

TABU SEARCH BASED HYPER-HEURISTIC APPROACHES TO EXAMINATION TIMETABLING

By

Naimah Mohd Hussin, MSc

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy
School of Computer Science and Information Technology
November 2005

Table of Contents

Acknowledgments	i
Publications from PhD period	ii
Thesis abstract.....	iv
Chapter 1 Introduction	1-6
1.1 Motivation	1-6
1.2 Scope and Objectives	1-8
1.3 Thesis Overview	1-9
1.4 Research Contribution	1-12
Chapter 2 Algorithmic Techniques for Examination Timetabling.....	2-14
2.1 Introduction to Timetabling.....	2-14
2.2 Examination Timetabling Problem Description.....	2-17
2.3 Algorithmic Strategies.....	2-22
2.3.1 Cluster Method	2-24
2.3.2 Sequential Methods and Constructive Heuristics.....	2-25
2.3.3 Meta-heuristics	2-29
2.3.3.1 Local Neighbourhood Search	2-31
2.3.3.2 Simulated Annealing and the Great Deluge Algorithm.....	2-33
2.3.3.3 Tabu Search.....	2-35
2.3.3.4 Greedy Randomised Adaptive Search Procedure	2-40
2.3.3.5 Genetic Algorithms	2-41
2.3.3.6 Memetic Algorithms	2-43
2.3.3.7 Scatter Search Algorithms	2-43
2.3.3.8 Multiobjective Approaches.....	2-44
2.3.3.9 Case-Based Reasoning Approaches.....	2-45

2.3.4	Other Methods	2-46
2.4	Conclusions	2-46
Chapter 3	Hyper-heuristics	3-51
3.1	Introduction	3-51
3.2	University Timetabling Applications	3-51
3.3	Other Applications	3-52
3.4	Conclusions	3-54
Chapter 4	A Survey and Case Study of a Practical Examination Timetabling Problem	4-55
4.1	Introduction	4-55
4.2	Questionnaires	4-56
4.2.1	Student Questionnaire.....	4-57
4.2.2	Invigilator Questionnaire.....	4-62
4.3	Examination Timetabling at the MARA University of Technology (UiTM)	4-67
4.3.1	Examination Timetable.....	4-68
4.3.2	Invigilator and Room Assignment.....	4-71
4.4	Conclusions	4-72
Chapter 5	Examination Timetabling: Models and Data Specification.....	5-74
5.1	Introduction	5-74
5.2	Graph Model.....	5-75
5.3	Mathematical Model.....	5-76
5.4	UiTM Examination Timetabling Problem Formulation.....	5-79
5.4.1	Mathematical Model.....	5-79
5.4.2	Constraints	5-81
5.4.3	Costs	5-81
5.5	Benchmark Datasets and UiTM Dataset	5-82

5.6	Conflict Matrix and Greedy Constructive Heuristic	5-86
5.7	Conclusions	5-93
Chapter 6	Initial Study on a Tabu Search Based Hyper-heuristic for the Examination Timetabling Problem	6-94
6.1	Introduction	6-94
6.2	Hyper-heuristic Framework and Strategy	6-95
6.2.1	Tabu Search Based Hyper-heuristic Module.....	6-96
6.2.2	Low-level Heuristics Module	6-102
6.3	Experimental Results.....	6-104
6.3.1	Datasets.....	6-105
6.3.2	Experimental Results and Analysis	6-105
6.4	Conclusions	6-112
Chapter 7	Control Techniques in the Tabu Search Based Hyper-heuristic .	7-114
7.1	Introduction	7-114
7.2	The Tabu Search Based Hyper-heuristic.....	7-115
7.2.1	A Tabu Search based Hyper-heuristic with Hill Climbing (TSHH-HC)	7-118
7.2.2	A Tabu Search based Hyper-heuristic with Great Deluge (TSHH-GD)	7-119
7.2.3	Low-level Heuristics.....	7-121
7.3	Experimental Results and Analysis	7-122
7.3.1	Datasets.....	7-122
7.3.2	Tabu Search Based Hyper-heuristic Results Analysis	7-123
7.3.3	UiTM-03 Performance Analysis.....	7-126
7.3.4	Tabu Search Based Hyper-heuristic Results Compared to Published Methods.....	7-132
7.4	Conclusions	7-135
Chapter 8	A Tabu Search Based Hyper-heuristic Approach to Examination Timetabling Problem at the MARA University of Technology ...	8-137

8.1	UiTM Examination Timetable Problem Description	8-138
8.1.1	Constraints	8-141
8.1.2	Costs	8-142
8.2	The Tabu Search Based Hyper-heuristic.....	8-143
8.2.1	Constructive Heuristic	8-143
8.2.2	Hyper-heuristic Module.....	8-144
8.2.3	Low-level Heuristics Module	8-144
8.3	Experimental Results.....	8-145
8.3.1	Proximity and Weekend Cost Evaluation Method	8-146
8.3.2	Results Analysis.....	8-148
8.3.3	New Initial Solution and Dynamic Unscheduled Examination Weight.....	8-150
8.3.4	Benchmark Datasets with Dynamic and Fixed Unscheduled Weights	8-154
8.4	Conclusions	8-156
Chapter 9	Reactive Hyper-Heuristic with Tabu Heuristic List	9-158
9.1	Introduction	9-158
9.2	Methodology	9-160
9.3	Results and Analysis	9-165
9.3.1	Results of Nine Experiments for Benchmark Datasets.....	9-166
9.3.2	Reactive Tabu Search Based Hyper-heuristic (Experiment 9) Analysis	9-167
9.3.3	Performance Analysis.....	9-170
9.3.4	Reactive Tabu Search Based Hyper-heuristic with Fixed and Dynamic Unscheduled Examinations Weight (UiTM Dataset)	9-172
9.4	Conclusions	9-174
Chapter 10	Conclusions	10-175
Appendix A	Student and Invigilator Questionnaire.....	181

Appendix B Data File Format and Specification	189
Bibliography	194

List of Tables

Table 2.1: Published Benchmark Results (Uncapacitated Examination Timetabling Problem)	2-48
Table 4.1: Characteristics of UiTM dataset compared to benchmark datasets	4-69
Table 5.1: Characteristics of real problems	5-84
Table 6.1: Comparing results of TSHH with 10 minutes run time (with deterministic tabu duration value) and 4 hours run time	6-106
Table 6.2: Results of TSHH for 12 benchmark datasets with fixed deterministic tabu duration and random dynamic tabu duration (4 hours run)	6-108
Table 6.3: Comparing TSHH-4 hour run and published results	6-111
Table 7.1: Comparing results between the three methods: TSHH-B, TSHH-HC and TSHH-GD	7-125
Table 7.2: Comparison of result for UiTM dataset with other solution methods	7-127
Table 7.3: Comparing our best results and published results	7-134
Table 8.1: Characteristics of UiTM dataset categorised by campuses.	8-139
Table 8.2: Uitm-03 dataset (minimised proximity and weekend cost)	8-149
Table 8.3: Data characteristic of solution with 7.12 proximity and weekend cost (by TSHH-GD with TD = 3).	8-150
Table 8.4: Data characteristic of solution with 9.27 proximity and weekend cost (by TSHH-B with TD = 1).....	8-150
Table 8.5: Results with two different initial solutions and two unscheduled weight mode.	8-152

Table 8.6: Data characteristic of solution with 4.40 proximity and weekend cost. Best solution for TSHH-B with new initial solution and dynamic weight	8-153
Table 8.7: Data characteristic of solution with 6.98 proximity and weekend cost. Best solution for TSHH-B with old initial solution and dynamic weight	8-154
Table 8.8: Data characteristic of solution with 6.94 proximity and weekend cost. Best solution for TSHH-HC with new initial solution and fixed weight.....	8-154
Table 8.9: Benchmark datasets with dynamic unscheduled weight (1 hour run)	8-155
Table 8.10: Benchmark datasets with fixed unscheduled weight (1 hour run).....	8-156
Table 9.1: Results from 9 experiments	9-166
Table 9.2: Reactive Hyper-heuristic Methods	9-168
Table 9.3: A snapshot of low-level heuristic performance	9-172
Table 9.4: Comparison of results for uitm-03 dataset.....	9-173

List of Figures

Figure 2.1: Framework of a general local search technique	2-31
Figure 4.1: Preference to have examinations spread evenly (40% neutral, 53% agree)	4-58
Figure 4.2: Students preference to sit for the most difficult examinations earlier.....	4-58
Figure 4.3: Students preference for minimum days gap between two examinations ..	4-59
Figure 4.4: Percentage of students preferring weekend examinations.....	4-59
Figure 4.5: Students preference for afternoon examinations	4-60
Figure 4.6: Students preference for morning examinations	4-60
Figure 4.7: Percentage of lecturers' preference for minimum number of invigilation duties.....	4-64
Figure 4.8: Percentage of lecturers' preference for maximum number of invigilation duties.....	4-64
Figure 4.9: Lecturers preference for a minimum day gap between two invigilation duties.....	4-65
Figure 4.10: Lecturers response on weekend invigilation and total invigilation duties, and opinion on previous schedule	4-65
Figure 5.1: Graph model of a very simple timetabling problem.....	5-76
Figure 5.2: Student datafile (<fnm.stl>) format specification.....	5-87
Figure 5.3: Conflict Matrix	5-88
Figure 5.4: Sample of a conflict graph file	5-89

Figure 5.5: Algorithm to create a conflict graph.....	5-90
Figure 5.6: Largest Degree Constructive Algorithm	5-91
Figure 5.7: Modified Largest Degree Constructive Algorithm for UiTM dataset.....	5-92
Figure 6.1: Hyper-heuristic Framework.....	6-97
Figure 6.2: A general tabu search based hyper-heuristic procedure.....	6-98
Figure 6.3: Tabu search based hyper-heuristic algorithm.....	6-101
Figure 6.4: Tabu search base hyper-heuristic performance with different TD.....	6-109
Figure 6.5: Heuristic 6 – Select examination at random and move to random slot...	6-110
Figure 6.6: Heuristic 7 – Move examination with maximum penalty from randomly selected examinations	6-110
Figure 7.1: Tabu Search based Hyper-heuristic Framework	7-116
Figure 7.2: Tabu search based hyper-heuristic with hill climbing algorithm	7-119
Figure 7.3: Tabu search based hyper-heuristic with great deluge algorithm.....	7-121
Figure 7.4: Hyper-heuristic methods on uitm-03 dataset.....	7-129
Figure 7.5: Hyper-heuristic methods on Car-f91 dataset.....	7-129
Figure 7.6: Simple Tabu Search Based Hyper-heuristic for UiTM dataset.....	7-131
Figure 7.7: Tabu Search Based Hyper-heuristic with Hill Climbing for UiTM dataset.....	7-131
Figure 7.8: Tabu Search Based Hyper-heuristic with Great Deluge for UiTM dataset.....	7-132

Figure 7.9: Comparing TSHH-GD and Great Deluge Algorithm performance.....	7-132
Figure 9.1: Algorithm for experiment number 9.....	9-165
Figure 9.2: A snapshot of Reactive Tabu Search Based Hyper-heuristic performance graph for car-s91 dataset.....	9-169
Figure 9.3: A snapshot of Reactive Tabu Search Based Hyper-heuristic performance graph for sta-f83 dataset.....	9-169
Figure 9.4: One cycle of Reactive Tabu Search Based Hyper-heuristic search graph (sta-f83 dataset).....	9-171
Figure 9.5: A snapshot of two cycles in the search.....	9-173

Acknowledgments

There are a number of people that I would like to acknowledge for their help and support during my PhD research.

First of all I would like to thank MARA University of Technology and the Malaysian government for the doctoral scholarship and other financial support during my PhD study.

Thank you to the University of Nottingham for accepting me in their PhD programme within the Automated Scheduling, Optimisation and Planning Research Group (ASAP), School of Computer Science and Information Technology and a very special thanks to Dr. Graham Kendall for your help, support and guidance, not forgetting my former supervisor, Prof. Peter Cowling (University of Bradford) who started me off in my first year.

It is also with great pleasure that I thank the ASAP research group under the leadership of Prof. Edmund Burke for providing a very pleasant and academically enriched environment for my research. I am sure others will agree with me that the interaction through discussions, seminars and social events has made our studies here in Nottingham a fruitful and meaningful experience.

I would also like to thank the anonymous referees of the PATAT, MISTA and MIC conferences for providing me with very good feedback on my research. Also thanks to Puan Norafida and the Center for Integrated Information Systems for providing me with the data for my research.

Thank you to my colleagues and friends in Malaysia. Thank you to the Nottingham Malaysian Community for making Nottingham feels like home.

A special thank you goes to my brothers, sisters, nieces and nephews for always being supportive and caring for me. And to my husband, Kris Nor, with all my love and thanks.

Publications from PhD period

Papers published

Cowling, P., Kendall, G. and Mohd Hussin, N. (2002b) A survey and case study of practical examination timetabling problems. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT2002), Gent, 2002, pp 258-261. (Extended Abstract)

Kendall, G. and Mohd Hussin, N. (2005a) An investigation of a tabu search based hyper-heuristic for examination timetabling. In: Multidisciplinary Scheduling: Theory and Applications, G. Kendall, E. K. Burke, S. Petrovic and M. Gendreau (Eds.), Springer (2005) 309-328. An extended abstract of this paper also appeared in the Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA2003) pp. 226-233.

Kendall, G. and Mohd Hussin, N. (2005b) A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology. In: Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds), Lecture Notes in Computer Science, Volume 3616, Springer-Verlag, pp. 270-293. An earlier version of this paper also appeared in the Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds.), pp. 199-217.

Kendall G. and Mohd Hussin N. (2005c), Control techniques in a tabu search based hyper-heuristic, Accepted for publication in the 6th Metaheuristics International Conference (MIC2005).

Papers submitted

Kendall G. and Mohd Hussin N. (2005d) A reactive tabu search based hyper-heuristic, submitted to the Journal of the Operational Research Society.

Other papers¹

Kaplansky, E., Kendall, G., Meisels, A. and Mohd Hussin, N. (2004) Distributed examination timetabling. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT2004), E. K. Burke and M. Trick (Eds.), pp.511-516 (Extended Abstract).

¹ This research was undertaken during the course of this PhD, but does not form part of this thesis.

Thesis abstract

The problem of constructing a conflict-free examination timetable is a difficult task faced by colleges and universities across the world. The problem also varies with respect to the size, structure and constraints and with the importance of each constraint. Therefore, a known solution approach for a particular problem may not be suitable for another problem instance. Current area of research interest is to explore how we can raise the level of generality of state of the art algorithm, so that we can address a wider range of problems. One such methodology that forms the basis of this thesis is, a hyper-heuristic. This thesis describes a hyper-heuristic framework which uses a tabu list to monitor the performance of a collection of low-level heuristics and then make tabu heuristics that have been applied too many times, thus allowing other heuristics to be applied. We propose three tabu search based hyper-heuristic methods: a simple tabu search based hyper-heuristic, a tabu search based hyper-heuristic with hill climbing and a tabu search based hyper-heuristic with great deluge. Each of these methods has one parameter, i.e. the tabu duration (how long a heuristic is made tabu). The tabu list, tabu duration and the acceptance criteria are the control management tool used to manage the low-level heuristics. A parameter free reactive tabu search hyper-heuristic is also implemented that uses a feedback mechanism to automate an adaptive tabu duration parameter. This method does not require any parameter tuning and can be easily adapted to another set of problem. These methods, when tested on the benchmark datasets produce promising results compared to other solution approaches.

In this thesis we also introduce, and solve, an examination timetabling problem from the MARA University of Technology (UiTM). UiTM is the largest university in Malaysia and has 13 branch campuses and offers 144 programs delivered by 18 faculties. We discuss their extremely complex examination timetabling problem with respect to its size, complexity and constraints. We analyse and process their real world data, and feed it into our tabu search base hyper-heuristic framework. This dataset has an additional criterion that needs to be minimised, i.e. weekend costs. The UiTM campuses do not have the same weekend, i.e. some campuses have weekends on Thursday and Friday while others is on Saturday and Sunday. UiTM

has to schedule examinations at weekends because it cannot schedule all its exams in only three days per week. Therefore, to minimise weekend costs, we need to consider another factor, i.e. the branch campus. This problem is solved in two ways, firstly by minimising proximity cost and secondly, by minimising both proximity and weekend cost. The same tabu search based hyper-heuristic methods that were applied on the benchmark datasets are applied on this new dataset.

The main contributions of this thesis are: a flexible, parameter free and reactive hyper-heuristic approach to solve timetabling problems; the proposal of a control management tool for a hyper-heuristic framework; the formulation and solution of a new, large (in terms of the number of students and examinations) and complex (i.e. highly constrained) examination timetabling problem.

Chapter 1

Introduction

1.1 Motivation

Most academic institutions face the problem of scheduling both courses and examinations in every semester (or term). In a university, the scheduling of courses may be localised within departments or faculties and students may have to adjust their schedule so that they can attend their chosen courses. In a way that is different from course scheduling, examinations are normally scheduled at the university level (Burke et al, 1996). In some institutions, examinations for specialised courses are scheduled by the relevant faculties, at times that will not clash with the main examination timetable. If the examination timetable is (potentially) produced at different levels, students may find it unsuitable because examinations may be scheduled too close to each other. Indeed, even if the examination timetable is prepared centrally, it may still contain elements which some students find unacceptable.

Frequently, a timetable that has been used previously, and found to be acceptable is recycled and used again in the next semester (term). Some minor adjustment may need to be made and this can be done manually or with the help of a decision support system. As the difficulty of the problem increases, due to a large number of students, courses and examinations and room and invigilator constraints, (amongst others), an automated timetabling system that can produce feasible, high quality timetables quickly is often required.

