if (e with $SD = 0$) then
        Select timeslot $t$ at random;
        Empty the timeslot $t$ by moving all events into the poolOfRescheduled
        events;
        Assign event $e$ into the now empty timeslot $t$;
        trial = 0;
        while (poolOfRescheduled events is not empty ) do
            Select event $e$ from the poolOfRescheduled events at random;
            Assign event $e$ into any feasible timeslot $t$ which is selected at random;
            if (trial > trial maximum) then
                assign event $e$ into any timeslots without respecting the conflict
                between the events;
            else if (e with $SD = 0$ and trial < trial maximum) then
                Eject event $e$ into the poolOfunscheduled events;
            trial++;
    else
        chose feasible timeslot $t$ at random for event $e$;
        Update the new solution;
    if (poolOfUnscheduled events list $E$ is not empty and $time_U$ has elapsed) then
        One by one, place events from the unscheduled events list into any random
        selected timeslot without respecting the conflict between the events;
$S$ = current solution;
loop = 0;
while (S is not feasible ) do
    if (loop < 10) then
        if ( coinflip() ) then
            $S^* = M1(S)$; // apply M1 to S
        else
            $S^* = M2(S)$; // apply M2 to S
        if ( $f(S^*) \leq f(s)$) then
            $S \leftarrow S^*$ //accept new solution;
    else
        EHC = set of events that violate hard constraints;
        e = randomly selected member of EHC;
        $S^* = M2b(S, e)$; //Perform one iteration tabu search with move M2b using $e$;
        if ( $f(S^*) < f(S)$ then
            $S \leftarrow S^*$; //accept new solution
        if (loop == $ts_{max} + 10$ ) then
            loop = 0;
    loop++;
Output: S feasible solution (timetable)
```

4.3.3 Largest Degree, Saturation Degree, Local Search and Tabu Search (IH3)

The pseudo-code for this third initialisation heuristic is shown in Algorithm 9. Two well-known graph colouring heuristics are incorporated, Largest Degree (LD) and Saturation Degree (SD). First, the events in the pool of unscheduled events are sorted based on LD. After that, we choose the event with highest LD and calculate its SD. In the first while loop, the initialisation heuristic attempts to place all events into timeslots while avoiding conflicts. In order to do that, the heuristic uses the SD criterion and a list of rescheduled events to temporarily place conflicting events. The heuristic tries to do this for a given $time_U$ but once that time has elapsed, all remaining unscheduled events are placed into random timeslots. That is, if by the end of the first while loop the solution is not yet feasible, at least the penalty due to hard constraint violations is already very low. In the second while loop, the heuristic uses simple local search and tabu search to achieve feasibility. Two neighbourhood moves M1 and M2 (as the same as moves described in step two - local search in section 4.3.1) are used. The local search attempts to improve the solution but it also works as a disturbing operator, hence the reason for the maximum of ten trials before switching to tabu search. The tabu search uses move M2b only and is carried out for a fixed number of iterations ts_{max}. In our experiments, this initialisation heuristic always finds a feasible solution for all the problem instances considered.

Algorithm 9: Initialisation Heuristic 3 (IH3)

```
Input: set of events in the $poolOfUnscheduled$ events list $E$;
Sort the events in $E$ by using Largest Degree (LD) heuristic;
while (poolOfunscheduled events list $E$ is not empty) do
    Choose event $e$ from $E$ with the LD (tie break at random);
    Calculate SD for event $e$;
    if ($e$ with $SD = 0$) then
        Select a timeslot $t$ at random;
        From those events already scheduled in timeslot $t$ (if any), move those that
        conflict with event $e$ (if any) to the poolOfRescheduled events list;
        Place event $e$ into timeslot $t$;
        for (each event $e$ in the poolOfRescheduled events list with SD $> 0$) do
            Select a feasible timeslot $t$ for event $e$ at random;
            Recalculate SD for all events in the poolOfRescheduled events list;
        Move all events that remain in the poolOfRescheduled events list (those with
        SD $= 0$) to the poolOfUnscheduled events list $E$;
    else
        Select a feasible timeslot $t_i$ at random to place $e$;
    if (poolOfUnscheduled events list $E$ is not empty and $time_U$ has elapsed) then
        One by one, place events from the poolOfUnscheduled events list into any
        random selected timeslot without respecting the conflict between the events;
$S$ = current solution;
loop = 0;
while ($S$ is not feasible ) do
    if (loop < 10) then
        if ( coinflip()) then
            $S^* = M1(S)$; // apply M1 to S
        else
            $S^* = M2(S)$; // apply M2 to S
        if ( $f(S^*) \leq f(s)$) then
            $S \leftarrow S^*$ //accept new solution;
    else
        EHC = set of events that violate hard constraints;
        e = randomly selected member of EHC;
        $S^* = M2b(S, e)$; //Perform one iteration tabu search with move M2b using $e$;
        if ( $f(S^*) < f(S)$ then
            $S \leftarrow S^*$; //accept new solution
        if (loop == $ts_{max} + 10$ ) then
            loop = 0;
    loop++;
Output: S feasible solution (timetable)
```

4.3.4 Constraint Relaxation Approach (IH4)

In this fourth approach, we introduce extra timeslots to place events with zero SD. This initialisation method works as follows. First, we sort the events in the pool of unscheduled events using LD. The event e_j with the LD is chosen to be scheduled first. In the case that, there is no available resource for the chosen event e_j (event with zero SD), the event e_j will be distributed randomly into the extra dummy timeslots. The number of extra dummy timeslots needed is determined by the instance size. In our experiments, we added ten extra timeslots for instances that $100 > |E| \leq 200$ and $100 > |S| \leq 200$ respectively, whereas, 15 extra timeslots added when instances having $200 < |E| \leq 400$ and $200 < |S| \leq 400$ respectively. By introducing extra timeslots we managed to find free-conflict timetables in short computational time and then the search can concentrate on satisfying the soft constraints by moving all events in the extra timeslots into the 45 valid timeslots.

4.3.4.1 Improvement of the Dummy Soft Constraint

Once the algorithm managed to assign all events in the valid timeslots plus the extra timeslots without conflicts, we then perform great deluge to reduce the number of timeslots down to 45 valid timeslots if necessary. However, the great deluge we employed here is a bit different to the original great deluge proposed by Deuck [68]. In our approach, we allowed the water level to go up when the water level and the penalty cost are about to converge. Therefore, the water level is increased to five from the last value when it converges to the penalty cost. The rationale behind the selection increasing the water level to five is that this parameter value work well in our experiment (if too low the the algorithm will stuck in local optimum and if the value is to high the algorithm tend to diversify the search). In this stage, we employ two neighbourhood moves (M1 and M2). Move M1 moves an event selected at random and assign it to a random feasible timeslot. In this move, only the 45 valid timeslots are considered, so no events are allowed to move into any of the extra timeslots. Move M2 selects two events at random and a swaps their assigned timeslots.

The above moves help to place all the events into feasible timeslots. This is because an event with zero saturation degree and that needed to be placed into an extra timeslot, may now fit in some of the 45 valid timeslots because other events in the valid timeslots move to another conflict-free timeslots. The timetable is said to be feasible when all the events that were allocated to the extra timeslots are now scheduled into the 45 valid timeslots without conflict. However, to reduce the timeslot into allowed number of timeslots take longer time and it was too slow to find conflict-free timeslots. The pseudo-code for this initialisation heuristic is shown in Algorithm 10.

Algorithm 10: Initialisation Heuristic 4 (IH4)

```
Input: set of events in the poolOfUnscheduled events list $E$;
if (Large problem) then
    number of timeslots = 60;
else
    number of timeslots = 55;
Sort the events in $E$ by using Largest Degree (LD) heuristic;
while (poolOfunscheduled events list E is not empty) do
    Choose event $e$ from $E$ with the LD (tie break at random);
    Calculate event $e$ SD;
    if ($e$ with $SD == 0$) then
        Select the extra feasible timeslot at random to place event $e$;
        else
            Chose valid feasible timeslot for event $e$ at random;
            Update the new solution;
$S$ = current solution;
Calculate initial cost function $f(S)$;
Initial level $B = f(S)$;
$\Delta B = 0.01$;
while (extra timeslots are not empty) do
    if ( coinflip()) then
        $S^*$ = M1(S); // apply M1 to S
    else
        $S^*$ = M2(S); // apply M2 to S
    if $(f(s^*) \leq f(s))$ or $(f(s^*)(\leq B))$ then
        $S \leftarrow S^*$; //accept new solution
    Lower the level $B = B - \Delta B$;
    if ($B$ - $f(S) \leq 1$) then
        B = B + 5; //increase the water level
Output: S feasible solution (timetable);
```

4.4 Experimental Results and Analysis

To evaluate the performance of the proposed hybrid heuristic initialisation methods, we applied them to the Socha et al. instances and also to the ITC 2002 instances. We coded our algorithm in visual C++ version 6.0 and carried out 10 runs per instance on a Pentium duo with 1.86GHz processor. We did not impose time limit as a stopping condition. Instead, each algorithm stops when it finds a feasible solution. This is because we want to estimate the time that each algorithm takes to find an initial feasible solution.

All methods successfully generate initial solution for small instances in just few seconds using only the 45 timeslots available. The medium and large Socha et al. instances are more difficult as well as all ITC 2002 instances. However, the proposed methods generated feasible solutions for all instances demonstrating that the hybridisation compensates weakness in one approach with strengths in another one in order to produce feasible solutions in reasonable computation times.

Table 4.1 and Table 4.2 compare the performance of each method against each other. From these tables we see that initialisation heuristic one (IH1), initialisation heuristic two (IH2) and initialisation heuristic three (IH3) outperform the initialisation heuristic four (IH4). The results also show that the time taken by initialisation methods IH1, IH2 and IH3 to construct feasible solutions is quite similar. It can also be said that the performance of the first three methods is not very different from each other.

We recorded and analyse the computational time taken to produce feasible solutions based on the instance category. Thus, for every category, we check the minimum and maximum time. For example, we identify what is the minimum and maximum time taken no matter what number of instance they are as long as they are in the same category. Based on their time, we then make our conclusion on the initialisation heuristics performance. For the Socha et al. [152] instances, all four initialisation heuristics are capable to find feasible solutions in short computation time for all small instances. Table 4.1 and Table 4.2 show that IH1, IH2, IH3 and IH4 able to generate feasible solutions for all small instances in a short time. Therefore, the times taken as shown in Table 4.1 and Table 4.2 to generate the feasible solutions indicate that the proposed initialisation heuristics have the same capability to construct feasible solutions for small instances. In the medium instances case, the algorithms now start to show their different capabilities. Table 4.1 and Table 4.2 show that M5 is a more difficult instance and it took bit longer for all heuristics to construct feasible solutions for this dataset. Obviously, IH1 was the faster heuristic to construct feasible solutions and IH4 was the slowest heuristic to generate feasible solutions. Finally, for the large instance, IH4 shows better performance in computation time compared with the other methods. Finally, it is also worth to mention that all methods are capable to generate feasible solution, which show robustness of the methods.

For the second set of experiments, we tested the proposed hybrid initialisation heuristics on the ITC 2002 instances. The results are shown in Table 4.3 and Table 4.4 and indicate that IH1, IH2 and IH3 generate feasible solutions in short computation times of around 1.05-85.61, 1.085-85.345, 0.085-50.675 and 24.546-1007.288 seconds respectively. Theses results also tell us that the initialisation heuristics have almost the same capability and speed to generate feasible solutions in these data sets. When we tested IH4 on the ITC 2002 datasets, we found that the performance is very similar to that on the Socha et al. instances, for which IH4 takes around 24.546-1355.859 seconds to generate feasible timetables.

In Table 4.4 we include Kostuch [97] initial feasible solution results. From the table, it shows that Kostuch approach is capable to generate feasible solutions in a short time and produce good quality of initial solutions compared to our initial solutions quality (penalty cost). In comparison, Kostuch approach takes one of the soft constraints into account (no students are encouraged to attend last slot of the day) during the construction of initial solution, in our approach we did not take into account any soft constraints as the aim is finding feasible solutions without considering the soft constraints violations.

The details of the abbreviations in Table 4.1, 4.2, 4.3 and 4.4 are as follows: IN in column one is Instance Name, IH1 is Initialisation heuristic 1, IH2 is initialisation heuristic 2, IH3 is Initialisation heuristic 3, IH4 is initialisation heuristic 4. SC-P is soft constraints violation penalty, T(sec) is time taken to generate initial feasible solutions in seconds. SD is standard deviation calculated based on the time taken by the algorithms to generate feasible solutions for ten runs. In addition, in Table 4.1 and Tables 4.2, S1-S5 represent small problem instances 1 to 5, M1-M5 represent medium problem instances 1 to 5, L represents the large problem instance, all in the Socha et al instances. In Table 4.1 and Table 4.2 Comp01-Comp20 represent problem instances 1 to 20 in the ITC 2002 instances.

IN	IH1					IH2				
	SC-Penalty		T(sec)		SD	SC-P		T(sec)		SD
	Min	Max	Min	Max		Min	Max	Min	Max	
S1	173	219	0.078	0.125	19.832	198	233	0.077	0.093	14.131
S2	211	268	0.790	0.109	24.296	217	239	0.078	0.109	12.401
S3	176	251	0.068	0.110	38.674	190	244	0.062	0.124	19.854
S4	250	198	0.047	0.110	11.304	174	221	0.078	0.093	19.071
S5	229	260	0.078	0.110	18.352	238	274	0.078	0.109	15.215
M1	817	861	7.546	9.313	16.813	772	941	6.046	9.53	63.217
M2	793	917	9.656	10.937	46.328	782	882	6.342	14.952	40.431
M3	795	901	13.437	21.702	37.685	867	800	10.952	23.358	26.235
M4	735	825	6.891	7.766	41.601	785	858	5.828	6.468	30.435
M5	773	863	16.670	143.560	101.484	771	875	34.102	85.999	40.410
L	1340	1630	300	3000	76.541	1345	1647	1578.567	4500.345	153.317

Table 4.1: Results Obtained by IH1 and IH2 on the 11 Socha et al. Instances.

IN	IH3					IH4				
	SC Penalty		T(sec)		SD	SC Penalty		T(sec)		SD
	Min	Max	Min	Max		Min	Max	Min	Max	
S1	207	273	0.093	0.124	26.864	200	261	0.077	0.109	26.957
S2	189	294	0.093	0.115	39.732	208	281	0.078	0.109	27.468
S3	188	264	0.078	0.108	29.154	209	239	0.062	0.124	10.991
S4	203	235	0.108	0.125	12.457	192	234	0.078	0.093	16.531
S5	226	271	0.093	0.124	18.460	217	263	0.078	0.109	20.169
M1	802	862	5.531	19.29	22.952	774	854	22.702	101.905	33.417
M2	784	871	7.468	9.264	36.789	802	887	60.015	285.265	31.635
M3	828	885	6.64	35.313	22.421	817	865	30.515	230.671	21.651
M4	811	888	5.874	11.564	30.898	795	857	32.624	150.608	24.361
M5	784	835	33.827	91.109	37.278	769	829	115.946	358.561	25.822
L	1686	1807	2050.983	6300.278	45.686	1670	1801	760.811	1914.655	68.461

Table 4.2: Results Obtained by IH3 and IH4 on the 11 Socha et al. Instances.

IN	IH1					IH2				
	SC Penalty		T(sec)			SC Penalty		T(sec)		
	Min	Max	Min	Max	SD	Min	Max	Min	Max	SD
Com01	805	895	2.35	5.347	16.431	786	877	4.696	6.301	20.754
Com02	731	836	2.145	4.347	13.711	776	885	3.917	10.814	59.304
Com03	760	863	1.34	16.155	21.147	812	867	1.146	3.682	27.754
Com04	1201	1266	4.464	62.515	50.796	1178	1269	5.015	46.930	51.694
Com05	1246	1280	2.987	14.411	60.235	1243	1267	4.733	20.317	2.121
Com06	1206	1334	1.780	3.271	59.082	1219	1269	2.452	4.942	21.654
Com07	1391	1539	2.644	20.9	44.128	1388	1504	2.102	21.335	59.253
Com08	1001	1095	1.82	51.421	12.070	968	1079	1.810	13.749	57.726
Com09	841	893	1.464	11.086	36.909	859	968	14.594	8.452	54.671
Com10	786	931	34.678	85.61	13.285	816	858	43.233	85.345	29.698
Com11	852	920	1.05	9.84	37.199	877	896	2.880	7.967	10.016
Com12	814	874	3.016	34.687	38.720	831	873	2.218	36.155	23.180
Com13	1008	1174	2.26	6.976	48.769	1010	1026	2.46	11.567	8.020
Com14	1040	1350	6.816	50.675	44.916	1032	1497	3.716	51.952	328.804
Com15	1165	1259	2.564	8.956	50.423	1162	1364	2.064	9.046	104.887
Com16	887	929	2.592	6.415	24.213	911	967	1.651	4.675	28.213
Com17	1227	1294	3.536	13.048	52.252	1032	1336	1.136	14.952	159.904
Com18	793	871	1.892	3.035	32.144	724	860	1.892	4.249	51.381
Com19	1184	1336	3.928	20.753	71.234	1212	1264	4.228	17.421	26.312
Com20	1137	1254	1.072	1.804	78.014	1161	1243	1.085	1.952	38.837

Table 4.3: Results Obtained by IH1 and IH2 on the ITC 2002 Instances.

IN	IH3					IH4					Kostuch [97]	
	SC Penalty		T(sec)			SC Penalty			T(sec)			
	Min	Max	Min	Max	SD	Min	Max	Min	Max	SD	SC	Time
Com01	805	895	1.93	5.492	32.422	805	882	184.124	365.952	28.112	537	3.4
Com02	731	836	1.36	2.644	39.518	778	844	97.687	471.124	27.481	493	1.5
Com03	760	863	1.34	2.22	36.537	777	856	37.921	358.296	27.674	545	8.4
Com04	1201	1266	4.464	28.98	24.582	1236	1266	116.405	547.007	51.159	760	6.5
Com05	1246	1280	2.112	11.028	13.341	1135	1274	24.546	287.202	55.895	846	8.6
Com06	1206	1334	1.33	3.272	57.365	1133	1307	35.39	163.608	79.892	800	0.5
Com07	1391	1539	2.644	42.402	64.317	1265	1495	33.656	444.843	83.754	859	0.4
Com08	1001	1095	1.82	11.086	35.175	1006	1085	112.999	488.015	31.205	634	1.0
Com09	841	893	1.496	8.088	19.659	843	901	187.624	276.562	23.156	547	0.7
Com10	786	931	4.644	29.045	51.910	799	859	509.827	1355.859	22.887	539	7.9
Com11	852	917	4.768	15.632	32.645	839	908	121.968	536.796	27.613	557	5.3
Com12	814	879	3.016	12.632	27.745	788	879	102.515	426.609	34.178	535	8.4
Com13	1008	1174	2.26	6.976	65.796	1009	1070	268.75	477.12	26.334	662	6.7
Com14	1040	1473	5.816	50.675	171.906	1355	1501	128.174	1007.288	60.087	889	2.7
Com15	1165	1259	1.564	8.956	43.339	1161	1270	135.14	227.89	45.079	761	1.6
Com16	887	929	1.092	3.884	15.630	888	970	116.75	354.34	30.867	582	0.3
Com17	1227	1294	2.136	13.048	28.151	1199	1433	160.78	812.405	88.204	820	12.3
Com18	793	871	1.292	2.948	32.337	763	799	119.75	218.52	15.449	516	1.5
Com19	1184	1336	3.228	20.753	58.668	1209	1294	136.296	405.904	35.525	786	3.8
Com20	1137	1254	0.085	02.104	44.283	1205	1273	36.499	95.03	25.667	761	0.2

Table 4.4: Results Obtained by IH3, IH4 and Kostuch on the ITC 2002 Instances.

As we mentioned above, the sequential heuristic method can be very efficient for generating initial timetables [23, 29, 97]. However, the sequential heuristic alone does not guarantee the construction of feasible solutions even with the combination of more than one heuristics [5, 4]. Then, we conducted a third set of experiments. Table 4.5 and Table 4.6 show the results of the sequential heuristic when applied to the Socha et al. and the ITC 2002 problem instances. Note however that we ran experiments on the medium and large Socha et al. only and all the 20 problems in the ITC 2002 datasets. We did not run experiments with the sequential heuristic on the small Socha et al. instances because it is known that the sequential heuristics generates feasible solution easily for these problems.

In these experiments we also observed that extending the computation time limit does not make any difference in terms of reducing the hard constraints violations. The reason is that the pool of unscheduled events keeps shrinking and growing during the assignment process. This experiments also showed that the sequential heuristic is capable of reducing the hard constraints violations and usually the penalty cost is very low. However it is unable to find feasible solutions.

The details of the abbreviations in Table 4.5 and 4.6 are as follows: IN in column one is Instance Name, HCV(Min) is the minimum hard constraints violation, HCV(Max) is the maximum hard constraints violations, Avg is the average hard constraints violations for ten runs and SD is the standard deviation in the hard constraints violations for ten runs.

IN	HCV(Min)	HCV(Max)	Avg	SD
M1	56	75	66.2	6.906
M2	31	47	40	6.59
M3	30	52	38.4	8.619
M4	48	69	58.2	8.043
M5	21	34	27.4	5.128
L	111	127	119.4	6.387

Table 4.5: Results Obtained by the Sequential Heuristic on the Socha et al. Instances.

IN	HCV(Min)	HCV(Max)	Avg	SD
Comp01	63	78	70.6	5.412
Comp02	48	73	56.8	9.523
Comp03	51	71	63.2	8.871
Comp04	81	100	90.2	8.348
Comp05	41	55	47.4	5.594
Comp06	22	40	33.8	7.429
Comp07	12	18	15.6	2.302
Comp08	26	53	40.4	11.392
Comp09	35	55	41	8.485
Comp10	90	112	100	8.573
Comp11	28	52	41.2	8.927
Comp12	32	47	39.8	5.848
Comp13	24	66	48	16.355
Comp14	15	43	30.6	10.807
Comp15	28	44	35.8	5.761
Comp16	38	75	56.2	14.923
Comp17	26	48	39.8	9.1487
Comp18	48	58	53.2	4.207
Comp19	22	32	26.8	3.563
Comp20	40	64	50.8	11.031

Table 4.6: Results Obtained by the Sequential Heuristic on the ITC 2002 Instances.

The results obtained from these third set of experiments and reported in Table 4.5 and Table 4.6, show that the sequential heuristic alone is unable to generate feasible solutions, no matter how long we extend the computation time. The results in this chapter support our claim that when we disable the local search elements in our hybrid heuristics, the methods are not capable of generating feasible timetables. These experiments also indicate that complex combinatorial problems need additional strategies to help the search escape the local optima, and find a probably optimal solution, in our case being to find feasible timetables. Furthermore, as shown in 4.4 our approaches are able to generate feasible solutions that are competitive with times recorded in Kostuch [97].

4.5 Discussion and Summary

The methods proposed in this chapter are necessary to give different options for constructing timetable, as not many works have been devoted solely to initialisation techniques for the UCTTP problem. In this chapter, we have designed and analysed different initilisation approaches for course timetabling problem that combines the local searches and well known sequential heuristics. The experimental results for each method provides important evidence as a prove that by using the sequential heuristic or the combination of more than one sequential heuristics alone are unable to find feasible solution especially when the size of problem is very large. These methods also give some options for new researchers in combinatorial optimisation area of what techniques can be used to generate feasible solutions. Note that we do not considered the issue of soft constraints with the proposed algorithms in this chapter. This was never our aim, and indeed providing better starting solutions is not the main concern of the first stage of the two-stage optimization algorithms strategy. Furthermore, we want to encourage researchers to contribute on initialisation algorithms.

In this chapter, four hybrid initialisation heuristic approaches employing graph colouring, local search, tabu search and great deluge were proposed as the first stage to tackle the UCTTP. From our experiments, we found that each components on these hybrid initialisation heuristic relies on each other to reach feasibility in the timetable. The graph colouring alone only produces feasible solutions for small instances. The graph colouring method alone is only able to produce close to feasibility for the rest of the problem instances. Moreover, the proposed methodologies were able to generate feasible solutions in reasonable time except for the Initialisation Heuristic 4 (IH4). The methods presented in this chapter are more reliable to reach feasibility compared to sequential heuristics in Abdullah et al. [5, 4]. These methods also able to generate feasible solutions in short time with different quality of solutions.

Chapter 5

An Investigation of the Great Deluge Algorithm

5.1 Introduction

In this chapter we present an investigation into the application of the *Great Deluge Algorithm* to solve the UCTTP. The great deluge algorithm is a meta-heuristic approach proposed by Dueck [68] in 1993 and is inspired by the behaviour that could arise when someone seeks higher ground to avoid the rising *water level* during constant rain. For a maximisation problem, the algorithm seeks to find the highest point on a certain surface with hills, valleys and plateaus (search space). Then, it starts to rain constantly and the algorithm walks around (explores the neighbourhood) but never makes a step into the increasing water level. As it continues raining, the algorithm can explore higher and lower ground (improving and non-improving positions) but is continually pushed to a high point (hopefully close to the optimum) until eventually it cannot escape the rising water level and it stops. The initial water level is set to a value below the fitness of the initial solution and then is increased in a linear fashion as the search progresses. Note that for a minimisation problem, the water level starts on a value above the fitness of the initial solution and decreases constantly. In this case, the algorithm seeks to find the lower point by exploring the surface and maintaining its head below the decreasing water level. One can see that great deluge is similar to simulated annealing (SA) [2] but while SA accepts non-improving solutions based on probability, great deluge does this in a more deterministic manner by controlling the water level. Moreover, great deluge is said to be less dependent upon parameter tuning compared to simulated annealing. In fact, great deluge needs only two parameters. These parameters are: 1) the amount of computational times that user wishes to spend on the search and 2) the expected quality of the final solution [22].

In principle, the decay rate at which the water level decreases determines the speed of the search. The higher the decay rate the faster the water level goes down. Burke et al. [22], initialised the value of the water level equal to the initial cost function. The decay rate at each iteration is decreased gradually at constant decay rate and they interpreted the parameter as a function of expected search time they wish to spend for the entire search process. Besides that they also interpreted the parameter function as expected solution quality they might get at the end of the search process. In order to set the decay rate $\triangle B$, they estimate the desired objective function value $f(S^{'})$ and then calculate $\triangle B$ by applying the following formula $\triangle B = B_0 - f(S^{'})/N_{moves}$ (where B_0 is initial level and N_{moves} is the desired number of iterations). To estimate the desired result $f(S^{'})$ they conducted preliminary runs of a simple Hill-Climbing algorithm. The pseudocode of the so-called extended version of great deluge proposed by Burke et al. [22] is given in Algorithm 11. Figure 5.5 illustrates the behaviour of great deluge in the case of a minimising problem. As we can see in Figure 5.5, the water level decreases linearly guiding the search until at certain point when the water level and the penalty cost both converge and then the algorithm becomes greedy accepting only improving solutions.

The great deluge algorithm was applied to course timetabling by Burke et al. [22]

using the ITC 2002 datasets. Burke et al. observed good performance of great deluge on all the 20 problem instances. Overall, their experimental results showed superiority of great deluge.

Algorithm 11: Great Deluge Algorithm

```
Set the initial solution $s$;
Calculate initial cost function $f(s)$;
Initial level $B_0 = f(s)$;
Specify input parameter $B = B_0$;
while (stopping condition not satisfied do) do
    Define neighbourhood $N(s)$;
    Randomly select the candidate solution $s* \in N(s)$;
    if $(f(s*) \leq f(s))$ or $(f(s*)(\leq B))$ then
        accept $s*$;
Lower the level $B = B - \Delta B$;
```

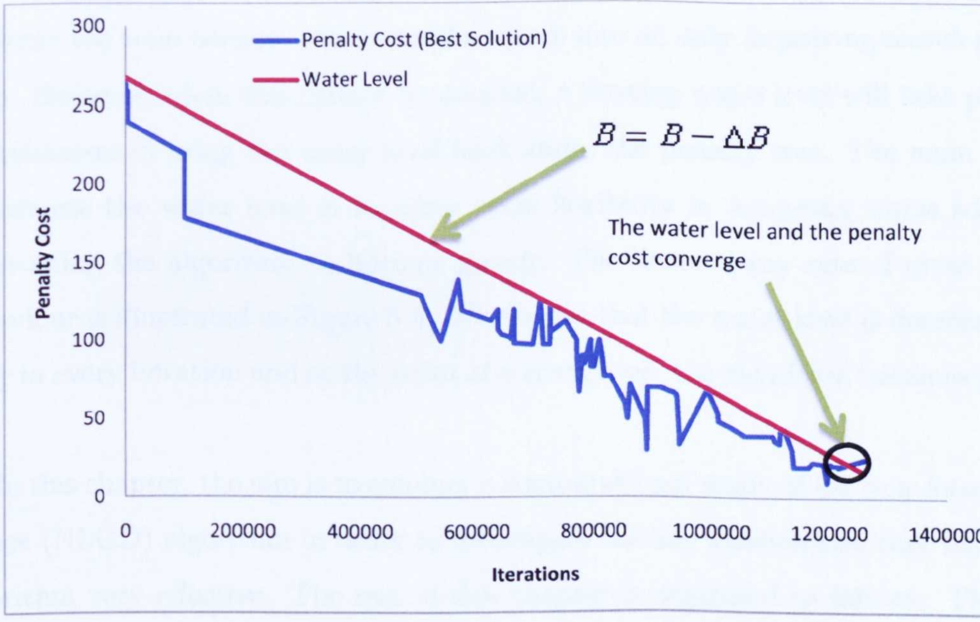


Figure 5.1: Linear Great Deluge Behaviour

In this chapter we propose a modification of the decay rate in the great deluge algorithm. In the original great deluge method, the water level is set to a value equal to the penalty of the best solution at the start of the search. Then, the water level is decreased in a linear fashion until it reaches a value of zero. During the search, the algorithm explores solutions in the neighbourhood of the best solution. A new solution with a lower penalty is accepted straight away replacing the best solution. A new solution with a higher penalty is accepted only if this worse penalty is not higher than the current water level. The modification to the conventional great deluge method proposed in this chapter is on the decay rate of the water level. We propose a non-linear great deluge algorithm in which the water level decay rate is controlled by an exponential function and floating water level (increase the water level) when the penalty cost and the water level is about to converge. While Burke et al. [22] run the hill climbing algorithm in order to estimate the initial penalty cost, the only thing we need is to control the water level speed and avoid it to converge with penalty cost too frequently during the search. We do not want the water level and the penalty cost to converge too soon because this turns the search into an only improving search mechanism. However, when this cannot be avoided, a floating water level will take place as a mechanism to bring the water level back above the penalty cost. The main reason to increase the water level is to allow some flexibility in accepting worse solutions by avoiding the algorithm to become greedy. The linear decay rate of great deluge behaviour is illustrated in Figure 5.5. We can see that the water level is decreased linearly in every iteration and at the point of convergence, the algorithm becomes greedy.

In this chapter, the aim is to conduct a computational study of the *non-linear great deluge* (NLGD) algorithm in order to investigate the key mechanisms that make this algorithm very effective. The rest of this chapter is organised as follows. The non-linear great deluge algorithm proposed in this chapter as well as its application to the UCTTP is described in Section 5.2. Experiments and results are presented and discussed in Sections 5.3 and 5.4. Section 5.3 focuses on the overall performance of the proposed method and Section 5.4 studies in more detail the effect that the non-linear decay rate has on the overall performance of the algorithm. The work presented in this chapter was published in the proceeding of the 2008 IEEE Conference on Intelligent Systems (IS 2008). (see Landa-Silva and Obit [100]) and Intelligent Systems - From Theory to Practice Obit and Landa-Silva [87].

5.2 Great Deluge With Non-Linear Decay Rate Approach

This section presents a modified great deluge algorithm using a non-linear decay rate. The motivation behind this modification with a non-linear decay rate and floating water level is to enhance the feedback between the search activity and the water level. Early in the search the algorithm is able to reduce the penalty cost considerably and the gap between the water level and the penalty cost is usually very large. Therefore, the algorithm must prevent the cost function to go back near to the water level and for this reason it is important to reduce the gap between the water level and the penalty cost. Later in the search, it becomes more and more difficult to find the improvement moves. To manage this situation, we float the water level to prevent the algorithm becoming greedy. By floating the water level the algorithm tries to diversify the search by extending its search to a different region of the search. Therefore, at the early stage of the search this algorithm performs more on the intensification and less diversification. However, when the search stuck in the local optima the algorithm begins to diversify the search by floating the water level (increasing the water level).

The main weakness with the linear decay for the water level is that the water level decreases too quicks in the later stages of the search. At the beginning, the algorithm seems to produce several successful moves. However when the search is in the middle or approaching the end of the search and the water level converges with the value of the current best solution, most of the neighbourhood solutions are rejected and this situation hinders the algorithm in diversifying the search. Therefore, the algorithm suffers on its own greediness by trapping itself in local optimum. In the conventional great deluge approach, there is no mechanism to help escaping local optima once the water level and the best solution penalty cost converge.

5.2.1 Initial Feasible Timetables and Neighbourhood Structures

To construct feasible timetables, we took the hybrid initialisation heuristic 1 (IH1) proposed in chapter 4 (Since the performance of the IH1, IH2 and IH3 are similar, therefore we can select any of this hybrid heuristic to generate feasible solutions). In addition, we employed neighbourhood moves M1, M2 and M3 in the improvement search. Moves M1 and M2 were described above. M3 selects three events at random and performs a permutation between their timeslots at random. Note that the three neighbourhood moves are based on random search but always seeking the satisfaction of hard constraints. The three neighbourhood moves used here might consider very small but this is because we want to better assess the effectiveness of the non-linear decay rate in the proposed algorithm for guiding the local search.

5.2.2 Non-linear and Floating Water Level Decay

Consider a problem in which the goal is to find the solution that minimises a given objective function. The distinctive feature of the conventional great deluge algorithm is that when the candidate solution S^* is worse than the current solution S, then S^* replaces S depending on the current water level B. The water level is initially set according to the quality of the initial solution, that is, $B > f(S^0)$ where $f(S^0)$

denotes the objective function value of the initial solution S^0. The decay, i.e. the speed at which B decreases, is determined by a linear function in the conventional great deluge algorithm:

$$B = B - \Delta B \text{ where } \Delta B \in \Re^+ \tag{5.1}$$

The non-linear great deluge algorithm uses a non-linear decay for decreasing the water level. The decay is given by the following expression:

$$B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta \tag{5.2}$$

The various parameters in Eq. (5.2) control the speed and the shape of the water level decay rate. Parameter β represents the minimum expected value corresponding to the optimal solution. In this paper, we set $\beta = 0$ because we want the water level to reach that value by the end of the search. This is because we know that an optimal value of zero is possible for the problem instances tackled in this thesis. If for a given minimisation problem we knew that the minimum objective value that can be achieved is let's say 100, then we would set β around that value. If there is no previous knowledge on the minimum objective value expected, then we suggest to tune β through preliminary experimentation for the problem in hand. The role of the parameters δ, min and max (more specifically the expression $\exp^{-\delta(rnd[min,max])}$) is to control the speed of the decay and hence the speed of the search process. A random min and max are drawn from the uniform distribution interval [min,max] and the min and mix are integer numbers. By changing the value of these three parameters, the water level goes down faster or slower. Therefore, The lower the values of min and max, the faster the water level goes down, and in consequence, the search quickly achieves improvement but it also gets stuck in local optima very early. To escape from the local optima, the algorithm needs to increase the water level.