Examination timetabling is a very buoyant topic of research at the moment

attracting the interest of academic researchers from all over the world. Carter (1986), Carter and Laporte (1996), Burke et al. (1997) and Schaerf (1999) have conducted surveys and overviews on various methods and strategies applied by researchers to solve timetabling problems up to the end of 1990s. Many of these methods have successfully solved given problems and some algorithms/heuristics were reported to work well with particular datasets whilst others performed better when presented with different datasets.

Burke and Petrovic (2002) and Petrovic and Burke (2004) present an overview of university timetabling problems and recent approaches in automated timetabling developed within the Automated Scheduling, Optimisation and Planning (ASAP) research group at the University of Nottingham. These methods include hybridising heuristic methods, memetic algorithms (incorporating hill climbing, decomposition and the diversity of initial populations), multi-criteria approaches, fuzzy methods, case-base reasoning and hyper-heuristic methods. The authors also highlight several important future research directions in timetabling: research on methods with relevant real world parameters that are less dependent on parameter tuning, integrating meta-heuristics and multi-criteria approaches to cater for different objectives in real world problems, case based reasoning approaches that can reuse previous solutions and methods, and lastly, the development of approaches that are more generalised and so applicable to a wider range of problems than the current state of the art.

A collection of educational timetabling problems and their solution approaches can be found in several conference volumes (e.g. see Burke and Ross, 1996; Burke and Carter, 1998; Burke and Erben, 2001; Burke and De Causmaecker, 2003; Kendall et al., 2005). The timetabling problem is known to be an NP-hard optimisation problem (see Cooper and Kingston, 1996). Meta-heuristics are a class of heuristic techniques that have been successfully applied (e.g. see Osman and Kelly, 1996; Voß et al., 1999; Ribeiro and Hansen, 2001; Ibaraki et al, 2005) to solve a wide range of combinatorial optimisation problems over the years.

The No Free Lunch Theorem (NFL) of Wolpert and Macready (Wolpert and Macready, 1995; Wolpert and Macready, 1997) states that “... *all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions*”. In other words, “*if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems*”. This is an important results and it is entirely consistent with one of the main motivating goal of hyper-heuristics research which is to explore how to build more general algorithms and what the limit are on the level of generality that can be obtained. Burke et al. (2003a) state that “*A hyper-heuristic could be thought of as a (meta)-heuristic which operates on lower level (meta-)heuristics. Hyper-heuristics are broadly concerned with intelligently choosing the right heuristic or algorithm in a given situation*”. A hyper-heuristic can be implemented as a generic module that has a common interface to the various low-level heuristics and other domain specific knowledge (typically the evaluation function) of the problem being solved. To solve a different problem we only need to modify the low-level heuristics and the evaluation function while the high level part remain unchanged.

1.2 Scope and Objectives

This thesis is concerned with examination timetabling problems. A real examination timetabling problem has various constraints specific to the requirement of an institution. Different institutions will place different emphasis upon the various constraints and it would be impossible to model a generic examination timetabling model that is applicable to every institution. Another obstacle that needs to be overcome is the fluidity of problem specification. At the beginning of a project, the problem specification may be agreed but the final solution may not be as required because some implicit processes are not included or there has been a change in management requirements (e.g. Burke et al., 2004d; Burke et al., 2006d). Therefore, in this thesis, we model only the basic examination timetabling problem and apply a solution approach that has the flexibility of adding other specific changes in

requirements.

The aim of this thesis is to present a hyper-heuristic framework that can solve the basic examination timetabling problem of assigning examinations to slots without violating a clash constraint. A large number of solution approaches have been successfully developed to solve this problem with good quality solutions (See table 2.1). Each time a new solution methodology is proposed, its performance is measured against the current best value found for a set of benchmark datasets (first presented in Carter et al., 1996). It is not our aim to beat the results found previously by other methods. Moreover, these methods are mostly very problem instance specific (e.g. methods prior to 1995 described in Carter and Laporte, 1998) or the algorithm requires a rigorous parameter tuning (e.g. Thompson and Dowsland, 1998). Our aim in utilising hyper-heuristic approaches is to demonstrate that the approaches we develop can operate across a wide range of differing timetabling instances.

In addition, we introduce a new examination timetabling dataset that has a different (and much more complex) structure from existing benchmark datasets. This enables us to further test our hyper-heuristic approaches on a problem that is different from those in the literature, in at least one significant way. A side effect is that we present the academic world with a new and more complex real world dataset that will hopefully be used as a future benchmark problem.

In summary, this thesis will demonstrate the flexibility of the hyper-heuristic framework for solving examination timetabling benchmark datasets (with competitive results) and a new examination timetabling dataset with different problem characteristics, constraints and evaluation function.

1.3 Thesis Overview

This thesis is presented in ten chapters. The first chapter presents the scope and

objectives of the research and how it contributes to the research community. Chapter two describes the general examination timetabling problem and the solution methodologies that have been developed over the years. Chapter three focuses on the hyper-heuristic method and some application examples.

Chapter four presents the findings from two questionnaires distributed to university students and lecturers with the objective of ascertaining their opinion on their current examination timetable and what they feel should be included in the next generation of examination timetables. The chapter also gives a brief introduction to another instance of an examination timetable problem faced by a very large university (MARA University of Technology) which has several branch campuses.

In chapter five, the basic examination timetabling problem of assigning examinations to slots is formulated using a graph colouring model. The mathematical model used by other researchers and also in this thesis is also presented. The next section in chapter five presents a mathematical model, with objectives and constraints, for the MARA University of Technology examination timetabling problem. This chapter also demonstrates the data specification format, data representation (conflict matrix), penalty cost evaluation and a constructive heuristic algorithm used in the hyper-heuristic framework.

In chapter six, a new hyper-heuristic method with tabu list (tabu search based hyper-heuristic) is developed and tested on 12 examination timetabling benchmark datasets. This initial investigation aims to establish a well-defined relationship between a high-level non-domain specific meta-heuristic (employed a hyper-heuristic) and several low-level problem dependent heuristics. Various low-level heuristics specific to examination timetabling are also designed, implemented and tested in conjunction with the hyper-heuristic. It also aims to identify how a tabu list or memory can help a hyper-heuristic manage and control its low level heuristics. The tabu list size or tabu duration (as we refer it in this thesis) is implemented as a fixed pre-determined tabu duration and also as a random dynamic tabu duration.

The basic tabu search based hyper-heuristic method from chapter six is further enhanced and evaluated on 13 datasets (12 benchmark datasets and MARA University of Technology (uitm-03) dataset) and presented in chapter seven. Two additional hyper-heuristics are implemented: a tabu search based hyper-heuristic with hill climbing (steepest descent) and a tabu search based hyper-heuristic with great deluge. In this chapter, the MARA University of Technology examination timetabling problem is formulated in a similar way as the benchmark datasets. We also apply a modified great deluge and simulated annealing algorithm on the uitm-03 dataset so as to compare the hyper-heuristics against more standard meta-heuristic approaches.

Chapter eight provides solutions for the MARA University of Technology examination timetabling problem using the tabu search hyper-heuristics from chapter six and seven. The problem is different from the benchmark datasets because of an additional soft constraint, i.e. a weekend constraint. A new constructive heuristic algorithm and a new objective function are included to accommodate the additional constraint. The unscheduled examinations penalty is also modified to reflect the number of students involved in the unscheduled examinations. A dynamic unscheduled examinations weight is also applied on the uitm-03 dataset and benchmark datasets.

In chapter nine, a new control technique adapted from the lessons learnt from the previous chapters is introduced in a new and final hyper-heuristic methodology. The objective in designing a hyper-heuristic is to allow it to learn and adapt necessary parameters suitable for solving the current problem at hand. This new technique (reactive tabu search based hyper-heuristic) is applied on the twelve benchmark datasets and the UiTM dataset so as to demonstrate its flexibility and efficiency in finding good quality solutions.

Chapter ten concludes the thesis by identifying and summarising lessons learned in this work and how it can be applied in future research.

1.4 Research Contribution

The main contributions of this thesis can be summarised as follows:

- § Three tabu search based hyper-heuristic methods, that incorporate a tabu list (with tabu duration parameter), to manage a set of low-level heuristics that guide a search towards a good quality solution. These methods are as competitive as other known methods applied on the same benchmark datasets and using a similar quality measure.
- § The hyper-heuristic uses a tabu list as a guide to manage the co-operation between low-level heuristics and to choose the low-level heuristics at each decision point in the search.
- § A well-defined relationship is established between a hyper-heuristic module and its domain specific low-level heuristics module. The hyper-heuristic framework utilises only general knowledge (such as changes in the objective function, a move into an infeasible or feasible region, etc.) about any application and does not need to rely upon domain knowledge to make its decisions.
- § A well-defined module that encapsulates a set of low-level heuristics and a cost evaluation function for the examination timetabling problem is presented.
- § A parameter free reactive tabu search based hyper-heuristic is presented. It uses a feedback mechanism to automate an adaptive tabu duration parameter by monitoring the performance of the low-level heuristics. In addition to the deterministic tabu duration for all low-level heuristics, a suitable performance based tabu duration is also assigned to each low-level heuristic.

- § A new final examination dataset for a very large university is presented with the same format specification as benchmark datasets so that it can be made public and tested with existing and new methods. This new dataset will be the largest dataset in the literature and also incorporates additional constraints which are not present in current benchmark problems.
- § The hyper-heuristic framework is easily adaptable to other problems with additional soft constraints by simply adding a new low-level heuristic that optimises a specific constraint and amending the objective function.
- § We present a solution methodology that is generic, robust and able to produce good solutions across all current benchmark datasets as well as a newly introduced benchmark instance.

In summary, this thesis presents a new examination timetabling dataset with different characteristics from benchmark datasets and a reactive, parameter free and flexible hyper-heuristic solution approach that can solve and produce good quality solution across all the standard benchmark instances of the examination timetabling problem.

Chapter 2

Algorithmic Techniques for Examination Timetabling

This chapter provides an introduction to the process of automatically generating university examination timetables. Section 2.1 begins with a general introduction to timetabling with a special focus on educational timetabling. Section 2.2 gives a description of the university examination timetabling problem and how timetables are constructed in universities. Finally, section 2.3 considers the different algorithmic techniques that have been applied to the examination timetabling problems that appear in the literature.

2.1 Introduction to Timetabling

Work on timetabling problems has been published since the 1960's (Appleby et al., 1960), (Csima and Gotlieb, 1964). The *Chambers English Dictionary* defines timetable as ‘*a table of times, e.g. of classes, events, trains, etc*’. Timetabling is a special case of a scheduling problem (Wren, 1996). According to Wren, ‘*a timetable shows when particular events are to take place and it does not necessarily imply an allocation of resources*’. Numerous researchers have been working on problems ranging from sports timetabling (Trick, 2001; Easton et al., 2004), railway timetabling (Isaai and Singh, 2001; Caprara et al., 2001), school timetabling (Abramson et al., 1999; Colorni et al., 1998; Ribeiro and Lorena, 2001; Hansen, 1995; Schaerf and Schaerf, 1995; Cambazard et al., 2004; Kingston, 2004) and university timetabling (Awad and Chinneck, 1998; Burke and Newall, 1999; Burke et al., 2006b; Marti et al., 2000; Ross et al., 1998; Thompson and Dowsland, 1998;

Terashima Marin et al., 1999a; White and Xie, 2001).

Most academic institutions face the problem of scheduling both courses and examinations for every semester (or term). As the difficulty of the problem increases, due to a large number of students, courses and examinations and room and invigilator constraints, (amongst others), an automated timetabling system that can produce feasible and high quality timetables is often required. The timetabling procedure at universities and schools varies from manual timetabling, semi-automated timetabling to fully automated timetabling. A survey conducted by Burke et al. (1996a) received feedback from 56 registrars of British Universities with regards to the nature of their examination timetabling problem, how they solved it (manual or automated) and what qualities were considered for a good examination timetable. They discovered that only 58% (32 universities) of their respondents use a computer at some stage in producing their examination timetable and 21% (11 universities) of these have a scheduling system. Only two universities use commercial software whilst the other systems were developed in house. Thus, whilst commercial software is available, such as, EXAMINE (Carter, 1997; Paechter et al., 2000), SyllabusPlus (Forster, 1995), ConBaTT (Goltz and Matzke, 2000), OPTIME (McCollum and Newall, 2000), and CelCAT (Rogalla 1997), many universities are yet to be convinced that an automated system will provide a satisfactory solution. Universities may need to develop their own system or customise a commercial system to fulfil their specific needs and requirements in timetabling. Once a customised system is developed, it will also require full support with frequent updates and maintenance due to changes in academic policy or education structure.

An early survey by Comm and Mathaisel (1988) involving 1,494 U.S. college registrars concluded that there was a large market for a computerised timetabling system and most registrars were unhappy with their current systems. The survey showed that a computerised system must produce good quality timetables allowing some user intervention; it must be easy to use, comprehensive and compatible with any previous systems. JISC (Joint Information System Committee) Technology

Applications Programme (JTAP 1998) published findings from a questionnaire from which they received replies from 16 universities in the U.K. The universities were asked whether a central computerised system was in use and for their views on its effectiveness. The report concluded that centralising the whole process of room bookings, examination timetabling and lecture timetabling was done in phases using a wide variety of software packages and there was a need for a full and complete management support for such systems.

In his survey on automated timetabling, Schaerf (1999) classifies educational timetabling problem into three classes:

1. **School Timetabling** (Class-teacher Timetabling as classified by Burke et al., 2004b): The scheduling of classes (with a pre-assigned teacher and a group of students) into any number of time slots over a one week period, such that no teacher or group of students has to attend two classes at the same time.
2. **University Course Timetabling**: The process of assigning a set of courses to time periods and classrooms so that there are no clashes (teacher, student or room) in each period and that room capacities are not exceeded.
3. **Examination Timetabling**: The assignment of a set of examinations to a finite number of slots over a time period such that all examinations are scheduled and no student has to sit for more than one examination at the same time.

The scheduling of courses in schools and universities may seem to be similar but they are different since a school usually has very similar class sizes for all its courses and the same group of students is associated with a set of courses. University courses have a more varied range of enrolments and students have more freedom to select which courses to enrol upon. University course timetabling and examination timetabling also differ with respect to the specification and constraints involved. In course timetabling student conflicts are not as crucial as in examination timetabling since students can have the option of choosing another course but students cannot

choose to take one examination over another examination.

Invigilator assignment (Marti et al., 2000) and office space allocation (Burke et al., 2001c) are other problems closely related to the problems in educational timetabling described above. This chapter continues with a general description of the examination timetabling problem (which is the main focus of this thesis) and an overview of various algorithmic techniques that have been applied to the examination timetabling problem.

2.2 Examination Timetabling Problem Description

A timetable normally tells you when and where events are to take place. Carter and Laporte (1996) defined the basic problem in examination timetabling as “*the assigning of examinations to a limited number of available time periods in such a way that there are no conflicts or clashes*”. In a more comprehensive definition, we can define examination timetabling as follows:

Examination timetabling problem is the assigning of a set of examinations into time slots over a finite period of time such that it satisfies constraints associated with some resources.

Examples of resources associated with the slot assignments are students sitting for the examinations, available rooms and invigilators on duty. Slot assignments must satisfy some pre-determined set of constraints. These constraints can be classified as hard or soft. Hard constraints must be satisfied to produce a feasible timetable. Examples of common hard constraints are:

§ **Examination constraint:** All examinations must be scheduled in a time slot.

§ **Student constraint:** There should not be any student scheduled to sit for more than one examination in the same time slot (often referred to as first order conflict).

- § **Room constraint:** The total number of students scheduled to sit for the examination(s) in a time slot must not exceed room capacity.
- § **Invigilator constraint:** An invigilator should not be scheduled to invigilate more than once in the same time slot. Note that invigilation scheduling is often done separately from the scheduling of examinations to time slot.

Other hard constraints, specific to certain universities, may also be included, thus further increasing the complexity of the problem. An example of an additional hard constraint is *examination constraint* where more than one examination may have to be assigned concurrently, in sequence, within a certain period, or in earlier or later slots. General-purpose automated timetabling software may not be able to cater for these specific constraints and custom-made software must be commissioned. In some universities, with a large number of examinations and students, it may be difficult to produce a timetable that satisfies all hard constraint. One hard constraint that may be violated is the student constraint where a number of students might be scheduled to sit for more than one examination in the same time slot. In this scenario, students have to be quarantined and required to sit for the conflicting examination in the next time slot. This involves additional resources, such as invigilators and rooms, which we are trying to reduce.

Often, we can find one or more timetables that satisfy the basic hard constraints described above. But, which timetable should we choose? Thus, we have to use a measure to evaluate good and practical timetables that satisfy most, if not all, of the people affected by the timetable. In the educational timetabling research community, we categorise another set of constraints called soft constraints that can be violated if necessary. Each violation of a soft constraint is penalised and a weighted penalty value is associated with each constraint (a higher weight if the constraint is very important). Minimising these violations (penalty values) will be the objective of producing a high quality timetable. Burke et al. (1996a) gives a comprehensive list of constraints from their surveys on 56 British universities. The following are examples of soft constraints:

§ Student satisfaction:

- Students should not sit for two examinations in sequence, often referred to as second order conflict (Burke et al., 1996b).
- A general requirement that students should only sit for x examinations within y days. (Leong and Yeong, 1987;1990)
- A gap of one period between sittings referred to as third order conflict (White and Xie, 2001).
- Maximise paper gaps over the examination period for each student by having a consistent gap between examinations and maximise student study time (Ho, Lim and Oon, 2001).
- Minimise weekend conflicts when students have to sit for their examination during a weekend (See chapter 7 and Kendall and Mohd Hussin, 2005b).