In this chapter, the value of the parameters in Eq. (5.2) were determined by experimentation. We tested different combination of parameter values ($-\delta$ and $rnd[min, max]$) and observe the effect of each combination in order to find suitable parameters for given problem. Based on the preliminary experiments, we now then assigned, δ the values of 5×10^{-10}, 5×10^{-8} and 5×10^{-9} for small, medium and large instances respectively. As said before, the value of β for all problem instances is $\beta = 0$. The values of min and max in Eq. (5.2) are set according to the size of the problem instance. For medium and large problems we used $min = 100000$ and $max = 300000$. For small problems we used $min = 10000$ and $max = 20000$. The parameter values for small instance is only apply when the penalty cost reach to 10 points. Therefore, it means that from the first iteration the NLGD uses the same parameter used for medium instances and changes the parameter when it reaches the penalty cost to 10 points. The use of the non-linear decay rate is shown in the last **else** of Algorithm 12 below.

In addition to using a non-linear decay rate for the water level B, we also allow B to go up when its value is about to converge with the penalty cost of the candidate solution S^*. This occurs when $range \leq 1$ in Algorithm 12 (range is the difference between the water level and the penalty cost). We increase the water level B by a random number within the interval $[B_{min}, B_{max}]$. All the parameter values in $[B_{min}, B_{max}]$ were identified by experimentation. For small problem instances the interval used was [2,5]. For the large problem instance the interval used was [1,3]. For medium problem instances, we first check if the penalty of the best solution so far $f(S_{best})$ is lower than a parameter f_{low}. If this is the case, then we use [1,4] as the interval $[B_{min}, B_{max}]$. Otherwise, we assume that the best solution so far seems to be stuck in local optima ($f(S_{best}) > f_{low}$) so we make $B = B + 2$. The concept of floating water level might be similar to reheating concept in simulated annealing, however in simulated annealing to reheat the temperature, it uses the geometric reheating method. In our method we increase the water level at random. In addition,

Algorithm 12: Non-linear Great Deluge Algorithm

```
Construct initial feasible solution $S$
Set best solution so far $S_{best} \leftarrow S$
Set $timeLimit$ according to problem size
Set initial water level $B \leftarrow f(S)$
while $elapsedTime \leq timeLimit$ do
  Select move at random from M1,M2,M3;
  if (move == M1 ) then
    $S^* = M1(S)$; {apply M1 to S};
  end if
  if (move == M2 ) then
    $S^* = M2(S)$ {apply M2 to S};
  end if
  if (move == M3 ) then
    $S^* = M3(S)$ {apply M3 to S};
  end if
  if ( $f(S^*) \leq f(S)$ or $f(S^*) \leq B$ ) then
    $S \leftarrow S^*$ {accept new solution}
    $S_{best} \leftarrow S$ {update best solution}
  end if
  // $range$ is the difference between the water level ($B$) and the current best solution $f(S^*)$
  $range = B - f(S^*)$
  if ($range < 1$) then
    if (Large or Small Problem) then
      $B = B + rand[B_{min}, B_{max}]$
    else
      if ( $f(S_{best}) < f_{low}$ ) then
        $B = B + rand[B_{min}, B_{max}]$
      else
        $B = B + 2$
      end if
    end if
  else
    if $f(S_{best<=20})$ and Small then
      $B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta$
      (Apply small instances parameter)
    else
      $B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta$
    end if
  end if
end while
```

acceptance in simulated annealing uses probability while great deluge does not employ probability. Full details of this strategy to control the water level decay rate in the modified great deluge are shown in Algorithm 12.

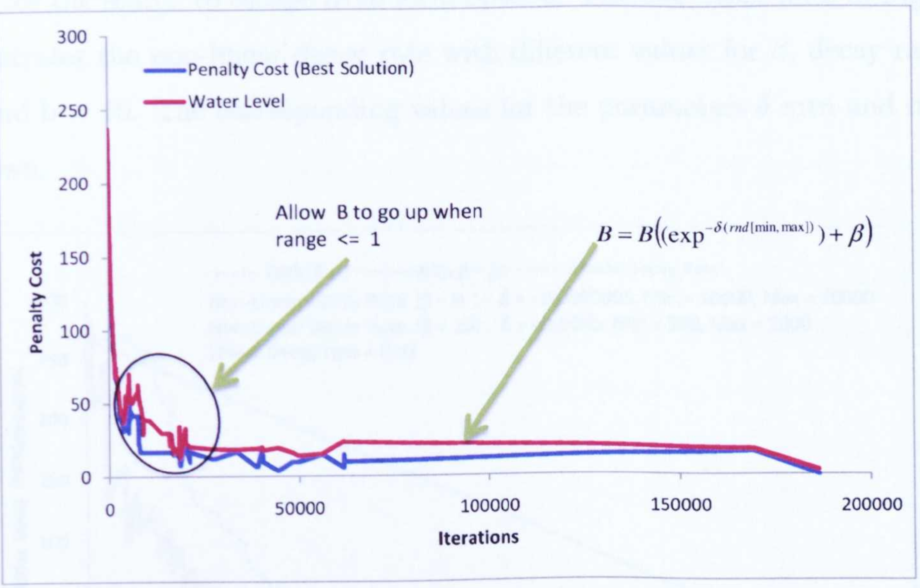


Figure 5.2: Behaviour of Non-Linear Great Deluge With Non-linear and Floating Water Level Decay Rate.

The behaviour of the proposed NLGD is illustrated in Figure 5.6. From the outset, the water level is equal to the current penalty cost. When the search progresses the current penalty cost improves as shown by the blue line. The water level decreases quickly to prevent a huge gap between the water level and the current penalty cost. But when the water level current penalty cost is about to converge the algorithm then increases the water level as shown by the red line.

Figure 5.3 illustrates the difference between the linear and non-linear decay rates. The graph also illustrates the effect of parameters β, δ, min and max on the non-linear decay rate. The straight line in Figure 5.3 corresponds to the linear decay rate (with $\Delta B = 0.01$) originally proposed by Dueck [68]. In this case, a non-improving candidate solution S^* is accepted only if its objective value $f(S^*)$ is below the water level B. When $f(S^*)$ and B converge the algorithm becomes greedy and it is more difficult for the search to escape from local optima. The two other lines in Figure 5.3 also illustrates the non-linear decay rate with different values for β, decay rate with b = 0 and b = 10. The corresponding values for the parameters δ min and max are also shown.

With β= 0
With β = 10
Linear Decay Rate
Non-Linear Decay Rate (β = 0): δ = - 0.0000005, Min = 10000, Max = 30000
Non-Linear Decay Rate (β = 10) : δ = - 0.0005, Min = 500, Max = 1000
Linear Decay rate = 0.01
Water Level (Minimisation)
300
250
200
150
100
50
0
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
Iterations (x 0000)

Figure 5.3: Comparison Between Linear (Eq. 5.1) and Non-linear (Eq. 5.2) Decay Rates and Illustration of the Effect of Parameters β, δ, min and max on the Shape of the Non-linear Decay Rate in the Great Deluge Algorithm.

5.3 Experiments and Results

To measure the impact of the modified non-linear water level decay rate, we conducted several experiments using the UCTTP benchmark instances by Socha et al. [152]. It is known that for each instance there exists at least one solution (timetable) with an penalty function value equal to zero. For each type of dataset (in terms of size), a fixed computation time (timeLimit) in seconds was used as the stopping condition: 2500 for small problems, 4700 for medium problems and 6700 for the large problem. This fixed computation time is only for the improvement phase, i.e. the non-linear great deluge starting from a feasible solution. For each problem instance we executed the non-linear great deluge algorithms for 20 times.

In the first set of the experiments, we compared the NLGD to other algorithms reported in the literature. The algorithms considered include MAX-MIN Ant System [152], choice function hyper-heuristic [27], fuzzy multiple heuristic ordering [15], VNS with tabu search [5], Randomised Iterative Improvement with neighbourhoods [4], Graph-based hyper-heuristic [29] and Hybrid Evolutionary Algorithm [3]. In these experiments we want to evaluate how beneficial it is to modify the water level decay rate from the linear to non-linear and floating water level in the great deluge algorithm.

Table 5.1 shows the results obtained by the NLGD and by the original great deluge algorithms alongside other results reported in the literature. Table 5.2 shows the penalty of the initial solution provided to the great deluge approaches.

Table 5.1 shows the comparison of non-linear great deluge with the existing state-of-the-art approaches found in the literature. The best results are in bold whereas, the term "x%Inf" in Table 5.1 indicates a percentage of runs in which the approaches failed to generate feasible solutions. From the table it is clear that our approach is able to produce feasible solutions for all 11 instances. The results for the NLGD also show that the proposed modified decay rate makes the algorithm competitive to other approaches that have been applied on these instances. When compared to a local search method and ant algorithm by Socha et al [152], the Graph hyper-heuristic by Burke

Table 5.1: Comparison of Results Obtained by the Non-Linear Great Deluge (NLGD) Proposed in this Chapter Against the Best Known Results from the Literature for the 11 Socha et al. Problem Instances.

	Best (NLGD)	Avg (NLGD)	RRLS	MMAS	GALS	RIICN	GBHH	CFHH	VNS-T	HEA	FMHO
S1	3	3.9	8	1	**0**	**0**	6	1	**0**	**0**	10
S2	4	5	11	3	**0**	**0**	7	2	0	**0**	9
S3	6	7	8	1	**0**	**0**	3	**0**	**0**	**0**	7
S4	6	7.3	7	1	**0**	**0**	3	1	**0**	**0**	17
S5	**0**	2	5	**0**	**0**	**0**	4	**0**	**0**	**0**	7
M1	**140**	152	199	195	175	242	372	146	317	221	243
M2	**130**	147.2	202.5	184	197	161	419	173	313	147	325
M3	**189**	207.2	77.5%Inf	248	216	265	359	267	357	246	249
M4	**112**	129.8	177.5	164.5	149	181	348	169	247	165	285
M5	141	174.7	100%Inf	219.5	190	151	171	303	292	**130**	132
L	876	962.2	100%Inf	851.5	912	-	1068	80%Inf	-	**529**	1138

NLGD is Non-Linear Great Deluge [100].

RRLS is the Local Search and Ant System in [153]

MMAS is the MAX-MIN Ant System in [152]

GALS is Genetic algorithm and local search by Abdullah and Turabieh [7].

RIICN is Randomised iterative improvement algorithm by Abdullah et al. [3].

GBHH is Graph-based Hyper-heuristic by Burke et al. [29].

CFHH is the Choice Function Hyper-heuristic in [27]

VSN-T is Variable neighbourhood search with tabu by Abdullah et al. [5].

HEA is Hybrid evolutionary approach by Abdullah et al. [4].

FMHO is fuzzy multiple heuristic ordering [15]

S1-S5 represent small problem instances 1 to 5

M1-M5 represent medium problem instances 1 to 5

L represents the large problem instance

et al. [29] and the fuzzy approach by Asmuni et al. [15], the Non-linear great deluge (NLGD) produced better solution or competitive results on the small instances. Moreover, NLGD is able to find the global optimum for small5 unlike GBHH and FMHO approaches. Compared with other algorithms like genetic algorithm and local search by Abdullah and Turabieh [7], Randomised iterative improvement algorithm by Abdullah et al. [4], Variable neighbourhood search with tabu by Abdullah et al. [5] and Hybrid evolutionary approach by Abdullah et al. [3], we see that NLGD performs less competitive for instances S1, S2, S3 and S4. For medium instances, NLGD produced better solutions in four out of five compared to all approaches. Even though NLGD could not beat the best result for instance M5, the best result it managed to find is not too far from the best. Finally for the large instance, NLGD produces comparable results when compared to other approaches.

Table 5.1 shows the comparison of NLGD and the other results reported in the literature. Table 5.1 make an interesting evaluation of our proposed modification of the decay rate of the great deluge algorithm. It can be seen that NLGD outperformed some of the best known results obtained by other algorithms. In addition NLGD is able to solve one out of the five small instances to optimality within 20 runs. It also can be seen that the NLGD approach produces better or equivalent results on the small instances when compared against RRLS, GBHH and FMHO. In more detail NLGD produced better results for all small instances when compared to RRLS and FMHO. For S1 and S2, and S5, NLGD produced better results when compared to GBHH. However, there are alteration in the penalty cost for the set of medium instances specifically M1, M2, M3 and M4, where NLGD has shown significant improvement over other algorithms. The first two comparisons are results against MMAS and CFHH. NLGD are better than the MMAS for all medium instances. Comparisons are also made with results from other methods such as VNST, GBHH and RIICN. NLGD results are better than VNST, GAHH and RIICN in all 5 medium instances. For M5, even though NLGD unable to improve the solution when compared to FMHO and HEA, however, the quality of the solution is still competitive.

Table 5.2 presents a comparison of the results obtained for all three categories (small, medium and large) instances by the modified great deluge (NLGD) and the reference algorithm, the conventional great deluge (GD), in addition to comparing to best results obtained by other approaches. It is clear that the modified great deluge (NLGD) performs significantly better than any of the reference algorithms on instances M1, M2, M3 and M4. It is however interesting to observe that while NLGD performs better than GD and is competitive for M5 with HEA, the HEA produced solution better than those produced by NLGD in the large instance. Hence, the proposed non-linear great deluge seems particularly effective on the medium problem instances producing new best results four out of five medium instances. Based on the results shown in Table 5.1 and Table 5.2 we can claim that our algorithm outperforms some of the previous published results and it is also competitive on the rest of the instances.

Table 5.2: Comparison of Results Obtained by the Non-linear Great Deluge (NLGD) Proposed in this Paper Against the Best Known Results from the Literature for the 11 Socha et al. Problem Instances.

	Init. Sol.	GD	NLGD	Best Known
S1	198	17	3	**0** (VNS-T)
S2	265	15	4	**0** (VNS-T)
S3	214	24	6	**0** (CFHH)
S4	196	21	6	**0** (VNS-T)
S5	233	5	**0**	**0** (MMAS)
M1	858	201	**140**	146 (CFHH)
M2	891	190	**130**	147 (HEA)
M3	806	229	**189**	246 (HEA)
M4	846	154	**112**	164.5 (MMAS)
M5	765	222	141	**130** (HEA)
L	1615	1066	876	**529** (HEA)

MMAS is the MAX-MIN Ant System in [152]
CFHH is the Choice Function Hyper-heuristic in [27]
VNS-T is the Hybrid of VNS with Tabu Search in [5]
HEA is the Hybrid Evolutionary Algorithm in [4]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L represents the large problem instance

In the second set of experiments we tested NLGD on the International Competition instances. Table 5.3 presents the best values obtained by different algorithms including NLGD. The table gives us an idea about the variability of the performance for different algorithm proposed in the competition. From the Table 5.3, shows that even though our NLGD did not obtained the best results but it is still competitive when compared to the results obtained by the fifth to ninth places in the competition. The proposed algorithm is mainly tuned for the 11 instances and no modification have been done in order to tackle the international timetabling competition instances. Therefore the results obtained are only moderate when competing with solvers designed specifically for the competition.

In more detail, Figures 5.4-5.9 summarise the performance of NLGD compared to other allgorithms. In these graphs, the x-axis represents the instance type while the y-axis represents the penalty cost. Figure 5.5 shows the strong performance of NLGD on the medium and large instances. Figures 5.6-5.9 show details of the results achieved by NLGD when compared to the algorithms from the competition.

Table 5.3: Comparison of results obtained by the non-linear Great Deluge (NLGD)with other approaches for the International Timetabling Competition on the 20 instances. Details of the competition algorithms are available at: http://www.idsia.ch/Files/ttcomp2002/results.htm.

Instances	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	NLGD
com01	**45**	61	85	63	132	148	178	211	257	153
com02	**25**	39	42	46	92	101	103	128	112	118
com03	**65**	77	84	96	170	162	156	213	226	120
com04	**115**	160	119	166	265	350	399	408	441	358
com05	102	161	**77**	203	257	412	336	312	299	398
com06	13	42	**6**	92	133	246	246	169	209	129
com07	44	52	**12**	118	177	228	225	281	99	99
com08	**29**	54	32	66	134	125	210	214	194	111
com09	**17**	50	184	51	139	126	154	164	175	119
com10	**61**	72	90	81	148	147	153	222	308	153
com11	44	53	73	65	**35**	144	169	196	273	149
com12	107	110	**79**	119	290	182	219	282	242	229
com13	**78**	109	91	160	251	192	248	315	364	240
com14	52	93	**36**	197	230	316	267	345	156	282
com15	**24**	62	27	114	140	209	235	185	95	172
com16	**22**	34	300	38	114	121	132	185	171	91
com17	86	114	**79**	212	186	327	313	409	148	356
com18	**31**	38	39	40	87	98	107	153	117	190
com19	**44**	128	86	185	256	325	309	334	414	228
com20	7	26	**0**	17	94	185	185	149	113	72

1st SA-based Heuristic (Three-Phase Approach): Kostuch [97]

2nd Tabu Search : Brigitte Jaumard, Jean-Franois Cordeau and Rodrigo Morales [43]

3rd Great Deluge : Yuri Bykov [22]

4th Local Search Paradigm (Hill Climbing, Tabu Search and Multi-swap shake): Luca Di Gaspero and Andrea Schaerf [63]

5th local search heuristic : Halvard Arntzen, Arne Løkketangen [13]

6th tabu search. : Alexandre Dubourg, Benoît Laurent, Emmanuel Long and Benoît Salotti [67]

7th tabu Search : Gustavo Toro, Victor Parada [157]

8th Guided Simulated Annealing and Local Searches : Roberto Montemanni [119]

9th Local Search : Tomáš Müller [125]

NLGD Non-Linear Great Deluge

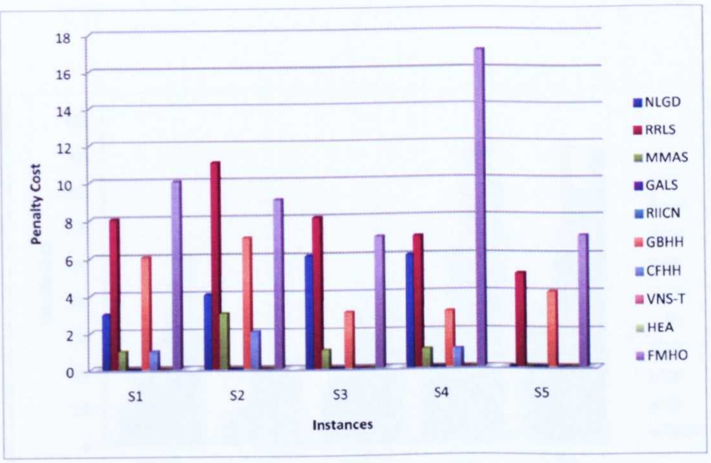


Figure 5.4: Detailed comparison of non-linear great deluge against other algorithms for small instances.

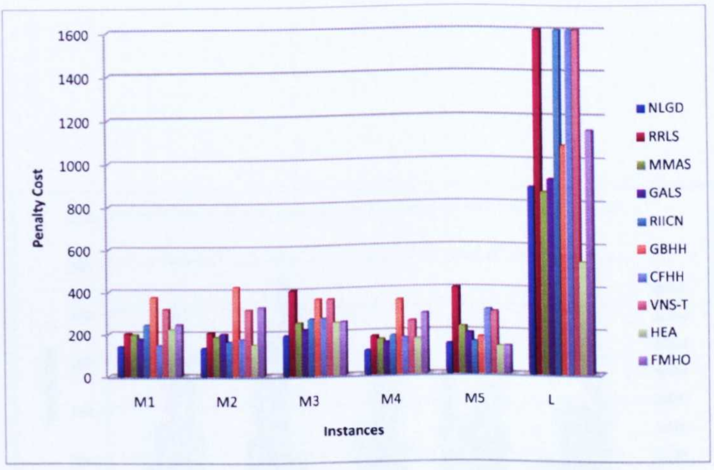


Figure 5.5: Detailed comparison of non-linear great deluge against other algorithms for medium instances.

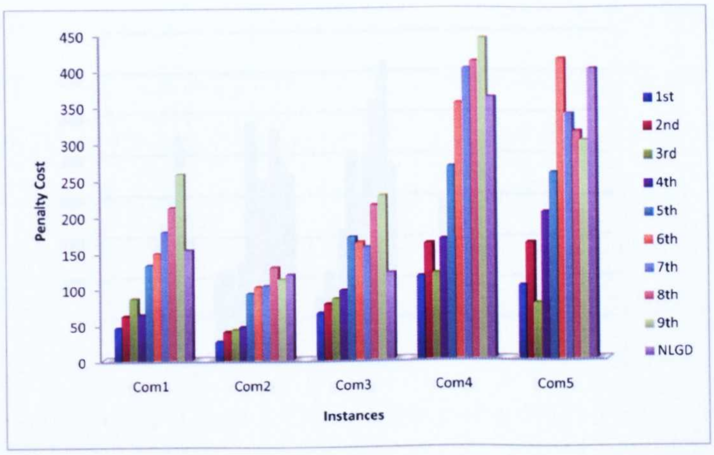


Figure 5.6: Detailed comparison of non-linear great deluge against other algorithms for com01-com05 instances.

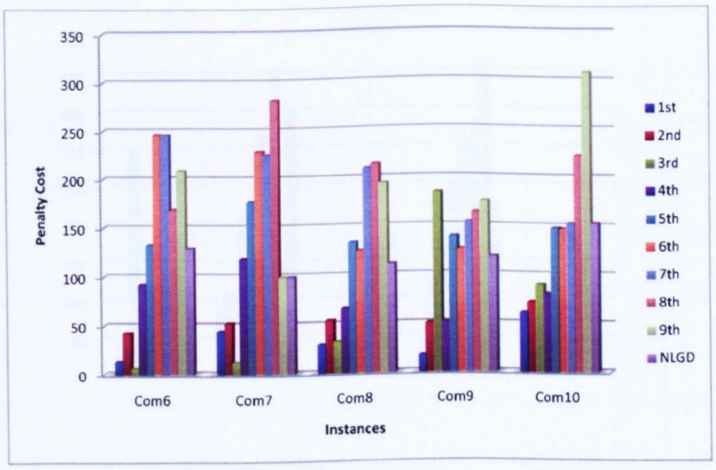


Figure 5.7: Detailed comparison of non-linear great deluge against other algorithms for com06-com10 instances.

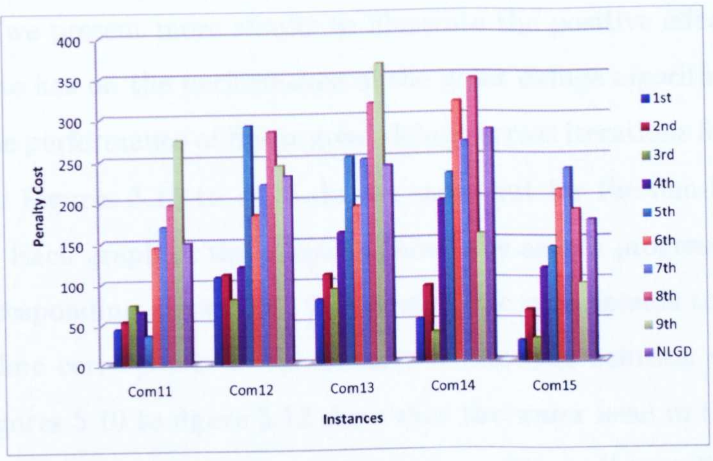


Figure 5.8: Detailed comparison of non-linear great deluge against other algorithms for com11-com15 instances.

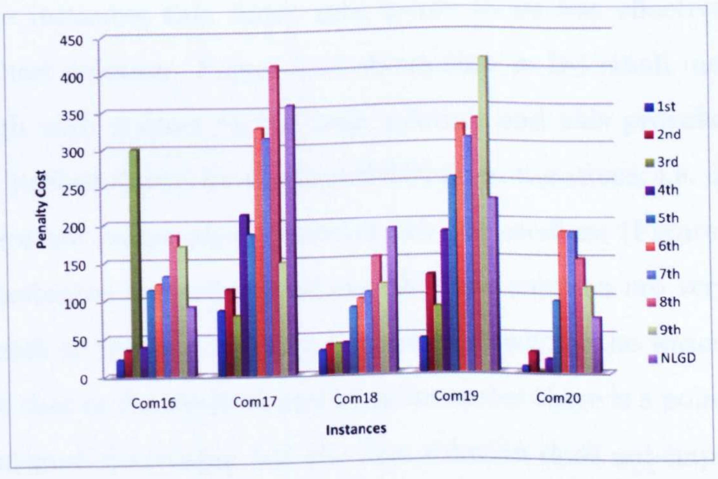


Figure 5.9: Detailed comparison of non-linear great deluge against other algorithms for com16-com20 instances.

5.4 Effect of the Non-linear Decay Rate

In this section we present more results to illustrate the positive effect that the non-linear decay rate has on the performance of the great deluge algorithm. Figures 5.10 to 5.12 show the performance of linear great deluge across iterations for three problem instances while Figures 5.13 to 5.15 do the same but for the non-linear version of the algorithm. Each graph in these figures shows the search progress for one sample run of the corresponding algorithm. The dotted line corresponds to the water level and the solid line corresponds to the penalty of the best solution which should be minimised. Figures 5.10 to figure 5.12 show that the water level in the original great deluge decreases at the same rate in every iteration while in the modified great deluge proposed in this paper the water level decreases exponentially according to Eq. (5.2).

The first interesting observation is that the relation between the water level and the best solution varies for different instance sizes. The rigid and pre-determined linear decay rate appears to suit better the medium problem instances while for the small and large instances this decay rate seems to be less effective in driving the search for the best solution. Figure 5.10 shows that in the small instance the water level is too high with respect to the best solution and this provokes that the best solution is not 'pushed down' for the first 60000 or so iterations, i.e. improvements to the best solution are rather slow. However, for the medium (Figure 5.11) and large (Figure 5.12) instances the water level and the best solution are very close from the start of the search so the best solution is 'pushed down' as the water level decreases. We can also see that in the medium and large instances there is a point after which the water level continues decreasing but the best solution does not improve further, i.e. the search stagnates. That is, when the water level and the best solution 'converge', the search becomes greedy and improvements are more difficult to achieve while the water level continues decreasing. This occurs around iteration 110000 in the medium instance and around iteration 8000 in the large instance. We argue that the simple linear water level decay rate in the original great deluge algorithm does not adapt easily to the quality of the best solution. This is precisely the shortcoming that we tackle in this chapter and hence our proposal for a non-linear great deluge algorithm.

Then, in the non-linear version of the algorithm, the decay rate is adjusted at every iteration and the size of the problem instance being solved is taken into account when setting the parameters β, δ, min and max as explained in Section 5.2.2. We can see in Figures 5.13 to 5.15 that this modification helps the algorithm to perform a more effective search regardless of the instance size. We can see that in the three sample runs of the non-linear great deluge algorithm, if drastic improvements are found then the water level also decreases more drastically. But when the improvement to the best solution becomes slower than the decay rate also slows in reaction to this. Moreover, to avoid (as much as possible) the convergence of the water level and the best solution, the water level is increased from time to time as explained in Section 5.2.2. This 'floating' feature of the water level explains the small increases on the best solution penalty observed in the graphs of Figures 5.13 to 5.15. As in many heuristics based on local search, the rationale for increasing the water level is to accept slightly worse solutions to explore different areas of the search space in the hope of finding better solutions.

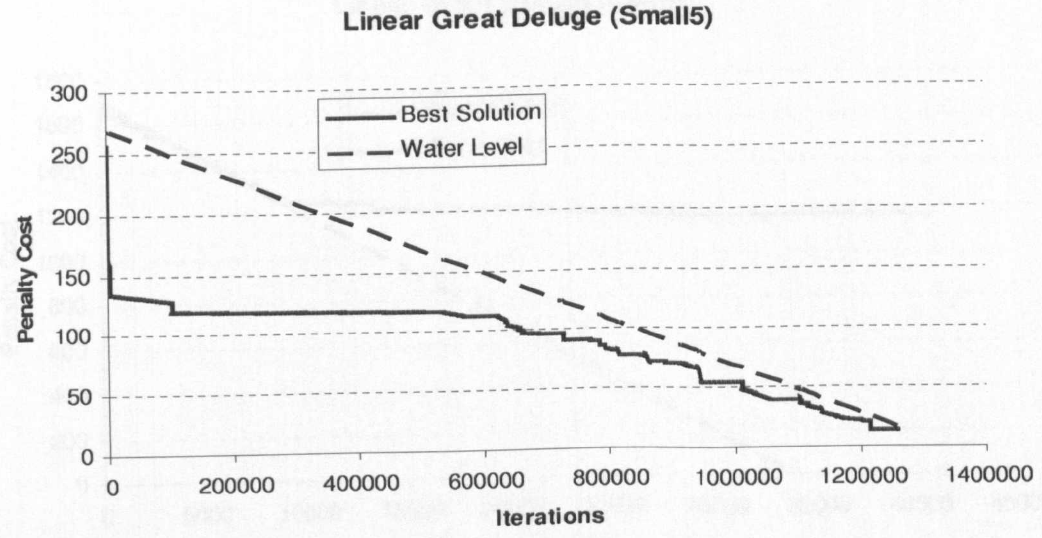


Figure 5.10: Behaviour of Linear Great Deluge on Instance small5.

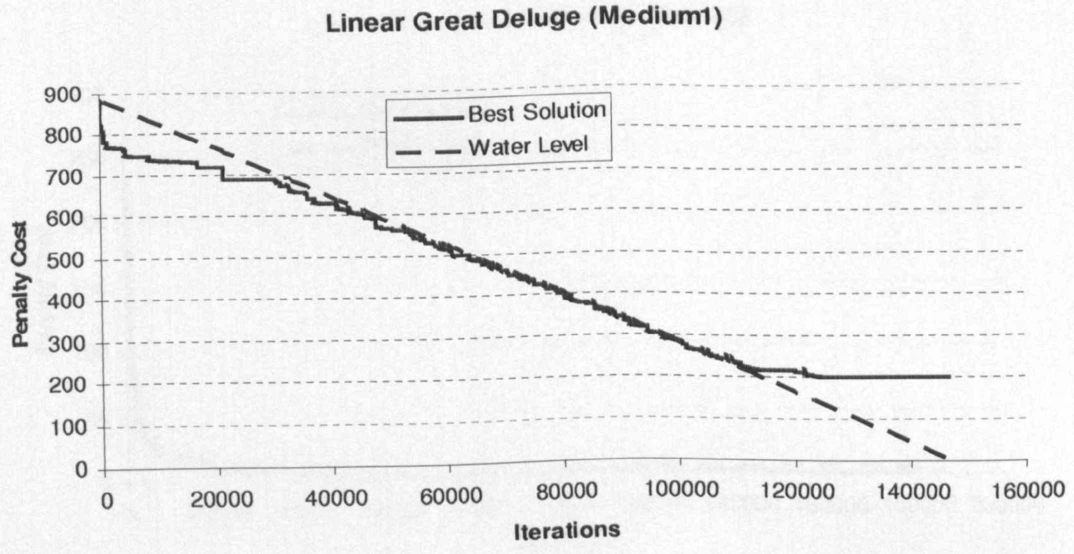


Figure 5.11: Behaviour of Linear Great Deluge on Instance medium1.

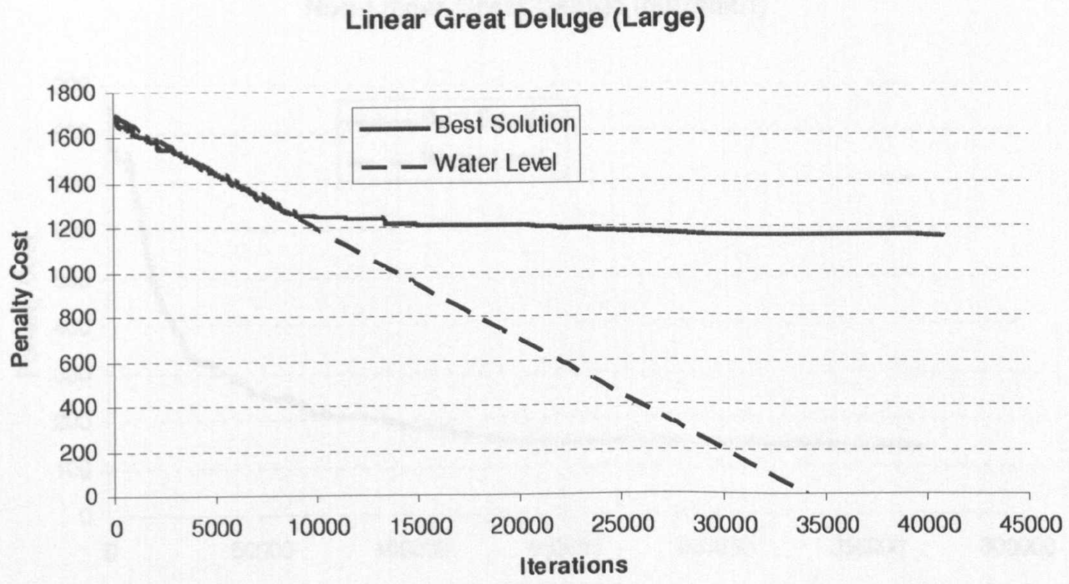


Figure 5.12: Behaviour of Linear Great Deluge on Instance large.

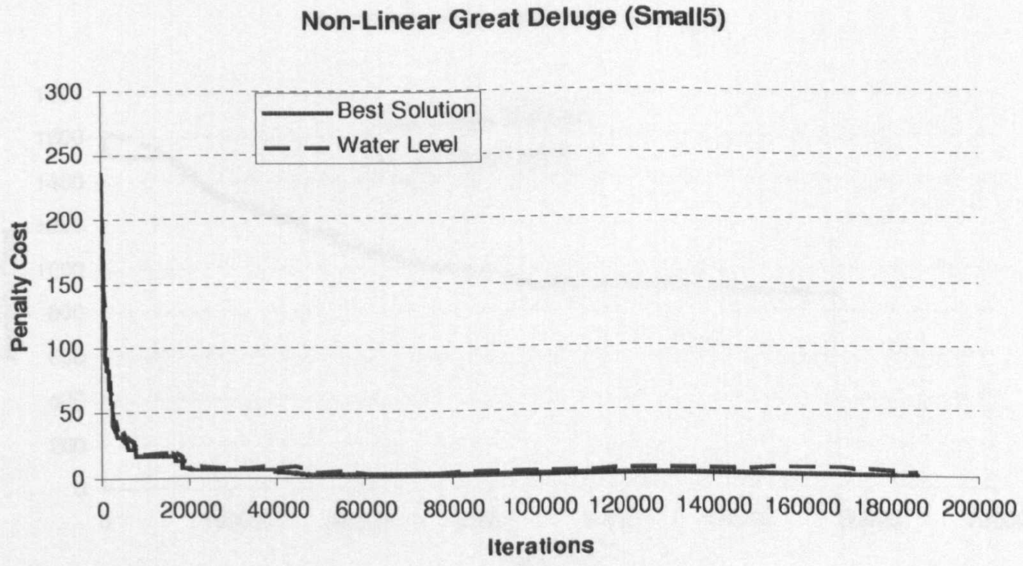


Figure 5.13: Behaviour of Non-Linear Great Deluge on Instance small5.

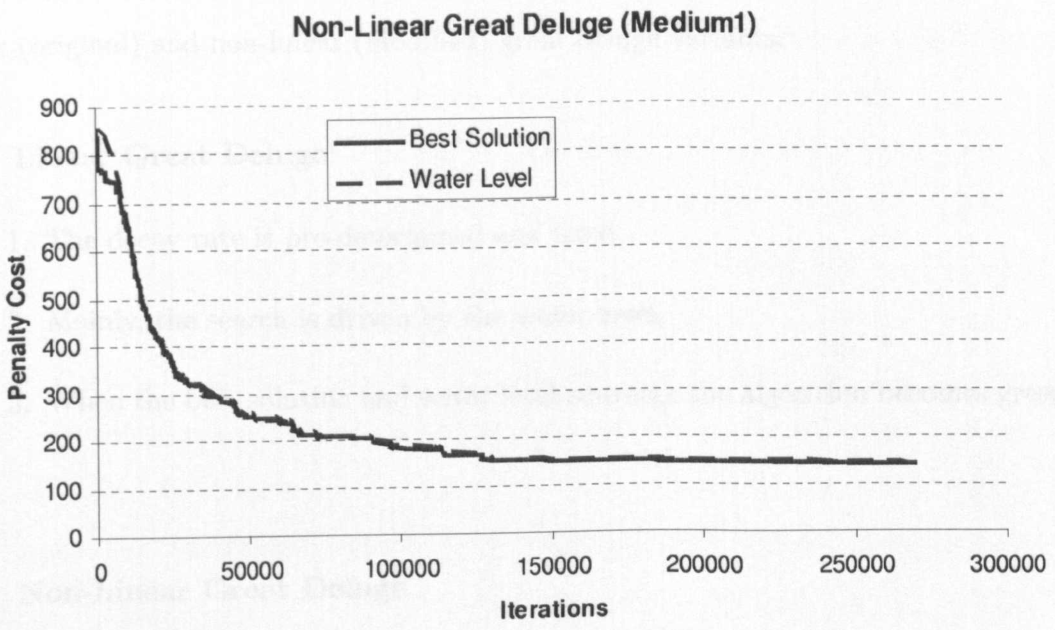


Figure 5.14: Behaviour of Non-Linear Great Deluge on Instance medium1.

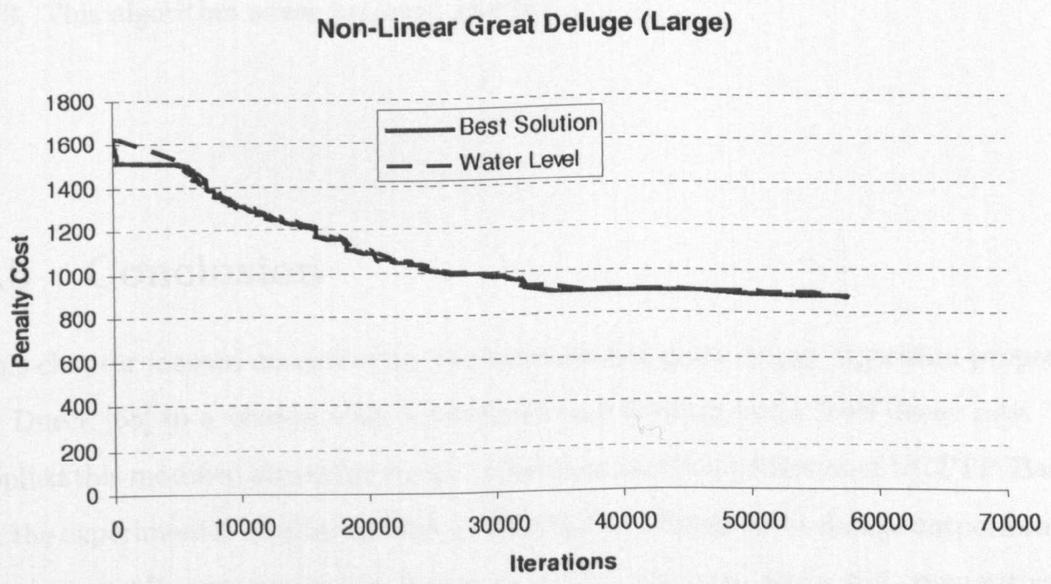


Figure 5.15: Behaviour of Non-Linear Great Deluge on Instance large.

The above observations help us to summarise the key differences between the linear (original) and non-linear (modified) great deluge variants:

Linear Great Deluge

1. The decay rate is pre-determined and fixed.
2. Mainly, the search is driven by the water level.
3. When the best solution and water level converge the algorithm becomes greedy.

Non-Linear Great Deluge

1. The decay rate changes every iteration based on Eq (5.2).
2. Mainly, the water level is driven by the search.
3. This algorithm never becomes greedy.

5.5 Conclusion

This chapter focused on extending the conventional great deluge algorithm proposed by Dueck [68] to a version with a non-linear and floating water level decay rate. We applied this modified algorithm to the 11 Socha et al. [152] instances of UCTTP. Based on the experimental results, we showed that the non-linear great deluge outperformed previous results reported in the literature in four instances while still competitive in the other seven instances. The proposed approach found new best solutions in four of the five medium problem instances. Unfortunately, it seems that this method is not very effective on the small instances. We speculate that this is because the size of the neighbourhood in those instances is not that large to allow the non-linear and floating water level to take its time to diversify and intensify the search repeatedly. Another potential explanation is that the neighbourhood structures might not be effective anymore once the penalty cost reaches a very low value. In order to prove these speculations, future work could run tests as follows: Firstly, when the penalty cost is very low, increase the neighbourhood size by adding additional moves. If increasing the water level could result in an improvement in the solutions or reaching the optimality, then it proves that the neighbourhood size is not large enough to allow the diversification and intensification. Secondly, we record the success rate for each move, counting how many time each move changes or improves the solution. If each moves stop showing any significant success rate, then, we can say that the moves are not effective anymore when the penalty cost reaches a very low value. Another possible test which could be done is we could apply single moves whenever the penalty cost reaches the lowest point, and observe whether any of the moves improve or change the solution.

Chapter 6

An Evolutionary Non-Linear Great Deluge Algorithm

6.1 Introduction

The central aim of this chapter is to investigate an extension of the non-linear great deluge algorithm presented in the previous chapter to an evolutionary version by incorporating a population and a mutation operator, as an approach for the university course timetabling problem. This technique might be seen as a variation of memetic algorithms in particular as presented in [3, 34, 135, 136]. The popularity of evolutionary computation approaches has increased and become an important technique in solving complex combinatorial problems. They are powerful techniques and have been applied to many complex problems e.g. the travelling salesman problem [123, 84, 83], university exam timetabling [30] and university course timetabling problems [33, 136, 135].

The chapter starts with an overview of evolutionary algorithms in Section 6.2 and reviews the different variants of evolutionary approaches found in the literature. Section 6.3 gives the description of the evolutionary non-linear great deluge proposed here for solving the university course timetabling problem. Experiments and results are presented in Section 6.4 and the chapter ends with a conclusion in Section 6.5.

6.2 Overview of Evolutionary Algorithm

Although different variants of evolutionary algorithms exist in the literature, there is a common underlying idea that underpins the basic structure of these algorithms [70]: many evolutionary algorithms are population-based meta-heuristics(here we consider only this type of approaches). These algorithms maintain a population of solutions and conduct the search process by simulating natural selection based on Darwin's theory of survival of the fittest. This means that only strong individual solutions who fulfil the given criteria will survive and participate in the selection for reproduction before being subject to the process of recombination and mutation. Sastry et al. [147] explained various types of recombination and mutation operators. Recombination is an operator which combines two or more individuals from the mating pool in order to create one or more new candidate solutions, whereas mutation is usually designed to add more diverse solutions to increase the chances of exploring large areas of the search space [147]. Different than recombination, mutation is only applied to one candidate solution and produces one new solution. Even though numerous variations of evolutionary algorithms have been proposed in the literature, according to Eiben and Smith [70] they are all followed the same framework as shown in pseudo-code 13 and Figure 6.1 (both pseudo-code and Figure are cited from [70]). That is, evolutionary algorithms consist of a population which evolves and produces a population of individuals (hopefully diverse and with high fitness) in every generation.

Algorithm 13: General Procedure of an Evolutionary Algorithm.

```
Set the initial solution with random candidate solutions
Evaluate each candidate solution
while ( Termination condition is not satisfied) do
    Step 1: Select parents
    Step 2: Recombine pairs of parents
    Step 3: Mutate the resulting offspring
    Step 4: Evaluate the new candidate
    Step 5: Select individuals for the next generation;
```

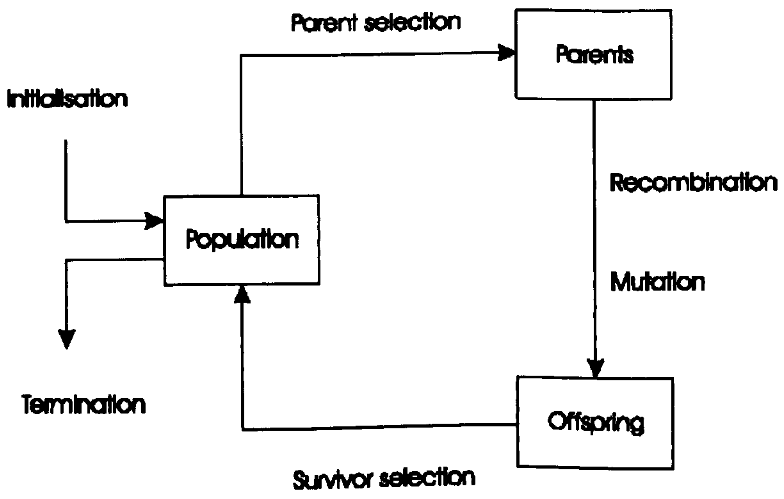


Figure 6.1: Flow Chart Showing the General Procedure of an Evolutionary Algorithm.

Even though crossover is one of the main components in Genetic Algorithms and other Evolutionary Algorithms, Moscato and Norman [123] and Radcliff and Surry [140] have argued whether crossover should the main operator in Genetic Algorithms. It is not an unusual practice that some papers present different implementations of Evolutionary Algorithms in which local search are used as a replacement for crossover. For example, Ackley [8] proposed a genetic hill-climbing approach in which the crossover operator only plays a small role in the algorithm. In addition, according to Bäeck et al. [16] the Evolutionary Strategies community has emphasised on mutation rather than crossover.

6.3 Evolutionary Non-Linear Great Deluge

As discussed in the introduction and suggested in the literature, crossover can be replaced by other operators. For example Ackley [8] used hill-climbing as an operator in place of crossover after arguing that crossover was not effective and played less dominant role. Gorges-Schleuter [82] also provides empirical evidence that local search can be an operator that provides better improvement when added as one of the operators in evolutionary algorithms. Therefore, in this work, we don't use crossover. Instead, we only focus on population management, mutation and local search (non-linear great deluge) as our operators. Using crossover would cause damage to the solutions produced, in the sense that crossover may yield infeasible solutions thus needing a repair operator to bring the solution back into the feasible region.

The motivation behind the introduction of evolutionary operators into our great deluge algorithm come from the interest for striking a good balance between diversification and intensification, which are the main strategic forces in metaheuristic approaches. Therefore, a good search technique must balance these two forces in order to achieve robustness and effectiveness in the search as well as to help the search activity to find optimal or near optimal solutions. *Diversification* is the ability to reach not yet visited regions in the search space and it can be achieved by disturbing some of the solutions using special operators (in our case, we use mutation) when necessary. *Intensification* is about exploiting the current search space regions by using local search (non-linear great deluge in our case) to obtain better quality of solutions.

We now describe the overall hybrid strategy, an extension of our previous NLGD algorithm which produced very good results on the UCTTP as was shown in the previous chapter. Our previous algorithm maintains a single-solution during the search. Here, we extend it to a population-based evolutionary approach by incorporating tournament selection, a mutation operator and a replacement strategy.

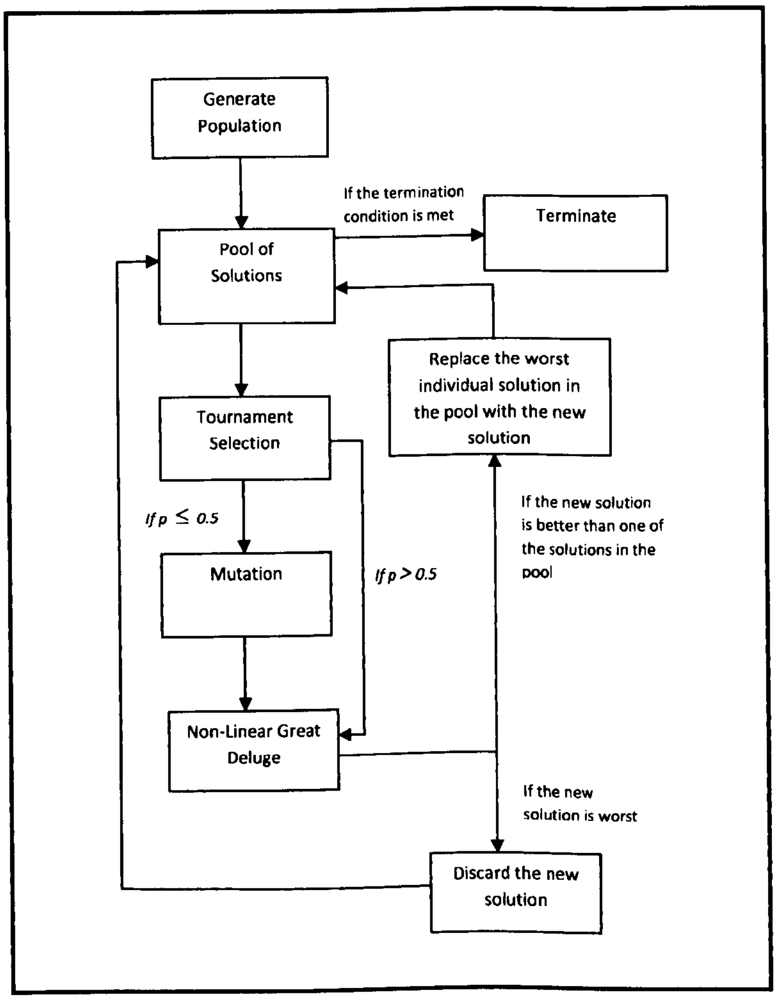
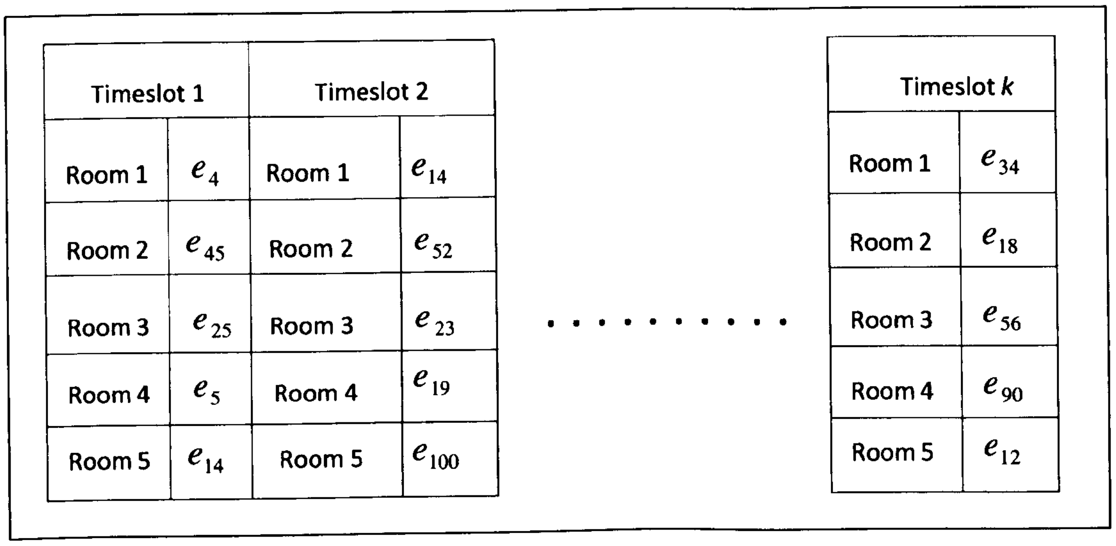

Generate Population
If the termination condition is met
Terminate
Pool of Solutions
Replace the worst individual solution in the pool with the new solution
Tournament Selection
If p ≤ 0.5
If the new solution is better than one of the solutions in the pool
Mutation
If p > 0.5
Non-Linear Great Deluge
If the new solution is worst
Discard the new solution

Figure 6.2: The Evolutionary Non-linear Great Deluge Algorithm.

Figure 6.2 shows the components of the proposed hybrid algorithm. It begins by generating an initial population of size P which becomes the pool of solutions. Then, a number of generations take place and in each of them the algorithm works as follows. First, tournament selection takes place where five individuals are chosen at random from the pool of solutions and the one with the best fitness is selected (x^t). With probability less or equal to 0.5, a mutation operator is applied to x^t while maintaining feasibility and obtaining solution x^m. The probability value was determined by experimentation (If we apply the mutation to often or too low, no much improvement can be found). This is followed by applying the non-linear great deluge algorithm to x^m and obtaining an improved solution x^i. Then, the worst solution in the pool of solutions, x^w (ties broken at random) is identified and if x^i is better than x^w then x^i replaces x^w in the pool of solutions. This hybrid algorithm is then executed for a pre-determined amount of computation time according to the size of the problem instance. Note that this is a steady-state evolutionary approach that uses non-linear great deluge for intensification and a mutation operator for diversification. The following subsections describe each of the algorithm components is more detail.

6.3.1 Solution Representation

Each solution in the population uses a direct representation, consisting of a chromosome with information on what events or courses are assigned into a pair of timeslot-room. In addition, the chromosome is also used to keep information on forbidden assignments for a particular timeslot and room. Figure 6.3 illustrates the direct encoding of an individual solution used in the population. The given example shows that, e_i is an event number i, $i \in \{1, ...n\}$ where n is the number of events that need to be scheduled in available timeslot t, $t \in \{1, k\}$ where k is the number of available timeslots. For example event e_4 is assigned to timeslot 1 in room 1.



Timeslot 1		Timeslot 2	
Room 1	e_4	Room 1	e_{14}
Room 2	e_{45}	Room 2	e_{52}
Room 3	e_{25}	Room 3	e_{23}
Room 4	e_5	Room 4	e_{19}
Room 5	e_{14}	Room 5	e_{100}

. . . .

Timeslot k	
Room 1	e_{34}
Room 2	e_{18}
Room 3	e_{56}
Room 4	e_{90}
Room 5	e_{12}

Figure 6.3: Solution Representation (direct encoding) of a Timetable where Events are Assigned to Pairs timeslot-room.

6.3.2 Initialisation of the Population

We use initialisation heuristic 3 (IH3) to create the initial timetables. That heuristic was described in chapter 4. We would remember that the IH3 initialisation method incorporates two well-known graph colouring heuristics in order to generate feasible solutions, largest degree (LD) and saturation degree (SD). This algorithm attempts to place all events into timeslots while avoiding conflicts. The detail of the initialisation algorithm can be found in Algorithm 9 of chapter four, subsection 4.3.3, in the initialisation heuristic.

6.3.3 The Evolutionary Operator: Mutation

With a probability less or equal to 0.5 ($p \leq 0.5$), the mutation operator is applied to the solution selected from the tournament (x^t). The mutation operator selects at random one out of three types of neighbourhood moves in order to change the solution while maintaining feasibility. These moves are described below.

1. Move M1. Selects one event at random and assigns it to a feasible timeslot and room.

2. Move M2. Selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained.

3. Move M3. Selects three events at random, then it exchanges the position of the events at random and ensuring feasibility is maintained.

6.3.4 Non-linear Great Deluge Search

In this chapter we propose two different stopping conditions for the algorithm. Since non-linear great deluge plays the main role in the proposed evolutionary algorithm, we want to investigate which are the adequate criteria for stopping the non-linear great deluge search before it goes to the next process which is update of the pool of solutions (see Figure 6.2). It should be clear that the non-linear great deluge search promotes intensification in the overall evolutionary method. The detail of the non-linear great deluge can be found in the previous chapter. The use of a population of solutions and the mutation operator promote diversification. Then, by setting the stopping condition for the non-linear great deluge search, we are effectively setting (in a simple manner) the balance between intensification and diversification in the overall evolutionary approach. The first strategy for this balance is to stop the non-linear great deluge after 8000 idle iterations or 30 seconds of computational time, whatever happens first. The second strategy is to stop the non-linear great deluge after 3 seconds of computational time. The first strategy gives more time to intensification while the second strategy attempts to promote diversification more by stopping intensification sooner.

In general, the whole hybrid evolutionary process can be described as follows. After generating the initial set of solutions, this population then becomes the pool of individual solutions (refer to Figure 6.2). After the tournament selection of a solution s, this solution is mutated or not as explained above according to the set probability. Then, the non-linear great deluge search takes place over the solution s. The non-linear great deluge search continues until the given stopping condition, one of the two strategies explained above, is satisfied.

We implemented three variations of the proposed evolutionary algorithm in order to examine the performance of the algorithm when each of the two stopping conditions is used and also when the mutation operator is removed. The three algorithm variants are: Evolutionary Non-linear Great Deluge Without Mutation (ENLGD-M), Evolutionary Non-linear Great Deluge using stopping condition 1 (ENLGD-1) and Evolutionary Non-linear Great Deluge using stopping condition 2 (ENLGD-2). Both ENLGD-1 and ENLGD-2 have the mutation operator incorporated. The aim of examining these algorithm variants is to assess the robustness of the proposed evolutionary algorithm with different settings. By robustness we mean the reliability of the algorithm to produce high-quality of solutions under different settings. Table 6.1 shows the various parameter settings for the three algorithm variants examined here.

Table 6.1: Parameter Settings for the Three Variants of the Proposed Evolutionary Non-linear Great Deluge Algorithm.

Parameter	ENLGD-M	ENLGD-1	ENLGD-2
Mutation	no mutation applied	0.5	0.5
Stopping condition	idle 8000 iterations or 30 seconds	idle 8000 iterations or 30 seconds	every 3 seconds of computation time
Replacement	Steady state	Steady state	Steady state
Stopping time for whole search process	small (2600 seconds) medium (7200 seconds) large (10000 seconds)	small (2600 seconds) medium(7200 seconds) large (10000 seconds)	small (2600 seconds) medium (7200 seconds) large (10000 seconds)

6.4 Experiments and Results

We now evaluate the performance of the proposed evolutionary algorithm. For each problem size, a fixed computation time ($time_{max}$) in seconds was used as the stopping condition for the overall algorithm: 2600 for small problems, 7200 for medium problems and 10000 for the large problem. This fixed computation time is for the whole process including the construction of the initial population. We executed the proposed evolutionary algorithm 20 times for each problem instance.

Table 6.2 shows the experimental results on the Socha et al. instances of the university course timetabling problems of the three algorithm variants described above, i.e ENLGD-M, ENLGD-1 and ENLGD-2. The Table shows the best and the average results obtained by each method. For each dataset, the best results are indicated in bold. As shown in Table 6.2, the hybrid evolutionary algorithm described here (ENLGD-1 and ENLGD-2) clearly outperforms our previous single-solution algorithm (NLGD). The results also show that both ENLGD-2 and ENGLD-1 produce better results when compared to ENLGD-M. This means that the tailored mutation operator makes a significant impact to the good performance of ENLGD. Besides that, the results also show that ENLGD-2 outperforms ENLGD-1 and ENLGD-M. This means that balancing the intensification and diversification helps the Evolutionary Non-linear Great Deluge (ENLGD) approach to better explore the search space rather than run the intensification for longer which makes the local search to converge earlier (as in the ENLGD-1 case). The intensification phase is mainly carried out by the non-linear great deluge procedure. In great deluge phase, it intensively looks for quality solutions. Notice that the main task of non-linear great deluge is the search intensification, however this approach also able to diversify the search procedure by accepting the worst solution as long as the new found solution is less than or equal to the water level and by allowing the water level to go up whenever the objective function and the water level are about to converge. The evolutionary operator (mutation, selection and replacement) objective, on the other hand is mainly associated with diversification search. Similar to non-linear great deluge, evolutionary algorithm is also able to perform the intensification by mean of replacing the individual solution in the population pool in every generation. In results of balancing the intensification and diversification, now the proposed ENLGD able to satisfying all the soft constraints at least one solution, for all small instances that previously unable to reach by NLGD. Obvious improvement can be seen also in the medium and large instances, where ENLGD-2 managed to improve the penalty values obtained by NLGD.

Table 6.2: Comparison of NLGD, ENLGD-M, ENLGD-1 and ENLGD-2 on the Socha et al. UCTTP Instances.

IN	NLGD		ENLGD-M		ENLGD-1		ENLGD-2	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
S1	3	3.6	**0**	1.55	**0**	0.95	**0**	0.7
S2	4	4.85	**0**	2.2	**0**	1.45	**0**	0.3
S3	6	6.85	1	2.7	**0**	1.3	**0**	1.05
S4	6	6.85	**0**	1.7	**0**	1.35	**0**	1.25
S5	**0**	1.75	**0**	**0**	**0**	**0**	**0**	**0**
M1	140	160.75	144	176.65	125	140	**59**	84.8
M2	130	156	140	162	123	149.1	**51**	93.8
M3	189	212.1	182	204.8	178	199.3	**75**	121.05
M4	112	138.3	135	164.6	116	130.2	**48**	72.8
M5	141	192.6	123	173.15	129	168.6	**65**	110.2
L	876	974.3	970	1026	821	946.1	**703**	819.2

Further investigation was also carried out to inspect the overall performance of the proposed evolutionary algorithm. Figures 6.4, 6.5 and 6.6 summarise the performance of the various versions of the algorithm together with our previous single-solution approach. The x-axis corresponds to the instance type while the y-axis corresponds to the penalty cost. Figure 6.4 shows the strong performance of ENLGD-2 on the medium and large instances, while also obtaining optimal solutions with the same quality as the other algorithms for small instances. In addition, Figure 6.5 and Figure 6.6 show details of the results achieved by the proposed algorithms. Both figures show that according to the average results, ENLGD-2 outperformed the other algorithms.

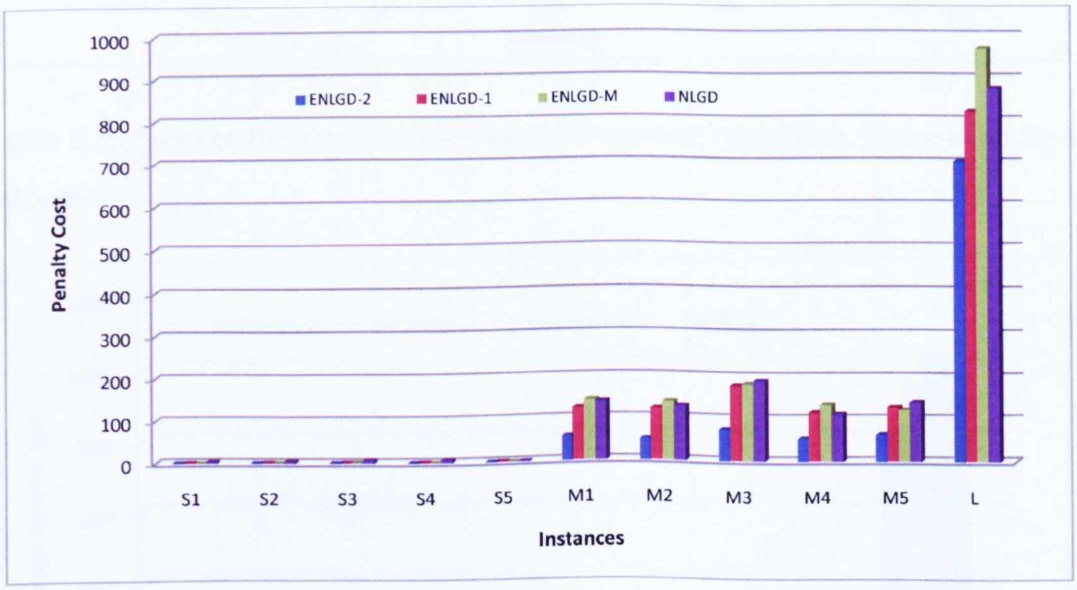


Figure 6.4: Best Results Obtained by the Proposed Algorithm Variants.

Overall, this experimental evidence shows that by combining some key evolutionary components with our previous single-solution non-linear great deluge approach, we have been able to produce a hybrid evolutionary approach that is still quite simple

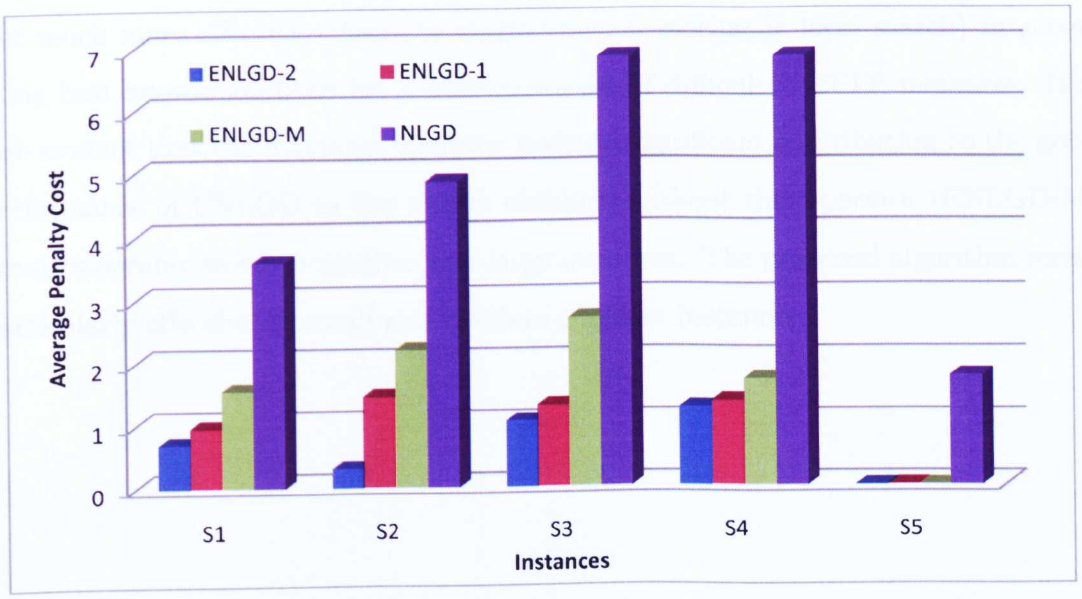


Figure 6.5: Average Results Obtained by the Proposed Algorithm Variants on Small Instances.

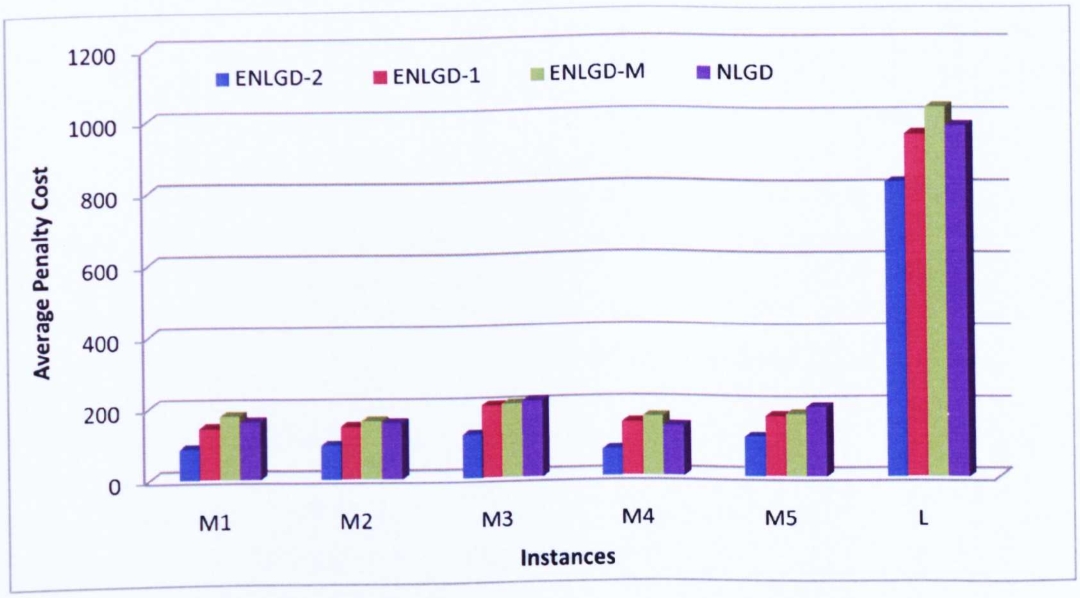


Figure 6.6: Average Results Obtained by the Proposed Algorithm Variants on Medium and Large Instances.

but much more effective (than the single-solution stochastic local search) in generating best known solutions for a well-known set of difficult UCTTP instances. It is also evident that the mutation operator makes a significant contribution to the good performance of ENLGD as the results obtained without this operator (ENLGD-M) are considerably worse in medium and large instances. The proposed algorithm seems particularly effective on small and medium problem instances.

Table 6.3: Comparison of results obtained by the Evolutionary Non-Linear Great Deluge (ENLGD) proposed in this chapter against the best known results from the literature for the 11 Socha et al. UCTTP instances.