§ Room satisfaction: All examinations in the same room should end at the same time so as not to disturb the examinees. All rooms must have the necessary equipment for an examination (disability access, tables, chairs, audio-visual equipment etc.)

§ Lecturer satisfaction: Schedule examinations with large enrolments earlier to help in the marking process. Satisfy lecturer preference for a specific time slot.

§ Management satisfaction: Schedule all large examinations within a shorter period to minimise the number of days required to rent an external

examination hall. Schedule examinations in the minimum number of rooms and larger halls to minimise the number of invigilators.

§ Invigilator satisfaction: Invigilators must be assigned an equal amount of invigilation duties. Satisfy invigilator preference for specific time slots. There is a requirement for one chief invigilator (with experience) and n number of other invigilators.

§ Other additional constraints (see Burke et al., 1996a for more details).

The management and invigilator satisfaction are related to the UiTM problem, which will be discussed in chapter 5. The above list of hard and soft constraints may be categorised differently by different institutions. Some soft constraints may go under the category of hard constraints while some soft constraints will be considered more important than other soft constraints. Thus, it is important for an automated scheduling system to be interactive and able to allow the user to enter their specific requirements and allocate higher weights to more important constraints such that these constraints can be minimised.

Examination timetabling problems in universities begin with the integration of examination data and processes from various departments, centres, faculties and/or branches. It is a complex problem due to the number of examinations that it needs to schedule. See chapter 8 for a detailed description of a timetabling problem for a large university. Thus, the easiest way to solve this complex problem is to tackle it in several stages:

1. Collect relevant data (such as list of courses, list of student course registrations, list of rooms, list of invigilators and other relevant constraints determined by departments). This requires the full cooperation of registrars, departments and faculties (McCollum, 1997). The frequent update of information is necessary to produce a good quality and accurate timetable at the appropriate time.

2. Schedule all examinations into available time slots so that no students are scheduled to sit for two examinations at the same time (conflict-free). Many researchers have modelled examination timetabling as a graph colouring problem, for example, (Burke et al., 1994a), (Terashima-Marin et al., 1999a), (Welsh and Powell, 1967) and therefore graph colouring heuristics are often used to construct the timetable. In addition to these heuristics, and taking into consideration other side constraints (Carter and Laporte, 1996), many researchers solve examination timetabling problems using meta-heuristics such as tabu search (e.g. White and Xie, 2001; Gaspero and Schaerf, 2001), simulated annealing (e.g. Thompson and Dowsland, 1996; 1998), genetic algorithms (e.g. Burke et al., 1994b, 1995, 1998; Terashima-Marin 1999b; Erben 2001; Ergul 1996) and memetic algorithms (e.g. Burke et al 1996b). This problem may be considered as an un-capacitated problem if the actual room capacity is not considered. A capacitated problem for examination timetabling includes a limit on the number of students scheduled in a slot but with no actual assignment of examinations to specific rooms. The benchmark datasets have one big room (e.g. sports hall) available for each time slot. More details of the algorithmic techniques are given in section 2.3.
3. Assign examinations to rooms. This problem is similar to a bin-packing problem (Burke et al., 1994a) where examinations have to be assigned to rooms such that the number of students sitting the examination does not exceed the room capacity. The problem becomes more complex when two examinations can be assigned into the same room or one examination can be split across several rooms.
4. Assign invigilators (proctors) to each room (Awad and Chinneck, 1998). This problem is modelled as an assignment problem (Marti et al., 2000), where invigilators are assigned to an examination room such that no invigilator is assigned to two places at the same time. Other factors, such as, invigilator preferences for particular slots and limited number of slots increases the problem complexity. Invigilator scheduling is included here because some universities (e.g. MARA University of Technology) use their own academic lecturers to

invigilate and it is highly desirable to have a schedule that is fair to the lecturers and does not affect their other academic responsibilities (lecturing, marking, research, conference participation etc.). However, this rule may not apply to institutions that hire others to do the invigilation. The invigilators assignment is usually done after the timetabling and is relatively straightforward.

The main objective of an examination timetable is to guarantee that all examinations are scheduled and students can sit all the examinations that they are required to do. A timetable is of good quality if all other constraints under consideration are minimised. For example, if the examination period is short, a gap of two slots between two examinations that a student has to sit (third order conflict) might be unavoidable and should be minimised, but if the examination period is long, third order conflict should be avoided and minimised. The objective function in timetabling is an aggregated weighted penalty where a weighted penalty is assigned to violated constraints. This is in contrast with the multi-criteria approaches where different objective functions are used to measure different constraints. Since the importance of each constraint can vary from one institution to another it can be difficult to determine the general weighted penalty for each constraint. This makes it difficult to specify a precise and generalised mathematical formulation of the real problem. Adding too many constraints can also increase the problem complexity and finding a feasible solution can be difficult and require plenty of resources. We can simplify the problem by taking into consideration the generalised form of the timetabling problem. The timetabling problem can be thought of as an optimisation problem since its objective is to minimise the number of violated constraints or penalties. A more technical description of the problem model and a specification of the examination timetabling problem is given in chapter 5.

2.3 Algorithmic Strategies

In most of the literature related to the educational timetabling problem, many

researchers propose solutions that have been developed for schools or universities. These proposals were implemented by applying various techniques that were developed for some instance of a real problem. Several surveys on automated timetabling problems have been published that classify timetabling problems and their solution methodologies.

An early survey by Carter (1986) presented an overview of practical applications for the examination timetabling problem from as early as 1964 until 1984. The applications were based on graph colouring heuristics designed to solve specific problems in particular schools. There was no integration and comparison between the approaches and as such a novice examination scheduler had little basis for selecting the best approach to work with.

Carter and Laporte (1996) extended the survey by classifying the algorithms into: cluster methods, sequential methods, generalised search strategies and constraint based approaches. At that point in time, most algorithms solved only the basic timetabling problem with simple constraints. To encourage more advanced research into timetabling problems, the authors made public a set of test problems that was first tested and reported by Carter et al. (1996). These benchmark datasets are also used in this thesis and details about these datasets can be found in chapter 5.

Burke et al. (1997) stressed the need and importance of an automated timetabling system and presented a very brief overview of some timetabling methods.

Schaerf (1999) conducted a survey on automated timetabling, categorising it into: school, course and examination. The paper gives a mathematical description of the basic search and optimisation problems, variants of the problems, and solution approaches published in the literature.

Burke and Petrovic (2002) and Petrovic and Burke (2004) presented an

overview of university timetabling problems and recent approaches in automated timetabling (see section 1.1).

The next section discusses some of the methods used and some application examples.

2.3.1 Cluster Method

This method is classified by Carter and Laporte (1996) and is solved in two phases:

Phase 1: Combine compatible (no or few conflicts) examinations in a block or cluster.

Phase 2: Sequence the cluster into specific periods minimising some objective or satisfying some constraints.

In the first phase, White and Chan (1979) group conflict free examinations by selecting examinations in order of their size while Leong and Yeong (1990) choose to select examinations in descending order of the number of conflicts. Lotfi and Cerveney (1991) arrange examinations by the number of events in conflict multiplied by the number of students in conflict with all other examinations. In the second phase, to sequence the group and to minimise the number of students with consecutive examinations, (White and Chan, 1979) modelled the problem as a travelling salesman problem where each cluster is a city and the cost to travel from city ' i ' to city ' j ' is the number of students who must sit examinations in both group i and group j . Therefore, finding a tour with minimum cost is equivalent to finding a sequence for the group that minimises the number of students with consecutive examinations.

Burke et al. (1994a) use a heuristic algorithm based on graph colouring to split

examinations into groups that can be scheduled together, and another algorithm to fit the examinations into rooms. The examinations are grouped together by successively selecting a colour set (vertices that can have the same colour), removing the coloured vertices and finding the next colour set in the reduced graph. The second algorithm fits examinations into rooms by selecting the smallest examination from a set of those scheduled together and placing it in the smallest room that it can fit into. If, by selecting a room, the capacity has been exceeded, the smallest examination is displaced into the next larger room and so on. If the examination cannot fit into any room, it is placed into an unassigned room list.

2.3.2 Sequential Methods and Constructive Heuristics

It is more efficient to find near optimal (or at least good quality) solutions using techniques known by the term heuristics. The following, by Reeves (1995), defines a heuristic:

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

Examples of heuristics are constructive heuristics or greedy heuristics. These heuristics construct initial solutions by sequentially initialising variables with values in the domain using a strategy that optimises cost. Several earlier papers (Broder, 1964; Cole 1964; Descroches et al., 1978; Laporte and Descroches, 1984) use sequential methods which are discussed in Carter (1986). These methods are normally divided into two phases; a construction phase and an improvement phase. The construction phase consist of a constructive heuristic that produces an initial timetable. The algorithm then selects examinations according to a sequencing strategy and assigns it to either the first feasible slot (to minimise number of slots) or

to the best feasible period (to minimise the cost of its objectives). The second phase is the improvement phase that makes modifications and improvements. The improvements may just be a manual process of selecting from different generated timetables from the first phase. It may also involve performing ' k -opt' interchanges looking for improvement. For example, 1-opt moves examination to a better slot and 2-opt swaps the slots of two examinations.

Constructive heuristics are used in the initialisation phase of sequential methods and in constructing near optimal (or at least reasonable quality) initial solutions. Constructive heuristics are normally an iterative procedure that selects an element in the variable list and assigns it a feasible value in the domain. The constructive heuristics vary from random techniques, greedy techniques and adaptive techniques.

In the examination timetabling problem, a constructive heuristic will include strategies to order and select which examination is to be scheduled next and the strategies to assign an examination to a feasible slot. The following list describes some strategies that can be used to order and select which examination to schedule. The first six strategies are the simplest methods that have evolved from graph colouring heuristics and are used by many researchers referenced in other parts of this thesis.

- 1) Largest degree – order examinations in descending order by the number of other examinations that are in conflict. Examinations that have too many other examinations in conflict should be scheduled first.
- 2) Least saturation degree – order examinations in ascending order by the number of available slots. Examinations with the least number of available slots should be scheduled first.

- 3) Largest weighted degree – order examinations by the total number of students in conflict with other examinations. Examination with the largest total number of students in conflict should be scheduled first.
- 4) Largest enrolment – order examinations by the total number of student enrolments. Examination with the largest total number of student enrolments should be scheduled first.
- 5) Colour degree – order examinations by the number of other examinations that are in conflict and have already been scheduled. Examination with the greatest number of conflicting examinations that have already been scheduled should be scheduled first.
- 6) Random – Select examination at random or next examination in the input list.
- 7) Linear combination of two criteria (enrolment size of examination and the number of conflicts). By varying the relative weights of each criterion, several timetable solutions can be produced (Johnson 1990).
- 8) Tournament selection – Select based on other simpler strategies (the first six strategies described above) from a randomly generated subset of examinations to be scheduled (Burke et al., 1998).
- 9) Bias selection – Select examination at random from the best (based on other simpler strategies) n examinations (Burke et al., 1998).
- 10) Examinations in the largest clique – This method finds the largest clique and selects examinations from the largest clique first (Carter and Johnson, 2001). A clique in a graph G is a sub-graph where any two vertices in the

sub-graph are connected by an edge and the size of a clique is the number of vertices it contains.

- 11) Adaptive ordering – Select examinations based on an order. The ordering is adapted over time with a heuristic modifier that gives a new measure of difficulty for an examination. For example, if an examination is difficult or cannot be scheduled within a given iteration, the examination will be moved to the front of the list in the next iteration so that it can be scheduled earlier (Burke and Newall, 2004).
- 12) Fuzzy technique – Use fuzzy weights (based on a two ordering criteria) to determine the next examination to be scheduled (Asmuni et al., 2004)
- 13) Graph based constructive hyper-heuristic – A hyper-heuristic that chooses next exam to be scheduled from a set of graph colouring heuristics (Burke et al., 2005a; 2005b).

In a constructive heuristic, once an examination is selected, the next decision is to select a feasible slot that does not violate any hard constraints (capacity of a slot can be easily included). The following strategies can be found in the literature:

- § Random – select a feasible slot at random (selected from all slots or restricted to n slots).
- § First or last feasible slot.
- § The slot that causes minimum penalty cost.
- § The feasible slot that has the least number of examinations already scheduled in the slot. This will induce a spread of examinations during the examination period.
- § The feasible slot that has the largest number of examinations already scheduled in the slot. This is to minimise the number of slots used.

§ Peckish-assign (Corne and Ross, 1996) – Find the best slot from a set of slots selected at random.

If no feasible slot can be found for an examination, a ‘limited backtracking’ (Carter et al. 1996) is used to ensure that the initial solution is feasible. The backtracking strategy involves bumping the examination with the least number of conflicts and reassigning it to another slot, and so on. This process may cause a cycle and Casey and Thompson (2003) introduce a tabu list to prevent cycling. Some constructive heuristics may not produce a feasible solution and the examinations are scheduled in a dummy slot (Merlot et al., 2003), thus increasing the length of a schedule. The length of the schedule can be reduced later during exploration of the search space. The constructive heuristic can also present a feasible solution and a list of unscheduled examinations (Kendall and Mohd Hussin, 2005a). These unscheduled examinations can be rescheduled during exploration of the search space.

2.3.3 Meta-heuristics

Heuristics are normally problem-dependent such that methods that work for an instance of one problem cannot be used to solve another instance. Therefore, there is a need to design techniques that are more general and can be applied to a larger number of applications. One such technique is the local neighbourhood search heuristic that will be discussed in the next section.

The more modern local search algorithms are categorised as meta-heuristics because they include some control or guidance over the heuristics. Osman and Kelly (1996) define a meta-heuristic as follows:

“A meta-heuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces

using learning strategies to structure information in order to find efficiently near-optimal solutions.”

Glover and Laguna (1997) defines a meta-heuristic as follows:

“A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.”

Voß et al. (1999) defines a meta-heuristic as

“an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method.”

Meta-heuristics are a class of heuristic techniques that have been successfully applied to solve a wide range of combinatorial optimisation problems over the years (Reeves, 1995; Osman and Kelly, 1996; Osman and Laporte, 1996; Aarts and Lenstra, 1997; Voß et al., 1999; Corne et al, 1999; Glover and Kochenberger, 2003). Examples of meta-heuristics (Voß, 2000) include simulated annealing, tabu search, evolutionary algorithms, GRASP and VNS.

2.3.3.1 Local Neighbourhood Search

The basic idea of finding a local optimum motivates the classical neighbourhood search method known as local search. To solve a combinatorial optimisation problem using local search, we make the following assumptions:

- § S = a set of solutions and s = a solution
- § $f(s)$ is a cost function applied to solution s
- § an objective, such as *minimise* $f(s)$
- § a neighbourhood function, $N(s)$, mapping any solution s in S to a set of ‘related’ s' solutions in S .

Local search is an iterative algorithm that begins with an initial solution, searches the solution space by generating a sequence of solutions and at each step it selects the next solution to move to from the neighbourhood of the current solution. A framework for general local search techniques is as shown in Figure 2.1.

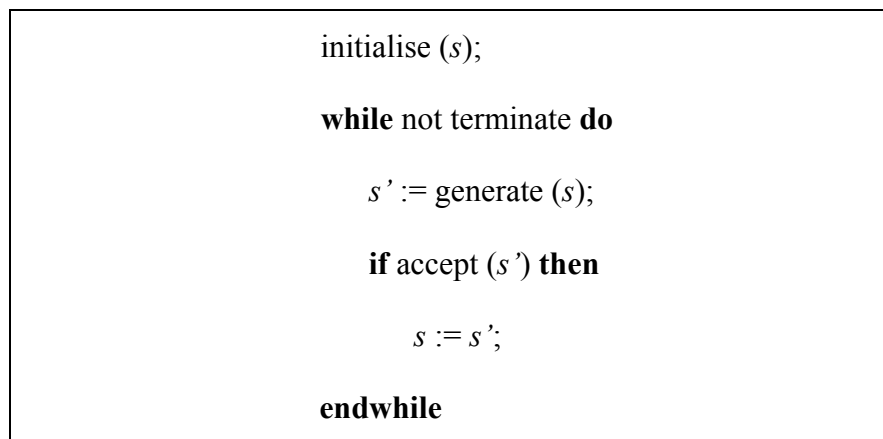


Figure 2.1: Framework of a general local search technique

The *initialise(s)* procedure generates the starting solution. The *terminate(s)* function determines when to stop searching. The *generate(s)* function determines and returns a new solution s' selected from the neighbourhood. The *accept(s')* function determines whether the current solution s will be updated with the new

modified solution s' . The looping structure (begins with *while*) and the functions within the loop (*terminate(s)*, *generate(s)* and *accept(s')* functions) are the improvement phase that determines which of the solutions in neighbourhood $N(s)$ will replace the current solution.

The nature of the improvement phase determines the type of heuristic algorithm used. Examples include:

- § **Random Walk.** The most basic and probably least useful. It selects a solution at random from $N(s)$ and accepts it. This technique is occasionally used in combination with other techniques as a diversification strategy.
- § **Simple Descent.** Often referred to as hill climbing. This is the classic local search technique. At each step a candidate solution, s' is selected at random from $N(s)$. If $f(s')$ is an improvement over $f(s)$ then s' is accepted to replace s .
- § **Steepest Descent.** Each solution in $N(s)$ is evaluated and the solution with the best $f(s)$ value returned by the *generate(s)* function. If $f(s')$ is an improvement over $f(s)$ then s' is accepted to replace s , otherwise the algorithm terminates. Steepest descent can be slow with large neighbourhood. Limiting evaluation to a candidate list is sometimes used to increase speed. The candidate list is some subset of $N(s)$, selected at random or according to some preset rules.