	(ENLGD-2)	(NLGD)	RRLS	MMAS	GALS	RIICN	GBHH	CFHH	VNS-T	HEA	FMHO	EGD
S1	**0**	3	8	1	**0**	**0**	6	1	**0**	**0**	10	**0**
S2	**0**	4	11	3	**0**	**0**	7	2	**0**	**0**	9	**0**
S3	**0**	6	8	1	**0**	**0**	3	**0**	**0**	**0**	7	**0**
S4	**0**	6	7	1	**0**	**0**	3	1	**0**	**0**	17	**0**
S5	**0**	**0**	5	**0**	**0**	**0**	4	**0**	**0**	**0**	7	**0**
M1	**59**	140	199	195	175	242	372	146	317	221	243	80
M2	**51**	130	202.5	184	197	161	419	173	313	147	325	105
M3	**75**	189	77.5%Inf	248	216	265	359	267	357	246	249	139
M4	**48**	112	177.5	164.5	149	181	348	169	247	165	285	88
M5	**65**	141	100%Inf	219.5	190	151	171	303	292	130	132	88
L	703	876	100%Inf	851.5	912	-	1068	80%Inf	-	**529**	1138	730

ENLGD-2 is Evolutionary Non-Linear Great Deluge with stopping strategy 2.
NLGD is Non-Linear Great Deluge [100]. NLGD is Non-Linear Great Deluge [100].
RRLS is the Local Search and Ant System in [153]
MMAS is the MAX-MIN Ant System in [152]
GALS is Genetic algorithm and local search by Abdullah and Turabieh [7].
RIICN is Randomised iterative improvement algorithm by Abdullah et al. [3].
GBHH is Graph-based Hyper-heuristic by Burke et al. [29].
CFHH is the Choice Function Hyper-heuristic in [27]
VSN-T is Variable neighbourhood search with tabu by Abdullah et al. [5].
HEA is Hybrid evolutionary approach by Abdullah et al. [4].
FMHO is fuzzy multiple heuristic ordering [15]
EGD is Extended Great Deluge [116]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L represents the large problem instance

Table 6.3 compares the results obtained by the approach proposed in this chapter against the other available approaches in the literature. The term x%Inf in Table 5.1 illustrates a percentage of runs that were unable to achieve feasibility. The figures in bold indicate the best results. Results in the Table indicate that some of the algorithms were unable to produce feasible solutions. However, in contrast, our approach was able to achieve feasible solutions. It can be seen that the proposed hybrid evolutionary approach (ENLGD-2) matches the best known solution quality for all small problem instances. For medium instances, ENLGD-2 was able to achieve better quality solutions when compared against all other methods listed in Table 5.1. More interestingly ENLGD-2 is able to produce high quality solutions and outperformed the best known results obtained by other algorithms as reported in the literature. Only on the case of the large problem instance, we see that our algorithm does not match the best known result reported by Abdullah at al. [4]. However, our result is still comparable to other results reported in the literature.

6.4.1 Statistical Analysis

The main goal of this section is to examine using statistical analysis, the performance of our four different methods, and to inspect whether extending the non-linear great deluge to a hybrid evolutionary approach helps to produce better solutions. In this section we want to inspect whether different strategies contribute to the improvement on the solution quality. In order to make the comparison more clearly, three questions have been set as follows:

1. Does mutation help to improve the solution quality?
2. Can Non-Linear Great Deluge become an effective operator in EA replacing the typical crossover?

3. Since the NLGD is playing the main role in this search activity, it is worth comparing the different strategies in order to measure the effect on each search?

Before we proceed to the analysis, it is essential to verify the compatibility of the models with the sample data. There are important hypotheses that need to be verified: normality, independency and homogeneity of the sample data. After running the descriptive analysis, we found that our sample data fulfils the hypothesis requirements. For that reason variance analysis (ANOVA) is considered suitable for the sample data hypothesis ensuring the validity of the experiment. ANOVA is one of the existing statistical models used to test significant differences between means and this tool is very useful to make comparison when dealing three or more means.

Even though conclusions can usually be made based on the best and average results obtained by each algorithms, those conclusions and analysis might be premature. Therefore, ANOVA was used to determine whether there is a significant difference in performance among ENLGD-2, ENLGD-1, ENLGD-M and NLGD. The analysis showed that there are statistically significant differences among the proposed algorithms with the p-value very close to zero as shown in Figure 6.7. The p-value stands for probability ranging from zero to one. Therefore, the p-value is used to measure the different in population means and used as an evidence to reject or accept the null hypothesis. In our case the null hypothesis H_0 is that there are no significant differences in performance between the algorithms. Therefore, if we reject H_0 then we accept that there are significant differences in performance among the algorithms.

ANOVA analysis only reveals that at least one mean is significant difference among the algorithms. However, we still do not know whether the performance of the algorithm is totally different to each other or perhaps only one algorithm is different. Thus, Post-Hoc Test was carried out to test the differences between pairs of algorithms. Post-Hoc test is also known as post-hoc comparison tests. In practise this tool is used in the second stage of ANOVA. The main aim of this analysis is to evaluate whether there exists a significant difference among groups in respect to the mean values. Tables 6.6, 6.7 and 6.8 clearly show that there are significant differences between the algorithms as described below:

- For small instances, the p-value are less than the confidence level at 0.05 for every pair of algorithms (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M), (ENLGD-2, NLGD), (ENLGD-1, ENLGD-M) and (ENLGD-M, NLGD).
- For medium instances there are significant differences in performance between (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M), (ENLGD-2, NLGD), (ENLGD-1, ENLGD-M) where the p-value are less than the confidence level at 0.05. However, there is no significant difference in performance between NLGD and ENLGD-M, where the Post-Hoc analysis shows that the p-value is 0.659 (greater than 0.05).
- Finally for the large instance, there are significant differences in performance between (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M) (ENLGD-2, NLGD) and (ENLGD-1, ENLGD-M) where the p-value for the respective pairs are less than 0.05 significance level. Interestingly, the Post-Hoc test shows that there is no significant difference in performance between (ENLGD-1, NLGD) and (NLGD-M, NLGD) where the p-value are 0.697 and 0.063 respectively, where both p-value is greater than significant level at 0.05.

The Post-Hoc analysis clearly showed that all four algorithms perform differently. However, at this stage we still do now know which algorithm is actually outperforming the others across the eleven instances. Thus, to evaluate this, we plot the mean of each algorithm with Least Significant Difference (LSD) intervals at 95% confidence level for the different algorithms as shown in Figures 6.8 to 6.10. LSD is used to measure the significant differences between group means in ANOVA. From the mean plot, we see that ENLGD-2 outperforms the other algorithms followed by ENLGD-1, NLGD and the worst algorithm is ENLGD-M.

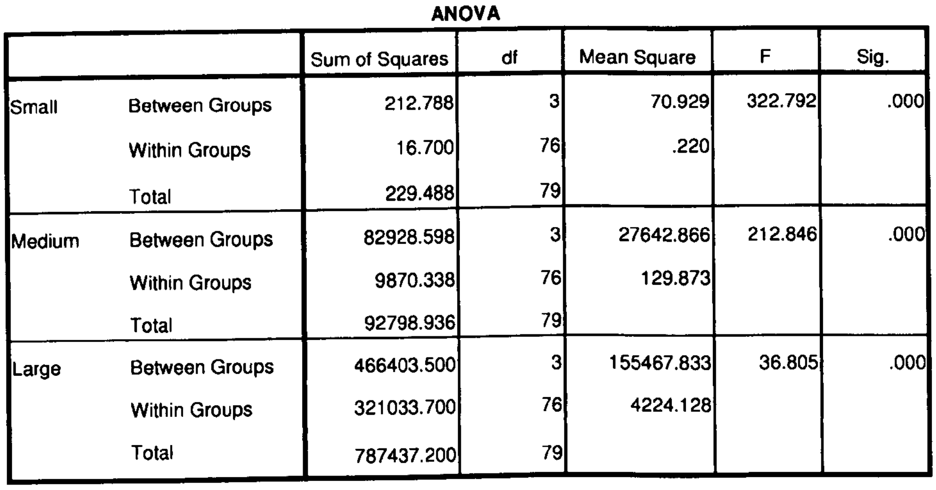
Figure 6.8, Figure 6.9 and Figure 6.10 present the means plot of each algorithm, for the specific instances. Figure 6.8 shows that there are three homogenous groups for small instances (ENGLD-1, ENLGD-2), (ENLGD-M) and (NLGD). The best algorithm is ENGLD-2 followed by ENLGD-1 and ENLGD-M, the worst algorithm is NLGD. In medium instances we also found three homogenous groups as shown in Figure 6.9 and they are (ENGLD-1), (ENLGD-2) and (ENLGD-M, NLGD). The algorithm that performs well in medium instances is ENGLD-2 followed by ENLGD-1 and two algorithms which perform slightly worst are ENLGD-M and NLGD. Finally, for the large instance, we found that there are three homogenous group (ENGLD-1, NLGD), (ENLGD-2) and ENLGD-M. In the large instance case, we found that ENLGD-2 outperforms the other algorithms and ENLGD-M is the worst. In conclusion, considering the overall performance, ENLGD-2 is the best algorithm followed by ENLG-1, NLGD and the worst algorithm is ENLGD-M (mutation operator removed).

	ENLGD-2			ENLGD-1		
Run	Small	medium	Large	Small	medium	Large
1	0.8	95.6	703	0.2	159	821
2	0.4	85.8	927	1.4	165.4	940
3	0.4	95.4	835	1	167.8	963
4	0.4	93.6	968	1.2	163.6	879
5	0.4	108.6	895	1	165.2	954
6	0.4	99.8	730	1.2	162	952
7	0.2	81.2	782	0.8	146.4	938
8	0.4	91.6	711	1.2	148.2	976
9	0.8	110.4	777	1	147.4	1018
10	1	96.4	838	0.6	144.4	1020
11	0.4	96.6	808	1	171.6	968
12	1	98.4	944	1.6	178	904
13	0.8	91.2	870	1.2	158.8	958
14	0.4	96.4	807	0.4	159.2	876
15	0.8	83.6	849	1.8	165	876
16	1.2	90.6	713	1.6	156	970
17	0.4	117.8	852	1.2	169.6	918
18	0.6	102.2	795	0.6	172.8	1003
19	1.6	106	779	0.6	148.2	1031
20	0.8	89.4	801	0.6	175.2	1072

Table 6.4: Average Penalty Cost of ENLGD-2 and ENLGD-1 Across the 11 Socha et al. Instances.

Run	ENLGD-M			NLGD		
	Small	medium	Large	Small	medium	Large
1	2	186.2	1023	3.8	142.4	966
2	2	176.6	1070	4.8	165	1070
3	1.4	191.6	998	6	165.6	876
4	2	177.6	1142	5.2	162.2	935
5	1.4	205.8	1114	5	165.2	971
6	1	189.8	984	4.6	166.8	942
7	1	184	923	5	165.4	895
8	1.8	179.6	970	5.2	156.8	976
9	2	166.4	1082	5.4	160.4	986
10	1.4	185	1023	5.4	172.8	1005
11	1.8	192.2	1023	3.8	185	966
12	2	159.2	1070	4	171.6	1070
13	2	178.8	998	4.2	177	935
14	2.2	156.4	1142	4.2	181	1024
15	1.6	167.6	984	4	172.4	942
16	2	166.6	923	5	188.4	958
17	1.6	168.6	970	4.2	179.6	978
18	0.8	168.8	1082	5.4	182.6	1005
19	1.4	156.6	1023	5.4	196	1078
20	1.2	166.8	982	5	183.8	907

Table 6.5: Average Penalty Cost of ENLGD-M and NLGD Across the 11 Socha et al. Instances.



ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Small	Between Groups	212.788	3	70.929	322.792	.000
	Within Groups	16.700	76	.220		
	Total	229.488	79			
Medium	Between Groups	82928.598	3	27642.866	212.846	.000
	Within Groups	9870.338	76	129.873		
	Total	92798.936	79			
Large	Between Groups	466403.500	3	155467.833	36.805	.000
	Within Groups	321033.700	76	4224.128		
	Total	787437.200	79			

Figure 6.7: ANOVA Results.

	ENLGD-2	ENGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.041	0.000	0.000
ENLGD-1	0.041	-	0.000	0.000
NLGD-M	0.000	0.000	-	0.000
NLGD	0.000	0.000	0.000	-

Table 6.6: Post Hoc Tests - Small Instances

	ENLGD-2	ENGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.000	0.000	0.000
ENLGD-1	0.000	-	0.001	0.019
NLGD-M	0.000	0.000	-	0.649
NLGD	0.000	0.019	0.649	-

Table 6.7: Post Hoc Tests - Medium Instances

	ENLGD-2	ENGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.000	0.000	0.000
ENLGD-1	0.000	-	0.003	0.697
NLGD-M	0.000	0.003	-	0.063
NLGD	0.000	0.697	0.063	-

Table 6.8: Post Hoc Tests - Large Instance

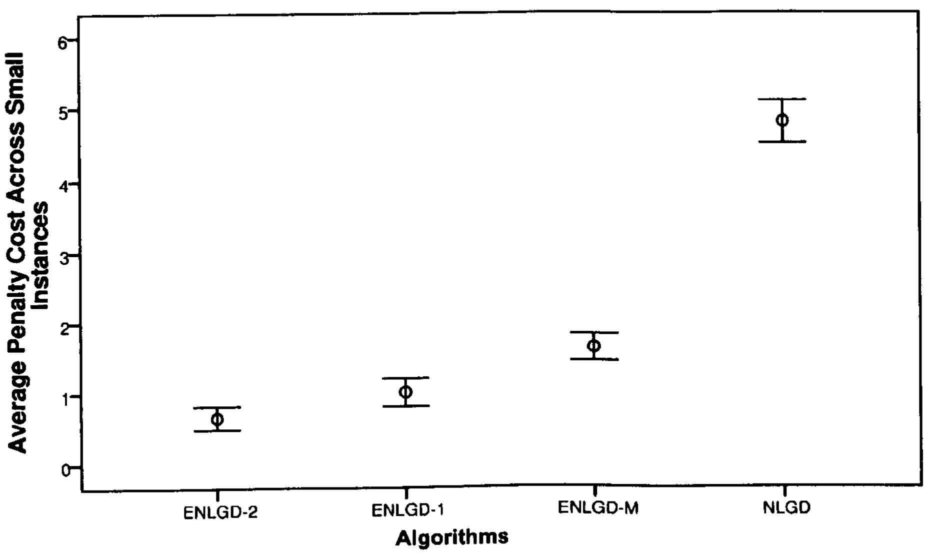


Figure 6.8: Mean Plot and LSD Intervals (Small Instances).

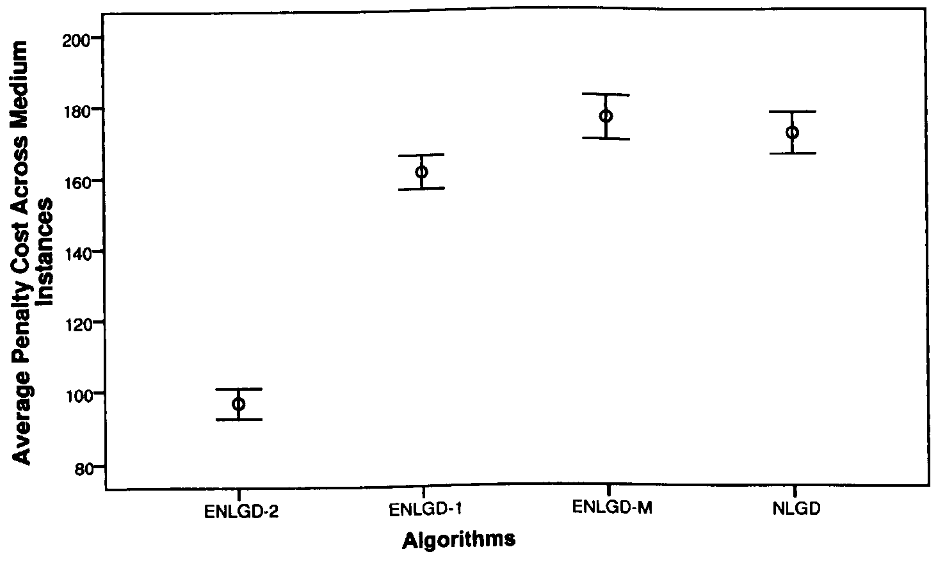


Figure 6.9: Mean Plot and LSD Intervals (Medium Instances).

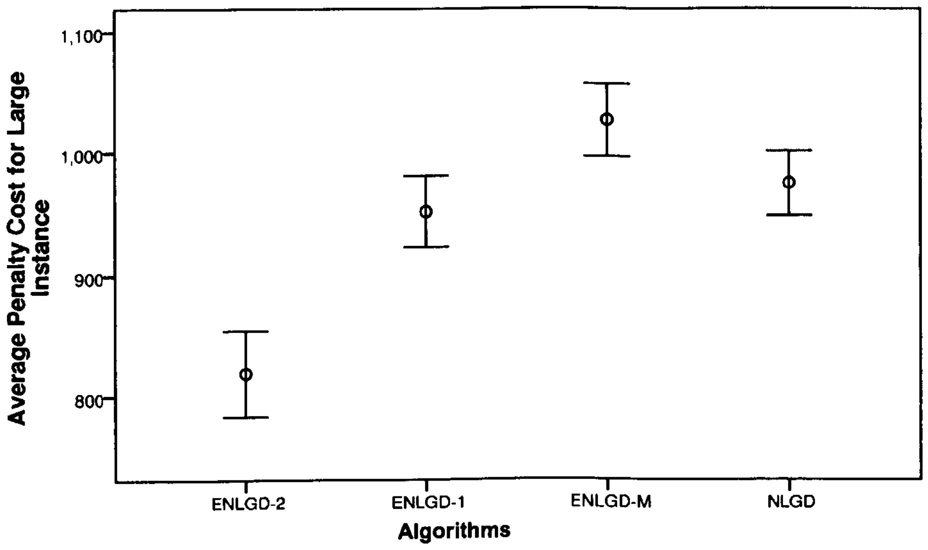


Figure 6.10: Mean Plot and LSD Intervals (Large Instance).

The statistical analysis presented here suggests that each algorithm has different behaviour and performs differently across all 11 Socha et al. instances. This analysis also showed that ENLGD-2 outperforms the three other algorithms across all instances. It is also evident that the mutation operator makes a significant contribution to the good performance of ENLGD-2 as the results obtained by ENLGD-M are considerably worse. Moreover, the strategy applied in ENLGD-2 to balance intensification and diversification proves to be a good strategy as it managed to further improve the solution quality compared to ENLGD-1. As a conclusion, the proposed hybrid evolutionary approach matches the best known solution quality for almost all small problem instances and improves the best known results for most all medium instances. Only on the case of the large problem instance, we see that our algorithm does not match the best known result published in the literature, however the result is still competitive when compared to the results obtained by other algorithms as reported in the literature.

6.5 Conclusions

The overall endeavour of this chapter was to extend our previous approach, a non-linear great deluge algorithm, towards an evolutionary variant by incorporating some key operators like a population of solutions, tournament selection, a mutation operator and a steady-state replacement strategy. The performances of the various versions of evolutionary non-linear great deluge were compared along with the single-solution NLGD algorithm. Preliminary comparisons illustrate that ENLGD-2 outperforms the results produced by other versions of ENLGD and NLGD algorithms. The results from our experiments also provide evidence that our hybrid evolutionary algorithm is capable of producing best known solutions for a number of the test instances used here. Obtaining the best timetables (with penalty equal to zero) for the medium and large instances is still a challenge. However, when compared to the results obtained by ENLGD-2 to the best know results reported in the literature, obviously, ENLGD-2 outperform all the results of medium instances and produced comparable for large instance.

Chapter 7

Non-Linear Great Deluge with Modified Choice Function

7.1 Introduction

This chapter presents a non-linear great deluge hyper-heuristic incorporating a modified choice function mechanism for the selection of low-level heuristics and non-linear great deluge acceptance criterion. The proposed hyper-heuristic only deals with complete feasible solutions. The learning mechanism allows the hyper-heuristic to select low-level heuristics based on their previous performance. The learning mechanism provides a measure of performance for each low-level heuristic and ranks them from the highest to the lowest. Therefore, the higher ranked the low-level heuristic is, the highest the probability that the low-level heuristic will be selected next. The low-level heuristics are local search operators which operate in the solution space. We propose two learning mechanisms: learning with static memory length and learning with random change in learning rate. In static memory length, the reward and punishment are constant and the reward of the low-level heuristics is normalised at every predefined learning period. In random change in learning rate, a low-level heuristic is rewarded and penalised when it is selected to change the current solution and the learning rate is changed automatically at random at every predefined learning period. The performance of the proposed hyper-heuristics is assessed using the standard course timetabling benchmark instances of Socha et al. [152] and our results are compared to results published in the literature.

As discussed in the literature review of this thesis, the UCTTP has been tackled using a wide range of exact methods, heuristics and meta-heuristics. In recent years, the term *hyper-heuristic* has emerged for referring to methods that use (meta-) heuristics to choose (meta-) heuristics [25]. Then, a hyper-heuristic is a process which, given a particular problem instance and a number of *low-level heuristics*, manages the selection and acceptance of the low-level heuristic to apply at any given time, until a stopping condition is met. Low-level heuristics are simple local search operators or domain dependent heuristics. Typically, a hyper-heuristic is meant to search in the space of heuristics instead of searching in the solution space directly. One of the main challenges in designing a hyper-heuristic method is to manage the low-level heuristics with minimum parameter tuning.

The rest of this chapter is organised as follows. Section 7.2 describes the Non-Linear Great Deluge With Learning Mechanism Framework while Section 7.3 presents and discusses the experimental results. Conclusions and future research are presented in Section 7.4.

7.2 The Non-linear Great Deluge Hyper-heuristic

In this chapter, we use our non-linear great deluge algorithm (NLGD) [102] developed in chapter 5 as an acceptance criterion and incorporate learning mechanism called modified choice function to select the low-level heuristics to apply at each step of the search process. That is, while in a NLGD meta-heuristic *candidate solutions* are accepted or not based on the great deluge criterion, in the proposed Non-Linear Great Deluge Hyper-heuristic (NLGDHH) it is *candidate low-level heuristics* which are accepted or not, i.e. the method operates in the heuristic search space.

Figure 7.1 illustrates the proposed hyper-heuristic approach in which the low-level heuristics are local search operators that explore the solution space while the modified choice function and the NLGD acceptance criterion explore the heuristic space. We use the non-linear great deluge criterion because of its simplicity and less dependent nature upon parameter tuning compared to simulated annealing [22, 102]. The low-level heuristics implemented in this work are the same neighbourhood moves uses earlier in this thesis and are described again below. These heuristics are based on random search but always ensuring the satisfaction of hard constraints.

H1: selects 1 event at random and assigns it to a feasible pair timeslot-room also selected at random.
H2: selects 2 events at random and swaps their timeslot-room while ensuring feasibility.
H3: selects 3 events at random and exchanges timeslot-room at random while ensuring feasibility.

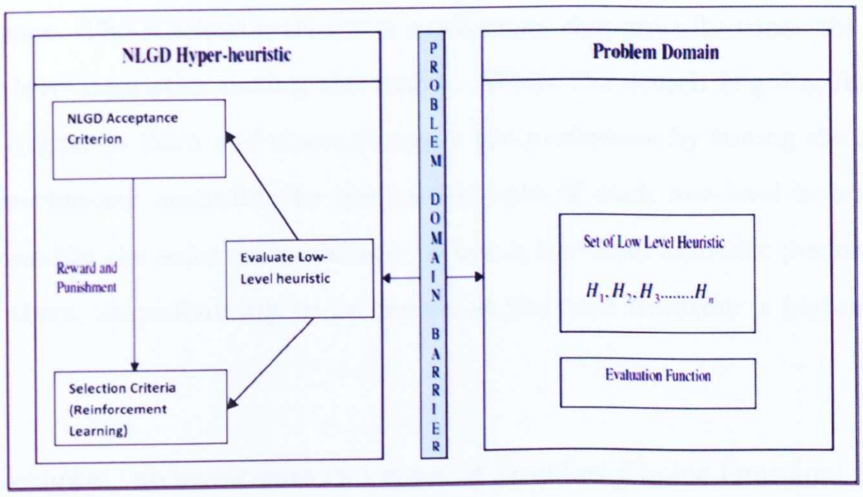

NLGD Hyper-heuristic
NLGD Acceptance Criterion
Reward and Punishment
Evaluate Low-Level heuristic
Selection Criteria (Reinforcement Learning)
PROBLEM DOMAIN BARRIER
Problem Domain
Set of Low Level Heuristic
$H_1, H_2, H_3 .... H_n$
Evaluation Function

Figure 7.1: Non-Linear Great Deluge Hyper-heuristic Approach.

7.2.1 Learning Mechanism

A choice function proposed by Soubeiga [154] is mainly used to rank low-level heuristics. The main task of the choice function is to select low-level heuristic at each decision point based on its previous performance and the area of the search space currently under search. Three main features of choice function are: The choice function assigns each heuristic a sum of weight that indicate the recent improvement (change in the objective function), new improvement generate by a consecutive pair of heuristics and the last time a heuristic was called. Different than the standard choice function proposed by Soubeiga [154], we employed a learning mechanism which adapted from Bai et al. [17] and no consecutive pair of heuristic will take into account (only single heuristic performance is counted). This learning mechanism is used to guide the selection of low-level heuristics during the search. Thus, a low-level heuristic is more likely to be selected at each iteration if it has the highest number of success in the past. We adopt the stochastic ranking of the low-level heuristics and the stochastic selection mechanism to select the low-level heuristics. Initially, the low-level heuristics are treated equally and have the same probability to be selected to change the solution space. The stochastic selection mechanism dynamically tunes the priorities of the low-level heuristics during the search. While the search is going further, the algorithm begins to learn and starts to apply the preference by tuning the stochastic selection mechanism according to the success rate of each low-level heuristic when chosen to modify the solution landscape. When a low-level heuristic performs better than the others, its probability to be chosen at the next iteration is higher than the others.

In this chapter, we investigate two types of Modified Choice Function: *MCF with static memory length* and *MFC with random change in learning rate.* In static memory length, the reward and punishment are constant and the reward obtained by the low-level heuristics is normalised at every predefined learning period. Whereas in random changer learning rate, a low-level heuristic is rewarded and penalised when it is chosen to change the current solution and the learning rate is changed automatically at random at every predefined learning period. We propose two Non-Linear Great Deluge Hyper-heuristic algorithms using learning with static memory and learning with random changer in learning rate namely NLGDHH-SM (with static memory length) and NLGDHH-RCLR (with random changer in learning rate). The detail of these two approaches is given below in subsections 7.2.1.1 and 7.2.1.2.

7.2.1.1 MCF with Static Memory Length

In each iteration, a low-level heuristic i is selected with probability p_i given by Eq. (7.1) where n is the number of heuristics and w_i is the weight assigned to each heuristic.

$$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{7.1}$$

Initially, every weight is set to $w_i = 0.01$. At each iteration, the algorithm starts to reward or punish the heuristics according to their performance. When the chosen heuristic improves the current solution, a reward of 1 point is given to the heuristic. If the heuristic does not improve the solution, the punishment is to award no points. This amount of reward/punishment never changes. However, the algorithm updates the set of weights w_i in every learning period (lp) given by $lp = max(K/500, n)$, where K is the total number of feasible moves explored.

We use the following counters to track the performance of each low-level heuristic: $Ctotal_i$, is the number of times that low-level heuristic i is called; $Cnew_i$ is the number of times that low-level heuristic i generates solutions with different fitness value (objective function); and $Caccept_i$ is the number of times that low-level heuristic i meets the non-linear great deluge acceptance criterion. Each heuristic weight w_i is updated at every learning period lp and normalised by the ratio $Caccept_i/Ctotal_i$ when $range > 1$ and by $Cnew_i/Ctotal_i$ when $range < 1$. At every learning period lp if the $range < 1$ (range is the difference between water level and current penalty cost), the water level then increases to $B = B + rand[B_{min}, B_{max}]$. We call this mechanism *surge B* (we allow the water level to go up to explore different region of search). We set B_{min} equal to 1 and B_{max} equal to 4 regardless to the size of the dataset. Note that the water level can increase due to the *floating B* (continuous) mechanism or the *surge B* (every lp feasible moves) mechanism.

7.2.1.2 MCF with Randomly Change Learning Rate

In each iteration, a low-level heuristic i is selected with probability p_i given by Eq. (7.2) where n is the number of heuristics, w_i is the weight assigned to each heuristic and $w_{min} = min\{0, w_i\}$.

$$p_i = \frac{w_i + w_{min}}{\sum_{i=1}^{n} w_i + w_{min}} \tag{7.2}$$

Initially, every weight is set to $w_i = 0.01$ as before, however, each w_i is updated every time the algorithm performs a feasible move. When the selected heuristic improves the current solution, the heuristic is rewarded, otherwise the heuristic is punished. The value $\Re_{ij}$ of reward/punishment applied to heuristic i at iteration j is as given below where $r = 1$, $\Im = 0.1$ and Δ is the difference between the best solution (lowest penalty) so far and the current solution (current penalty).

$$\Re_{ij} = \begin{cases} r & \text{if } \Delta < 0 \\ -r & \text{if } \Delta > 0 \\ \Im & \text{if } \Delta = 0 \text{ and new solution} \\ -\Im & \text{if } \Delta = 0 \text{ and no new solution} \\ 0 & \text{if not selected} \end{cases}$$

Then, at each iteration h, each weight w_i is calculated using Eq.(7.3) where σ is learning rate that scales the weight of the reward that obtained at every iteration during the search history.

$$w_{ih} = \sum_{j=1}^{h} \sigma^j \Re_{ij} \tag{7.3}$$

In every learning period lp, the algorithm updates σ with a random value in $(0.5, 1.0]$. Here, we also set $lp = max(K/500, n)$ as before. At every learning period lp and if $range < 1$, the water level increases to $B = B + rand[B_{min}, B_{max}]$. We set B_{min} equal to 1 and B_{max} equal to 4 regardless to the size of the dataset.

7.2.2 Illustration of the Weights Adaptation

Before presenting our experimental results in detail, we further illustrate the weight adaptation mechanism. As explained above, the weight w_i for each of the low-level heuristics is set to 0.01 at the start of the search. Then, these weights are updated depending on the success or failure of the low-level heuristics to improve the current solution. In order to appreciate how this works, Figures 7.2 to 7.12 show the weight values for a particular run of the NLGDHH-SM algorithm on each of the test instances. The initial weights have the same value for all the low-level heuristics but as the search progress, we can see that these weights are adapted for each instance. For example, Figures 7.2 to 7.6 show that for small instances, the probability of low-level heuristic H3 being selected is reduced quickly down to zero. However, Figure 7.7 to 7.12 show that in the case of three medium instances and the large one, this probability remains above zero and fluctuating for most of the search. We can also see in these Figures that the weights for heuristics H1 and H2 are tuned for each test instance and there is no clearly defined common pattern.

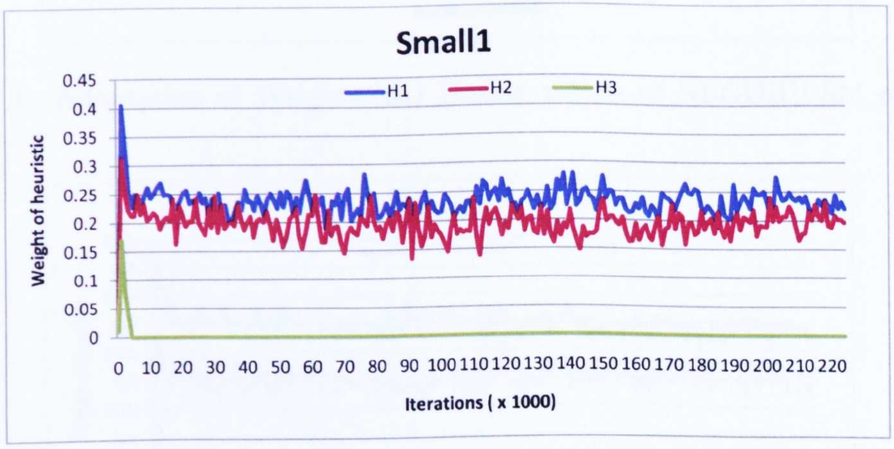


Figure 7.2: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on small1 Instance.

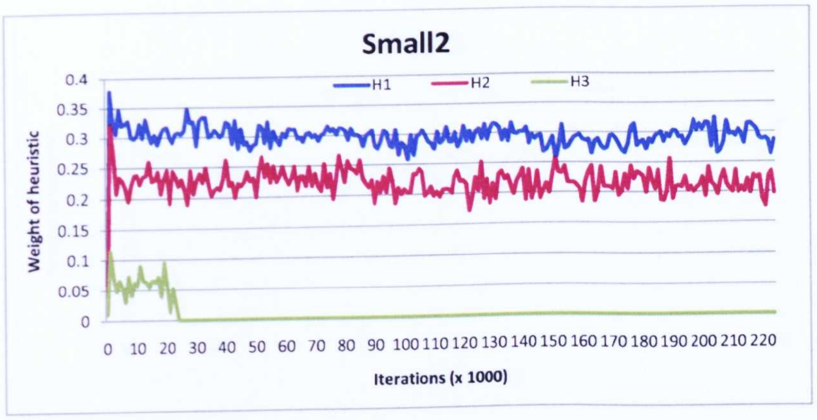


Figure 7.3: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on small2 Instance.

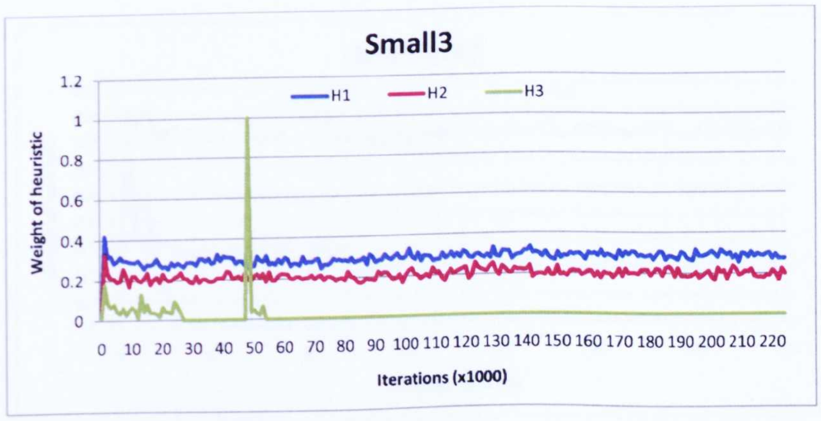


Figure 7.4: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on small3 Instance.

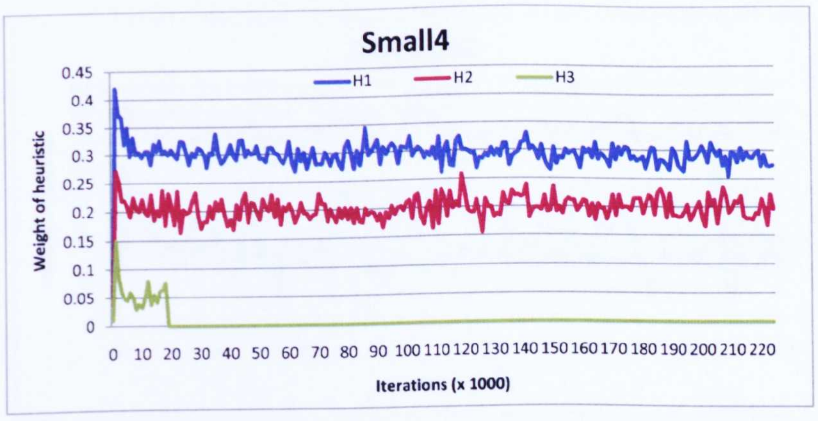


Figure 7.5: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on small4 Instance.

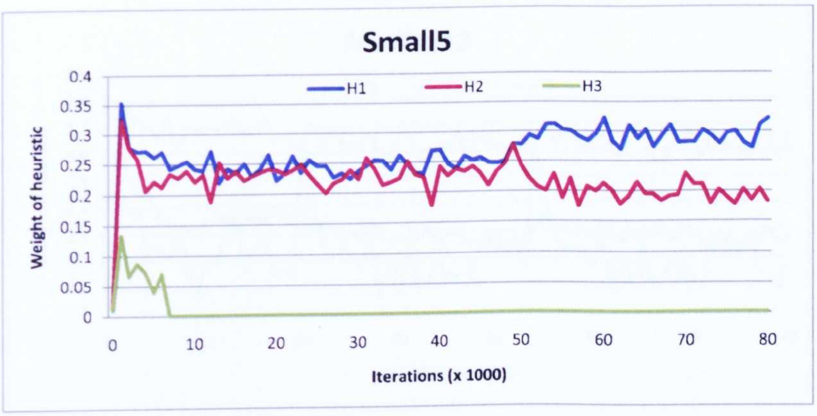


Figure 7.6: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on small5 Instance.

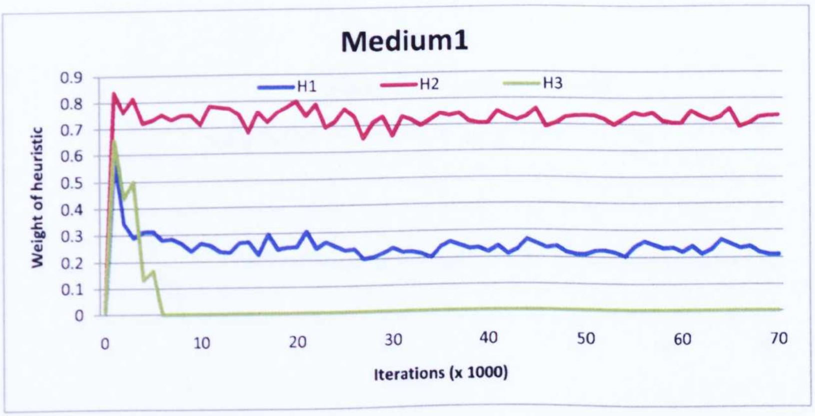


Figure 7.7: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on medium1 Instance.

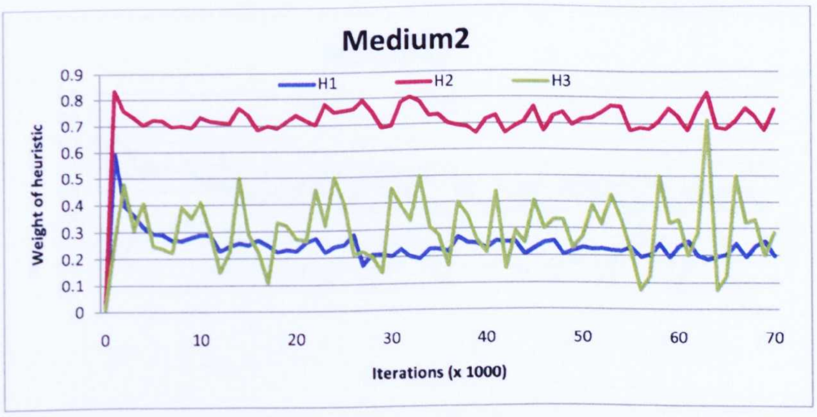


Figure 7.8: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on medium2 Instance.

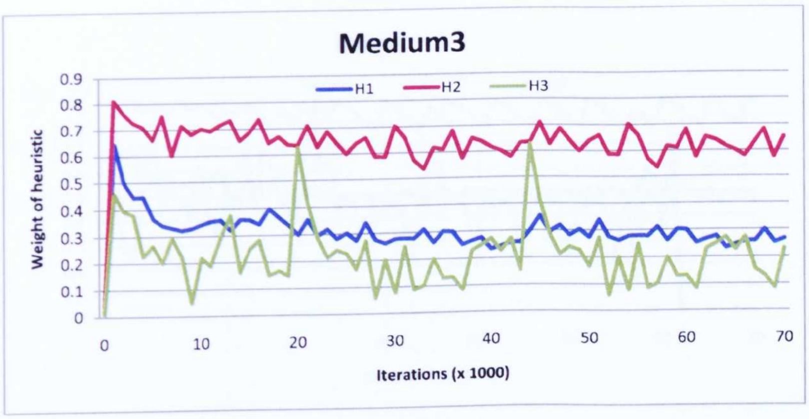


Figure 7.9: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on medium3 Instance.

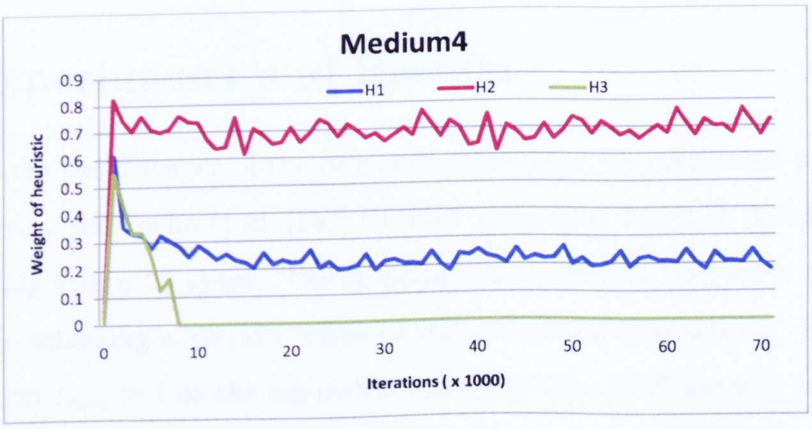


Figure 7.10: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on medium4 Instance.

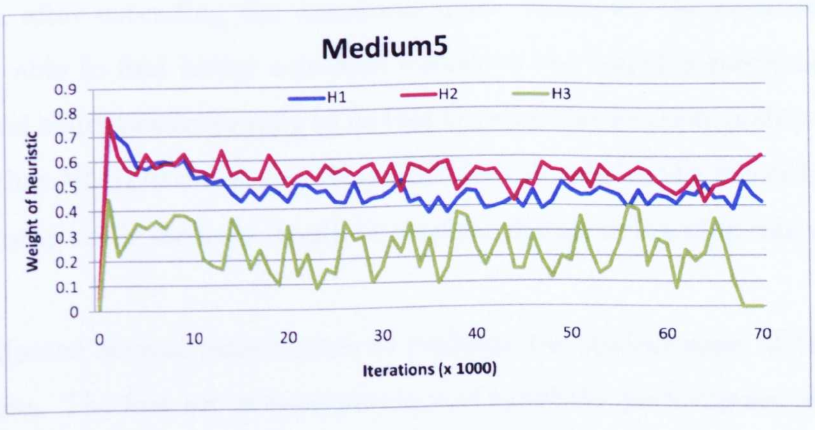


Figure 7.11: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on medium5 Instance.

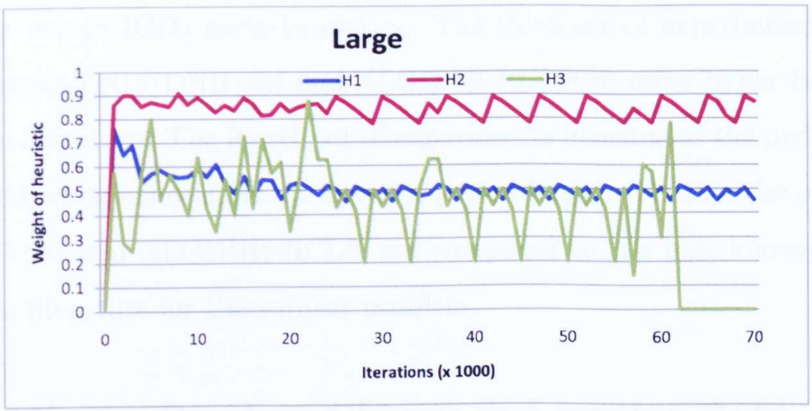


Figure 7.12: Adaptation of Weights (w_i) During a Run of NLGDHH-SM on large Instance.

7.3 Experiments and Results

To evaluate the performance of the proposed algorithms, we conducted a range of experiments using the Socha et al. [152] UCTTP instances. For each problem instance we run the algorithm 20 times. The stopping condition is a maximum computation time t_{max} or achieving a penalty value of zero, whatever was sooner. For small instances, we set $t_{max} > 1$ as the algorithm takes less than 2500 seconds (42 minutes). For medium instances, we set $t_{max} = 2.5$ hours. For the large instance, we set $t_{max} = 5$ hours. Our previous NLGD meta-heuristic [102] of chapter 4 was not able to improve results even after extending the execution time. However, the approach proposed here is now able to find better solutions thanks to the learning mechanism that selects low-level heuristics accurately to further improve the solution quality. We remind the reader that NLGDHH-SM and NLGDHH-RCLR refer to the algorithm proposed here when using static memory length or random change in learning rate respectively.

We conducted several experiments to evaluate the performance of the two algorithm variants. The first set of experiments compared the performance of NLGDHH-SM and NLGDHH-RCLR. A second set of experiments compared these methods to various great deluge (GD) meta-heuristics. The third set of experiments compared the performance of NLGDHH-SM and NLGDHH-RCLR to other hyper-heuristics reported in the literature. The fourth set of experiments investigate the performance of NLGDHH-SM when using different learning period length. Finally, the performance of NLGDHH-SM and NLGDHH-RCLR are compared to the best known results reported in the literature for the subject problem.

7.3.1 Static vs. Random Change in Learning Rate

We first compare NLGDHH-SM to NLGDHH-RCLR with the objective of examining the effect of the modified choice function mechanism when using Static Memory (SM) or Random Change in Learning Rate (RCLR). Figure 7.13 shows the best results obtained by the algorithm with each type of memory. We can see that both learning mechanisms are able to produce optimal solutions for all small instances for at least one out of 20 runs. For medium instances, both mechanisms perform well and the results obtained with the RCLR are competitive with those obtained with the static memory, particularly for the M1 instance (for which NLGDHH-SM obtained a value of 51 while NLGDHH-RCLR obtained a value of 54). The exact values are reported in Table 7.1. For instances M2, M3, M4, M5 and L, the results show that NLGDHH-SM obtained better solution quality compared to NLGDHH-RCLR.

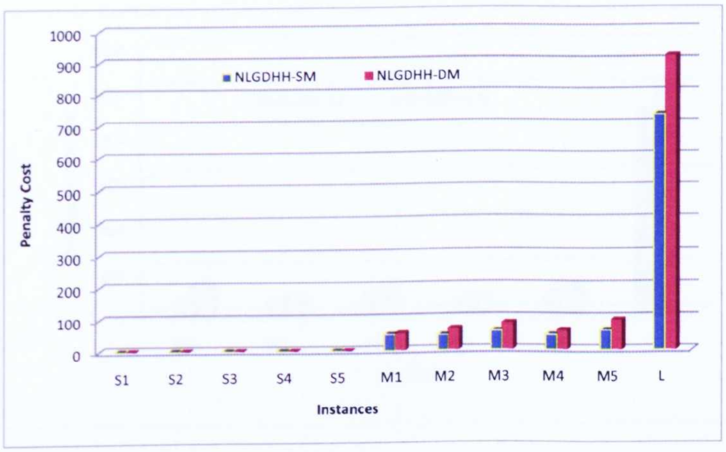


Figure 7.13: Best Results Obtained by NLGDHH-SM and NLGDHH-RCLR.

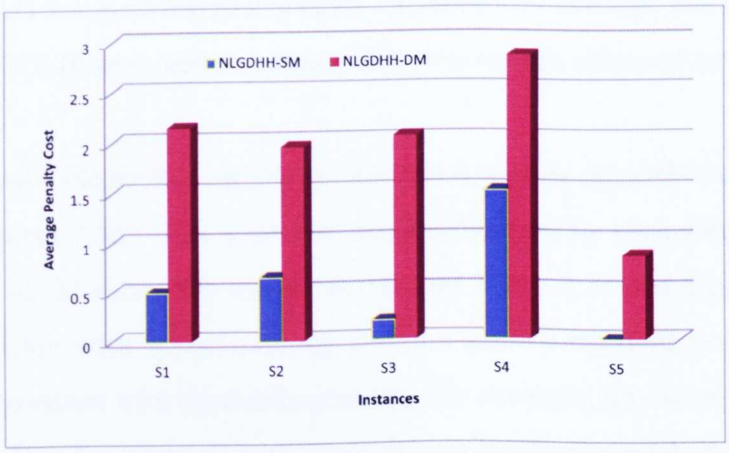


Figure 7.14: Average Results Obtained by NLGDHH-SM and NLGDHH-RCLR on Small Instances.

In addition to reporting the best results obtained from the 20 runs, we also report in Figure 7.14 and Figure 7.15, the average results over the 20 runs for each of the approaches. We can see that although both algorithms reach optimal solutions for all small instances, NLGDHH-SM does this more often compared to NLGDHH-RCLR. The overall results obtained by NLGDHH-SM are better than those achieved by NLGDHH-RCLR. It was shown above that the best results obtained by both algo-

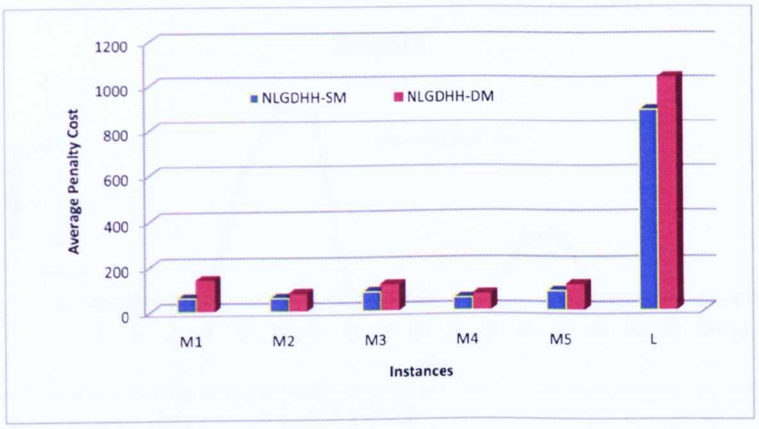


Figure 7.15: Average Results Obtained by NLGDHH-SM and NLGDHH-RCLR on Medium and Large Instances.

rithms on the M1 instance are pretty close. However, on average, the results obtained by NLGDHH-RCLR seem less consistent than the results achieved by NLGDHH-SM.

We now have a closer look at the performance of each algorithm on instances S1, M1 and L. Figures 7.16 - 7.21 show the results obtained by each algorithm on these instances over all 20 runs. We can see in Figures 7.16 - 7.17 and Figures 7.18 - 7.21 that the algorithm with static memory shows a more consistent performance compared to the algorithm with dynamic memory. For example, for the small instance S1, NLGDHH-SM found a solution with penalty zero in 15 runs while NLGDHH-RCLR did it only for 9 of the 20 runs. On the medium instance M1, the algorithm with static memory found better results in almost all the 20 runs and with less variability compared to the results obtained by the algorithm with dynamic memory. However, for the large instance, Figures 7.20 - 7.21 show that the algorithm with dynamic memory shows a more consistent performance over the 20 runs although the results obtained with the static memory are still better overall.

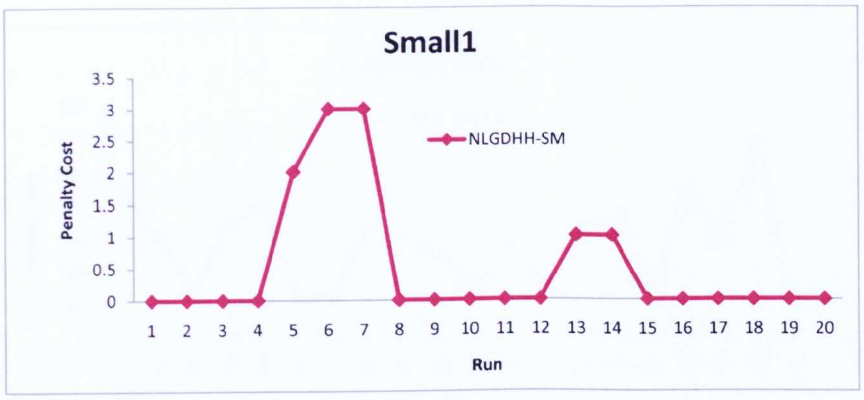


Figure 7.16: All Results Obtained by NLGDHH-SM on Small1 Instance.

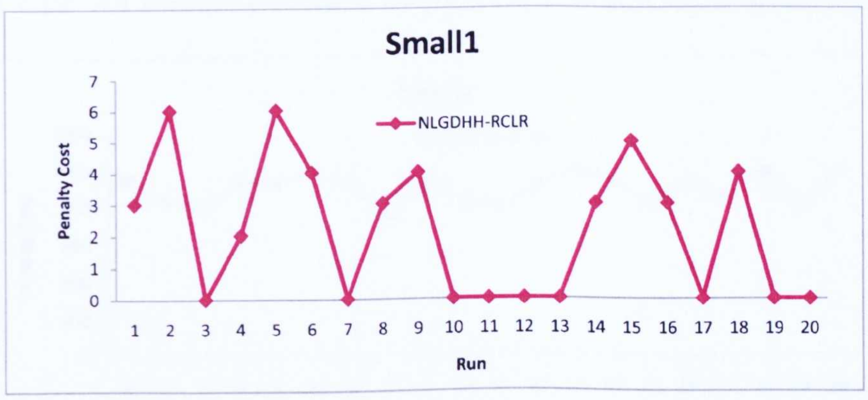


Figure 7.17: All Results Obtained by NLGDHH-RCLR on small1 Instance.

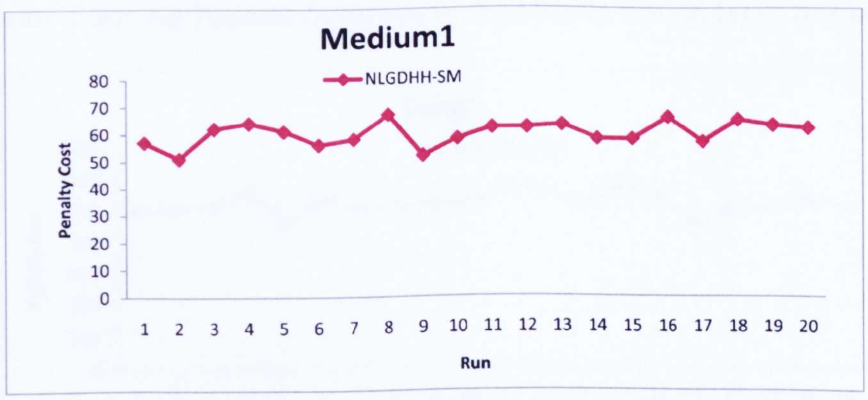


Figure 7.18: All Results Obtained by NLGDHH-SM on medium1 Instance.

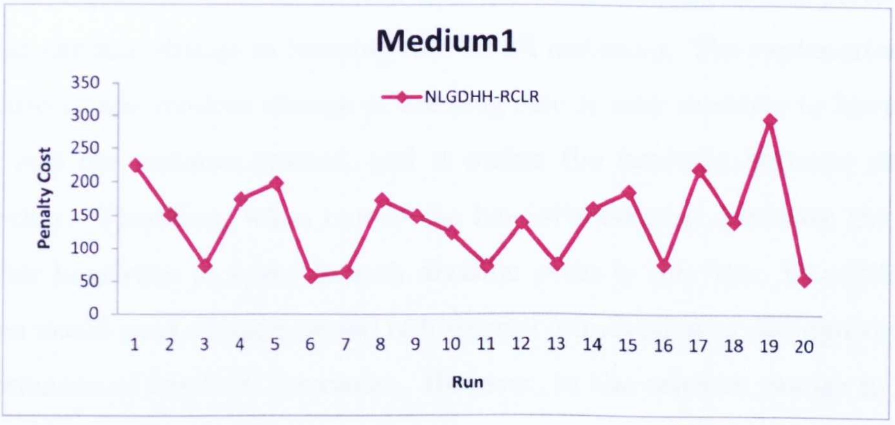


Figure 7.19: All Results Obtained by NLGDHH-RCLR on medium1 Instance.

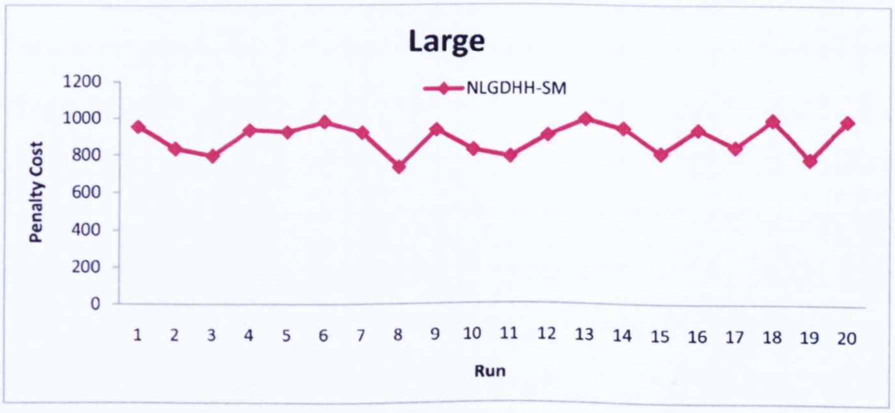


Figure 7.20: All Results Obtained by NLGDHH-SM on large Instance.

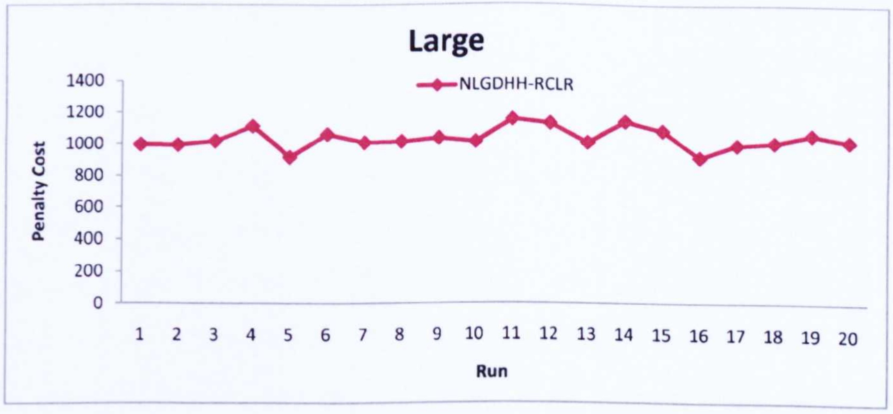


Figure 7.21: All Results Obtained by NLGDHH-RCLR on large Instance.

From the experiments it can be seen that the static memory length performs a lot better than random change in learning rate for all instances. The explanation behind this scenario is, the random change in learning rate is very sensitive to learning rate changing and the instance reward, and it makes the heuristic becomes dominates to each other. Therefore, when one of the heuristic become dominate the chances of the other heuristics to select at each decision point is very low. In addition both algorithms would need a longer period of historical information to distinguish between the performance of low-level heuristics. However, in the random change in learning rate the algorithm fails to give enough time to measure the performance of the low-level heuristics.

7.3.2 Comparison to Previous Great Deluge

The second set of experiments compared the proposed NLGDHH (with static and with random change in learning rate) to previous great deluge meta-heuristics in order to assess the performance of the non-linear acceptance criterion and the modified choice function mechanism. Table 7.1 shows the results obtained by NLGDHH-SM and NLGDHH-RCLR, the extended great deluge (EGD) [116], the non-linear great deluge (NLGD) [99] of chapter 4, the evolutionary non-linear great deluge (ENLGD) [101] of chapter 5, and the conventional great deluge (GD). We can see in Table 7.1 that NLGDHH-SM mostly outperforms NLGDHH-RCLR in terms of the number of best solutions found across all instances. Both variants of the proposed method obtained equal or better results than the other approaches, except for instance L where EGD found better solutions. However, NLGDHH-SM produced better solutions for 10 out of the 11 instances. In fact, NLGDHH-SM improved the solutions obtained by EGD for all medium instances. The average results for all medium instances obtained by NLGDHH-SM also outperform the average results obtained by EGD. However, for the large instance, EGD outperform NLGDHH-SM. The overall performance of both NLGDHH-SM and NLGDHH-RCLR is quite good according to these results.

7.3.3 Comparison to Other Hyper-heuristics

We now compare the proposed NLGDHH to other hyper-heuristics reported in the literature. Table 7.2 shows the results obtained by the following approaches: NLGDHH-SM, NLGDHH-RCLR, choice function hyper-heuristic (CFHH) [27], case-based hyper-heuristic (CBHH) [29], simulated annealing hyper-heuristic (SAHH) [17] and distributed-choice function hyper-heuristic (DCFHH) [141]. The results show that the proposed method finds equal or better solutions for 5 out of the 11 instances. For all small instances, both NLGDHH-SM and NLGDHH-RCLR are able to find the optimal solutions. For all medium instances, the NLGDHH variants achieve a significant improvement over the other hyper-heuristics. The NLGDHH approaches are also quite

Table 7.1: Comparison of the Proposed Great Deluge Based Hyper-heuristic and other Great Deluge Methods from the Literature.

Instance	NLGDHH-SM		NLGDHH-RCLR		EGD		NLGD	ENLGD-2	GD
	Best	Avg	Best	Avg	Best	Avg	Best	Best	Best
S1	**0**	**0.5**	**0**	2.5	**0**	0.8	3	**0**	17
S2	**0**	**0.65**	**0**	1.9	**0**	2	4	**0**	15
S3	**0**	**0.20**	**0**	2.05	**0**	1.3	6	**0**	24
S4	**0**	1.5	**0**	2.85	**0**	**1**	6	**0**	21
S5	**0**	**0**	**0**	0.85	**0**	0.2	**0**	**0**	5
M1	**51**	60.1	54	139	80	101.4	140	59	201
M2	**48**	59.05	67	78.2	105	116.9	130	51	190
M3	**60**	83.9	84	115.45	139	162.1	189	75	229
M4	**47**	54.9	60	72.05	88	108.8	112	48	154
M5	**61**	84.15	93	112.8	88	119.7	141	65	222
L	731	888.65	917	1035.25	730	834.1	876	**703**	1066

NLGDHH-SM is Non-linear great deluge hyper-heuristic with static memory length

NLGDHH-RCLR is Non-linear great deluge hyper-heuristic with Dynamic memory length

EGD is Extended Great Deluge Algorithm

NLGD is Non-linear Great deluge algorithm

ENLGD is the Evolutionary Non-linear Great Deluge (the ENLGD-2 variant of chapter 6)

GD is Great Deluge Algorithm

Best is the best result found by the algorithm

Avg is the average over the 20 runs

STD is Standard Deviation

S1-S5 represent small problem instances 1 to 5

M1-M5 represent medium problem instances 1 to 5

L represents the large problem instance

competitive in the large instance when compared to the results obtained by SAHH.

Table 7.2: Comparison of NLGDHH-SM, NLGDHH-RCLR and all types of Hyper-heuristic Algorithms Reported in the Literature.

	NLGDHH-SM	NLGDHH-RCLR	CFHH	GBHH	SAHH	DCFHH
S1	**0**	**0**	1	6	**0**	1
S2	**0**	**0**	2	7	**0**	3
S3	**0**	**0**	**0**	3	1	1
S4	**0**	**0**	1	3	1	1
S5	**0**	**0**	**0**	4	**0**	**0**
M1	**51**	57	146	372	102	182
M2	**48**	69	173	419	114	164
M3	**72**	93	267	359	125	250
M4	**47**	66	169	348	106	168
M5	**61**	100	303	171	106	222
L	731	915	1166	1068	**653**	-

CFHH is the Choice Function Hyper-heuristic in [27]
GBHH is the Graph-based Hyper-heuristic in [29]
SAHH is Simulated Annealing Hyper-heuristic [17]
DCFHH is Distributed Choice Function Hyper-heuristic [141]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L represents the large problem instance

7.3.4 Experiments With Different Memory Lengths

Since NLGDHH-SM produced better results, we conducted experiments with different learning period length (lp). We ran experiments with $lp = 250$, $lp = 500$, $lp = 1000$, $lp = 2500$, $lp = 5000$ and $lp = 10000$. The best and average results are presented in Table 7.3.

Instance	lp=250		lp=500		lp=1000		lp=2500		lp=5000		lp=10000	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
S1	**0**	0.7	**0**	0.5	**0**	0.5	**0**	0.3	**0**	0.35	**0**	0.35
S2	**0**	0.95	**0**	0.9	**0**	0.65	**0**	0.4	**0**	0.2	**0**	0.35
S3	**0**	0.35	**0**	0.4	**0**	0.20	**0**	0.2	**0**	0.3	**0**	0.40
S4	**0**	1	**0**	0.85	**0**	1.5	**0**	0.8	**0**	0.5	**0**	0.55
S5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
M1	54	61.6	53	56.9	51	60.1	**38**	53	42	51.35	44	52.15
M2	51	61.6	52	63.35	48	59.05	**37**	50.3	44	51.4	44	52.75
M3	70	101.2	62	78.4	60	83.9	61	75.45	**60**	79.5	61	79.65
M4	40	56.45	53	61.25	47	54.9	41	49.35	**39**	47.2	43	49.1
M5	68	87.8	62	77.15	61	84.15	61	76.95	**55**	79.05	62	78.45
L	818	937.4	755	939.85	731	888.65	**638**	829.05	713	875.1	831	918.75

Table 7.3: Comparison of the NLGDHH-SM with Different Learning Period Length lp.

We can see that for different values of lp, the proposed methods perform differently. All static memory (SM) variants are able to find the optimal solution for small instances. For medium and large instances $lp = 2500$ and $lp = 5000$ give better results. For the large instance $lp = 2500$ gives better results than all other values of lp. NLGDHH-SM performed worst with $lp = 250$. The overall performance for different lp values is shown in Figures 7.22 - 7.24. From these experiments, we can conclude that longer length of learning period produces better quality solutions than lp with shorter values. However, we must not set the lp too long as it gets worst if we over extend the lp. As shown in Table 7.3, when we set $LP = 2500$, it gives better results, and when we extend it to $LP = 5000$ the algorithm still performs very well by improving some of the solutions which are not able to achieve by $LP = 2500$. However, when we set the $LP = 10000$, the quality of the solutions become worse.

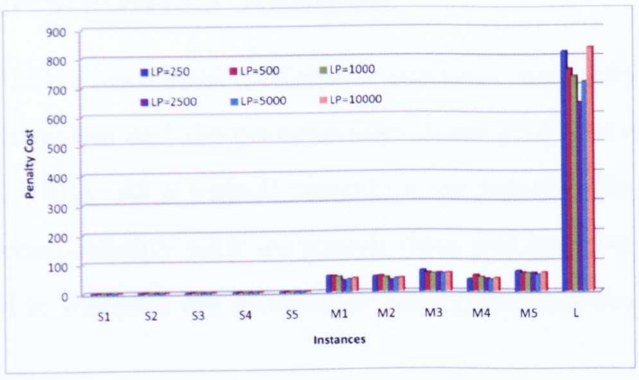


Figure 7.22: The Best Results Obtained from Different lp Values

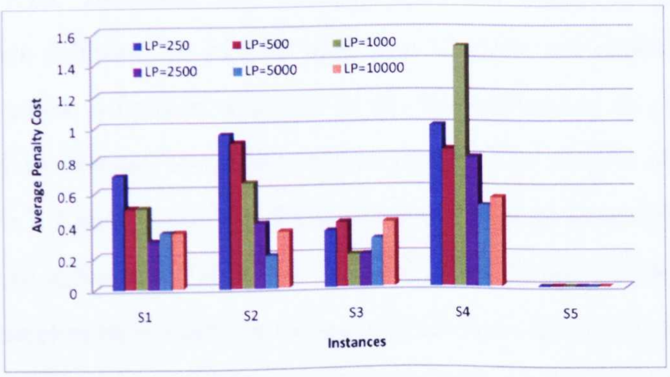


Figure 7.23: Average Results from Different lp Values - Small instance.

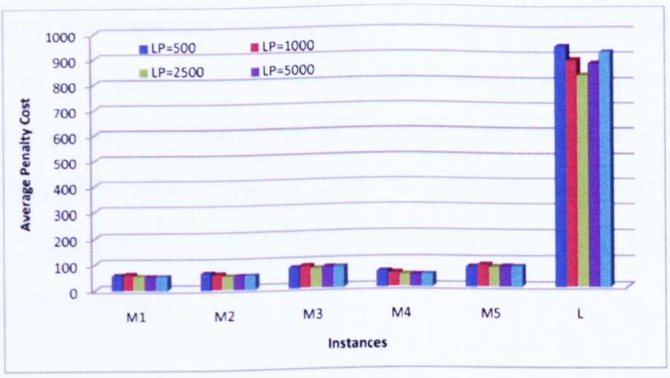


Figure 7.24: Average Results from Different lp Values - medium and large instances.

7.3.5 Further Statistical Analysis

We now conduct further statistical analysis to examine the performance of the proposed Learning Mechanism and the previous algorithms proposed earlier in this thesis (see also [100] and [101]). As a normal procedure we run the descriptive analysis to check the model's compatibility with the sample data and found that ANOVA is suitable statistical tool to measure the differences among the proposed algorithms.