Local search techniques do not necessarily explore all the solution space and cannot guarantee to find an optimal solution. The main problem with the above techniques is that they can easily get stuck in a valley of local minima, where all neighbouring solution states are worse, or it may be cycling on a plateaux of equal solution states. The performance of a local search is dependent upon how it decides to accept new solution states, the type of neighbourhood structure (single- or multi-neighbourhood and the size of the neighbourhood), efficient neighbourhood search techniques, and the initial solution (Ribeiro and Hansen 2002). These issues are addressed in the more modern meta-heuristics such as simulated annealing, tabu search, variable neighbourhood search, genetic algorithms, etc.

In educational timetabling problems, hill climbing (simple descent) is now frequently used in a hybridised form with other meta-heuristics (Burke et al. 1996b; Burke and Newall 1999; Schaerf and Di Gaspero 2001; Merlot et al 2003; Kendall and Mohd Hussin 2005a, 2005b). It is also used as a control algorithm whose performance is compared against other new algorithms (Corne and Ross 1995; Burke and Newall 2003).

2.3.3.2 Simulated Annealing and the Great Deluge Algorithm

Simulated annealing (SA) is an extension of the simple descent algorithm but applies a less strict acceptance rule. Improving moves are still always accepted, but other (worse) moves are accepted probabilistically. This method is based on ideas inspired by the Metropolis algorithm for statistical mechanics (Metropolis et al., 1953). Annealing is the process of cooling material in a heat bath. The material is heated to high energy where there are frequent state changes. It is then gradually cooled to a low energy state where state changes are rare. The Metropolis algorithm simulates the change in energy of the system when subjected to a cooling process, until it converges to a steady state. Kirkpatrick et al. (1983) suggested that simulated annealing could be used to search for feasible solutions in an optimisation problem whose objective is to converge to an optimum state. In simulated annealing, the temperature and the change in the evaluation function determine how likely it is that a worsening move will be accepted. A high temperature gives a high probability of acceptance and a low temperature causes moves to be rejected with a high probability. Acceptance probability is calculated as $\exp(-\delta/t)$ where δ is the magnitude of $f(s') - f(s)$ (difference between the current and new solution cost function) and t is the current value of a temperature parameter. As the algorithm proceeds, the temperature is lowered. The cooling schedule (start temperature, end temperature, how and when the temperature is lowered) and choice of neighbourhood structure is important (Aarts and Korst, 1998). The search starts with high temperature to allow a large proportion of worse moves to be accepted and then the temperature is slowly reduced so as to reach an equilibrium.

Simulated annealing has been applied to various optimisation problems. Dowsland (1995) presented a comprehensive introduction to simulated annealing techniques and references to research publications.

Abramson (1991) and Abramson et al. (1999) have successfully applied simulated annealing to school timetabling problems. Thompson and Dowsland (1996; 1998) solved examination timetabling problem in two phases. The first phase produced a feasible timetable, which eliminated clashes and satisfied resources constraints. The second phase removed all infeasible solutions from the solution space and searched the remainder for a solution that optimised the secondary objectives. Their experiments show that the solution quality depends not only on the cooling schedule and neighbourhood but also on the way it is sampled.

Criticisms against simulated annealing have been made because of the necessity to find a cooling schedule for a particular instance of a problem. The great deluge algorithm (GDA) introduced by Dueck (1993) is an alternative to simulated annealing where the algorithm begins with a *water level* equivalent to its initial solution quality, and a pre-determined rate to decrease *water level*. The pre-determined rate is the only parameter for this algorithm. The algorithm accepts worsening solutions if the penalty cost is less than the *water level*. The *water level* is decreased by the pre-determined rate at each iteration. Dueck applied this algorithm to the travelling salesman problem. The level was decreased by at least 0.01 or by the difference between the level and the length of the current tour divided by 500.

Due to the advantage of using less parameter, the great deluge algorithm has been used favourably in several other implementations of meta-heuristics. Burke and Newall (2003) apply it to an examination timetabling problem and produced good results on 13 benchmark datasets (the same datasets used in this thesis). They refer to the pre-determined rate to decrease the *water level* as *decay rate*. They compare performance between simulated annealing, hill climbing and the great deluge algorithm (with good initial solutions). The *decay rate* is computed as the initial

solution quality multiplied by a user provided factor divided by the number of iterations. The algorithm ran for 20,000,000 and 200,000,000 iterations, terminating if there is no improvement in the last 1,000,000 iterations. The great deluge algorithm was found to perform better when compared to simulated annealing and hill climbing. It also performs better in most cases compared to other methods such as tabu search method (Di Gaspero and Schaerf, 2001) and graph colouring heuristics (Carter et al., 1996). Burke et al. (2004a) implemented a time-predefined simulated annealing and time-predefined great deluge algorithm. The modified great deluge algorithm includes two user-defined parameters i.e. time to run the algorithm and an approximation of the desired solution quality. The decay rate is now computed as the difference between the initial solution quality and the desired solution quality divided by time (or number of moves in the desired time). It also enhances the acceptance criteria of great deluge algorithm by accepting improving moves even if they are greater than the level. Again, the time-predefined great deluge algorithm was found to be superior compared to other methods.

2.3.3.3 Tabu Search.

The basic form of tabu search (TS) was proposed by Glover (1986) to solve combinatorial optimisation problems. Since then, TS-based methods have been shown to be successful in several areas of optimisation. For detailed discussions and references see Glover (1986), Glover and Laguna (1997), Voß (2000), Voß et al (1999), Osman and Laporte (1996). The following is a definition of tabu search by Glover and Laguna (1997):

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

The basic concept of tabu search is an extension of steepest descent by incorporating adaptive memory and responsive exploration. It uses memory not only to keep track of the current best solution but it also stores information related to the exploration process. Starting from the initial solution S_0 , the algorithm iteratively

explores a subset $N'(s)$ of the neighbourhood, $N(s)$, of the current solution s . The member with the lowest (assuming minimisation) value becomes the current solution irrespective of whether its value is better or worse than the current solution. Accepting a non-improving move will allow the search to continue to explore areas beyond local optima. However this will typically lead to cycling, that is, repeatedly moving between some small set of solutions. To avoid this, it uses memory to store a tabu list. This list contains moves that satisfy some *tabu restriction* criteria and these moves are prohibited for a predetermined number of iterations (*tabu tenure*). Moves that are in the tabu list are said to have a *tabu-active status*.

An *aspiration criteria* is used to make a solution tabu free if the resultant evaluation is of sufficient quality and can prevent cycling. It plays the role of guiding the search process. An appropriate choice of aspiration criteria is important to produce good results. The aspiration criteria can be time-dependent, time-independent or aspiration by default. Other aspiration criteria discussed by Reeves (1995) are aspiration by objective, aspiration by search direction and aspiration by influence.

Tabu restrictions are used to avoid repetitions of a search path. Tabu restriction is typically activated only in the case where its attributes occur within a limited number of iterations prior to the present iteration (a recency-base restriction) or occur with a certain frequency over a larger number of iterations (a frequency-based restriction). In other words, a tabu restriction is enforced only when the attributes of the underlying solution satisfy certain thresholds of recency or frequency.

Some stopping conditions for tabu search could be, a maximum number of iterations, or the number of iterations since the last improvement is larger than a specified number. Other important control parameters that need to be considered are the length of the tabu list, the aspiration function and the cardinality of the set of neighbour solutions tested at each iteration.

Glover and Laguna (1997) also describe two important strategies used in tabu search: *intensification* and *diversification*. *Intensification* strategies involve changing the choice rules to intensify the search to examine neighbours of elite solutions. The idea is that if certain regions contained good solutions in the past they may possibly yield better solutions in the future. The *diversification* stage encourages the search process to examine unvisited regions and to generate solutions that differ significantly. There are at least three approaches for defining the tabu list size; fixed to a predetermined value, randomly chosen from a specific range or dynamically changed by adjusting the value.

More advanced research using tabu search includes longer term memory, with associated intensification and diversification strategies. This approach is referred to as Adaptive Memory Programming (AMP) (Glover and Laguna, 1997; Taillard et al. 2001).

One of the earliest papers that uses tabu search to solve educational timetabling problems is Hertz (1991) who deals with school timetabling and some restricted form of soft constraints. Schaerf and Schaerf (1995) apply tabu search techniques in scheduling lectures to periods for a large high-school. They represented their timetable as an integer-valued matrix M_{mp} such that each row j of M represents the weekly assignment for teacher t_j . The type of moves used are atomic: moving a lecture to another period and double moves which are moves made by a pair of atomic moves. The algorithm used a tabu search with atomic moves interleaved with a randomised non-ascendant method (RNA) using double moves. The RNA is used to generate the initial solution and is applied again after TS has given no improvement for a given number of iterations. The cycle is repeated allowing TS to start in a different direction. The tabu list is of variable size. Each move is inserted into the tabu list with a number of iterations i selected at random within a predetermined range. The *tabu tenure* therefore varies for each move. Each time a move is inserted the value i of all moves in the list will be decremented and once it reaches zero the move is removed. The algorithm uses the simplest *aspiration*

criteria of accepting a tabu move only if it improves the current best solution. The algorithm gave good results for schools of various types, and for different settings of the weights of the objective functions. The timetable produced is better than the manual timetable and it was able to schedule 90-95% of the lectures.

Di Gaspero and Schaerf (2001) continued this research using tabu search for the examination timetabling problem. They modified their objective function using a shifting penalty mechanism (varying weights on soft and hard constraints) thus causing the search to explore different solution spaces. In order to decide which examinations are to be moved, they maintain two violation lists: a list of examinations that violate either hard or soft constraints and a list of examinations that violate hard constraints only. During the search, they experiment on various strategies using shifting penalty mechanisms and the two violation lists. These two features combined with a variable-size tabu list and starting the search with a good initial solution, were found to be helpful in directing the search into promising regions.

Di Gaspero (2002) and Di Gaspero and Schaerf (2003) further enhanced their algorithm by employing a multi-neighbourhood strategy applied to examination timetabling and course scheduling respectively. In examination timetabling, Di Gaspero (2002) applied a combination of tabu search with different neighbourhoods (union and composition). He categorised these combinations into local search that specialised in optimising the objective function (*recolour*), perturbing current solution (*shake*) or obtaining more improvement (*kick*). The *recolour* and *shake* algorithms were applied in sequence until no further improvement and the algorithm ended with the *kick*. The final results on seven benchmark datasets were better compared to the basic tabu search with a single neighbourhood.

Dowland (1998) showed that it is possible to design robust solutions based on simulated annealing and tabu search by applying the algorithms on different case studies of scheduling, timetabling and staff-rostering problems in the education and

hospital sectors. She applies a varying *tabu restriction* on different moves and uses a frequency-based *diversification* mechanism that penalised attributes that occur very frequently. Some of the modifications improved the tabu search but the implementation frequently depends on the precise details of the problem. Some of these modifications are different cost functions, variable tabu length list, combining moves into chains, strategic oscillation that force the search into different regions and prominent candidate list strategies.

White and Xie (2001) called their algorithm OTTABU and used it to provide an examination timetable using data provided by the University of Ottawa. The problem is modelled as a graph. The initial solution was generated using an algorithm derived from bin packing algorithms (largest enrolment first). The initial solution does not guarantee a feasible solution. A new solution is obtained by an atomic move. Their system used recency based short-term memory (TS) and frequency based long-term memory (TL) to improve the solution quality. The tenure of the short-term tabu list is found not to be critical if both longer term and short-term memory are used. Their experiments showed that longer term memory produced better schedules and since a longer term memory list can reduce its effectiveness, a quantitative analysis method is used to estimate the appropriate length of the longer term tabu list and a controlled tabu relaxation technique (emptying entries in TS and TL) is used to diversify the search. White, Xie and Zonjic (2004) expand their research to include comparisons between their results and other published algorithms.

Wright (2001) incorporated sub cost guided search in both simulated annealing and tabu threshold acceptance methods. In tabu thresholding, the intensification and diversification are explicitly divided into two separate phases - the improving (intensifying) phase and the mixed (diversifying) phase. He used a focussed form of diversification by accepting a solution even though the overall cost had increased but one of the sub costs had decreased. He experimented on modified school timetabling data and found that it significantly improved the results.

2.3.3.4 Greedy Randomised Adaptive Search Procedure

The greedy randomised adaptive search procedure (GRASP) is a multi-start iterative procedure and each GRASP iteration consists of a construction phase and an improvement phase (Resende and Ribeiro, 2003). The construction phase consists of an adaptive and probabilistic greedy constructive heuristic that builds a solution from scratch. First, it arranges all candidate elements based on the benefit of choosing an element. The benefit of choosing an element is adaptive and dynamic since the benefit is associated with the previous choice of element. Once a list of the best candidates, called the restricted candidate list (RCL), is constructed, the algorithm randomly or greedily (depending on a parameter that controls the amount of greediness and randomness) selects a candidate element to be initialised. This step is repeated until all candidates are initialised. The construction phase ensures that each GRASP iteration starts with a different solution. The improvement phase can be a simple local search procedure. The solution from the improvement phase is kept if it is the best so far and, if the terminating condition is not met, the algorithm repeats with a new solution. The GRASP procedure can be viewed as a local search with multi-start that relies on a good sampling of initial points that lie in promising regions of the search space. The improvement phase produces a number of locally optimal solutions that can be utilised as single solution or multiple solutions.

A number of enhancements to the basic GRASP algorithm can be found in other methods presented by Resende and Ribeiro (2003, 2005). The methods have been applied in many other operations research problems and industrial applications (survey by Resende and Ribeiro, 2003; 2005).

Casey and Thompson (2003) applied an enhanced GRASP algorithm on examination timetabling problems. The problem solved is a basic examination timetabling problem of scheduling examinations into timeslots so that clashing examinations are assigned in different timeslots. The objective is to minimise the proximity of examinations in students' timetables. Some parameter tuning was

carried out to find the best constructive heuristic and an appropriate size n for the restricted candidate list (RCL) that produce feasible solutions to a set of examination timetabling benchmark datasets. Examinations are ordered according to one of the criteria proposed by Carter et al. (1996) with the next examination to be scheduled being chosen from the RCL using a roulette wheel selection and is assigned to the first available period. Similar to findings by Carter et al. (1996), a saturation degree method of ordering examinations was found to be more successful. If there is no feasible period for an examination, a backtracking technique (similar to Carter et al., 1996) is applied using a tabu list to prevent cycling. The improvement phase was implemented with a neighbourhood of Kempe chains, including a limited form of simulated annealing and memory function to allow for diversification of the search. The method was found to be robust, performing well across all datasets. It also performs well by putting more emphasis on the improvement phase rather than increasing GRASP iterations. If time is not an important factor, GRASP iterations can be increased, thus producing an advantage of finding more local minima.

2.3.3.5 Genetic Algorithms

In addition to the type of single solution search algorithm that was described in the previous sections, there is a growing interest in population-based meta-heuristics. Those meta-heuristics include evolutionary algorithms (EA: genetic algorithms (GA) (Holland, 1975), evolution strategies (ES) (Rechenberg, 1965), genetic programming (Koza, 1992), etc.), ant colonies (AC) (Dorigo et al., 1996), scatter search (SS) (Glover et al., 2000), and so on.

Genetic algorithms (GA) are an evolutionary algorithm that is a directed random search technique (Holland 1975). Earlier work by Fraser (1957) (discussed in Fogel and Fraser, 2000) and Bremermann 1962 (discussed in Fogel and Anderson, 2000), provided the foundations of genetic algorithms, which were later popularised by Holland (1975) and De Jong (1975). Goldberg (1989), Michalewicz (1996) and Mitchell (1996) provide excellent introductions on genetic algorithms and their

various applications.

Genetic algorithms operate on a population of solutions represented by some encoding. Each member of the population consists of a number of genes, representing a unit of information. The algorithm begins by creating an initial population (maybe randomly generated). Next the solution members are evaluated by computing their fitness. The selection procedure then reproduce more copies of individuals with higher fitness values. The selection procedure influences the search direction towards promising areas. Genetic operators such as crossover (combining promising parts of two parents) and mutation (small random changes) are used to create new populations. Important control parameters of a simple genetic algorithm include the population size, crossover rate and mutation rate. A large population size increases computational time, the crossover rate determines the frequency of crossover operation. A mutation rate that is too high may cause diversity in the population and might cause instability.

Ross et al. (1994) have successfully solved university lecture timetabling problems using genetic algorithm where each chromosome is a timetable solution containing a list of events. Each event is denoted by the time, place and lecturer involved. A fitness function is calculated based on the number and kinds of constraints violated (spreading events, capacity constraint etc.). The algorithm uses a population size of 50, uniform crossover, gene-to-gene mutation, elitist reproduction and tournament selection.

Ross et al. (1998) applied genetic algorithm to examination timetabling problems using a simple direct representation where a chromosome is a timetable solution arranged as a list of examinations where the i^{th} number indicate the timeslot for examination i .

Burke et al. (1998) studied the diversity of solutions produced by several graph colouring heuristics that are used to initialise solution population of a genetic

algorithm. They used a random element in their solution initialisation by using bias selection and tournament selection (see section 2.3.2).