Figure 7.25 shows that there is significant difference between the learning mechanisms and our previous proposed algorithms, with p-value very close to zero as shown in Figure 7.25. However, this analysis does not shows the detail whether all four algorithms are different or maybe only one of them are different. Therefore it was necessary to make a further analysis to see the difference in pairs. A Post-Hoc analysis was used to inspect the difference in pairs. The results of the analysis are presented in Table 7.7 and Table 7.8. First of all we want to examine if the two learning mechanisms are different. Table 7.6, Table 7.7 and Table 7.8 clearly suggest that the two learning mechanisms static memory and random change in learning rate have difference performance across small, medium and large instances, where the p-value is very close to zero (p=0.05) significant level.

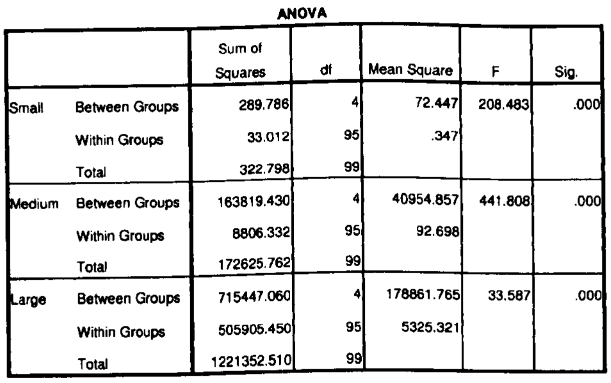
We also compared the learning mechanisms with our previous algorithm and found that no similarity of the performance of our previous algorithms. In detail, for all size of instances our proposed algorithm performed differently where the p-value for each pair (NLGDHH-SM, NLGDHH-RCLR), (NLGDHH-SM, ENGD), (NLGDHH-SM, NLGD), (NLGDHH-RCLR, ENGLD)(NLGDHH-RCLR, NLGD) were close to zero at p=0.05 significant level.

As the Post-Hoc test indicates, all four algorithms have different performance. However at this point the analysis did not tell us which algorithm is actually outperforming across all 11 instances. To examine which algorithm are more superior than the other, we refer to the mean plot of each algorithm at 95% confidence level for the different algorithms as shown in Figure 7.26, Figure 7.27 and Figure 7.28. Firstly, Figure 7.26 indicates that there are four homogenous groups and they are (NLGDHH-SM (LP=2500), NLGDHH-SM(LP=5000)), (NLGDHH-RCLR), (ENLGDS2) and (NLGD). And two algorithms perform better are (NLGDHH-SM (lp=2500), NLGDHH-SM (lp=5000) followed by ENLGDS2 and in third by NLGDHH-RCLR. Among the proposed algorithms, NLGD is the worst algorithm. Figure 7.27 shows that there are three homogenous groups, NLGDHH-SM (lp=2500), NLGDHH-SM(lp=5000)), (NLGDHH-RCLR, NLGDS) and (NLGD). The two best algorithms are NLGDHH-SM (lp=2500), NLGDHH-SM (lp=500) followed by ENLGDS and NLGDHH-RCLR, and the worst algorithm is NLGD. Finally for large instance, we found that there are three homogenous groups (NLGDHH-SM (lp=2500), NLGDHH-SM (lp=5000), ENLGDS), (NLGDHH-RCLR) and NLGD. In the large instance case, we found that three algorithms perform equally and they are NLGDHH-SM (lp=2500), NLGDHH-SM (lp=5000) and NLGDHH-RCLR. These three algorithms outperform the other algorithms, and the worst algorithm is NLGDHH-RCLR. In conclusion, overall, the mean plots show that both NLGDHH-SM (lp=2500 and lp=5000) outperform the other algorithms and the worst algorithm is NLGD.

The statistical analysis suggests that each algorithm has different behaviour and performs differently across all 11 instances. These analysis also show that NLGDHH-SM outperforms five other algorithms across all instances. It is also evident that the learning mechanism with static memory is better than random change learning rate and the rest of our proposed algorithms. Moreover, with learning mechanisms proven to be a good strategy as it managed to further improve the solution quality compared to our previous algorithms without learning mechanism. As a conclusion, the proposed learning mechanisms incorporated with Non-Linear Great Deluge matches the best known solution quality for all small problem instances and improves the best known results for all five medium instances. Only in the case of the large problem instance, we see that our algorithms do not match the best known result.

	NLGDHH-SM(lp=2500)			NLGDHH-SM(lp=5000)			NLGDHH-RCLR		
Run	Small	Medium	Large	Small	Medium	Large	Small	Medium	Large
1	0	61	870	0	63.4	813	1.4	119.6	995
2	0	65.6	827	0.6	61.2	824	1.8	110.6	993
3	0	63.4	739	0	61.8	911	2	89.6	1017
4	0.6	61.8	905	0	58	786	0.4	110.6	1112
5	0	62.8	806	0	58.8	958	3	112.2	915
6	0.2	64.2	936	0	54.4	962	2.4	92	1062
7	1	56.8	918	0.6	63.8	813	2.2	88	1008
8	0.2	61.2	800	1.8	68	824	2.4	109.8	1014
9	0.6	56.8	813	0	64.8	853	2.8	101.8	1036
10	0	62.6	638	0	64.2	813	0.8	99	1011
11	0	61.2	843	0.6	64.6	824	2.6	87.2	1153
12	0.2	67	817	0.8	62.6	880	0.8	108.6	1130
13	1	60.8	845	0.2	64.2	914	2.6	92	1010
14	1	59.4	817	0.2	55.4	879	2.8	102	1146
15	0.6	59	912	0	64.4	713	3.8	118.6	1088
16	1	58.6	857	0.4	66	1030	2.8	92.4	917
17	0	57.2	855	0	60.4	1083	1.4	121	996
18	0.4	63.4	788	0.2	61.2	985	1.4	106.8	1012
19	0	56.6	725	0	61.2	813	1.4	134	1065
20	0	60.8	870	0	55.6	824	0.6	76.6	1025

Table 7.4: Average Penalty Cost of NLGDHH-SM(lp=2500), NLGDHH-SM(lp=5000) and NLGDHH-RCLR Across the 11 Socha et al. Instances.


ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Small	Between Groups	289.786	4	72.447	208.483	.000
	Within Groups	33.012	95	.347		
	Total	322.798	99			
Medium	Between Groups	163819.430	4	40954.857	441.808	.000
	Within Groups	8806.332	95	92.698		
	Total	172625.762	99			
Large	Between Groups	715447.060	4	178861.765	33.587	.000
	Within Groups	505905.450	95	5325.321		
	Total	1221352.510	99			

Figure 7.25: ANOVA Results

	ENLGD-2			NLGD		
Run	Small	Medium	Large	Small	Medium	Large
1	0.8	95.6	703	3.8	142.4	966
2	0.4	85.8	927	4.8	165	1070
3	0.4	95.4	835	6	165.6	876
4	0.4	93.6	968	5.2	162.2	935
5	0.4	108.6	895	5	165.2	971
6	0.4	99.8	730	4.6	166.8	942
7	0.2	81.2	782	5	165.4	895
8	0.4	91.6	711	5.2	156.8	976
9	0.8	110.4	777	5.4	160.4	986
10	1	96.4	838	5.4	172.8	1005
11	0.4	96.6	808	3.8	185	966
12	1	98.4	944	4	171.6	1070
13	0.8	91.2	870	4.2	177	935
14	0.4	96.4	807	4.2	181	1024
15	0.8	83.6	849	4	172.4	942
16	1.2	90.6	713	5	188.4	958
17	0.4	117.8	852	4.2	179.6	978
18	0.6	102.2	795	5.4	182.6	1005
19	1.6	106	779	5.4	196	1078
20	0.8	89.4	801	5	183.8	907

Table 7.5: Average Penalty Cost of ENLGD-1 and NLGD Across the 11 Socha et al. Instances.

	NLGDHH-SM (LP=2500)	NLGDHH-SM LP=5000	NLGDHH-RCLR	ENGD-2	NLGD
NLGDHH-SM(LP=2500)	-	1.000	0.000	0.097	0.000
NLGDHH-SM(LP=5000)	1.000	-	0.000	0.038	0.000
NLGDHH-RCLR	0.000	0.000	-	0.000	0.000
ENLGD-2	0.097	0.038	0.000	-	0.000
NLGD	0.000	0.000	0.000	0.000	-

Table 7.6: Post Hoc Tests - Small Instances.

	NLGDHH-SM LP=2500	NLGDHH-SM LP=5000	NLGDHH-RCLR	ENLGD-2	NLGD
NLGDHH-SM(LP=2500)	-	0.999	0.000	0.000	0.000
NLGDHHsSM(LP=5000)	0.999	0.000	0.000	0.000	0.000
NLGDHH-RCLR	0.000	0.000	-	0.483	0.000
ENLGD-2	0.000	0.000	0.483	-	0.000
NLGD	0.000	0.000	0.000	0.000	-

Table 7.7: Post Hoc Tests - Medium Instances.

	NLGDHH-SM LP=2500	NLGDHH-SM LP=5000	NLGDHH-RCLR	ENLGD-2	NLGD
NLGDHHsSM(LP=2500)	-	0.551	0.000	1.000	0.000
NLGDHHsSM(LP=5000)	0.551	-	0.000	0.332	0.002
NLGDHH-RCLR	0.000	0.000	-	0.000	0.030
ENLGD-2	1.000	0.332	0.000	-	0.000
NLGD	0.000	0.002	0.030	0.000	-

Table 7.8: Post Hoc Tests - Large Instance.

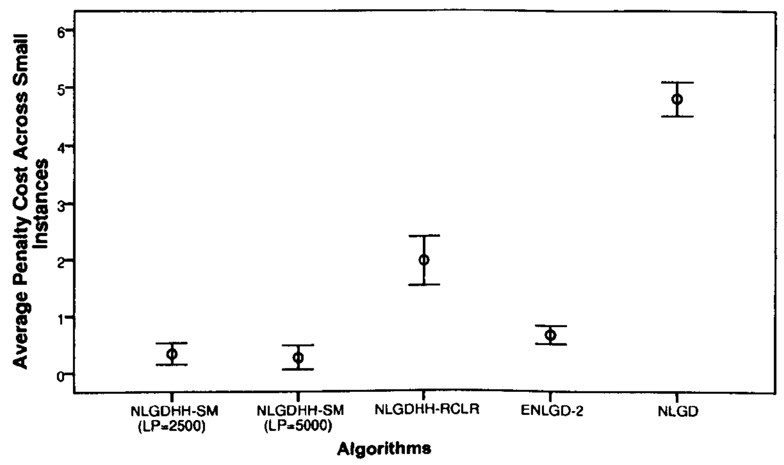


Figure 7.26: Means Plot and LSD Intervals (Small Instances).

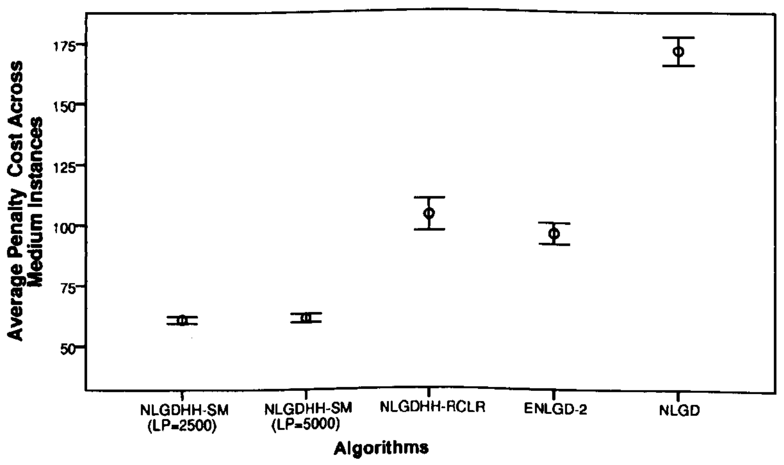


Figure 7.27: Means Plot and LSD Intervals (Medium Instances).

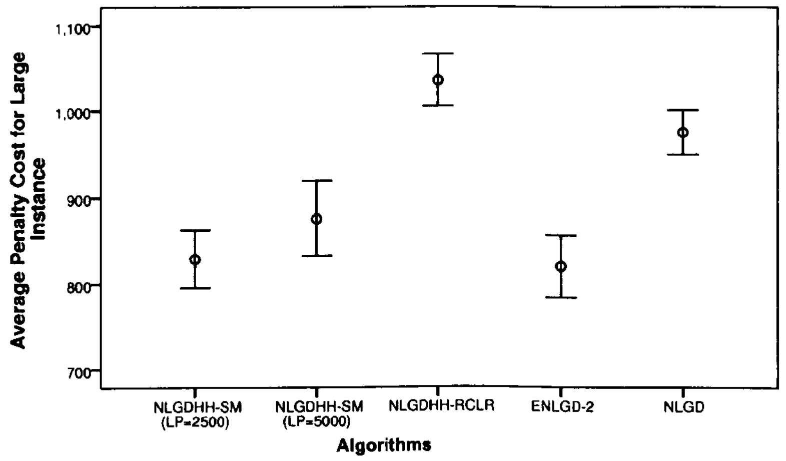


Figure 7.28: Means plot LSD Intervals (Large Instance).

7.3.6 Comparison with Best Known Results

Finally, we compare the results obtained by the NLGDHH with the best results reported in the literature for the subject problem. Columns 2 to 4 in Table 7.9 show the results obtained by NLGDHH, while the fifth column shows the best known results and the corresponding approaches. It should be noted that although a timetable with zero penalty exists for each problem instance (the data sets were generated starting from such a timetable [152]), to the best of our knowledge no heuristic method has found before the ideal timetable for the medium and large instances. Hence, these data sets are still very challenging for heuristic search methods. For all small instances, both approaches NLGDHH-SM and NLGDHH-RCLR produced optimal solutions. For medium instances, NLGDHH-SM improved the best solutions of M1, M2, M3, M4 and M5 while NLGDHH-RCLR improved the best solution of M1, M2, M3, and M4. For the large instance, neither NLGDHH-RCLR nor NLGDHH-RCLR improved the best solution reported in the literature but they are very competitive.

Table 7.9: Comparison of the Proposed Great Deluge Based Hyper-heuristic to the Best Results Reported in the Literature for the Socha et al. UCTTP Instances.

Instance	NLGDHH-SM LP=1000	NLGDHH-SM LP=2500	NLGDHH-SM LP=5000	NLGDHH-RCLR	Best Known
S1	**0**	**0**	**0**	**0**	**0** (VNS-T)
S2	**0**	**0**	**0**	**0**	**0** (VNS-T)
S3	**0**	**0**	**0**	**0**	**0** (CFHH)
S4	**0**	**0**	**0**	**0**	**0** (VNS-T)
S5	**0**	**0**	**0**	**0**	**0** (MMAS)
M1	51	**38**	42	54	80 (EGD)
M2	48	**37**	44	67	105 (EGD)
M3	60	61	**60**	84	139 (EGD)
M4	47	41	**39**	60	88 (EGD)
M5	61	61	**55**	93	88 (EGD)
L	731	638	713	915	**529**(HEA)

NLGDHH-SM is the Non-Linear Great Deluge Hyper-heuristic with fixed memory length
NLGDHH-RCLR is the Non-Linear Great Deluge Hyper-heuristic with dynamic memory length
MMAS is the MAX-MIN Ant System in [152]
CFHH is the Choice Function Hyper-heuristic in [27]
VNS-T is the Hybrid of VNS with Tabu Search in [5]
HEA is the Hybrid Evolutionary Algorithm in [3]
EGD is the Extended Great Deluge in [116]

7.4 Conclusions

In this chapter we have developed a hyper-heuristic approach that uses the modified choice function mechanism and a non-linear great deluge (NLGD) acceptance criterion to manage the selection of low-level heuristics during the search process. The proposed hyper-heuristics deals only with complete feasible solutions. Two types of modified choice function learning mechanism are investigated: learning with static memory length and learning with random change learning rate. The method focuses on trying to choose the most appropriate heuristic in each step of the search and hence it follows the *hyper-heuristic* concept. We applied the proposed method to well-known instances of the university course timetabling problem proposed by Socha et al. [152]. We did not employ large neighbourhoods in our problem, as the purpose of this research chapter is to test the effectiveness of the learning mechanisms approach when incorporated to our non-liner great deluge hyper-heuristic. In addition we want to make the comparison between static memory length and random change of learning rate. The employment of the large neighbourhoods will be investigate in the future. The experimental results show that the non-linear great deluge hyper-heuristic performs better using static memory length. Furthermore, the algorithm with static memory produced five new best results out of eleven instances while the algorithm with random change in learning rate produced four best results compared to the best known results from the literature. However, for the large instance, both algorithms produced only competitive results, rather than best.

Chapter 8

Developing Asynchronous Cooperative Multi-agent Search

8.1 Introduction

This chapter presents novel *asynchronous cooperative search* approaches to tackle the university course timetabling problem. The proposed algorithms are agent-based systems inspired in the particle swarm optimisation metaheuristic and implemented using CODEA [38], a programming framework for the development of multi-agent systems. We propose two asynchronous cooperative algorithms: the first one is an Asynchronous Cooperative Multi-heuristic (ACMH) and the second one is an Asynchronous Cooperative Multi-hyper-heuristic (ACMHH). Both algorithms are multi-agent based systems in which a number of autonomous agents cooperate within a distributed environment in order to improve the global solution. Like our previous search methods in this thesis, both approaches here start their search from complete feasible solutions and try to improve the satisfaction of soft constraints whilst always remaining in the feasible region of the search space. The performance of the proposed asynchronous cooperative algorithms are compared using the Socha et al. in [152] problem instances of the university course timetabling problem.

The rest of this chapter is organised as follows. Section 8.2 gives an overview of hyper-heuristics and parallel cooperative search. The literature review chapter in this thesis already made an account of previous work on the application of hyper-heuristics to the UCTTP, so here we concentrate in discussing the rationale of hyper-heuristics and cooperative search. Section 8.3 discusses the important issues to consider when developing asynchronous cooperative search algorithms and also outlines some of the cooperative strategies proposed in the literature. Section 8.4 gives an outline of Particle Swarm Optimisation, which inspired the multi-agent algorithms proposed in this chapter. The asynchronous cooperative multi-agent algorithms proposed in this chapter are described in Section 8.5, while Section 8.6 presents and discusses experiments and results. Finally, conclusion for this chapter are given in Section 8.7.

8.2 Hyper-heuristics and Parallel Cooperative Search

The emergence of parallel processing and cooperative search strategies as tools to develop more effective and efficient search methodologies has attracted the attention of researchers particularly in the last ten to fifteen years. Parallel cooperative search is an important stream of metaheuristics development that has resulted in the publication of many articles in the literature tackling complex combinatorial optimisation problems, e.g [19, 134, 51, 49, 48, 104, 99]. One of the motivations behind the interest on cooperative search is that diversification is inherent in cooperative search because various areas of the search space can be better explored with multiple explorers working in parallel than with sequential heuristics. In addition, cooperative search is capable of performing more efficient search due to the possibility of combining different and independent strategies with different parameter settings into a more robust system [48, 134]. Cooperative search is also capable of increasing the speed of the search and reduces the computation time when solving a problem instance in comparison to the execution of independent sequential heuristics. In this work, we propose asynchronous cooperative search in which each agent is free to communicate to each other as needed, this type of model is called island model according to [47].

8.3 Asynchronous Cooperative Search

Several cooperative search approaches have been proposed in the literature. Crainic et al. [52] presented a taxonomy for parallel tabu search. They classified approaches based on three features: the *control strategy* used to guide the search, the *information sharing mechanism* to exchange information between the threads and the *strategy to partition the search space.* Later, Crainic and Toulouse in [50, 51, 48] further investigated parallel tabu search and presented three types of strategies: *low-level parallelisation*, *parallelisation by domain decomposition*, and *cooperative/independent multi-thread.* In the first type of strategy, parallelism is usually realised within an iteration where moves are evaluated in parallel. In the second type of strategy, the problem search space is partitioned into several parts and an individual parallel search is conducted in each part accelerating the global search. The resulting partial explorations are combined by a master process to obtain an overall feasible solution. In the third type of strategy, several concurrent searches are conducted over the same solution space, this strategy is also called cooperative/independent multi-thread strategy. The threads may communicate during the search or only at the end of it to identify the best overall solution. Communication among the threads may be performed synchronously or asynchronously and may be even executed at predetermined or dynamically decided moments. Several studies in the literature have shown that multi-thread strategies yield better solutions than the corresponding sequential approach, even when the exploration time permitted to each thread is significantly lower than that of the sequential computation [51, 48]. Studies have also shown that the combination of several threads that implement different parameter settings increases the robustness of the global search relative to variations in problem instance characteristics.

Crainic and Toulouse [47] have emphasised the design of the *information exchange mechanism* as the key element that determines the performance of cooperative search methods. They also discussed that other design issues include: what information to exchange, when to exchange it, the logical inter-processor structure, synchronous or asynchronous communication and what each independent process does with the received information. Therefore, the design issues must be tackled carefully in order to come out with efficient and robust search method.

8.4 Particle Swarm Optimisation

Particle Swarm Optimization (PSO) was proposed by Russel Eberhart and James Kennedy [69, 94] as a population-based stochastic approach for solving continuous optimisation problems and was inspired by social-psychological individual behaviour of swarms like bird flocking or fish schooling. A swarm is made of particles that move in a multidimensional space. Each particle has a position $x_{i,j}$ given by its current solution and a velocity $v_{i,j}$ used to update the particle's position in each iteration of the algorithm. The particles in a PSO implementation fly through the hyperspace $\mathbb{R}^n$ and have important capabilities. Each particle memorises its own best position (solution) and also each particle makes this information available to its neighbouring particles. Particles also have the knowledge of the global best-so-far or the best position of its neighbourhood. With this information about their own solution and that of the other particles in the swarm, each particle's position and velocity is updated as shown in equations 8.1 and 8.2.

Equations 8.1 and 8.2 determine how a particle updates its velocity and position respectively. Equation 8.1 shows that the velocity of a particle is highly influenced by inertial and social coefficients. Inertial movement refers to the tendency of the particle to follow its own direction while social movement refers to tendency of the particle to follow other better positioned particles in its vicinity and also the whole swarm. Each of the weights c_0 to c_3 indicate the inertial and social influence when the particle moves or updates its position (solution). That is, c_0 is the inertial weight that corresponds to the particle's own explorative ability whilst c_1, c_2 and c_3 are the social weights that correspond to the influence that other particles have on the particle's movement. The social coefficients drive the particle to follow other better positioned particles such as the global best in the whole swarm (x_j^{gb}), the local best or best position that the particle has achieved during the search (x_j^{lb}) and the best particle's neighbour (x_j^{nb}). To avoid a predictable behaviour of the particle's movement, the coefficients are multiplied by random number (r_1, r_2 and r_3) drawn from a uniform distribution [0,1]. Once the particle's velocity is updated with Equation 8.1, the particle updates its position (changes its solution) by adding the new velocity to its current position using Equation 8.2. In basic particle swarm algorithm it consists of a number of iterations, where at each iteration every single particle in the swarm is updating its position (current solution) and velocity. Since all particles in the swarm have the knowledge of what is the best position in its neighbourhood (the best solution achieved by other particles in the swarm), therefore whole swarm a likely to move towards the better positions guided by the leading particle so far.

$$v_{i,j} = c_0 v_i + c_1.r_1(x_j^{gb} - x_{i,j}) + c_2.r_2(x_j^{lb} - x_{i,j}) + c_3.r_3(x_j^{nb} - x_{i,j}) \qquad (8.1)$$

$$x_{i,j} = x_{i,j} + v_{i,j} \qquad (8.2)$$

8.5 The Asynchronous Cooperative Multi-agent Algorithms

The Distributed Multi-agent system is a network of agents that work together to solve problems that a beyond their individual capabilities [124]. Whereas, in [133] defines a multi-agent systems as are distributed and autonomous systems made up of autonomous agents that support reactivity, and are robust against failures locally and globally. There are two prominent multi-agent architectures have been addressed in the literature: blackboard and autonomous architectures. Hayes-Roth [86] proposed multi-agent system based on blackboard architecture. The model proposed by Hayes-Roth is an inter-agent communication that share a common global database, called the blackboard, which can be accessed by a number of knowledge sources. The knowledge source is a set of problem solving modules. These knowledge sources communicate by manipulating the contents of the blackboard. The knowledge sources respond to changes on the blackboard and directly modifying and withdrawing solution elements within a common working area called blackboard. In blackboard architecture, the data are placed in central and each agent is able to read the data from the blackboard. If the agents in the system are capable to execute the task, then the agents record the result on the blackboard. The main features of blackboard architecture are: Relatively, this architecture is more on centralise controlled and lack of local memory. Different than blackboard architecture, the proposed method in this thesis is autonomous agent architecture, where the agents in the distributed environment are not controlled or managed by any other agents. Each agent is free to communicate and interact directly to each other to achieve the global objective as illustrated in Figure 8.1. Therefore, each agent in our proposed system embodies its own knowledge, meaning that each agent has the ability to knows its best position (current solution) and the best position of its neighborhood. In addition, every single agent is also have the knowledge of the global best-so-far solution. Furthermore, each agent embodies its own control, for example make the information (solution) available to all agents in the distributed environment and able to change their position (current solution) when necessary.

In this paper we propose two types of cooperative search algorithms: Asynchronous Cooperative Multi-heuristic (ACMH) and Asynchronous Cooperative Multi-hyper-heuristic (ACMHH). Both approaches operate from an initial complete feasible solution and maintain feasibility during the search. The initial solutions provided to these algorithms are constructed with the Initialisation Heuristic 3 (IH3) presented in chapter 4 of this thesis. Please refer to that chapter for details.

8.5.1 Asynchronous Cooperative Multi-agent Search Framework

Inspired by the particle swarm optimisation algorithm, we propose a decentralised agent-based system framework which consists of a population of agents. Since we are dealing with a discrete search space instead of a continuous one, we require the agents to 'jump' from position to position instead of moving continuously as in the original PSO algorithm. Then, in our framework all agents in the distributed environment cooperate among them and jump from one solution to another one searching for the global optima. The proposed decentralised multi-agent framework is illustrated in Figure 8.1. Each agent is able to perform four types of jump (changing the solution) depending on which agent acts as the attractor in each iteration. This model shows that each agent is able to communicate with each other, communication links are represented by arrows. Therefore, the system is designed in such a way that whenever an agent gets stuck in local optima, i.e. is unable to change its solution or find a new better solution, the agent can communicate freely with other agents in the distributed environment.

Each agent starts from a different feasible solution. In each iteration all agents update their positions based on work in [92, 114]. In order to select the new position, a random number r is drawn from the uniform distribution interval [0,1]. The interval is divided into four segments with length $\sum_{i=1}^{4} c_i = 1$. Therefore, the agent will choose its type of movement (inertial or social influence) based on which segment c_i the value of r falls. That is, the formulation of how the agents in the distributed environment change their position can be expressed by:

$$x_i = c_1 x_{i,j} \bigoplus c_2 x_j^{nb} \bigoplus c_3 x_j^{lb} \bigoplus c_4 x_j^{gb} \tag{8.3}$$

From the above formulation the resulting move can be as follows:

1. If $r \in c_1$, no agent acts as an attractor, therefore, the agent changes its position at random with respect to its current state $x_{i,j}$. This type of move called intertial move, and the purpose of this move is to allow the agent to explore and diversify the search.

2. If $r \in c_2$, the agent changes its position moving towards the best agent in its current neighbourhood x_j^{nb}.

3. If $r \in c_3$, the agent changes its position moving towards the best position achieved by this agent so far x_j^{lb}.

4. If $r \in c_4$, the agent changes its position moving towards the best global agent in the system x_j^{gb}

The asynchronous cooperative multi-agent system proceeds as follows:

1. The agents start the search each from an initial feasible solution.

2. Once all agent determine their new position (new solution), the agents apply their own heuristic to improve the quality of their own solution. That is, each agent is able to diversify its own search by:

(a) Jumping from one solution to another solution.

(b) Using their own stochastic acceptance criterion for accepting or not their new solution. We use two different acceptance criteria here: Non-linear great deluge and simulated annealing.

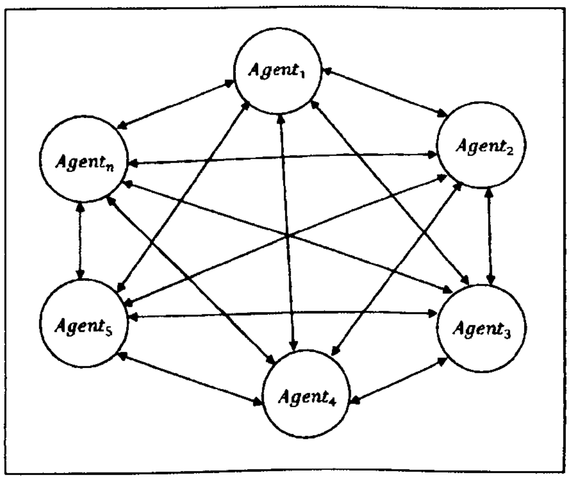


Figure 8.1: Island Model Multi-agent System: Many-to-Many Communication Model.

8.5.2 Low-Level heuristics

As previously in this thesis, we use the following three neighbourhood moves or low-level heuristics:

- H1: selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random.
- H2: selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained.

- H3: selects three events at random and exchanges their timeslots and rooms at random while ensuring feasibility is maintained.

8.5.3 Acceptance Criteria

Each agent moves by inertia or social influence as explained above. Also each agent uses its own heuristic to search for a new candidate solution. But also each agent has its own acceptance criterion to accept the new candidate solution. The criteria used here are as follows:

- **Non-linear great deluge:** A new candidate solution is accepted if it is better or equal than the current solution. A candidate solution worse than the current solution will only be accepted if the penalty of the candidate solution is less than or equal to a pre-defined limit called *water level.* The acceptance level decrease over time at non-linear fashion decay rate, expressed by following formulation:

$$B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta \tag{8.4}$$

  When the penalty cost and the water level are about to converge the algorithm tends to become greedy. Then, it is necessary for the algorithm to relax and allow accepting worse solutions and this is achieved by increasing the water level with certain probability. This acceptance criterion is of course the one used in our non-linear great deluge algorithm from chapter 5.

- **Simulated annealing:** Worse solutions are accepted with certain probability expressed as $P : e^{-\delta/T}$, where $\delta = f(S^*) - f(S)$ and the parameter T denotes the current temperature. The search starts with higher temperature and it is reduced gradually towards the end of the search as $T_{i+1} = T_i - T_i * \beta$ (geometric cooling schedule). Reheating the temperature is also activated when the temperature is very low; the new generated solution is only accepted if it does not worsen the overall value of the current solution. Therefore, the temperature is increased to the last improvement temperature (the temperature where the last best solution was found).

We used the asynchronous cooperative multi-agent framework described above to implement two algorithms in this chapter. The difference between the two algorithms is in the complexity of the agents. In the first algorithm, each agent is a simple low-level heuristic. In the second algorithm, each agent is a hyper-heuristic type of approach. The following subsections give details of these proposed methods.

8.5.4 Asynchronous Cooperative Multi-heuristic Algorithm (ACMHA)

Each agent acts as a low-level heuristic to improve its own solution. The agents in the system cooperate in order to improve the global best solution. Each agent conducts its own search independently and asynchronously as explained above. Once an agent i spends a number of idle iterations (no improvement and unable to change its solution), the agent compares its current solution $x_{i,j}$ with the best global solution so far x_j^{gb} and if $x_{i,j}$ is better than x_j^{gb}, the global best is updated, otherwise, the agent discards its own solution and moves to another position using the velocity and position equations explained in section 8.5.1. All agents in the distributed environment repeat the same procedure until the system terminates the whole search process. Each agent decides whether to accept new solutions based on their own acceptance criterion.

8.5.5 Asynchronous Cooperative Multi-hyperheuristic Algorithm (ACMHHA)

Each agent acts as a low-level heuristic to improve their own solution. As above, the agents in the system cooperate in order to improve the global best solution. Each agent conducts its own search independently and asynchronously. Once an agent i

spends a number of idle iterations (no improvement and unable to change its solution), the agent compares its current solution $x_{i,j}$ to the best global solution so far x_j^{gb} and if $x_{i,j}$ is better than x_j^{gb}, the global best is updated, otherwise, the agent discards its own solution and moves to another position using the velocity and position equations explained in section 8.5.1. The difference here is that each agent is able to choose between the three type of low-level heuristics H1, H2 and H3. As suggested in [154], each agent in the distributed environment selects the low-level heuristic at random in every iteration. Each agent decides whether to accept new solutions based on their own acceptance criterion.

8.6 Experiments and Results

The main aim of these experiments is to evaluate the performance of the proposed asynchronous cooperative multi-agent search algorithms ACMHA(6) and ACMHHA(2). We used the Socha et al. problems instances and executed the proposed algorithms 20 times on each instance. The stopping condition for both algorithms was a maximum of 100 cycles or 10 hours of computation time whatever was first. In this proposed algorithms, a cycle occurs when all agents complete their tasks. Although 10 hours of computation time might seem too long for medium and large instances, we should note that our implementation is not truly parallel or multi-threaded, but a simulation on a sequential computer. Thus, the time taken to complete one cycle of the asynchronous cooperative search for the medium and large instances is somehow slow in such a sequential machine. In addition, we should remember that computational time is generally considered to be non-critical for the UCTTP and most algorithms reported in the literature spent hours solving these problems (e.g. Abdullah et al. [4, 6] ran their algorithms for ten and twelve hours per instance respectively). Another reason for the provision of long execution times is that we are mainly interested in investigating the performance of the proposed multi-agent systems and hence our decision in allowing sufficient computation time for the agents to conduct their search and collaboration. The agents in the ACMHA(6) (low-level heuristics) stop applying their low-level heuristic after a number of idle iterations, we set this maximum according to the size of the problem instance, 100 for large, 1000 for medium and 20,000 for small respectively. In these experiments, we compare the performance of our algorithms to the best results reported in the literature.

Results in Table 8.1 clearly show that the proposed algorithms find better results outperforming between them all other approaches in the literature including our previous algorithms in these thesis (see also [128, 101, 100]). We see that for small instances the proposed algorithms match the best results reported in the literature. For all small instances, ACMHA(6) and ACMHHA(2) find the optimal solution in all the 20 runs. For the medium instances, ACMHA(6) produced better solutions in four instances. In term of percentage improvement, the proposed algorithms reduced the penalty by 47.37%, 48.65%, 13.11%, 73.17% and 63.93% for instances M1, M2, M3, M4 and M5 respectively and with respect to our previous best results in this thesis. For the large instance, the cooperative algorithms also produced an improvement over our previous best result, in this case the improvement was of 6.11%.

It is obvious that the asynchronous cooperative multi-agent algorithms are beneficial since new best results are reported for all the medium and large instances. It is very noticeable that ACMHHA(2) finds a considerably much better solution for the most difficult instance (the large one) in which our previous algorithms in this thesis have failed.

In second set of experiments we investigate the performance of ACMHHA using different number of agents (2, 4 and 6). For each problem instance we executed the algorithms variants 20 times. The results obtained are shown in Table 8.2. We can

Table 8.1: Comparison of Results Obtained by the Asynchronous Cooperative Multi-agent Algorithms Proposed in this Chapter Against the Best Known Results from the Literature for the 11 Socha et al. Problem Instances.

	ACMHA(6)	ACMHHA(2)	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
S1	**0**	**0**	**0**	**0**	3	**0**	**0**	6	**0**	**0**	**0**	**0**
S2	**0**	**0**	**0**	1	4	**0**	**0**	7	**0**	**0**	**0**	**0**
S3	**0**	**0**	**0**	**0**	6	**0**	**0**	3	**0**	**0**	**0**	**0**
S4	**0**	**0**	**0**	**0**	6	**0**	**0**	3	**0**	**0**	**0**	**0**
S5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	4	**0**	**0**	**0**	**0**
M1	**20**	28	38	126	140	175	242	372	317	221	80	78
M2	**19**	27	37	123	130	197	161	419	313	147	105	92
M3	53	**43**	61	185	189	216	265	359	357	246	139	135
M4	**11**	28	41	116	112	149	181	348	247	165	88	75
M5	**22**	24	61	129	141	190	151	171	292	130	88	68
L	460	**415**	638	821	876	912	-	1068	-	529	730	556

ACMHA(6) is Asynchronous Cooperative Multi-heuristic with 6 agents.
ACMHHA(2) is Asynchronous Cooperative Multi-hyperHeuristic with 2 agents
A1 is Non-Linear Great Deluge Hyper-heuristic with LP=2500 [128].
A2 is Evolutionary Non-Linear Great Deluge [101].
A3 is Non-Linear Great Deluge [100].
A4 is Genetic algorithm and local search by Abdullah and Turabieh [7].
A5 is Randomised iterative improvement algorithm by Abdullah et al. [3].
A6 is Graph hyper heuristic by Burke et al. [29].
A7 is Variable neighbourhood search with tabu by Abdullah et al. [5].
A8 is Hybrid evolutionary approach by Abdullah et al. [4].
A9 is Extended great deluge by McMullan [116].
A10 is Great Deluge and Tabu Search by Abdullah et al. [6].
S1-S5 represent small problem instances 1 to 5.
M1-M5 represent medium problem instances 1 to 5.
L represents the large problem instance.

Table 8.2: Comparison of ACMH(6), ACMHHA(2), ACMHHA(4) and ACMHHA(6).

	ACMHA(6)		ACMHHA(2)		ACMHHA(4)		ACMHHA(6)	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
S1	**0**	0.75	**0**	0.1	**0**	**0**	0	**0**
S2	**0**	1.9	**0**	0.1	**0**	**0.05**	0	1
S3	**0**	0.85	**0**	**0**	**0**	**0**	0	1
S4	**0**	1.4	**0**	0.7	**0**	**0**	0	1
S5	**0**	**0**	**0**	**0**	**0**	**0**	0	**0**
M1	**20**	37.65	28	48.85	28	47.65	30	61
M2	**19**	31.95	27	48.45	30	50.55	32	41
M3	53	73.45	**43**	71.45	66	88.4	64	102
M4	**11**	26.8	28	39.1	23	37.75	24	51
M5	22	38.45	**24**	50.1	34	58.35	32	45
L	460	675.5	**415**	510.1	450	527.4	468	560

ACMHA(6) is Asynchronous Cooperative Multi-heuristic with 6 agents.
ACMHHA(2) is Asynchronous Cooperative Multi-hyperheuristic with 2 agents.
ACMHHA(4) is Asynchronous Cooperative Multi-hyperheuristic with 4 agents.
ACMHHA(6) is Asynchronous Cooperative Multi-hyperheuristic with 6 agents.
Avg is Average over the 20 runs.
S1-S5 represent small problem instances 1 to 5.
M1-M5 represent medium problem instances 1 to 5.
L represents the large problem instance.

see that all ACMHHA variants managed to generate optimal solution at least in one out of the 20 runs for all small instances. However, for medium instances ACMHA(6) outperform the ACMHHA variants in 3 instances. For the large instance, the 2 agents approach ACMHHA(2) outperforms the other algorithms.

In the third set of experiments, we compare the performance of ACMHA(6) and ACMHHA with three variants to the best results reported in the literature instance by instance. Columns 2 to 5 in Table 8.3 show the best results obtained by ACMHA(6) and ACMHHA with three variants while column 6 shows the best known results and the corresponding approaches for each problem instances. It should be noted that although a timetable with zero penalty exists for each problem instance (the data sets were generated starting from such a timetable [152]), to the best of our knowledge no heuristic method has found the ideal timetable for the medium and large

Table 8.3: Comparison of the Results Obtained by the Asynchronous Cooperative Multi-agent Algorithms Proposed in this Chapter Against the Best Results Reported in the Literature.

	ACMHA(6)	ACMHHA(2)	ACMHHA(4)	ACMHHA(6)	Best Known
S1	**0**	**0**	**0**	**0**	**0**(VNS-T)
S2	**0**	**0**	**0**	**0**	**0** (VNS-T)
S3	**0**	**0**	**0**	**0**	**0** (CFHH)
S4	**0**	**0**	**0**	**0**	**0** (VNS-T)
S5	**0**	**0**	**0**	**0**	**0** (MMAS)
M1	**20**	28	28	30	80 (EGD)
M2	**19**	27	30	32	105 (EGD)
M3	53	**43**	66	64	139 (EGD)
M4	**11**	28	23	24	88 (EGD)
M5	**22**	24	34	32	88 (EGD)
L	460	**415**	450	468	529 (HEA)

MMAS is the MAX-MIN Ant System in [152].
CFHH is the Choice Function Hyper-heuristic in [27].
VNS-T is the Hybrid of VNS with Tabu Search in [5].
HEA is the Hybrid Evolutionary Algorithm in [3].
EGD is the Extended Great Deluge in [116].
S1-S5 represent small problem instances 1 to 5.
M1-M5 represent medium problem instances 1 to 5.
L represents the large problem instance.

instances. Hence, these data sets are still very challenging for heuristic search methods. For all small instances, both approaches ACMHA(6) and ACMHHA with three variants produce optimal solutions. For medium instances, they produce better solutions compared to the best results in the literature. For large instance ACMHHA(2) does improve the previous best known result (by HEA) by 21.55%. More interestingly, ACMHA(6), ACMHHA(4)and ACMHHA(6) also produce better solutions for medium and large instances compared to the best results found in the literature.

The reasons of using different number of agents in this experiments are :

- We employ six agents in ACMHA because there are only three heuristics and two different criteria applied. For that reasons, we set three agents with different heuristic for each acceptance criterion. Meaning that three agents with different heuristic with acceptance criterion of great deluge and three more agents with different heuristic with simulated annealing acceptance criterion.

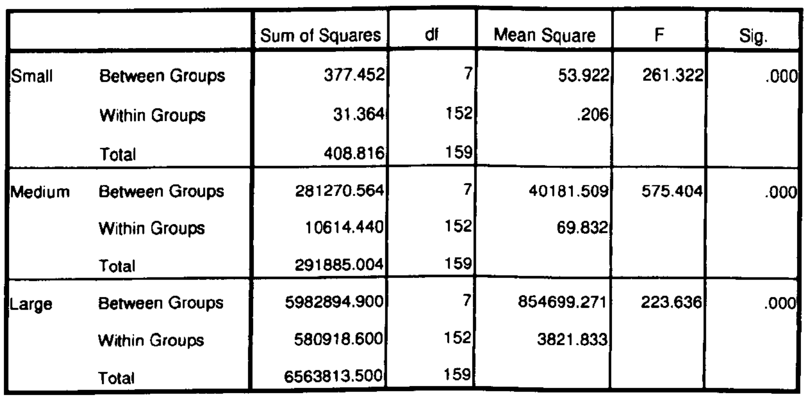
- For ACMHHA, since each agent able to perform all three different types of heuristics, therefore, we tested different number of agents (two, four and six). By employing different number of agents, we found that each different number of agents performed differently. Based on the experimental results, we found that, the more agents we employed the lower the quality of the solutions become. We only employed three different number of agents (two, four and six) as the more agents we employed the longer the time was needed to accomplished the job as the implementation of cooperative search was not truly parallel rather than a simulation of the parallel processing.

8.6.1 Statistical Analysis

We also carried out a statistical analysis to statistically examine the performance of ACMHA(6), ACMHHA with three variants and the previous algorithms proposed earlier in this thesis (also published in [100], [101] and [128]). Table 8.6 and 8.4 presents the average results for three categories of instances namely, small, medium and instances. The values are obtained from the average results for each category. First, we want to inspect whether all the proposed algorithms have different performance towards the sizes of the instances. As a normal procedure we run the descriptive analysis to check the model's compatibility with the sample data and found that ANOVA is suitable statistical tool to measure the differences among the proposed algorithms.

Figure 8.2 shows that there is a significant difference between the ACMHA(6), ACMHHA with three variants and our previously proposed algorithms,the p-value is very close to zero. This analysis does not show the detail whether all algorithms are different between them. Therefore we used Post Hoc analysis to inspect the difference between pairs of algorithms. The results are presented in Table 8.7, Table 8.8 and

Table 8.9.



		Sum of Squares	df	Mean Square	F	Sig.
Small	Between Groups	377.452	7	53.922	261.322	.000
	Within Groups	31.364	152	.206		
	Total	408.816	159			
Medium	Between Groups	281270.564	7	40181.509	575.404	.000
	Within Groups	10614.440	152	69.832		
	Total	291885.004	159			
Large	Between Groups	5982894.900	7	854699.271	223.636	.000
	Within Groups	580918.600	152	3821.833		
	Total	6563813.500	159			

Figure 8.2: ANOVA Results

	ACMHA(6)			ACMHHA(2)			ACMHHA(4)		
Run	Small	medium	Large	Small	medium	Large	Small	medium	Large
1	0.2	41	677	0	57	415	0	50.2	542
2	0	42.4	668	0	50.8	526	0	58	587
3	0	45.4	768	0	44.8	504	0	59.2	609
4	0	40.6	597	0.4	50.8	528	0	49.6	594
5	0	36.4	580	0	51.4	543	0	53.8	596
6	0	43.4	734	0	53.8	551	0	54.8	489
7	0	47.8	460	0	58	518	0	63.6	621
8	0.4	41.4	611	0	58	474	0	58.2	489
9	0	40.6	830	0.4	61	561	0	47.6	450
10	0.2	41.8	585	0	64.6	453	0	56	484
11	0	43.2	663	0.6	53.6	425	0.2	61.8	450
12	0	38.6	784	0.2	63.8	506	0	51.2	504
13	0.2	41.8	681	0	45.8	578	0	58.4	577
14	0.4	43.4	691	0.2	45	485	0	55.4	477
15	0.2	42.8	765	0.2	43.4	629	0	65	589
16	0.2	42.2	792	0.6	49.2	512	0	54.4	518
17	0.2	41.4	773	0.2	46	542	0	62.8	457
18	0.4	37.4	572	0.4	51	473	0	58.2	527
19	0	37.2	550	0.2	44.6	452	0	53.6	505
20	0	46.4	599	0.2	39.2	527	0	59	483

Table 8.4: Average Penalty Cost of ACMHA(6), ACMHHA(2) and ACMHHA(4) Across the 11 Socha et al. Instances.

	ACMHHA(6)			NLGDHH-SM (LP=2500)			NLGDHH-SM (LP=5000)		
Run	Small	medium	Large	Small	medium	Large	Small	medium	Large
1	0.2	54.4	562.0	0.0	61.0	870.0	0.0	63.4	813.0
2	0.2	55.4	484.0	0.0	65.6	827.0	0.6	61.2	824.0
3	0.2	54.4	497.0	0.0	63.4	739.0	0.0	61.8	911.0
4	0.2	51.0	484.0	0.6	61.8	905.0	0.0	58.0	786.0
5	0.2	61.4	540.0	0.0	62.8	806.0	0.0	58.8	958.0
6	0.4	59.2	493.0	0.2	64.2	936.0	0.0	54.4	962.0
7	0.2	55.4	530.0	1.0	56.8	918.0	0.6	63.8	813.0
8	0.0	55.8	526.0	0.2	61.2	800.0	1.8	68.0	824.0
9	0.0	57.8	527.0	0.6	56.8	813.0	0.0	64.8	853.0
10	0.0	55.0	523.0	0.0	62.6	638.0	0.0	64.2	813.0
11	0.0	51.8	531.0	0.0	61.2	843.0	0.6	64.6	824.0
12	0.2	58.0	611.0	0.2	67.0	817.0	0.8	62.6	880.0
13	0.2	57.2	547.0	1.0	60.8	845.0	0.2	64.2	914.0
14	0.8	49.8	515.0	1.0	59.4	817.0	0.2	55.4	879.0
15	0. 0	56.4	539.0	0.6	59.0	912.0	0.0	64.4	713.0
16	0.0	53.6	540.0	1.0	58.6	857.0	0.4	66.0	1030.0
17	0.0	52.8	546.0	0.0	57.2	855.0	0.0	60.4	1083.0
18	0.6	60.0	560.0	0.4	63.4	788.0	0.2	61.2	985.0
19	0.2	60.6	530.0	0.0	56.6	725.0	0.0	61.2	813.0
20	0.0	50.2	468.0	0.0	60.8	870.0	0.0	55.6	824.0

Table 8.5: Average Penalty Cost of ACMHHA(6), NLGDHH-SM (LP=2500) and NLGDHH-SM (LP=5000)) Across the 11 Socha et al. Instances.

First of all we want to examine if the proposed asynchronous cooperative searches are significantly different than each other. Table 8.7, Table 8.8 and Table 8.9 show that for small instances most of the cooperative searches have similar performance

	NLGDHH-RCLR			ENLGD-2			NLGD		
Run	Small	medium	Large	Small	medium	Large	Small	medium	Large
1	1.4	119.6	995.0	0.8	95.6	703.0	3.8	142.4	966.0
2	1.8	110.6	993.0	0.4	85.8	927.0	4.8	165.0	1070.0
3	2.0	89.6	1017.0	0.4	95.4	835.0	6.0	165.6	876.0
4	0.4	110.6	1112.0	0.4	93.6	968.0	5.2	162.2	935.0
5	3.0	112.2	915.0	0.4	108.6	895.0	5.0	165.2	971.0
6	2.4	92.0	1062.0	0.4	99.8	730.0	4.6	166.8	942.0
7	2.2	88.0	1008.0	0.2	81.2	782.0	5.0	165.4	895.0
8	2.4	109.8	1014.0	0.4	91.6	711.0	5.2	156.8	976.0
9	2.8	101.8	1036.0	0.8	110.4	777.0	5.4	160.4	986.0
10	0.8	99.0	1011.0	1.0	96.4	838.0	5.4	172.8	1005.0
11	2.6	87.2	1153.0	0.4	96.6	808.0	3.8	185.0	966.0
12	0.8	108.6	1130.0	1.0	98.4	944.0	4.0	171.6	1070.0
13	2.6	92.0	1010.0	0.8	91.2	870.0	4.2	177.0	935.0
14	2.8	102.0	1146.0	0.4	96.4	807.0	4.2	181.0	1024.0
15	3.8	118.6	1088.0	0.8	83.6	849.0	4.0	172.4	942.0
16	2.8	92.4	917.0	1.2	90.6	713.0	5.0	188.4	958.0
17	1.4	121.0	996.0	0.4	117.8	852.0	4.2	179.6	978.0
18	1.4	106.8	1012.0	0.6	102.2	795.0	5.4	182.6	1005.0
19	1.4	134.0	1065.0	1.6	106.0	779.0	5.4	196.0	1078.0
20	0.6	76.6	1025.0	0.8	89.4	801.0	5.0	183.8	907.0

Table 8.6: Average Penalty Cost of NLGDHH-RCLR , ENLGD-2 and NLGD Across the 11 Socha et al. Instances.

except for ACMHHA(2) and ACMHHA(4) where the p-value is less than significant level at 0.05. For medium instances ACMHHA(2), ACMHHA(4) and ACMHHA(6) have similar performance across instances. However, the performance of ACMHHA with different number of agents is very different to the performance of ACMHA(6), with p-value close to zero. Finally, for the large instance, the Post Hoc tests show that all three types ACMHHA have similar performance, but their performance are really different compared to ACMHA(6) with p-value close to zero.

Now we compare the performance of all asynchronous cooperative searches to our previous algorithms (NLGDHH-SM, NLGDHH-RCLR, ENLGD-2 and NLGD). Based on the results shown in Table 8.7, Table 8.8 and Table 8.9, we found the pairs (ACMHA(6), NLGDHH-SM), (ACMHHA(2),NLGDHH-SM),(ACMHHA(6),NLGDHH-SM) and (NLGDHH-SM,ENLGD-2) having similar performance on small instances. In medium instances, we found that only NLGDHH-RCLR and ENLGD-2 have similar performance, whereas the rest of the algorithms have different performance with p-value less than the significant level at 0.05. For the large instance case, results indicate that pairs (ACMHA(6),ENLGD-2), (NLGDHH-SM,ENLGD-2) and (NLGDHH-SM,NLGD-2) have similar performance.

	ACMHA (6)	ACMHHA (2)	ACMHHA (4)	ACMHHA (6)	NLGDHH-SM (LP=2500)	NLGDHH-SM (LP=5000)	NLGDHH -RCLR	ENLGD -2	NLGD
ACMHA(6)	-	1.000	0.135	1.000	0.566	0.989	0.000	0.000	0.000
ACMHHA(2)	1.000	-	0.048	1.000	0.967	1.000	0.000	0.000	0.000
ACMHHA(4)	0.135	0.048	-	0.069	0.048	0.377	0.000	0.000	0.000
ACMHHA(6)	1.000	1.000	0.069	-	0.971	1.000	0.000	0.000	0.000
NLGDHH-SM (LP=2500)	0.566	0.967	0.048	0.971	-	1.000	0.000	0.284	0.000
NLGDHH-SM (LP=5000)	0.989	1.000	0.377	1.000	1.000	-	0.000	0.122	0.000
NLGDHH-RCLR	0.000	0.000	0.000	0.000	0.000	0.000	-	0.000	0.000
ENLGD-2	0.000	0.000	0.000	0.000	0.000	0.284	0.122	-	0.000
NLGD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-

Table 8.7: Post Hoc Tests - Small Instances.

	ACMHA (6)	ACMHHA (2)	ACMHHA (4)	ACMHHA (6)	NLGDHH-SM (LP=2500)	NLGDHH-SM (LP=5000)	NLGDHH	ENLGD -RCLR	NLGD -2
ACMHA(6)	-	0.000	0.000	0.000	0.000		0.000	0.000	0.000
ACMHHA(2)	0.000	-	0.353	0.619	0.000	0.000	0.000	0.000	0.000
ACMHHA4)	0.000	0.353	-	1.000	0.040	0.018	0.000	0.000	0.000
ACMHHA(6)	0.000	0.619	1.000	-	0.000	0.000	0.000	0.000	0.000
NLGDHH-SM (LP=2500)	0.000	0.000	0.040	0.000	-	1.000	0.000	0.000	0.000
NLGDHH-SM (LP=5000)	0.000	0.000	0.018	0.000	1.000	-	0.000	0.000	0.000
NLGDHH-RCLR	0.000	0.000	0.000	0.000	0.000	0.000	-	0.861	0.000
ENLGD-2	0.000	0.000	0.000	0.000	0.000	0.000	0.861	-	0.000
NLGD	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-

Table 8.8: Post Hoc Tests - Medium Instances.

	ACMHA (6)	ACMHHA (2)	ACMHHA (4)	ACMHHA (6)	NLGDHH-SM (LP=2500)	NLGDHH-SM (LP=5000)	NLGDHH -RCLR	ENLGD -2	NLGD
ACMHA(2)	-	0.000	0.000	0.000	0.015	0.000	0.000	0.070	0.000
ACMHHA(2)	0.000	-	1.000	0.998	0.000	0.000	0.000	0.000	0.000
ACMHHA(4)	0.000	1.000	-	1.000	0.000	0.000	0.000	0.000	0.000
ACMHHA(6)	0.000	0.998	1.000	-	0.000	0.000	0.000	0.000	0.000
NLGDHH-SM (LP=2500)	0.015	0.000	0.000	0.000	-	0.911	0.000	1.000	0.000
NLGDHH-SM (LP=5000)	0.000	0.000	0.000	0.000	0.911	-	0.000	0.713	0.008
NLGDHH-RCLR	0.000	0.000	0.000	0.000	0.000	0.000	-	0.000	0.099
ENLGD-2	0.070	0.000	0.000	0.000	1.000	0.713	0.000	-	0.000
NLGD	0.000	0.000	0.000	0.000	0.000	0.008	0.099	0.000	-

Table 8.9: Post Hoc Tests - Large Instance.

As the Post Hoc tests indicate, some algorithms have similar and some have different performance. However at this point the analysis does not tell us which algorithm is actually outperforming best across all instances. We refer to the mean plot of each algorithm at 95% confidence level shown in Figure 8.3, Figure 8.4 and Figure 8.5. In Figure 8.3 shows that for small instances ACMHHA(4) outperformed all the other algorithms. For medium instances, Figure 8.4 shows that ACMHA(6) outperformed ACMHHA(2), ACMHHA(4), ACMHHA(6), NLGDHH-SM, NLGDHH-RCLR, ENLGD-2 and NLGD. Figure 8.5 shows the means plot for the large and it shows that even though ACMHHA(2) produced the best results this instance, on average ACMHHA(2), ACMHHA(4) and ACMHHA(6) actually have similar performance (more than 50% of their solutions have similar quality). Overall, the mean plot shows that ACMHHA(4) outperforms the other algorithms for small instances. ACMHA(6) shows the best performance across the medium instances. In addition, we found that the mean plot indicates that ACMHHA(2), ACMHHA(2) and ACMHHA(2) have similar performance where it can be said that around 50% of their solutions are of similar quality. Finally, for the large instance, three algorithms have similar performance namely ACMHHA(2), ACMHHA(4) and ACMHHA(6) outperforming the other algorithms.

The statistical analysis presented here suggests that some of the algorithms have similar and some have different performance according to the size of the problem instances. It is also evident that the asynchronous cooperative multi-agent search, inspired in particle swarm optimisation, is a very good strategy as it managed to further improve the solutions found with our previous algorithms and produce best known results for the medium and large instances.

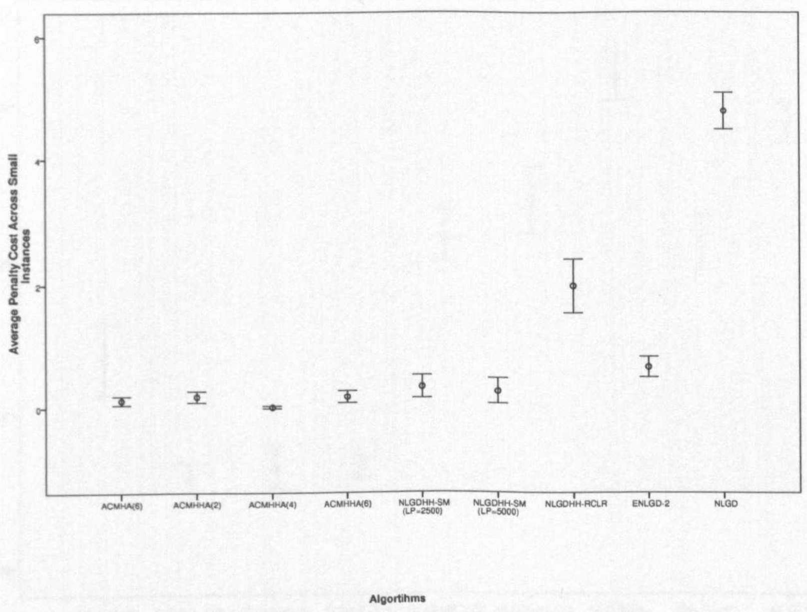


Figure 8.3: Min Plot and LSD Interval (Small Instances).

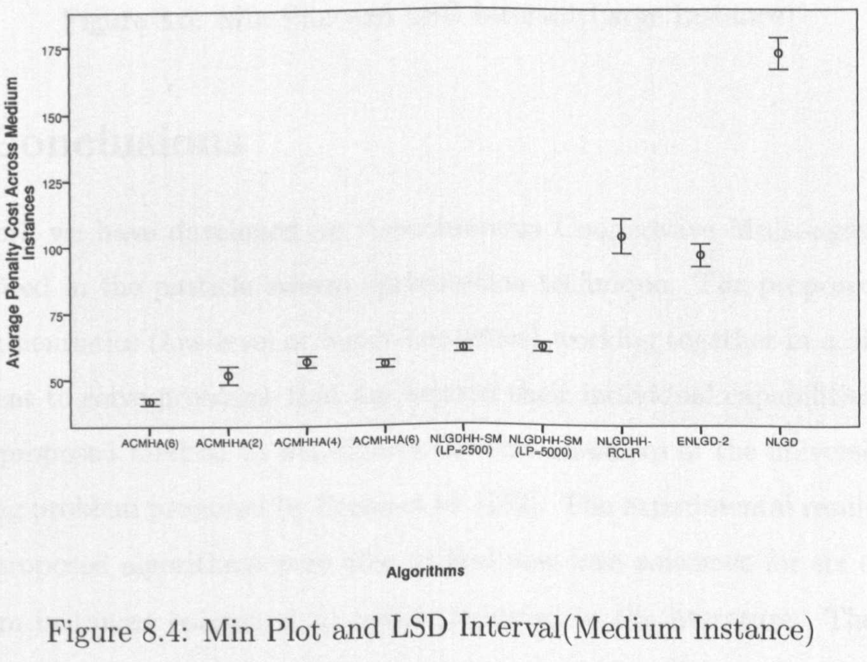


Figure 8.4: Min Plot and LSD Interval(Medium Instance)

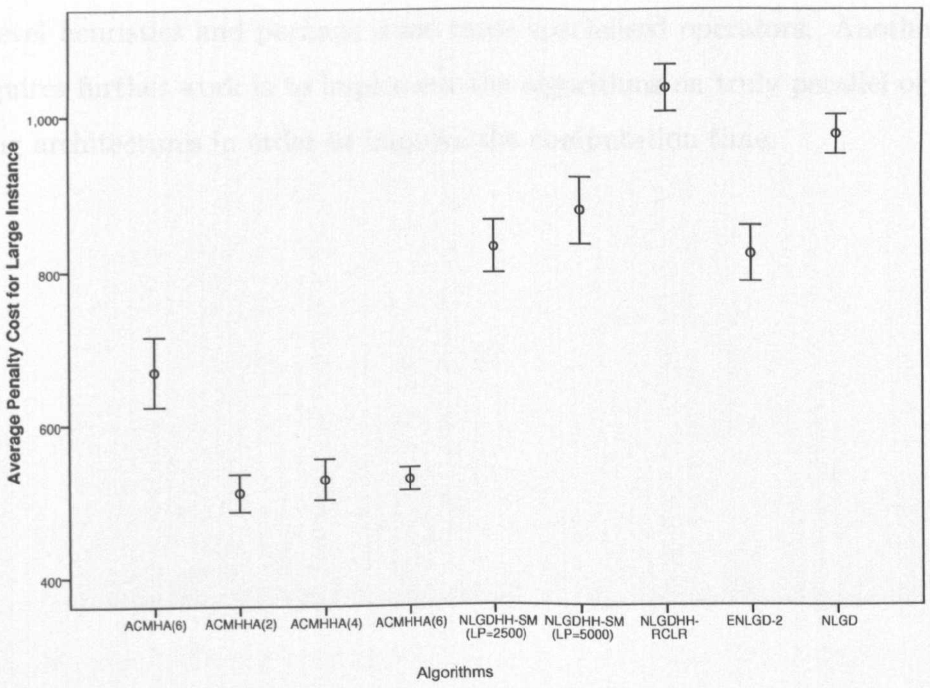


Figure 8.5: Min Plot and LSD Interval(Large Instance)

8.7 Conclusions

In this work we have developed an Asynchronous Cooperative Multi-agent Framework inspired in the particle swarm optimisation technique. The proposed method focuses on heuristics (low-level or hyper-heuristics) working together in a distributed environment to solve problems that are beyond their individual capabilities. We applied the proposed method to well-known difficult instances of the university course timetabling problem proposed by Socha et al. [152]. The experimental results showed that the proposed algorithms were able to find new best solutions for six out of the 11 problem instances compared to results reported in the literature. The propose algorithm variants are able to improve upon the best known solution for the most difficult problem, the large instance. Future work contemplates the decomposition of large problems into smaller ones. We also want to incorporate a larger number of low-level heuristics and perhaps some more specialised operators. Another issue that requires further work is to implement the algorithms on truly parallel or multi-threading architectures in order to improve the computation time.

Chapter 9

Conclusions and Future Work

In order to draw the conclusions of the study, this chapter sums up the major developments from the investigation presented in this thesis. As it was stated in chapter one, the aim and scope of this thesis was to study several approaches in a two-stage optimisation strategy, initialisation and minimisation of soft constraint violations, to tackle the university course timetabling problem. The purpose was to develop novel meta-heuristic, hyper-heuristic and cooperative search techniques capable of obtaining high-quality solutions for this difficult problem. The organisation of this chapter is as follows. Section 9.1 gives a summary of the research work carried out and highlights the overall contribution of this thesis. Section 9.2 outlines suggested further research directions that may be undertaken.

9.1 Research Work Summary

We investigated several approaches from the initialisation of feasible solutions (satisfy all hard constraints) to the improvement of solutions (minimise soft constraint violations) for the university course timetabling problem. This research also examined how cooperative low-level heuristics and cooperative hyper-heuristics improve the robustness and effectiveness of solution methods for the university course timetabling problems. Chapter 1 presented the outline and the aims of this investigation.

A comprehensive literature review of different optimisation techniques to tackle course timetabling problems was presented in Chapter 2 to identify previous relevant work in this area and gaps in the knowledge that could be investigated in this thesis. The review showed that the university course timetabling problem is extremely complex and is an NP-complete problem as there is no known efficient deterministic algorithm which will solve all instances in polynomial time. The review also revealed that since different educational institutions have different hard and soft constraints in their timetabling problems, this makes it more complex to model and tackle the various timetabling problems that exist in reality. Besides that, the introduction of a modular course structure has made the construction of timetables even more complex since students have much flexibility to enrol in almost any combination of courses.

Chapter 3 gave a detailed description of the university course timetabling problem (UCTTP) and the standard benchmark instances used in this thesis. These benchmark problem instances are inspired by real-world problems although they are still simplified. To the best of our knowledge, these problem instances are still very relevant as no heuristic method has found the optimal solutions for many of those benchmark instances.

According to reports in the literature, the sequential heuristic method has proven to be good at generating initial solutions for timetabling. However, that method alone does not guarantee the generation of feasible solutions even when combining of more than one heuristics. Abdullah and colleagues [5, 4] have employed the sequential heuristic approach, however, they failed to generate feasible solutions especially for large instances. Moreover, no details of the computation time taken by that sequential heuristic were reported by Abdullah et al. Thus, the main goal of Chapter 4 was to experiment with several hybrid approaches for the construction of initial and feasible timetables. A detailed analysis of the proposed approaches was also presented in that chapter.

Chapter 5 presented an investigation of the great deluge algorithm applied to the UCTTP. The aim of this chapter was to inspect in detail the performance of this simple yet effective method. We first extended that algorithm from using the traditional linear decay rate to using a non-linear decay rate with floating water level. We studied and compared the performance of the original great deluge and our extended one. In the original great deluge the decay rate is pre-determined and fixed, so the search is driven by the water level and when the best solution and water level converge the algorithm becomes greedy. In our proposed extended great deluge, the decay rate changes in every iteration according to the quality of the best solution so far. That is, the water level is driven by the search and the extended great deluge never becomes greedy. Our non-linear great deluge produced high quality solutions at reasonable computational time for the UCTTP.

Chapter 6 presented a hybrid evolutionary non-linear great deluge algorithm. The proposed approach is an extension of our non-linear great deluge algorithm. Several evolutionary features such as a population and mutation operator were incorporated. First, we generate a population of feasible solutions using our construction heuristics from chapter 4. Then, the population of feasible timetables is subject to a steady-state evolutionary process that combines mutation and stochastic local search. We evaluate the performance of the proposed evolutionary hybrid algorithm and in particular, the contribution of the evolutionary operators. The results showed that the hybrid between non-linear great deluge and evolutionary operators produces very good results on the instances of the UCTTP.

Chapter 7 proposed a great deluge hyper-heuristic framework. The aim of this research was to develop a hyper-heuristic that employed learning mechanism for the selection of low-level heuristics and a non-linear great deluge acceptance criterion. The hyper-heuristic deals with complete solutions i.e. feasible timetables are produced first using our initialisation methods. The low-level heuristics are local search operators which operate in the solution space. We choose the non-linear great deluge as a high level heuristic because of the simplicity of the algorithm itself, great deluge is less dependent upon parameter tuning compared to simulated annealing.

Chapter 8 investigated the development of asynchronous cooperative multi-agent methods inspired in the particle swarm optimisation algorithm. Two variants were implemented here, one in which the agents are low-level heuristics and another one on which the agents are hyper-heuristics. When the agents cooperate, they might follow their own inertial movement or follow other better positioned agents. The proposed asynchronous cooperative multi-agent algorithms produced the best results so far for all the medium and large instances of the Socha et al. datasets. To the best of our knowledge, these are the best results reported so far for this set of difficult benchmark problems.

9.1.1 Contributions

A number of contributions have been made as a result of the research work in this thesis. They are identified and presented according to their merits:

- Proposed a modification of the great deluge algorithm (using a non-linear decay rate and floating water level): the algorithm is able to produce good results for all medium instances and competitive for small and large instances.
- Developed asynchronous cooperative multi-agent algorithms: showed that these approaches are very effective for the university course timetabling problem,

findings new best solutions for the 5 medium and the 1 large instance of the Socha et al [152] datasets. On average, the asynchronous cooperative multi-agent search improved all 11 instances and proved to be the best approach so far.

- Investigated a learning mechanism for the non-linear great deluge, resulting in a hyper-heuristic method: the learning mechanism helped the low-level heuristics to produce improve the best known solutions at the time.
- Proposed a hybridisation of non-linear great deluge with elements of evolutionary algorithms: this algorithm managed to improve solutions for the five medium size instances of the Socha et al. problems and matched best known solutions for all the small instances.