2.3.3.6 Memetic Algorithms

Memetic Algorithms (MA) (Moscato, 1989; 1999) are an evolutionary algorithm that includes knowledge of the problem in the form of heuristics, approximate algorithms, local search techniques, specialised recombination operators, truncated exact methods, etc. The term was coined by Moscato (1989) from Richard Dawkin's (Dawkins, 1990) term 'meme'. The difference between memes and genes is that memes are ideas or concepts that are passed around and adapted to the environment while genes are passed down without alteration. In the evolutionary process, meme includes other characteristics or knowledge about the problem that are processed and enhanced by the communicating parts. Most memetic algorithms use genetic algorithm and local search on each population member after each generation.

Burke and Landa Silva (2004) presented a review on strategies used by researchers using memetic algorithms for scheduling and timetabling problems.

Burke et al. (1996b) used a hybrid genetic algorithm that was successfully tested on random data and real data. They used heuristically guided crossover operators and initialised their initial population with a set of feasible timetables.

2.3.3.7 Scatter Search Algorithms

Scatter search algorithms are an evolutionary population-based method (introduced by Glover, 1977) that combines solution vectors by linear combinations to produce new ones through successive generations. Marti et al. (2000) solved the problem of assigning invigilators to rooms using a scatter search solution procedure. The problem is formulated as an integer program with a weighted objective function that combines a preference function and a workload-fairness function. It uses the

concept of combining good solutions in order to yield a better solution. The solution combination method is based on a voting system where the solution votes for specific invigilator assignment for each examination. The vote is associated with a measurement of the merit of assigning an invigilator to an examination that takes into account the preference of the invigilator for some examination. A repair procedure is applied if the combination method results in a solution that violates the hard constraint of assigning an invigilator to two examinations at the same time.

2.3.3.8 Multiobjective Approaches

The objective of combinatorial optimisation problems is to find the optimal value of a set of variables subject to the satisfaction of a set of constraints. In many real world scheduling and timetabling problems, the quality of the solution is measured by the satisfaction of a number of criteria (Landa Silva et al., 2004). If each criterion is not in conflict with each other, we can solve the problem using a single linear objective function with weights attached to each criterion. In certain cases, some criteria are in conflict with each other, i.e. by improving one criterion we will inevitably have to compromise another criterion. For example, in an examination timetabling problem, two of the criteria to measure the quality of a solution are the schedule length (minimise) and the proximity cost (minimise) of students sitting examinations. Reducing the schedule length implies that students will have less of a time gap between examinations, thus increasing the proximity cost. In multiobjective approaches, each criterion is associated with an objective function that needs to be minimised or maximised. Landa Silva et al. (2004) provides a good introduction to multi-criteria decision-making and multiobjective optimisation in timetabling and scheduling with references to application examples in personnel scheduling, machine scheduling and educational timetabling.

A multi-criteria approach to solve the examination timetabling problem is described in (Burke et al, 2001b). Soft constraints imposed on the problem are translated into different criteria. User-defined weights associated with each criterion

indicate the importance of the criterion with respect to solution quality. In the initialisation phase, a sequential algorithm (based on graph colouring saturation degree algorithm) is used to generate high quality initial solutions with respect to each criterion separately. The constructive algorithm assigns an examination to a slot if the assignment causes the least increase violations of a criterion value. In the optimisation phase, trade-offs between criteria values are carried out in order to find a compromised solution with respect to all the criteria simultaneously. Petrovic and Bykov (2003) enhanced the great deluge algorithm by driving the search of the criteria space through a specified trajectory by dynamic change of criteria weights.

2.3.3.9 Case-Based Reasoning Approaches

Case-Based Reasoning (CBR) (Kolodner, 1993) is a knowledge-based paradigm that attempts to use previous knowledge stored as cases in a case base so as to build solutions for current problems. The main idea in this approach is not to reinvent the wheel by reusing similar solutions that are already stored in a case base, but to adapt a retrieved solution to the new problem and subsequently enrich the case base with new knowledge or cases. Details as to how CBR has been applied to university course timetabling problems can be found in Burke et al. (2001a; 2002) and Qu (2002).

An issue that requires attention in CBR is in the definition of similarity between two timetabling problems (Burke et al., 2001a;2004c; Yong and Petrovic, 2004). Burke et al. (2004c) studied features of examination timetabling problems and how these features should be included in a similarity measure. They conclude that the number of examinations plays an important role in determining how difficult a problem is and which techniques are better suited for this problem. Other features such as number of students and number of slots are important factors as well. Features of a conflict matrix such as its density, variation of degree of examinations from its average and cliques found within the conflicts gave a good indication of the problem structure. Yong and Petrovic (2004) compared two similarity measures

based on weighted and un-weighted graph representations and an improved fuzzy similarity measure based on weighted graph representation. A combination of a case base and similarity measure was employed to choose a sequential heuristic (in the initial solution construction phase) for the same set of examination timetabling benchmark problems used in this thesis. This approach is able to select the proper heuristic that produces good initial solutions for the great deluge algorithm. The results on benchmark problems were better compared to other initialisation methods techniques for great deluge algorithm and other state of the art meta-heuristics.

2.3.4 Other Methods

There are other approaches applied on the benchmark datasets that do not exactly fall under the categories discussed in the previous sections.

Caramia et al. (2001) use a greedy constructive heuristic with an optimiser. Their method has produce four of the best results for the benchmark datasets (see table 2.1). Merlot et al. (2003) use a hybrid algorithm (constraint method, simulated annealing and hill climbing) to find feasible schedule for the benchmark datasets and a new dataset from Melbourne University. Abdullah et al. (2005) use a sequential solution improvement technique that search efficiently over a large set of neighbourhood (Ahuja-Orlin) solutions. Their method has produce three of the best results for the benchmark datasets (see table 2.1). The drawback of this method is a long run time in constructing the improvement graph.

2.4 Conclusions

This chapter has introduced the general educational timetabling problem with emphasis on examination timetabling. Various heuristics and meta-heuristics techniques have been applied to timetabling and scheduling problems. The techniques and performance on examination timetabling benchmark datasets are well studied and analysed by several researchers in the examination timetabling

community. It seems that there is a tendency to hybridise several techniques (Talbi 2002) in an approach to solve optimisation problems. Obviously, we should exploit good features of a technique in our approach such that performance can be improved.

In 1996, Carter et al. (1996) made available examination timetabling benchmark datasets, thus making it easier for the timetabling community to compare which techniques are more efficient and robust. Since an optimal solution for these benchmark datasets is not known, we can only compare results from techniques that use the same method of evaluation. Table 2.1 presents a table of results from different techniques applied on the benchmark dataset.

Table 2.1: Published Benchmark Results (Uncapacitated Examination Timetabling Problem)

Datasets	Car-F92	Car-S91	Ear-F83	Hec-S92	Kfu-S93	Lse-F91	Pur-S93	Sta-F83	Tre-S92	Uta-S92	Ute-S92	Yor-F83	Penalty per student for all datasets tested by each method.	Penalty per student for all datasets (except pur-s93).
Number of Exams	543	682	190	81	461	381	2419	139	261	622	184	181		
Number of Slots	32	35	24	18	20	18	42	13	23	35	10	21		
Number of Students	18,419	16,925	1,125	2,823	5,349	2,726	30,032	611	4,360	21,267	2,750	941		
Number of Student Enrolment	55,522	56,877	8,109	10,632	25,113	10,918	120,681	5,751	14,901	58,979	11,793	6,034		
Conflict Matrix Density	13.80%	12.80%	26.70%	42.00%	5.60%	6.30%	2.90%	14.40%	17.60%	12.60%	8.50%	28.90%		
Abdullah et al. 2005; Ahuja-Orlin Large Neighbourhood	4.36	5.21	34.87	10.28	13.46	10.24		150.28	8.13	3.63	24.21	36.11	8.30	8.30
Asmuni et al. 2004; Fuzzy expert system with multiple ordering criteria.	4.56	5.29	37.02	11.78	15.81	12.09	-	160.42	8.67	3.57	27.78	40.66	8.96	8.96
Burke & Newall 2003; Adaptive initial solution generation and great deluge algorithm.	4.1	4.65	37.05	11.54	13.9	10.82	-	168.73	8.35	3.2	25.83	37.28	8.36	8.36
Burke et al. 2004a; Great deluge algorithm.	4.2	4.8	35.4	10.8	13.7	10.4	4.8	159.1	8.3	3.4	25.7	36.7	7.32	8.3
Burke et al. 2005a; Hybrid graph colouring heuristics within a hyper-heuristic.	-	-	45.60	-	-	-	-	158.2	-	4.52	35.40	-	13.26	-
Caramia et al. 2001; Greedy constructive heuristic with an optimiser.	6	6.6	29.3	9.2	13.8	9.6	3.7	158.2	9.4	3.5	24.4	36.2	7.51	8.98

Datasets	Car-F92	Car-S91	Ear-F83	Hec-S92	Kfu-S93	Lse-F91	Pur-S93	Sta-F83	Tre-S92	Uta-S92	Ute-S92	Yor-F83	Penalty per student for all datasets tested by each method.	Penalty per student for all datasets (except pur-s93).
Number of Exams	543	682	190	81	461	381	2419	139	261	622	184	181		
Number of Slots	32	35	24	18	20	18	42	13	23	35	10	21		
Number of Students	18,419	16,925	1,125	2,823	5,349	2,726	30,032	611	4,360	21,267	2,750	941		
Number of Student Enrolment	55,522	56,877	8,109	10,632	25,113	10,918	120,681	5,751	14,901	58,979	11,793	6,034		
Conflict Matrix Density	13.80%	12.80%	26.70%	42.00%	5.60%	6.30%	2.90%	14.40%	17.60%	12.60%	8.50%	28.90%		
Carter et al. 1996; Constructive heuristics with backtracking	6.2	7.1	36.4	10.8	14.0	10.5	3.9	161.5	9.6	3.5	25.8	41.7	7.93	9.5
Casey & Thompson 2003; Greedy randomised adaptive search procedure.	4.4	5.4	34.8	10.8	14.1	14.7	-	134.9	8.7	-	25.4	37.5	10.41	
Côté et al. 2005; Hybrid multi-objective evolutionary algorithm.	4.2	5.4	34.2	10.4	14.3	11.3		157.0	8.6	3.5	25.3	36.4	8.48	8.48
Di Gaspero 2002; Tabu search with multi-neighbourhoods.		5.7	39.4	10.9	-	12.6	-	157.4	-	4.1	-	39.7	9.19	
Di Gaspero and Schaerf 2001; Tabu search	5.2	6.2	45.7	12.4	18.0	15.5		160.8	10.0	4.2	29.0	41.0	10.03	10.03
Kendall & Mohd Hussin 2005a; 2005c; Tabu search based hyper-heuristic.	4.40	4.93	38.14	10.74	15.55	12.35	5.41	157.38	8.35	3.77	27.12	37.99	7.83	8.77
Kendall & Mohd Hussin 2005d; Reactive tabu search based hyper-heuristic.	4.61	5.38	38.57	10.82	16.22	13.00	5.84	154.19	8.98	4.04	27.28	38.58	8.18	9.09

Datasets	Car-F92	Car-S91	Ear-F83	Hec-S92	Kfu-S93	Lse-F91	Pur-S93	Sta-F83	Tre-S92	Uta-S92	Ute-S92	Yor-F83	Penalty per student for all datasets tested by each method.	Penalty per student for all datasets (except pur-s93).
Number of Exams	543	682	190	81	461	381	2419	139	261	622	184	181		
Number of Slots	32	35	24	18	20	18	42	13	23	35	10	21		
Number of Students	18,419	16,925	1,125	2,823	5,349	2,726	30,032	611	4,360	21,267	2,750	941		
Number of Student Enrolment	55,522	56,877	8,109	10,632	25,113	10,918	120,681	5,751	14,901	58,979	11,793	6,034		
Conflict Matrix Density	13.80%	12.80%	26.70%	42.00%	5.60%	6.30%	2.90%	14.40%	17.60%	12.60%	8.50%	28.90%		
Merlot et al. 2003; hybrid of constraint programming, simulated annealing and hill climbing with Kempe chain neighbourhood	4.3	5.1	35.1	10.6	13.5	10.5	-	157.3	8.4	3.5	25.1	37.4	8.37	8.37
Paquete & Stutzle 2002; Tabu search using a lexicographic formulation.	-	-	40.5	10.8	16.5	13.2	-	158.1	9.3	-	27.8	38.9	21.78	
White et al. 2004; Tabu search with longer-term memory and relaxation	4.63	5.73	45.8	12.9	17.1	14.7	-	158	8.94	4.44	29.0	42.3	9.72	9.72
Yong & Petrovic 2005; Fuzzy similarity measure in heuristics selection.	3.93	4.5	33.71	10.83	13.82	10.35	-	151.52	7.92	3.14	25.39	36.53	7.99	7.99
Best	3.93	4.5	29.3	9.2	13.46	9.6	3.7	134.9	7.92	3.14	24.4	36.11	7.32	

Chapter 3

Hyper-heuristics

3.1 Introduction

The term hyper-heuristic (Burke et al., 2003a) denotes a method that operates at a higher level of abstraction than most implementation of heuristics or meta-heuristics. A hyper-heuristic can be thought of as a (meta-) heuristic that is able to intelligently choose a (meta-)heuristic to be applied at any given time. We refer to Burke et al. (2003a) for further motivation and discussion on the emergence of hyper-heuristics to address optimisation problems. This includes references to earlier work that can be categorised as hyper-heuristic approaches, although they do not use this term.

3.2 University Timetabling Applications

One example of solving a large-scale university examination timetable problem using a hyper-heuristic approach can be seen in Terashima-Marin et al. (1999b). The problem is solved in phases. Each phase uses a different set of heuristics and a switch condition determines when to move from one phase to the other. A genetic algorithm, using an indirect chromosome representation, was used to evolve the choice of heuristics and the switch condition.

Burke and Newall (2004) proposed an adaptive method in constructing initial solutions for the examination timetabling problem. An initial ordering heuristic produced an order of examinations to be scheduled. The ordering heuristic provides a good solution if the order is ideal, otherwise, it will adapt and improve the order, thus improving the initial solution. The results showed that the method could

substantially improve the solution quality over the original heuristic (flat ordering, largest degree and smallest degree).

Burke et al. (2003b) have used a tabu search hyper-heuristic (although different to the one proposed in this thesis) and have successfully applied it to course timetabling and nurse rostering problems. They used a ranking mechanism to dynamically rank each low-level heuristic. The heuristic with the highest rank will be applied in the next iteration and if the heuristic does not improve the solution, it will be placed in a tabu list. This tabu list is used to prevent non-performing heuristics from being chosen again in the near future. The hyper-heuristic in this thesis differs with respect to how the tabu list is used and how heuristics are chosen and applied. Further details are given in chapters 6 - 9.

Another recent application of hyper-heuristic on timetabling problems is presented by Burke et al. (2006a). The hyper-heuristic uses tabu search to search for different permutations of graph heuristics that solve both examination and course timetabling problems. A further use of a hyper-heuristic is in solving multi-objective (Burke et al., 2005b) space allocation and timetabling problems. The idea is to choose, at each iteration, a heuristic that is suitable for the optimisation of a given, individual objective.

3.3 Other Applications

Cowling et al. (2001) use a choice function in their hyper-heuristic to determine which low level heuristic should be called next. The choice function adaptively ranks the low-level heuristics by considering the recent improvement of each low-level heuristic, recent improvement of consecutive pairs of low-level heuristics and the number of CPU seconds elapsed since a particular heuristic was last called. The method was successfully tested on different applications: sales summit scheduling (Cowling et al. 2001), nurse scheduling (Kendall et al. 2002; Cowling et al. 2002c) and project presentation scheduling (Cowling et al. 2002d).

Cowling et al. (2002a) and Han et al. (2002) introduced a genetic algorithm based hyper-heuristic (Hyper-GA) to evolve a sequence of heuristics that are applied to a trainer scheduling problem. An adaptive length chromosome is used since it was shown to produce superior results than a fixed length chromosome. The length is adaptive because during crossover and mutation, poor performing heuristics are removed from the chromosome and efficient heuristics are injected into the chromosome. A guided crossover and mutation operator compare length and average length of chromosomes to decide when to remove or inject genes (Han and Kendall, 2003a). In addition, this approach is also tested with student project presentation scheduling. The results with the guided hyper-GA are superior when compared to previous implementation of hyper-GA and a hyper-heuristic with choice function. A tabu list is also used in the next implementation of hyper-GA (Han and Kendall, 2003b) with the purpose of making non performing heuristics tabu and not being called in the next few generations (depending on the list size).

Nareyek (2003) proposed a learning procedure in a search process that learns to select promising heuristics based on weight adaptation. His empirical study was carried out on two problems: Orc Quest (a computer game) and Logistics Domain (a classical benchmark in the action-planning community).

Hart et al. (1998) developed an evolving heuristically driven schedule builder for a real-life chicken catching and transportation problem. They use a genetic algorithm in their schedule builder to evolve strategies that can handle each day's scheduling and then apply those strategies to produce a schedule or solution. The problem is solved in two stages using two genetic algorithms (the first feeding into the second): 1) assigning set of tasks to squads and 2) scheduling start times for each squad. Each chromosome in the first genetic algorithm encodes a sequence of orders, a set of heuristics to split loads and a second set of heuristics to assign loads. These set of heuristics are a set of rules used by a local firm. The solution built by the first genetic algorithm is then fed into the second genetic algorithm. The second genetic algorithm creates the schedules by searching the space of all start times for

each squad. The quality of schedule produced is measured by a fitness function which penalises if a certain set of constraints are violated. This method was found to be more robust by including more domain knowledge in the chromosomes and evolving a choice of heuristics, which eventually produce more reliable schedules.

Gratch and Chien (1996) developed an adaptive problem solver that integrates a wide range of heuristics and a statistical approach to automatically learn effective scheduling heuristics for Deep Space Network communications. The strategies learned by the adaptive problem solver were shown to significantly outperform the best expert human derived solution.

Research by Ross et al. (2002) on hyper-heuristic includes the use of a learning classifier system, XCS, to learn a set of rules that associates characteristics of the current state of a problem with specific heuristics. The approach is tested on a bin packing problem. The algorithm begins with an initial problem with associated characteristics. It then chooses a heuristic that packs a bin which alters the characteristics of the problems that remained to be solved. At each step, appropriate rules, suitable for current state of problem characteristic, are chosen until all items are packed.

3.4 Conclusions

There are other papers published on methods that are similar to the ones used in hyper-heuristics (for examples of early work see Glover, 1977; Kitano 1990; Fisher and Thompson 1963). This thesis has reviewed the recent papers in the area but it would be interesting to carry out a complete survey and categorise all papers that exhibit hyper-heuristic behaviour (see Soubeiga, 2003). From what we can see from existing papers on hyper-heuristics, we believe that further research should be carried out in order to inject intelligence into the hyper-heuristic that does not depend on domain knowledge.

Chapter 4

A Survey and Case Study of a Practical Examination Timetabling Problem

This chapter presents results from two questionnaires (distributed to students and invigilators) and a case study on a real world examination timetabling problem. The objectives of the questionnaires are to gather views from students and invigilators on their current examination or invigilation timetable and what they hope will be taken into account in the next generation of examination timetabling software. Section 4.2 presents the results and analyses the two questionnaires. The actual questionnaires are shown in Appendix A. Section 4.3 concludes with a presentation of a very large examination timetabling problem faced by MARA University of Technology (UiTM) in Malaysia, a discussion on its examination timetabling processes and the possibility of introducing an automated timetabling system.

4.1 Introduction

Surveys on examination timetabling (see chapter 2 and chapter 3) show that over the years, a lot of computerisation has taken place in educational timetabling. Ongoing research on timetabling problems has been discussed in chapter 2 and chapter 3. Much of this research has attempted to solve the basic timetabling problem as well as including some constraints that are considered to be important. Have they succeeded in solving the actual problem in timetabling? Trying to find the answer to this question is the motivation in carrying out the following survey.

4.2 Questionnaires

Romero (1982) categorised groups of people with different requirements and who are directly affected by the timetabling problem. These groups of people are the university administrator, departments and students. We have included another group of people, that of lecturers, who are directly affected by the timetable and have specific preferences in their invigilation schedule and the examination timetable. Previous surveys (Comm and Mathaisel, 1988; Burke et al., 1996a; JTAP, 1998; discussed in chapter 2) received feedback from university administrators or departments concerning their timetabling problems. No survey has been carried out which include the views of students or lecturers. For example, what specific requirements would they like to be considered by an examination timetabling system and what do they feel about their current examination timetable. Students normally prefer to sit examinations that are spread out over the examination period and to have a gap between two examinations. Lecturers would prefer to have a minimum number of invigilation duties and to have more time to mark examination scripts (i.e. schedule examinations earlier to allow more time for marking).

In conducting the survey for this paper, two questionnaires were distributed to students and invigilators. The main objective of these questionnaires was to get the opinion of people who are affected by the schedules they receive every semester (or term). A lot of computerisation has taken place in producing examination timetables and we would like to see whether the timetables produced are acceptable to students and invigilators. Normally, administrative staff responsible for the examination procedure receive many complaints with regards to the timetable and the staff have to solve the problem manually.

The first stage of questionnaire distribution was done via the web and an updated version was then distributed on paper. The student web-base questionnaire was sent to two mailing groups of students (University of Nottingham - Computer Science students and American Top Universities Students Association from

Malaysia). The lecturer/invigilator web-base questionnaire was sent to two mailing groups of lecturers (Faculty of Information Technology and Quantitative Sciences, UiTM and Faculty of Computer Science and Information Technology, University of Nottingham). These questionnaires were updated with more questions and were then transferred to a paper base questionnaire and distributed to 46 students and 50 lecturers of MARA University of Technology. See both questionnaires in Appendix A. The results were combined and are presented in the following sections.

4.2.1 Student Questionnaire

There were 46 (paper-based) responses from UiTM, 49 (web-based) responses from the University of Nottingham and 35 (web-base) from several universities in the United States. The main objective of the questionnaire was to get an opinion from students on their current examination timetable and what specific requirements they hope to be included in the next generation of examination timetabling system. It was interesting to find that students from UiTM, University of Nottingham and United States agree on most of the issues raised in the questionnaire.

Figures 4.1 through 4.6 show some of the results from the student questionnaire. Some questions were not included in the web-base questionnaire and are therefore not shown in the results. The questionnaire results suggest that:

About 53% of UiTM students prefer to have their examinations spread evenly while 40% have no preference (figure 4.1). A high percentage of students had no preference about uniform distribution of examinations most probably because they have projects due and project presentations during the examination weeks and students might prefer to finish their examinations earlier because they were eager to start their holidays. However, a high percentage prefer to spread their examinations and this is also indicated in their comments with regard to the weaknesses of previous examination schedules

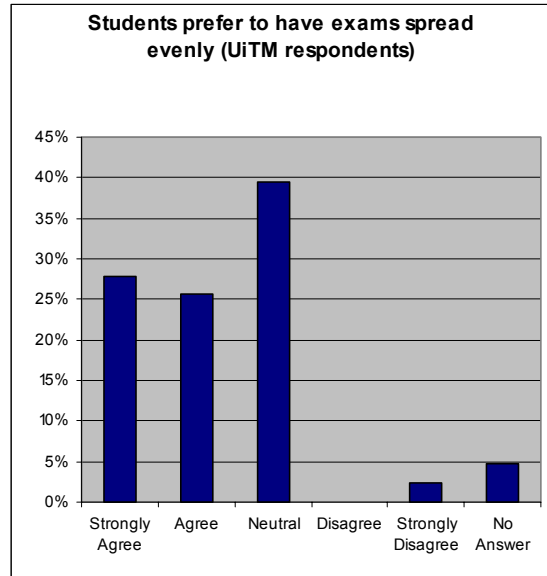


Figure 4.1: Preference to have examinations spread evenly (40% neutral, 53% agree)

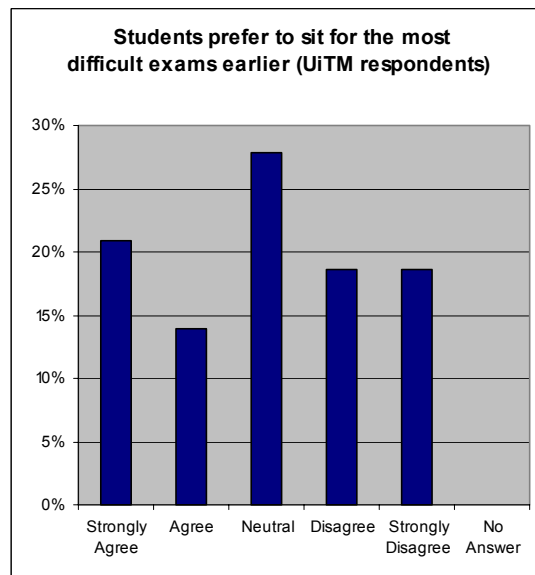


Figure 4.2: Students preference to sit for the most difficult examinations earlier

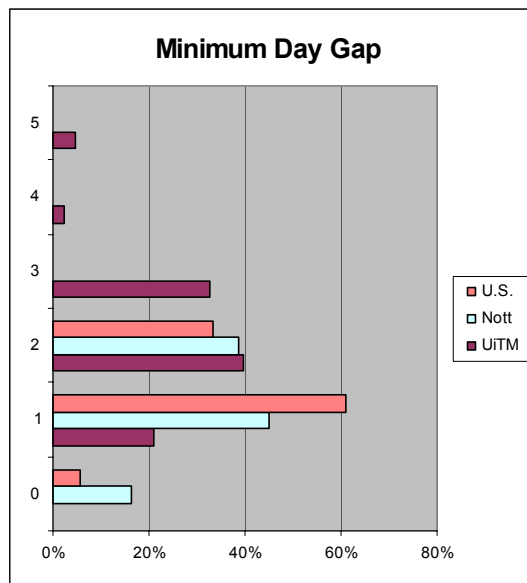


Figure 4.3: Students preference for minimum days gap between two examinations

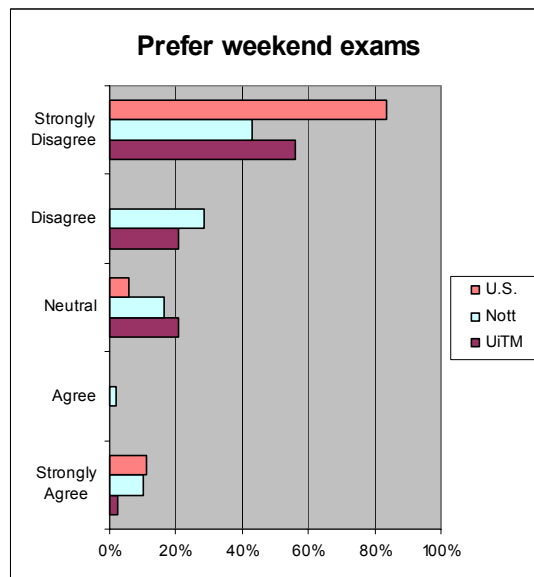


Figure 4.4: Percentage of students preferring weekend examinations

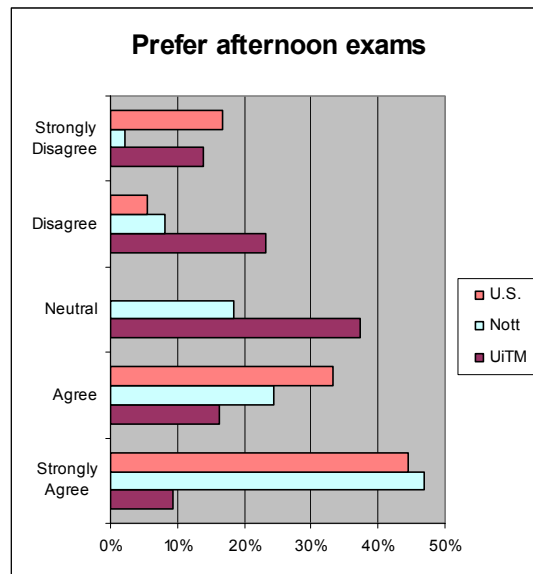


Figure 4.5: Students preference for afternoon examinations

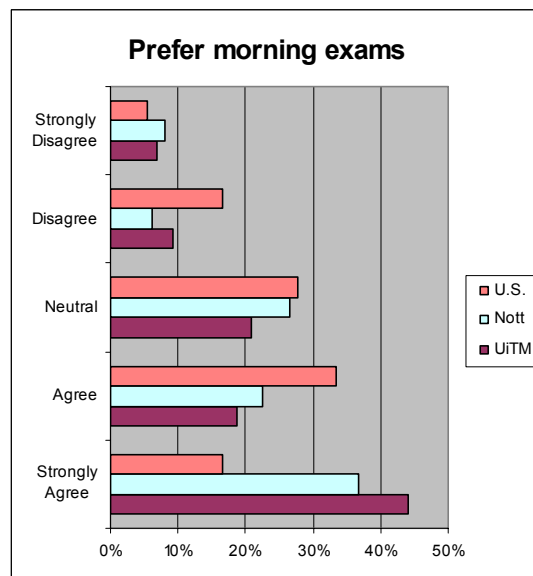


Figure 4.6: Students preference for morning examinations

- § Figure 4.2 shows that about an equivalent number of students agree, disagree and were neutral about sitting for the most difficult examination first. The examination scheduler may consider the constraint of uniformly distributing the difficult examinations rather than scheduling them in the beginning because students may require more time to prepare for the difficult examinations.

- § About 40% of UiTM students prefer a two-day gap and 33% prefer a three-day gap between examinations, while 45% of University of Nottingham students prefer a one-day gap and 39% prefer a two-day gap (figure 4.3). This preference is directly related to the number of examination days and number of examinations they have to sit. UiTM has an average of 6 examinations per student over about 20 days, while the University of Nottingham students have about 4.5 examinations per student over 14 days.

- § Figure 4.4 shows that most of the three groups of students do not like to have weekend examinations. UiTM has no alternative but to schedule examinations during weekends because not all UiTM campuses have the same weekend and they also have to accommodate part time and distance learning students (who prefer weekend examinations). This is one of the important criteria that we have included in solving UiTM examination timetabling problem where we try to minimise weekend slots (see chapter 8).

- § Figure 4.5 and figure 4.6 shows a mixed feeling amongst the three groups of students preferring to sit for a morning or afternoon examination. A comment from one University of Nottingham student was to delay the morning examination session by one hour.

When asked about what the students felt were the weaknesses of their previous examination timetable, 40% of UiTM students, 49% of University of Nottingham students and 72% of U.S. students complain that there is an inconsistent gap between two examinations. UiTM students and the University of Nottingham students face a similar problem of having two consecutive examinations and too wide a gap between other examinations while U.S. students had two to three examinations in a day. In addition, between 5% - 10%, of UiTM students indicate other weaknesses such as

frequent changes in the timetable, not updated accurately and in a timely manner, clashing timetables, weekend examinations, late timetable delivery and incomplete information such as examination venue.

28% of UiTM students, 35% of University of Nottingham students and 44% of U.S. students felt that the most important consideration while preparing the timetable is to have a uniform distribution of examinations over the examination period. 30% of UiTM students were more concerned about getting an examination timetable that included subject code, date, time and venue. The students were confused about their examination venue since several groups of students may be sitting for the examinations and each group may be split across several rooms. Friday schedule time slots were also different from other days because examinations begin earlier by half an hour to allow longer lunch break (Friday Muslim prayer time) but this time is not indicated on their examination slips. 19% of UiTM students and 2% of University of Nottingham students felt that the examination scheduler should consider examination difficulty by scheduling difficult examinations earlier or uniformly distributing them across examination periods. However, this may be unrealistic because of the subjective nature of "examination difficulty". One student suggested that all departments rank their subjects by difficulty level. One U.S. student complained because he/she had to sit for Physics, Chemistry and Economics in one day. 14% of UiTM students consider a clash free timetable as an important requirement. Even though a clash free timetable should be the ultimate consideration for any timetable, students did not indicate this requirement as part of their answer, most probably because they have assumed that an examination timetable should always be clash free.

4.2.2 Invigilator Questionnaire

The main objective of the invigilation questionnaire was to ascertain lecturer preferences for their invigilation schedule. The number of responses was 19 from the web-based questionnaire and 28 from the paper-based questionnaire. The results

from both the University of Nottingham and the MARA University of Technology (UiTM) were combined and presented in figure 4.7 through figure 3.10. UiTM do not hire staff for the purposes of invigilation and therefore invigilation duties are shared amongst the lecturers. The lecturers cannot invigilate during their own papers/modules examination session. The University of Nottingham do hire staff to invigilate and therefore the lecturers may be required to invigilate a minimum number of times or invigilate their own papers/modules only.

Figures 4.7 - 4.10 suggest that:

- § Since invigilation duties are considered as part of lecturers responsibility, the invigilators prefer to have between 2 - 3 invigilation duties (figures 4.7 and 4.8). If possible, 10% would prefer not to have any invigilation duties at all.
- § A high percentage (74%) of the invigilators prefer one or two day's gap between invigilation duties.
- § 45% of the invigilators liked the idea of weekend duty if it was counted as a double duty, while 55% would be unhappy to invigilate at weekend (figure 4.10). Since there was almost an equal split for this answer, it would be fairer to have the weekend duty assigned to everybody at least once or on alternate semesters.
- § 65% felt that lecturers with other responsibilities such as administrative work or research work should not be given the same number of invigilation duties (figure 4.10).
- § 35% felt that the previous invigilation schedule was not fair (figure 4.10). Some reasons given were too many invigilation duties, scheduled too close to each other, not enough invigilators, too many weekends, time clash (need to find own replacement) and need to travel to another campus.