- Constructed feasible course timetables by hybrid heuristics (graph colouring, local search and tabu search): we developed a set of methods that produce feasible solutions in reasonable computation time for all the Socha et al. and also the ITTC 2002 problem instances.

9.2 Discussion and Future Work

In general scheduling problems include a wide range of combinatorial optimisation problem and in fact, university course timetabling problems belong to the family of scheduling problems. Scheduling problem can be defined as the process of arranging a set of entity such as people, task, vehicle, exam, course, etc to limited resources in such a way that all predefined hard constraints are satisfied and soft constraints are minimised to achieve desirable objectives [163]. According to Ferland and Fleurent [76] many scheduling problems have common features with timetabling problems, for example sport leagues games scheduling, nurse rostering, examination timetabling, school timetabling, crew scheduling and Transport scheduling. Even though, the focus of this thesis was to development algorithms for university course timetabling problems, the algorithms developed in this thesis can be applied to a different types of scheduling problems. As shown in this research work, the ideas for solutions techniques for course timetabling problem that investigated in this thesis will be profitably exploited for other scheduling problems as they share common features with course timetabling studied in this thesis. Overall, our proposed algorithms produced good results for course timetabling problem. Therefore, it is always worth considering other different scheduling problem which share common features and structure with course timetabling problem.

This thesis used a two-stage approach, initial feasible solution construction followed by soft constraint violations minimisation, to tackle the university course timetabling methodology. We spent much of our time investigating the second stage of the approach, the improvement of timetables. We developed several new algorithms that produced best known solutions for most of the problem instances used. However, there are still a few issues that can be addressed in future research work.

One worthwhile future endeavour would be to investigate if the various algorithms described here are also able to tackle other range of problems. It would also be interesting to consider proper real-world problem instances of the UCTTP. It might also be a good idea to extend the proposed algorithms into multi-objective approaches.

We also proposed to test the non-linear great deluge approach on other instances of course timetabling problems available in the literature and other related timetabling problems, such as examination timetabling or school timetabling. We also suggest to investigate mechanisms to automatically adapt the non-linear decay rate to the size of the problem being tackled. Our algorithm is able to find good quality feasible solutions, however, it takes long time to do that for the large instance. It is also interesting to investigate a population-based version of the non-linear great deluge algorithm taking into consideration the diversity among a set of timetables.

Since our evolutionary hybrid algorithm does not check similarity of solutions during the replacement of individuals in the pool of solutions, some of the solutions in the pool might be the same. Future work could investigate the similarity of solutions in the pool of solutions. Therefore the algorithm could take into consideration the diversity in the population to better conduct the search.

Learning mechanisms such as supervised learning and unsupervised learning have been applied in machine learning and produced very promising results. Another worthwhile future endeavour could be to investigate if different learning mechanisms such as q-learning and inspect can help to select the right low-level heuristic at every decision point. A good learning mechanism is said to be intelligent enough to select good heuristics and discard bad heuristics. Therefore, it would be interesting to incorporate large number of low-level heuristics and develop a learning mechanism that reacts and balances intensification and diversification while selecting the low-level heuristics.

Finally, in chapter 8 we presented the asynchronous cooperative low-level heuristics and hyper-heuristics. The main issue in that work is the amount of computation time taken to complete the circle of communication in these algorithms. Although in real-world timetabling problems computational time is generally considered not very critical, still we propose to implement our algorithms on parallel or multi-threading computers to reduce the computation time and investigate other mechanisms for exchanging information among the agents. In addition, it might be worth investigating different models of cooperative search such as central memory model, diffusion communication scheme, low level parallelisation and search space decomposition.

References

[1] Website of the metaheuristics network. In *http://www.metaheuristics.org (Website of the Metaheuristics Network)*.

[2] E. Aarts, J. E. Korts, and W.Michiels. Simulated Annealing. In E. Burke and G. Kendall, editors, *Search Methodology*, pages 187–210. Springer, 2005.

[3] S. Abdullah, E. Burke, and B. McCollum. A Hybrid Evolutionary Approach to the University Course Timetabling Problem. In *proceedings of CEC: The IEEE Congress on Evolutionary Computation*, pages 1764–1768, 2007.

[4] S. Abdullah, E. Burke, and B. McCollum. Using a Randomised Iterative Improvement Algorithm with Composite Neighborhood Structures for University Course Timetabling. *Metaheuristics - Progress in Complex Systems Optimization*, pages 153–172, 2007.

[5] S. Abdullah, E. K. Burke, and B. McCollum. An Investigation of Variable Neighbourhood Search for University Course Timetabling. In *The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, NY, USA*, pages 413–427, 2005.

[6] S. Abdullah, K. Shaker, B. McCollum, and P. McMullan. Construction of Course Timetables Based on Great Deluge and Tabu Search. In *MIC 2009: The VIII Metaheuristics International Conference*, 2009.

[7] S. Abdullah and H. Turabieh. Generating University Course Timetable Using Genetic Algorithms and Local Search. In *The Third International Conference on Convergence and Hybrid Information Technology ICCIT*, volume I, page 25, 2008.

[8] D. H. Ackley. A Connectionist Machine for Genetic Hillclimbing. *Kluwer Academic Press, Boston*, 1987.

[9] M. A. Al-Betar and A. T. Khader. A hybrid search for university course timetabling. In *Multidisiplinary International Conference on Scheduling : Theory and Application (MISTA 2009)*, pages 157–179, 2009.

[10] A. Alkan and E. Ozcan. Memetic Algorithms for Timetabling. In *proceeding IEEE Congress on Evolutionary Computation*, pages 1796–1802, December 2003.

[11] R. Alvarez-Valdes, E. Crespo, and J. Tamarit. Assigning Students Sections Using Tabu Search. *Annals of Operations Research*, 96:1–16, 2000.

[12] R. Alvarez-Valdes, E. Crespo, and J. Tamarit. Design and Implementation of a Course Scheduling Systems Using Tabu search. *Production, Manufacturing and Logistics. European Journal of Operational Research*, 137:512–523, 2002.

[13] H. Arntzen and A. Løkketangen. A Local Search Heuristic for a University Timetabling Problem. In *http://www.idsia.ch/Files/ttcomp2002/arntzen.pdf.*, 2003=2.

[14] H. Arntzen and A. Løkketangen. A Tabu Search Heuristic for a University Timetabling Problem. *In Metaheuristics: Progress as Real Problem Solvers*, Computer Science Interfaces Series 32:65–86, 2005.

[15] H. Asmuni, E. Burke, and J. Garibaldi. Fuzzy Multiple Heuristic Ordering for Course Timetabling. In *proceedings of the 5th United Kingdom Workshop on Computational Intelligence UKCI*, 2005.

[16] T. Back, F. Hoffmeister, and H. Schwefel. A Survey of Evolution Strategies. In *proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, 1991.

[17] R. Bai, E. K. Burke, G. Kendall, and B. McCollum. Memory Length in Hyper-heuristics: An Empirical Study. In *proceedings of the IEEE Symposium on Computational Intelligence in Scheduling CISched*, Hilton Hawaiian Village, Honolulu, Hawaii, USA, April 2007.

[18] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. Technical report, Technical Report TR/IRIDIA/2001-13, IRIDIA, Belgium, 2001.

[19] P. Borovska. Efficiency of Parallel Metaheuristics for Solving Combinatorial Problems. In *proceeding International Conference on Computer Systems and Technologies - CompSysTech07*, 2007.

[20] D. Brelaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, 22(4):251–256, 1979.

[21] H. Bremmermann. *Optimisation Through Evolution and Re-Combination.* Spartan Books, 1962.

[22] E. Burke, Y. Bykov, J. P. Newall, and S. Petrovic. A Time-predefined Approach to Course Timetabling. *Yugoslav Journal of Operations Research (YUJOR)*, 13 No. 2:139–151, 2003.

[23] E. Burke, A. Eckersleym, B. McCollum, S. Petrovic, and Q. Rong. A Hybrid Variable Neighbourhood Approaches to University Exam Timetabling. Tech-

nical report, OTTCS-TR-2006-2, University of Nottingham, School of CSiT, 2006.

[24] E. Burke, D. Elliman, P. Ford, and R. Weare. Examination Timetabling in British Universities: A Survey. In E. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT '95). Lecture Notes in Computer Science*, volume 1153/1996, pages 76–90. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[25] E. Burke, K. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Hyper-Heuristic: An Emerging Direction in Modern Search Technology.* Handbook of Meta-heuristic F. Glover (ed), Kluwer, 2003.

[26] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. A survey of hyper-heuristic. Technical report, School of Computer Science Univeristy of Nottingham, 2009.

[27] E. Burke, G. Kendall, and E. Soubeiga. A Tabu-search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9:451–470, 2003.

[28] E. Burke and J. Landa-Silva. The Design of Memetic Algorithms for Scheduling and Timetabling Problems. In E. Willaim, N. Krasnogor, and J. Smith, editors, *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, volume 166, pages 289–312. Springer, 2004.

[29] E. Burke, B. McCollum, A. Meisels, S. Petrovic, and Q. Rong. A Graph Based Hyper-heuristic for Educational Timetabling Problems. *European Journal of Operational Research*, 176:177–192, 2007.

[30] E. Burke, J. Newall, and R. Weare. A Memetic Algorithm for University Exam Timetabling. In *Lecture Notes in Computer Science; Archive Selected papers from the First International Conference on Practiceand Theory of Automated Timetabling*, volume 1153. Springer-Verlag London, UK, 1995.

[31] E. Burke, S. Petrovic, and R. Qu. Case Based Heuristic Selection for Timetabling Problems. *Journal of Scheduling*, 9 (2):115–132, 2006.

[32] E. K. Burke, G. Kendall, J. D. Landa-Silva, R. O'Brien, and E. Soubeiga. An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem. In *proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2263–2270, Edinburgh, Scotland, 2005.

[33] E. K. Burke and J. P. Newall. A Multi-Stage Evolutionary Algorithm for the Timetable Problem. *IEEE Transactions on Evolutionary Computation*, 13(1):63–74, Apr 1999.

[34] J. P. Burke, E.K Newall and R. Weare. *A Memetic Algorithm for University Exam Timetabling.* The Practice and Theory of Automated Timetabling I: Selected Papers from 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT I). Springer-Verlag, Edinburgh, UK, Lecture Notes in Computer Science 1153 edition, 1996.

[35] Y. Bykov. *Time-Predefined and Trajectory-Based Search: Single and Multi Objective Approaches to Exam Timetabling.* PhD Thesis Department of Computer Science, University of Nottingham, UK, 2003.

[36] H. Cambazard, E. Hebrard, B. O'Sullivan, and A. Papadopoulos. Local search and constraint programming for the post-enrolment-based course timetabling problem. In *proceedings of Practice And Theory of Automated Timetabling (PATAT)*, 2008.

[37] M. W. Carter. A Decompotion Algorithm for Practical Timetabling Problems. *Dept. Industrial Eng, University Toronto, Working Paper 83-06*, 1983.

[38] J. P. Castro Gutierrez, B. Melian Batista, J. A. Moreno Perez, J. M. Moreno Vega, and J. Ramos Bonilla. Codea: An architecture for designing nature-

inspired cooperative decentralized heuristics. In *Proceedings of the 2007 Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), Series Studies in Computational Intelligence, Vol. 129*, pages 189–198. Springer, 2008.

[39] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An Effective Hybrid Algorithm for University Course Timetabling. *Journal of Scheduling*, 9:403–432, 2006.

[40] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In *proceedings of ECAL'91, European Conference on Artificial Life.* Elsevier, Amsterdam, 1991.

[41] A. Colorni, M. Dorigo, and V. Maniezzo. Meta-heuristics for High School Timetabling. *Computational Optimisation and Applications*, 9:275–298, 1998.

[42] T. Cooper and H. Kingston. The Complexity of Timetable Construction Problems. In *Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT 1995), LNCS*, volume 1153, pages 283–295. Springer, 1996.

[43] J. B. M. R. Cordeau, J-F. Efficient Timetabling Solution with Tabu Search. In *http://www.idsia.ch/Files/ttcomp2002/jaumard.pdf*, 2002.

[44] D. Costa. A Tabu Search Algorithm for Computing an Operational Timetable. *European Journal of Operational Research*, 76:98–110, 1994.

[45] P. Cowling, G. Kendall, and L. Han. An Investigation of a Hyper-heuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In *proceeding of the IEEE Congress on Evolutionary Computation*, pages 1185–1190, Honolulu, Hawaii, 2002.

[46] P. Cowling, D. Ouelhadj, and S. Petrovic. Multi-agent systems for dynamic scheduling. In *the proceedings of the Nineteenth Workshop of Planning and Scheduling of the UK, PLANSIG 2000*, pages 45–54, Ed. Garagnani, Max, UK, 2000.

[47] T. Crainic and M. Toulouse. *Explicit and Emergent Cooperation Schemes for Search Algorithms.* LION II, LNCS 5313, Springer-Verlag Heidelberg, 2008.

[48] T. G. Crainic. Parallel Computation, Cooperation, Tabu Search. In *Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search, C. Rego and B. Alidaee (eds.), Kluwer Academic*, pages 283–302. Norwell, MA, 2005.

[49] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. Cooperative Parallel Variable Neighbourhood Search for the p-Median. *Journal of Heuristics*, 10:293–314, 2004.

[50] T. G. Crainic, M. Gendreau, and J. Potvin. *Parallel Tabu Search: A New Class of Algorithms.* John Wiley and Son, 2005.

[51] T. G. Crainic and M. Toulouse. Parallel strategies for meta-heuristics. In *Handbook in Meta-heuristics, F. Glover, G. Kochenberger (eds.),Kluwer Academic*, 2003.

[52] T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a Taxonomy of Parallel Tabu Search Heurisitcs. *INFORMS JOURNAL ON COMPUTING*, 9(1):61–72, 1997.

[53] L. Davis and L. Ritter. Schedule Optimization with Probabilistic Search. In *proceedings of the 3rd IEEE Conference on Artificial Intelligence Applications Orlando, Florida, USA*, pages 231–236, 1987.

[54] P. De Lit, A. Falkenauer, and A. Delchambre. Grouping Genetic Algorithms: An Efficient Method to Solve the Cell Formation Problem. *Mathematics and Computers in Simulation*, 51:257–271, 2000.

[55] D. De Werra. An Introduction to Timetabling. *European Journal of Operational Research*, 19:151–162, 1985.

[56] K. Deb. A Population-Based Algorithm-Generator for Real-Parameter Optimization. *Soft Computing*, 9:236–243, 2005.

[57] R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Mateo, CA, 2003.

[58] S. Deris, S. Omatu, and H. Ohta. Timetable Planning Using the Constraint-based Reasoning. *Computers & Operations Research*, 27(9):819–840, 2002.

[59] S. Deris, S. Omatub, H. Ohtab, and P. Saada. Incorporating Constraint Propagation in Genetic Algorithm for University Timetable Planning. *Engineering Applications of Artifcial Intelligence*, 12:241–253, 1999.

[60] T. Desef, A. Bortfeldt, and H. Gehring. A Tabu Search Algorithm for Solving the Timetabling Problem for German Primary Schools. In *proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004.

[61] L. Di Gaspero, S. Mizzaro, and A. Schaerf. A Multiagent Architecture for Distributed Course Timetabling. In *proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004.

[62] L. Di Gaspero and A. Schaerf. Tabu Search Techniques for Examination Timetabling. In *E. Burke and W. Erben (Eds.) PATAT 2000, 2001. Springer-Verlag Berlin Heidelberg 2001*, volume LNCS 2079, pages 104–117, 2001.

[63] L. Di Gaspero and A. Schaerf. Timetabling Competition TTComp 2002: Solver Description. In *http://www.idsia.ch/Files/ttcomp2002/schaerf.pdf*, 2002.

[64] M. Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico Di Milano, Milano, 1992.

[65] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: An Autocatalytic Optimizing Process. Technical report, Technical Report TR91-016, Politecnico di Milano, 1991.

[66] N. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-Heuristic, New Ideas in Optimizatio.* McGraw-Hill Ltd., UK, 1999.

[67] A. Dubourg, B. Laurent, E. Long, and B. Salotti. In *http://www.idsia.ch/Files/ttcomp2002/laurent.pdf.*, 2002.

[68] G. Dueck. New Optimization Heuristic: The Great Deluge Algorithm and the Record-to-Record Travel. *Journal of Computational Physics*, 104:86–92, 1993.

[69] R. Eberhard and J. Kennedy. A new optimiser using particle swarm theroy. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan*, pages 39–43, 1995.

[70] A. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Natural Computing Series. Springer, 2003.

[71] N. Ejaz and J. M. Younus. A Hybrid Approach for Course Scheduling Inspired by Die-hard Co-operative Ant Behavior. In *proceedings of the IEEE International Conference on Automation and Logistics. Jinan, China*, 2007.

[72] K. H. Elloumi, Abdelkarim and J. Ferland. A Tabu Search Procedure for Course Timetabling Problem at a Tunisian. In *proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[73] M. Elmohamed, P. Coddington, and G. Fox. A Comparison of Annealing Techniques for Academic Course Scheduling. In E. Burke and M. Carter, editors, *The*

Practice and Theory of Automated Timetabling II: Selected Papers from 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT II), Lecture Notes in Computer Science 1408, pages 92–112, Toronto, Canada, Springer-Verlag, 1998.

[74] W. Erben and J. Keppler. A Genetic Algorithm Solving a Weekly Course-timetabling Problem. In E. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling I: Selected Papers from 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT I), Edinburgh, UK, Lecture Notes in Computer Science*, volume 1153, pages 198–211. Springer-Verlag, 1996.

[75] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetabling and Multicommodity Flow Problems. *SIAM Journal of Computation*, 5:691–703, 1976.

[76] J. A. Ferland and C. Fleurent. Computer Aided Scheduling for a Sport League. *INFOR*, 29:14-25, 1991.

[77] J. Frausto-Sols, F. Alonso-Pecina, and J. Mora-Vargas. *An Efficient Simulated Annealing Algorithm for Feasible Solutions of Course Timetabling*, volume 5317/2008. Springer Berlin / Heidelberg, 2008.

[78] Z. W. Geem, J.-H. Kim, and G. Loganathan. A New Heuristic Optimization Algorithm: Harmony Search. *Simulation*, 76(2):60–68, 2001.

[79] M. W. George and S. X. Bill. Examination Timetables and Tabu Search with Longer-Term Memory. In E. Burke and W. E. (Eds.), editors, *PATAT 2000, LNCS 2079*, pages 85–103. Springer-Verlag Berlin Heidelberg, 2001.

[80] F. Glover. Heuristic for Integer Programming Using Surrogate Constraints. *Decision Sci*, 8:156–166, 1977.

[81] F. Glover. Future Path for Integer Programming and Links to Articial Intelligence. *Comput. & Ops. Res*, 13(5):533–549, 1986.

[82] M. Gorges-Schleuter. ASPARAGOS: an Asynchronous Parallel Genetic Optimization Strategy. In *proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann(San Mateo), 1989.

[83] G. Gutin and D. Karapetyan. A memetic Algorithm for the Generalized Traveling Salesman Problem. *Natural Computing*, 2009.

[84] D. Haibin and X. Yu. Hybrid Ant Colony Optimization Using Memetic Algorithm for Traveling Salesman Problem. In *proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, 2007.

[85] W. Hart, N. Krasnogor, and J. Smith, editors. *Recent advances in memetic algorithms*, volume 166 of *Studies in Fuzzyness and Soft Computing*. Springer Berlin Heidelberg New York, 2004. ISBN 3-540-22904-3.

[86] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26(1-2):pages 251–321, 1985.

[87] J. Henry Obit and D. Landa-Silva. Computational Study of Non-Linear Great Deluge for University Course Timetabling. In V. Sgurev and M. Hadjiski, editors, *Intelligent Systems - From Theory to Practice*. Springer Verlag, 2009.

[88] P. V. Hentenryck and V. Sarawat. Constraint Programming: Strategic Directions. *Constraints: An International Journal*, 2:7–33, 1997.

[89] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Anna Arbor, 1975.

[90] J. Hurink and S. Knust. Tabu Search Algorithms for Job-Shop Problems with a Single Transport Robot. *European Journal of Operational Research*, 162:99–111, 2004.

[91] ITC2002. Website of the 2002 international timetabling competition. In *http://www.idsia.ch/Files/ttcomp2002/(Website of the International Timetabling Competition)*, 2002.

[92] F. Javier Martinez Garcia and J. Moreno Perez. Jumping Frogs Optimization: A New Swarm Method for Discrete Optimization. In *Documentos de Trabajo del DEIOC. N. 3/2008.* Universidad de La Laguna, 2008.

[93] K. Kaplansky and A. Meisels. Negotiation Among Scheduling Agents for Distributed Timetabling. In *proceeding of the 5th International Conference on the Practice and Theory of Automated Timetabling, Springer*, 2004.

[94] J. Kennedy and R. Eberhard. Particle swarm optimisation. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages IV, 1942–1948, 1995.

[95] J. H. Kingston. Resource Assignment in High School Timetabling. In *PATAT '08 proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 2008.

[96] G. Konstantinow and C. Coakley. Use of Genetic Algorithms in Reactive Scheduling for Course Timetable. In *proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004.

[97] P. Kostuch. The University Course Timetabling Problem with a Three-Phase Approach. In *Lecture Notes in Computer Science*, pages 109–125. Springer Berlin / Heidelberg, 2005.

[98] N. Krasnogor. A Memetic Algorithm with Self-adaptive Local Search; TSP as a Case Study. In *The 2000 International Genetic and Evolutionary Computation Conference (GECCO)*, 2000.

[99] D. Landa-Silva and E. K. Burke. Asynchronous Cooperative Local Search for the Office-Space-Allocation Problem. *INFORMS Journal on Computing*, 19, No.4:575–587, 2007.

[100] D. Landa-Silva and J. Henry Obit. Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling problems. In *proceedings of the IEEE Conference on Intelligent Systems, IEEE Press*, pages 8.11–8.18, 2008.

[101] D. Landa-Silva and J. Henry Obit. Evolutionary Non-linear Great Deluge for University Course Timetabling. In *proceedings of the International Conference on Hybrid Artificial Intelligence Systems (HAIS)*, 2009.

[102] D. Landa-Silva and J. H. Obit. Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. In *proceedings of the IEEE Conference on Intelligent Systems, IEEE Press*, pages 8.11–8.18, 2008.

[103] J. Landa-Silva. *Metaheuristics and Multiobjective Approaches for the Space Allocation Problem.* PhD thesis, School of Computer Science and Information Technology, University of Nottingham, November 2003.

[104] A. Le Bouthillier, T. G. Crainic, and P. Kropf. Towards a Guided Cooperative Search. In *MIC:The Sixth Metaheurisitcs International Conference*, 2005.

[105] W. Legierski. Constraint-based Reasoning for Timetabling. In *AI-METH 2002 Artificial Intelligence Methods*, 2002.

[106] R. Lewis and B. Paechter. New Crossover Operators for Timetabling with Evolutionary Algorithms. In *The 5th International Conference on Recent Advances in Soft Computing (RASC), Nottingham, UK*, volume 5, pages 189–195, 2004.

[107] R. Lewis and B. Paechter. Application of the Grouping Genetic Algorithm to University Course Timetabling. In *The 5th European Conference in Evolutionary Computation in Combinatorial Optimisation (EvoCop), Lausanne, Swizerland, Lecture Notes in Computer Science*, volume 3448, pages 144–153, 2005.

[108] R. Lewis and B. Paechter. Finding Feasible Timetable Using Group-Based Operators. In *IEEE Transactions on Evolutionary Computation*, volume 11(3), pages 397–413, 2007.

[109] R. M. R. Lewis. *Metaheuristics for University Course Timetabling.* PhD thesis, Napier University, 2006.

[110] Z. Lu and J. Hao. Adaptive Tabu Search for Course Timetabling. *European Journal of Operational Research*, 200:235–244, 2010.

[111] M. M. R. Garey and D. Johnson. A Guide to NP Completeness, first ed. san francisco: W. h. freeman and company. *Computers and Intractability*, 22(4):251–256, 1979.

[112] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[113] K. F. Man, K. Tang, and S. Kwong. Genetic Algorithms: Concepts and Design. *Springer*, 1999.

[114] F. Martinez and J. A. Moreno. Discrete Particle Swarm Optimization for the p-median Problem. In *Metaheuristics International Conference Montreal,Canada*, 2007.

[115] A. Mayer, C. Nothegger, A. Chwatal, and G. Raidl. Solving the Post Enrolment Course Timetabling Problem by Ant Colony Optimization. In *proceedings*

of the 7th International Conference on the Practice and Theory of Automated Timetabling, 2008.

[116] P. McMullan. An Extended Implementation of the Great Deluge Algorithm for Course Timetabling. *Springer-Verlag Berlin Heidelberg*, Part I, LNCS 4487:538–545, 2007.

[117] N. Mladenović and P. Hansen. Variable Neighbourhood Search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[118] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–33, 1986.

[119] R. Montemanni. Timetabling: Guided Simulated Annealing and Local Searches. In *http://www.idsia.ch/Files/ttcomp2002/montemanni.pdf*, 2002.

[120] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. *Report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena CA, USA.*, 1989.

[121] P. Moscato. *Memetic Algorithms: A Short Introduction.* McGraw Hill, 1999.

[122] P. Moscato and C. Cotta Carlo. *Memetic Algorithms.* Handbook of Approximation Algorithms and Metaheuristics, 2007.

[123] P. Moscato and M. G. Norman. A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In *proceedings of the International Conference on Parallel Computing and Transputer Applications*, 1992.

[124] B. Moulin and B. Chaib-draa. An Overview of Distributed Artificial Intelligence. In G. M. P. O 'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 3 – 55. Wisely, New York, 1996.

[125] T. Müller. TTComp02:Algorithm Description
. In *http://www.idsia.ch/Files/ttcomp2002/muller.pdf*, 2002.

[126] V. Nissen and J. Propach. On the Robustness of Population-Based Versus Point-Based Optimization in the Presence of Noise. *IEEE Transactions on Evolutionary Computation*, 2:107–119, 1998.

[127] K. Nonobe and T. Ibaraki. A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver. *European Journal of Operational Research*, 106:599–623, 1998.

[128] J. Obit Henry, D. Landa-Silva, D. Ouelhadj, and M. Sevaux. Non-Linear Great Deluge with Learning Mechanism for Solving the Course Timetabling Problem. In *proceedings of the 8th Metaheuristics International Conference (MIC)*, July 2009.

[129] Y. S. Ong and A. J. Keane. Meta-Lamarckian Learning in Memetic Algorithms, IEEE Transactions on Evolutionary Computation. In *IEEE Transactions on Evolutionary Computation*, volume 8, 2004.

[130] M. Oprea. MAS_UP-UCT: A Multi-Agent System for University Course TimetableScheduling. *International Journal of Computers, Communications & Control*, II, No. 1:94–102, 2007.

[131] D. Ouelhadj. *A Multi-agent System for the Integrated and Dynamic Scheduling of Steel Production*. PhD thesis, University of Nottingham, 2003.

[132] D. Ouelhadj, P. Cowling, and S. Petrovic. Utility and Stability Measures for Agent-based Dynamic Scheduling of Steel Continuous Casting. In *proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2003)*, pages 175–180, Taipei, Taiwan, 2003.

[133] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. A Multi-agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing. In *Lecture Notes in Computer Science*, volume 3470/2005, pages 651–660. Springer Berlin / Heidelberg, 2005.

[134] D. Ouelhadj and S. Petrovic. A Cooperative Distributed Hyper-heuristic Framework for Scheduling. In *proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC), Singapore*, pages 2560–2565, 2008.

[135] E. Özcan and A. Alkan. A Memetic Algorithm for Solving a Timetabling Problem: An Incremental Strategy. In P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, *proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Paris, France*, pages 394–401, August 2007.

[136] B. Paechter, A. P Cumming, M. Norman, and H. Luchian. *Extensions to a memetic timetabling system.* The Practice and Theory of Automated Timetabling I: Selected Papers from 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT I). Springer-Verlag, Edinburgh, UK, 1996.

[137] B. Paechter, R. C. Rankin, and A. Cumming. Improving a Lecture Timetabling System for University-Wide Use. In E. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling II: Selected Papers from 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT II), Lecture Notes in Computer Science*, volume 1408, pages 156–165, Toronto, Canada, 1998. Springer-Verlag.

[138] S. Petrovic and E. K. Burke. *University Timetabling*, pages 45–1 – 45–23. Chapman & Hall/CRCRC Press, 2004.

[139] S. G. Ponnambalam, P. Aravindan, and S. V. Rajesh. A Tabu Search Algorithm for Job Shop Scheduling. *International Journal Advanced Manufacturing Technology , Springer-Verlag London Limited*, 16:765–771, 2000.

[140] N. J. Radcliffe and P. D. Surry. Formal Memetic Algorithms. In *Appears in Evolutionary Computing: AISB Workshop, Ed: T.C. Fogarty*, volume 865, pages 1–16. Springer-Verlag LNCS, 1994.

[141] P. Rattadilok, A. Gaw, and R. S. K. Kwan. Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling. In E. Burke and T. M. (eds.), editors, *PATAT, LNCS*, volume 3616, pages 51–67. Springer-Verlag Berlin Heidelberg, 2005.

[142] C. Reeves. Genetic algorithms. In L. Chambers, editor, *The Practical Handbook of Genetic Algorithms Applications*, pages 55–82. Chapman&Hall/CRC, 2001.

[143] P. Ross, Schulenburg, Marin-Blazquez, and H. Hart. Hyper-heuristic: Learning to Combine Simple Heuristic in Bin-packing Problems. In *proceeding of the Genetic and Evolutionary Computation Conference, New York, USA*, pages 942–948, 2002.

[144] O. Rossi-Doria and B. Paechter. A Memetic Algorithm for the Universiy Course Timetabling. In *Book of Abstracts Lancaster: Lancaster University*, page 56, 2004.

[145] H. Santos, L. Ochi, and M. J. F. Souza. A Tabu Search Heuristic with Efficient Diversification Strategies for the Class/Teacher Timetabling Problem. In *proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, 2004.

[146] H. G. Santos, E. Uchoa, O. L. S, and N. Maculan. Strong Bounds with Cut and Column Generation for Class-Teacher Timetabling. In *PATAT '08 Proceedings*

of the 7th International Conference on the Practice and Theory of Automated Timetabling, 2008.

[147] K. Sastry, D. Goldberg, and G. Kendall. Genetic Algorithms. In E. Burke and G. Kendall, editors, *Search Methodology*, pages 97–125. Springer, 2005.

[148] A. Schaerf. A Survey of Automated Timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[149] S. Schaerf. Local Search Techniques for High-School Timetabling Problems. *IEEE Transactions on Systems, Man, and Cybernatic*, 29(4):368–377, 1999.

[150] N. Shahidi, H. Esmaeilzadeh, M. Abdollahi, and L. C. Self-Adaptive Memetic Algorithm: An Adaptive Conjugate Gradient Approach. In *IEEE Conference of Cybernetic and Intelligent Systems (CIS2004)*, 2004.

[151] K. Socha. The Influence of Run-Time Limits on Choosing Ant System Parameters. In *proceedings of GECCO - Genetic and Evolutionary Computation Conference*, 2003.

[152] K. Socha, J. Knowles, and M. Samples. A Max-Min Ant System for the University Course Timetabling Problem. In *Ant Algorithms: Proceedings of the Third International Workshop (ANTS)*, volume 2463, pages 1–13. LNCS,Springer, 2002.

[153] K. Socha, M. Sampels, and M. Manfrin. Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In *Applications of Evolutionary Computing: Proceedings of the EvoWorkshops*, volume 2611, pages 334–345. Lecture Note in Computer Science, Springer, 2003.

[154] E. Soubeiga. *Development and Application of Hyper-heuristic to Personnel Scheduling.* PhD thesis, School of Computer Science, University of Nottingham, UK, 2003.

[155] P. Tellier and G. M. White. Generating Personnel Schedules in an Industrial Setting Using a Tabu Search Algorithm. In *proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, 2006.

[156] J. M. Thompson and K. A. Dowsland. Variants of Simulated Annealing for the Examination Timetabling Problem. *Annals of Operations Research*, 63, No 1:105–128, 1996.

[157] G. Toro and V. Parada. The Algorithm to Solve the Competition Problem. In *http://www.idsia.ch/Files/ttcomp2002/parada.pdf*, 2002.

[158] E. Tsang. *Foundations of Constraint Satisfaction.* Academic, London, 1993.

[159] H. Ueda, D. Ouchi, K. Takahashi, and T. Miyahara. A Co-evolving Timeslot/Room Assignment Genetic Algorithm Technique for University Timetabling. In *The Practice and Theory of Automated Timetabling III: Selected Papers from 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT III), Konstanz, Germany, Lecture Notes in Computer Science 2079, Springer-Verlag*, pages 48–63, 2001.

[160] P. Van Hentenryck, Y. Deville, and C. Teng. A Generic Arc-Consistency Algorithm and its Specializations. *Artificial Intelligence*, 57:291–321, 1992.

[161] P. van Hentenryck, H. Simonis, and M. Dincbas. Constraint Satisfaction Using Constraint Logic Programming. *Artificial Intelligence*, 58:113–59, 1992.

[162] P. Vytelingum, D. Cliff, and N. R. Jennings. "Analysing Buyers' and Sellers' Strategic Interactions in Marketplaces: An Evolutionary Game Theoretic Approach". In *Proc. 9th Int. Workshop on Agent-Mediated Electronic Commerce, Hawaii, USA, 141-154.*, 2007.

[163] V. Wren. Scheduling, Timetabling and Rostering A Special Relationship ? In E. Burke and P. Ross, editors, *Selected Papers from 1st International Conference on Practise and Theory of Automated Timetabling (PATAT I)*, Lecture Notes in Computer Science, 1153, pages 46–75, Springer, Edinburgh, UK, 1996.

[164] H. Yan and S. Yu. A Multiple-Neighborhoods-Based Simulated Annealing Algorithm for Timetable Problem. volume 3033/2004, pages 474–481. Springer Berlin / Heidelberg, 2004.

[165] K. Zervoudakis and P. Stamatopolous. A Generic Object-Oriented Constraint-Based Model for University Course Timetabling. In E. Burke and W. Erben, editors, *Lecture Notes in Computer Science, The Practice and Theory of Automated Timetabling: Selected Papers from the Third International Conference*, volume 2079/2001, pages 28–47. Springer Berlin / Heidelberg, 2001.