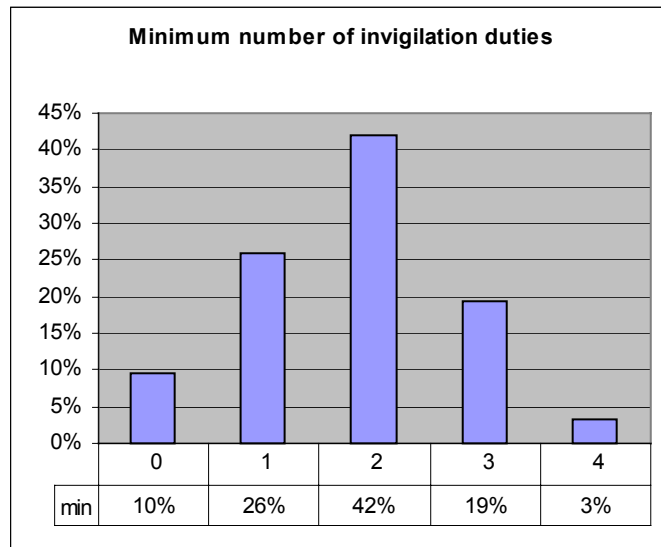


Figure 4.7: Percentage of lecturers' preference for minimum number of invigilation duties

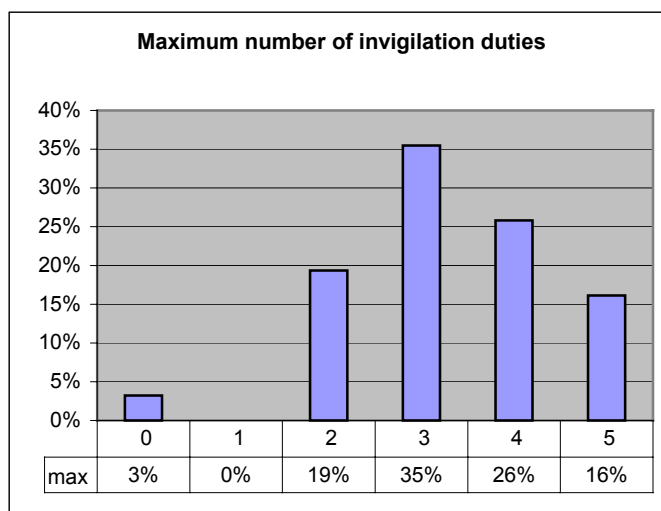


Figure 4.8: Percentage of lecturers' preference for maximum number of invigilation duties

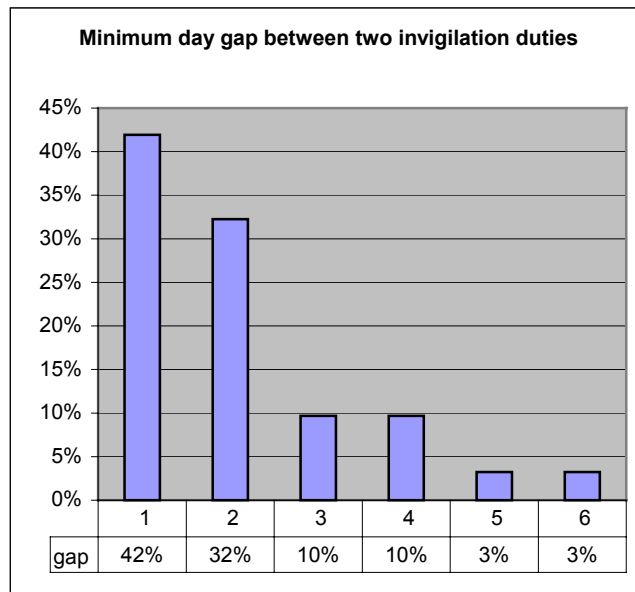


Figure 4.9: Lecturers preference for a minimum day gap between two invigilation duties

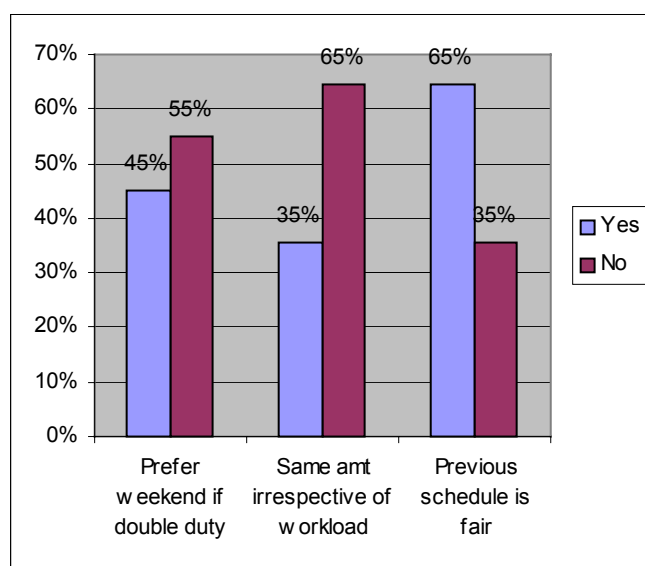


Figure 4.10: Lecturers response on weekend invigilation and total invigilation duties, and opinion on previous schedule

Some weaknesses of the previous schedule as reported by UiTM respondents are:

- § Too many chief invigilator assignments (2 out of 3).
- § Time clash because a lecturer receives two different invigilation schedules prepared by two different faculties. For example, a lecturer may be lecturing at two campuses and both campuses will schedule invigilation duties for him or her.
- § There are not enough gaps between invigilation duties.
- § The invigilation duty is too soon after the lecturers' own paper. This does not allow the lecturers enough time to do their marking and submit their grades within the required time (i.e. one week).
- § There was an insufficient number of invigilators because invigilators who did not turn up for duty were not replaced.

Another weakness that was reported by both UiTM and the University of Nottingham respondents was last minute changes to the invigilation schedule. Due to communication problems lecturers at UiTM were sometimes unaware of their invigilation duties especially for the first few days.

The following are the important requirements that the lecturers felt should be considered in their invigilation schedule:

- § The invigilation schedule should ensure that no lecturer has two examinations scheduled simultaneously.
- § Morning or afternoon sessions do not matter as long as there is a gap of at least one day between invigilation duties.
- § They would prefer the scheduler to consider their marking duties and not to assign them invigilation duties during that period.

- § The number of duties should be fair taking into account the number of times being appointed as head invigilator, the number of weekend duties and other administrative or research responsibilities.
- § An examination hall should not have different papers that require different sets of instructions and having different start and end times.
- § Schedules should be published as early as possible to allow enough time for the lecturers to be aware of their invigilation duty and to request alternative slots where clashes occur (which should never happen anyway) or lecturers are committed to other prior appointments they have made. It should also include the integration of scheduling activities between faculties and departments to avoid clashes and over assignment of invigilation duties.
- § Data held on the number of students and available room space should be carefully checked and collected early to avoid last minute changes or problems during the examination itself.
- § Allow the hiring of temporary invigilators so that lecturers will have a minimum number of invigilation duties.

4.3 Examination Timetabling at the MARA University of Technology (UiTM)

The MARA University of Technology (UiTM) is the largest university in Malaysia with a total number of students approaching 100,000. The university has 13 branch campuses, one in every state in Malaysia. Student applications for admission to the university are handled at the main campus in Shah Alam, Selangor and successful applicants are admitted to the branch campus closest to their home address. The students are registered as either full time, part time or distance-learning students. The students take courses from the 144 programs offered by 18 faculties. The branch campuses do not offer the same number of programs as the main campus but most of them offer major programs such as engineering, computer science and business. Some of the programs at UiTM are franchised and managed by private

colleges. All the students take the same examination papers and therefore have the same examination timetable.

The Center for Integrated Information System (CIIS), UiTM has the responsibility for planning and managing the overall Information Technology strategy in UiTM and to fulfil administrative and academic needs. One of its information system modules, Integrated Student Information Systems (iSIS), was designed and developed as a collaborative project. This system offers 6 main modules encompassing the complete Student Life Cycle process, from Intake to Convocation and Alumni. The 4 principal modules comprise the Student Intake System, Academic Affairs Systems, Student Affairs System and Student Accounting System. Thirteen personnel headed by a senior information system officer provide support for the system and two information system officers are assigned to the examination unit.

4.3.1 Examination Timetable

The Examination Unit at the Office of Academic Affairs liaises with the Examination Unit from CIIS to manage the preparation of the examination timetable. The examination timetabling processes are as follows:

- § Update examination data, including previous semester course codes, equivalent course codes, courses that have several papers and course codes with no final examination.
- § Prepare a draft timetable and distribute it to faculties, centres and branch campuses for corrections or amendments. The first draft is produced by using a program that was designed in house

- § Update the timetable based on feedback received. At this time all student course enrolments are completed and must be entered into the computer system database by each faculty or branch campus via the university computer network. The amendments to the timetable may produce clashes and a clash list is sent to the faculties/centre/branch campuses for further corrections.
- § Update timetable and distribute the final timetable. The timetable is also published online at the university's website (<http://www.uitm.edu.my>). The timetable only shows the date and time of each examination and does not indicate the venue.

Table 4.1: Characteristics of UiTM dataset compared to benchmark datasets

<i>Institution Code</i>		<i>No. of Examinations</i>	<i>No. of Students</i>	<i>No. of Student Exams</i>
Car-f92	Carleton University, Ottawa	543	18,419	55,522
Car-s91	Carleton University, Ottawa	682	16,925	56,877
Ear-f83	Earl Haig Collegiate Institute, Toronto	190	1,125	8,109
Kfu-s93	King Fahd Univ. of Petroleum & Mineral, Dharan	461	5,349	25,113
Lse-f91	London School of Economics	381	2,726	10,918
Pur-s93	Purdue University, Indiana	2,419	30,032	120,681
Rye-f92	Ryerson University, Toronto	486	11,483	45,051
Sta-f83	St. Andrews Junior High School	139	611	5,751
Tre-s92	Trent University, Peterborough, Toronto	261	4,360	14,901
Uta-s92	Faculty of Arts and Sciences, Univ. of Toronto	622	21,267	58,979
Ute-s92	Faculty of Engineering, University of Toronto	184	2,750	11,793
Yor-f83	York Mills Collegiate Institute, Toronto	181	941	6,034
UiTM-03	The MARA University of Technology	2,063	84,675	357,761

The total number of students enrolled in Semester 2, 2002/2003 was 84,675, course enrolment was 357,761 and the total number of courses to be scheduled was

2,063. This data was obtained from CIIS and is classified by faculty and branch campuses. Table 4.1 shows the problem characteristics compared to other benchmark datasets provided by Carter et al., (1996). More detailed analysis of this new dataset and the other benchmark datasets is given in chapter 5.

Wan Ya and Baharudin (2001) from UiTM used an examination scheduling program (developed in house about 30 years ago) to produce a first draft of the examination timetable. The timetable went through a manual update process by scheduling new courses and removing old ones. Apart from the constraints that are common for examination timetabling (Carter 1986; Carter and Laporte 1996), UiTM has to consider the following additional constraints:

- § If an examination falls on a state public holiday and there are students from that state sitting for the examination then the examination must be moved to another slot. Malaysia has a number of public holidays that are not shared between states.
- § Examinations have to be scheduled on weekends because different states have different weekend days (half-day on Thursday and full-day Friday; or half-day on Saturday and full-day Sunday). The scheduler tries to schedule courses with less students or courses with a large number of part time students during the weekend. Avoiding weekend slots will impose more constraints on the problem since it also needs to consider another variable i.e. campus.
- § Data on available space for examinations were not considered when moving examinations and therefore every faculty, centre and branch campus needs to give prompt feedback on space availability for examinations already scheduled. The main campus has to rent halls for their examinations and it would be beneficial to minimise the number of days when halls have to be rented.

The time taken to prepare the timetable may take up to 6 weeks involving personnel from the examination unit at the office of academic affairs, the examination unit from CIIS, the dean's office, the head of centre's office, and the provost of branch campuses office. A lot of data and information must transfer back

and forth before the final examination timetable is ready for distribution.

4.3.2 Invigilator and Room Assignment

Once the examination timetable is in its final draft, it is sent to all faculties and branch campuses for rooms and invigilator assignment. Faculties and branch campuses normally establish an examination committee that manages the overall preparation of the examination timetable (for room and invigilator assignment) and act as examination secretariat (collecting/distributing question booklets or answer scripts etc.) during the examination period.

The first task of the examination committee is to assign examinations to rooms. These assignments are normally done manually with some aid from a computerised spreadsheet. Some of the constraints for room assignment are as follows:

- § All examinations must be assigned to a room and the number of students sitting the examination must not exceed the maximum number of seats in the room. However, not many big examination halls are available to accommodate all students, so the examinations are assigned to several rooms.
- § To fully utilise available space and to reduce the number of invigilators, more than one examination may be assigned to the same room but the examinations should begin and finish at the same time.
- § Examinations that require special resources (such as science apparatus or audio visual aid) can only be assigned to certain rooms.

With regards to invigilation, one invigilator is scheduled as the chief invigilator and two invigilators are assigned as a standby for a particular slot. Data collection is quite difficult when the invigilators are also lecturers serving other faculties. More than one faculty may require the service of the same lecturer and when faculties do their own invigilator scheduling, some clashes may occur across faculties. The manual process of assigning invigilators and rooms involve many man-hours for all

faculties and branch campuses. Thus, it is highly desirable for UiTM to have a computer-based system that is deployed over all campuses.

For invigilator assignment, the examination committee must consider the following constraints:

- § Invigilators are not assigned to more than one room in a given time slot.
- § Invigilators are assigned fairly i.e. equivalent duties for all invigilators.
- § Invigilators do not invigilate their own papers.
- § Invigilators are not assigned to invigilate papers within a week of their own paper being sat. Lecturers need to submit their final examination grades one week after the examination time and the lecturers need to have time to mark and should not have additional invigilation duties in this period.
- § Project presentations and evaluation are also scheduled during the examination period and lecturers should not be scheduled during the project evaluation period.

4.4 Conclusions

From the survey we have seen that students and invigilators are not satisfied with their current timetable and they would like to see more work being done on improving the time between two examinations and invigilation duties. Both students and lecturers find it very inconvenient to have exams during the weekend. Automating the process of timetabling can avoid clashes but it may not take into account other constraints such as longer gaps between examinations and invigilation duties, and avoiding weekend slots. Lecturers who have to invigilate find it very inconvenient and would prefer to have hired personnel to do the invigilation.

The examination timetabling problem at UiTM was presented and due to the number of branch campuses, the number of students and the number of faculties, the

Center of Integration Information System was established to help in the integration of data. Personnel need to be trained and located at every branch campus to deal with the collection of data and manage the rooms and invigilator assignments. Since the assignment of examinations and invigilators to rooms is done manually at branch and faculty level, the examination timetable must be produced at least two months before the examination period so that students can get a full schedule (with date, time and venue) at least three weeks before examinations begin.

An automated scheduling process which integrated all branch campuses and faculties would be an improvement to the examination timetabling process at UiTM. Currently, an examination module is already part of the integrated student information system deployed over a wide area network based on COINS (Corporate Information Superhighway), a nationwide, high-speed broadband multimedia communication network. In this thesis, we will solve the main examination timetabling problem of assigning examinations to time slot subject to the hard constraints with the objective of minimising proximity and weekend slots. This module can be part of the examination module in the integrated student information system.

In the next chapter, we will present a mathematical formulation of the examination timetabling problem. We will also extend the model to suit the MARA University of Technology examination timetabling problem.

Chapter 5

Examination Timetabling: Models and Data Specification

This chapter describes the specification and formulation of the examination timetabling problem. Section 5.2 models the examination timetabling problem as a graph and section 5.3 presents a mathematical model of the problem. Section 5.4 presents a mathematical model of the MARA University of Technology (UiTM) examination timetabling problem. Section 5.5 shows some problem characteristics of the benchmark datasets and also the UiTM dataset. The conflict matrix graph definition and the constructive heuristic algorithm used in the initial solution construction phase are presented in Section 5.6.

5.1 Introduction

This chapter describes two modelling techniques for solving examination timetabling problems. The models presented in section 5.2 are simplified by considering only the important constraints whilst ignoring other less critical constraints. The important constraints are those that are common in many universities such as avoiding first order conflict and scheduling all examinations. There are other variants of the examination timetabling problem as presented by Schaerf (1999). This thesis presents two variants of an examination timetabling problem that can be solved using a hyper-heuristic framework. This chapter also provides the necessary background for the pre-processing phase and construction phase of a hyper-heuristic framework and other state-of-the-art meta-heuristic approaches. Appendix B is closely related to this chapter. It presents a formal

format specification for data files used in solving examination timetabling problems.

5.2 Graph Model

Burke et al. (2004b) relate how graph theory has played an important role for research on timetabling problems. The graph colouring problem has the objective of finding the minimum number of colours that can be applied on the nodes or vertices of a graph such that no two adjacent vertices have the same colour. Welsh and Powell (1967) discussed the close relationship between graph colouring and timetabling. The graph colouring problem is considered as NP-complete (Karp 1972) because there is no efficient polynomial-time algorithm that can find the minimum number of colours (chromatic number) for the graph.

The simplest form of a timetabling problem that can be modelled as a graph is to schedule all examinations into timeslot such that no two conflicting examinations (first order conflict) are scheduled in the same timeslots. Consider an examination timetabling problem with n examinations, e_1 to e_n and we need to schedule these examinations into a limited number of timeslots. This problem can be represented as a graph (see figure 5.1), consisting of a set of n vertices where the vertices correspond to examinations (e_1 to e_n). An edge exists between vertices i and j , if there is a conflict between examinations i and j (i.e., examination i and examination j cannot be scheduled in the same timeslot). The edges can be assigned a weight indicating the actual number of students involved in the conflict. The number next to the vertex is the student enrolment for the corresponding examination. The total number of edges from a vertex i is known as the vertex degree indicating the total number of examinations in conflict with examination i . Clearly, this is a graph colouring problem where we need to colour each vertex of the graph such that no two vertices connected by an edge have the same colour. The unique colour corresponds to a timeslot and therefore two adjacent vertices or examinations will not be assigned to the same colour or timeslot.

The next questions is how do we colour the vertices and is there a possible solution? Clearly, this is a decision problem that can be solved by continuously selecting a vertex and assigning it a new colour such that no two adjacent vertices have the same colour. A solution exists if we can solve the problem with c colours and c is less than or equal to the number of timeslots. There are several graph colouring heuristics that are adapted to solve examination timetabling problems (see section 2.3.2) and are still being applied in more modern approaches of meta-heuristics.

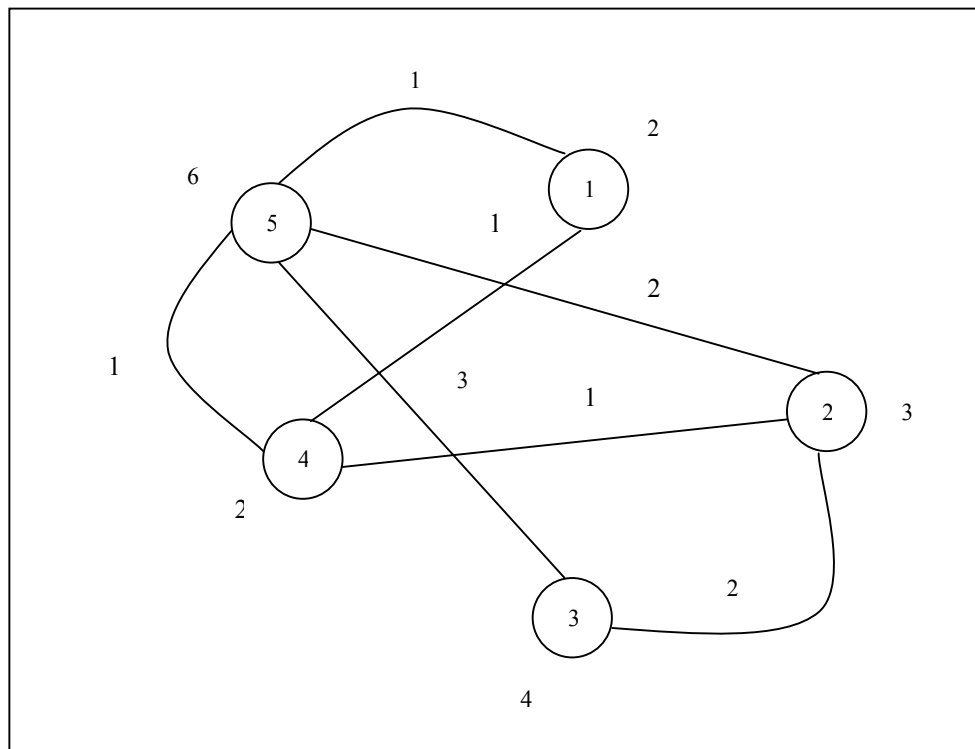


Figure 5.1: Graph model of a very simple timetabling problem

5.3 Mathematical Model

The combinatorial optimisation problem of an examination timetabling problem can be represented using a mathematical model. A mathematical model should have the following (Williams, 1999):

- § Choose decision variables that correspond to controllable input. Giving a value to each variable should give us a solution that can be implemented without ambiguity. In examination timetabling we wish to find the most appropriate slot for an examination. Therefore, the variables correspond to all the examinations and the value assigned is chosen from the slots available.
- § Find an objective function for each decision variable such that we can evaluate different decision outcomes. The value of an objective function determines the quality of a solution. If it is a minimisation problem, a lower value is better than a higher value and vice versa for a maximisation problem.
- § Determine constraints that represent restrictions or limitations on the values of decision variables.

The examination timetabling problem is to assign examinations to slots subject to some hard constraints that must be satisfied with the objective of minimising soft constraints violation. The following examination timetabling formulation is adapted from a thesis by Weare, 1995:

- § m = number of examinations
- § n = number of slots
- § **E**: A set of m examinations $E_1, E_2, \dots, E_m$
- § **S**: A set of n slots $S_1, S_2, \dots, S_n$
- § A final examination timetable $\mathbf{T}_{mn}$ such that: $T_{ik} = 1$ if examination i is scheduled in slot k , 0 otherwise.
- § A conflict matrix $\mathbf{C}_{mm}$ such that: C_{ij} = total number of students sitting for both examinations i and j .
- § P_{ik} is a penalty given if examination i is scheduled in slot k .

Hard constraints are those that cannot be violated. A timetable that violates a hard constraint will render it infeasible. Such infeasibilities may be unavoidable in

certain cases and universities have to take drastic measures to resolve the problem.

Hard constraints that must be satisfied are:

1. Examination constraint: All examinations must be scheduled and each examination ($E_1, E_2, \dots, E_m$) must be scheduled only once.

$$\sum_{k=1}^n T_{ik} = 1 \quad \text{For all } i = 1, \dots, m$$

2. Student Constraint (student conflict): No student should sit for more than one examination in the same slot. If examination i and examination j are scheduled in slot k , the number of students sitting for both examination i and j (C_{ij}) must be equal to zero, and this should be true for all examinations already allocated.

$$\sum_{i=1}^m \sum_{j=1}^m \sum_{k=1}^n T_{ik} T_{jk} C_{ij} = 0$$

We determine the quality of an examination timetable solution based on the penalty given if certain soft constraints are violated. The soft constraint that we would like to consider is the proximity constraint and a proximity cost is given when the proximity constraint is violated. A weighted proximity cost x_s is given whenever a student has to sit for two examinations scheduled s periods apart: these weights are $x_1 = 16$, $x_2 = 8$, $x_3 = 4$, $x_4 = 2$ and $x_5 = 1$.

P_{ik} , the total proximity cost if examination i is scheduled in slot k , is as follows:

$$P_{ik} = \sum_{j=1}^m x_{|k-l|} C_{ij}$$

where, j (an examination in conflict with examination i), is scheduled in slot l

Finally, the objective is to minimise the total proximity cost:

$$\sum_{i=1}^m \sum_{k=1}^n T_{ik} P_{ik}$$

2

If the timetable has unscheduled examination(s), a penalty of 5,000 for each unscheduled examination is added to the total cost. The proximity cost (and weights) above was first used in Carter et al. (1996) to evaluate solutions for a set of benchmark datasets. Therefore the same method of evaluating solution quality is used in this thesis so that we can compare our results with other results published in the literature.

5.4 UiTM Examination Timetabling Problem Formulation

The examination timetabling problem at the MARA University of Technology (UiTM) was first introduced in chapter 4. UiTM is a very large university with a total number of students approaching 100,000 and 13 branch campuses.

5.4.1 Mathematical Model

The examination timetabling problem at UiTM can be represented as follows:

- § m = number of examinations
 - § n = number of slots
 - § u = number of campus
 - § **E**: A set of m examinations $E_1, E_2, \dots, E_m$
 - § **S**: A set of n slots $S_1, S_2, \dots, S_n$
 - § **A**: A set of u campus $A_1, A_2, \dots, A_u$
-

- § A final examination timetable $\mathbf{T}_{mn}$ such that: $T_{ik} = 1$ if examination i is scheduled in slot k , 0 otherwise.
- § $\text{CampusType} = \{A, B\}$ where campus type A has a half-day Saturday and a full-day Sunday weekend and campus type B has a half-day Thursday and a full-day Friday weekend.
- § A vector $\mathbf{W}_n$ such that element $W_i = 1$ if slot i is a weekend, 0 otherwise.
- § A conflict matrix $\mathbf{C}_{mm}$ such that: C_{ij} = total number of students sitting for both examinations i and j categorised by campus type.
- § A co-schedule matrix $\mathbf{R}_{mm}$ such that: $R_{ik} = 1$ if examination i and examination k must be scheduled in the same time slot, 0 otherwise.

The examination timetabling problem is to assign examinations to n number of slots subject to various constraints, so as to minimise various costs. The number of slots is already fixed and the examination scheduler must schedule examinations into, at most n slots. There are 2,650 examinations with some examinations being held concurrently. Even though each examination may have a different duration (120, 150 or 180 minutes), we will treat each examination as occupying the whole slot of 180 minutes. The total number of examination days is 20 and each day will have 2 slots (morning and afternoon), i.e. we will have $n = 40$ slots into which to assign examinations. Each campus is categorised as $\{A, B\}$ to indicate which slots are weekend slots and must be avoided in assigning examinations taken by students in that particular campus. Campuses in category A has weekend: half-day on Saturday and full-day on Sunday while campuses in category B has weekend: half-day on Thursday and full-day on Friday. In this study, the examination dates were from 20th April 2003 to 10th May 2003 with 1st May as a national public holiday. For the entire country, we cannot schedule any exams on a national public holiday. For this dataset, we are not concerned with other public holidays since none occur on a different day for different states. On other occasions there might be a situation where a campus has a public holiday and others which do not.

5.4.2 Constraints

In addition to the hard constraints defined in section 5.3, UiTM has the following constraint :-

Co-schedule: Examinations that must be scheduled together must be assigned the same time slot.

For all examinations i :

$$\sum_{j=1}^m \sum_{k=1}^n T_{ik} T_{jk} R_{ij} = \sum_{j=1}^m R_{ij}$$

All examinations that must be scheduled with examination i should be assigned to the same slot k .

5.4.3 Costs

Other additional soft constraints that are specific to university requirements can be added to this problem. The cost of an examination timetable solution is based on the penalty given if certain soft constraints are violated. In addition to the proximity cost (see section 5.3), a weekend cost is also included. The current examination timetable scheduler schedules examinations during the weekend. We can try to produce a better quality timetable by penalising examinations that are scheduled during the weekend. A penalty cost of 16 is given whenever a student has to sit for a weekend examination. W_{ik} is the weekend cost if examination i is scheduled in slot k .

The total weekend cost is as follows:

$$\sum_{i=1}^m \sum_{k=1}^n T_{ik} W_{ik}$$

Finally, the objective is to minimise the proximity cost plus the weekend cost:

$$\frac{\sum_{i=1}^m \sum_{k=1}^n T_{ik} P_{ik} + \sum_{i=1}^m \sum_{k=1}^n T_{ik} W_{ik}}{2}$$

If the timetable has unscheduled examination(s), a penalty of 10,000 for each unscheduled examination is added to the total cost.

5.5 Benchmark Datasets and UiTM Dataset

Carter et al. (1996) used a set of benchmark examination timetabling instances from real examination problem datasets from various universities all over the world. They applied a variety of constructive algorithms, with backtracking, based on graph colouring heuristics and they solve the problem with and without room capacity constraint (capacitated and un-capacitated problem). The solution quality is measured by an objective function based on proximity cost. A lower proximity cost indicates that, on average, a student will have his/her examinations spread over the length of his/her examination period and, therefore, will have more time to concentrate on each examination.

Burke et al. (1996) introduced another dataset from the University of Nottingham that also includes room requirements and capacities. They used a memetic algorithm and a weighted objective function for adjacent, overnight and unscheduled examinations. Burke et al. (1998) applied some sequential heuristics and various ordering techniques to allocate examinations to slots while not violating the clash and capacity constraints.

Merlot et al (2003) introduced two new datasets from the University of

Melbourne that also includes two additional hard constraints: examination availability (examinations pre-assigned to specific slots) and large examinations (large examinations scheduled in the first n slots). They used a hybrid algorithm (constraint method, simulated annealing and hill climbing) to find a feasible schedule and found that they have to relax the constraints (by adding additional slots) so as to produce a clash free timetable.

All the datasets above use similar quality measure and are made public via these sites. (<ftp://ftp.mie.utoronto.ca/pub/carter/testprob>; <ftp://ftp.cs.nott.ac.uk/ttp/data> and <http://www.or.ms.unimelb.edu.au/timetabling>)

The hyper-heuristic framework will be tested with datasets taken from established datasets made public and used by a number of other examination timetabling researchers. We will use the datasets provided by Carter et al. (1996). Table 5.1 shows the characteristics of each dataset. The last dataset i.e. uitm-03 is an additional dataset introduced in this thesis.

Each benchmark dataset is stored in two files; one file contains a list of courses and their enrolment, and the other a list of student and their course selections. Both data files are used to construct a conflict graph and the largest degree algorithm (Carter et al., 1996) is used to construct an initial solution for each of the benchmark dataset.

The density of the conflict matrix in Table 5.1 is calculated as the average number of other examinations that each examination conflicts with, divided by the total number of examinations. For example, a conflict matrix density of 0.5 or 50% indicates that, on average, each examination conflicts with half of the other examinations.

No	File	No. of Slots	No. of Exams	No. of Students	No. of Enrol.	Density	Exams per period	Enrol. per student	Enrol. per exam (Avg)	Enrol per exam (max)	Student conflict (max)	Max degree	Max degree per exam
1	car-f92	32	543	18,419	55,522	13.6%	17.0	3.0	102.3	1,566	4,730	381	56.4%
2	car-s91	35	682	16,925	56,877	12.8%	19.5	3.4	83.4	1,385	4,718	472	56.4%
3	ear-f83	24	190	1,125	8,109	26.3%	7.9	7.2	42.7	232	1,665	134	44.0%
4	hec-s92	18	81	2,823	10,632	40.7%	4.5	3.8	131.3	634	2,315	62	35.0%
5	kfu-s93	20	461	5,349	25,113	5.4%	23.1	4.7	54.5	1,280	5,089	247	48.0%
6	lse-f91	18	381	2,726	10,918	6.0%	21.2	4.0	28.7	382	1,229	134	28.9%
7	pur-s93	42	2,419	30,032	120,681	2.9%	57.6	4.0	49.9	1,961	6,784	857	32.5%
8	sta-f83	13	139	611	5,751	13.7%	10.7	9.4	41.4	237	2,090	61	29.6%
9	tre-s92	23	261	4,360	14,901	17.6%	11.3	3.4	57.1	407	1,267	145	37.6%
10	uta-s92	35	622	21,267	58,979	12.4%	17.8	2.8	94.8	1,314	4,382	303	36.2%
11	ute-s92	10	184	2,750	11,793	8.2%	18.4	4.3	64.1	482	1,847	58	23.1%
12	yor-f83	21	181	941	6,034	28.7%	8.6	6.4	33.3	175	779	117	35.9%
13	uitm-03	40	2,063	84,675	357,761	1.1%	51.6	4.2	173.4	9,979	34,219	677	31.7%

Table 5.1: Characteristics of real problems

The number of slots are obtained from results reported by Carter et al. (1996). They used five different graph colouring heuristics to determine the minimum number of slots (sessions) required to produce a feasible solution subject to a no clash constraint.

The table also shows other features that are presented in Burke et al. (2004c) in their discussion on features and combination of features that contribute in measuring the similarity between two timetable problems. The number of examinations is a good indicator of problem difficulty. The conflict matrix density is not sufficient to indicate whether a problem is highly constrained, without considering variations in degree of examinations from the average and the cliques found within the conflict matrix (Burke et al., 2004c). If a large number of examinations have large degree (i.e. many examinations conflicting with each other), an algorithm will have to spend a long time to determine whether a move is feasible.

The last row in Table 5.1 is a description of the MARA University of Technology (UiTM) dataset. The number of slots is the actual length of the UiTM real timetabling problem whereas the number of slots for benchmark datasets is the minimum number of slots that can be used to schedule all examinations (from Carter et al., 1996). The total number of students enrolled in Semester 2, 2002/2003 was 84,675, course enrolment was 357,761 and the total number of courses to be scheduled was 2,063. The uitm-03 dataset is produced after further pre-processing of data supplied by the Central Integrated Information System of UiTM. The data was in the form of, a list of examinations that must be scheduled, a list of examinations that must be scheduled at the same time (concurrent), a list of students and their course selection (categorised by campuses) and an examination timetable that was used in the May 2003 semester. Details of the files needed to process the UiTM dataset are listed in Appendix B.

The conflict matrix density for uitm-03 is low compared to other datasets

because courses are offered by many faculties and therefore, on average, the courses are not clustered together. The low conflict matrix density does not imply that the problem is easier since other factors, such as the number of examinations, the number of students, the size of the maximum clique and the variations between the number of examinations with large degree and the conflict matrix density also play an important role. Recall that the conflict matrix density is an average number of other examinations that each examination conflicts with, divided by the total number of examinations. The last column in Table 5.1 shows the largest number of conflicts divided by the total number of examinations. This shows that, for uitm-03, the examination with the largest number of conflicts, conflicts with 31.7% of other examinations. Another factor that also contributes to the difficulty of this problem is the additional constraint imposed by scheduling examinations during a weekend. This constraint, if treated as a hard constraint, will make it very difficult to find a feasible solution because the number of valid slots is dramatically reduced. Therefore, the constraint is relaxed and we try to minimise the weekend slot assignments.

5.6 Conflict Matrix and Greedy Constructive Heuristic

A conflict matrix is a table, of size m by m (m is the number of examinations), such that an element in C_{mm} , denoted by C_{ij} , is the number of students sitting for both examination i and examination j . This is a static data where the student data is pre-processed and the conflict matrix is stored in a file and becomes an input to a timetable solver.

Example:

Given a set of $M = 12$ examinations and a list of 10 students

Figure 5.2 shows the data in file *fnm.stl* that contain a list of students and a list of examinations they need to sit during the examination period. The first row is the

field name for the datafile (student number, campus code, list of examinations), and each row that follows contains data for one student.

<StudentID><CampusCode> <E ₁ > <E ₂ > <E ₃ > <E _n > <eoln>						
20110284	P	KEE212	KEH215	KEL234		
20110291	P	KEE222	KEH215	KEL234	KJM203	
20110357	P	BEL250	KJM203	KEU110		
20110364	P	KJE211	KEU110			
20110371	P	KEE212	KEH215	KEU230	KJM203	
20110389	P	KEK213	KEE212	KEU230	KEL114	KEU210
20110405	P	KJE211	KEE212	KJM203	KEL114	KEU210
20110412	P	KEE222	KEH215	KEL234	KJM203	
20110437	P	KEE212	KEH215	KEU230	KJM203	
20110444	P	KEE212	KEL234	KEU230	KJM203	KEU210

Figure 5.2: Student datafile (<fnn.stl>) format specification

Figure 5.3 shows the conflict matrix for the above set of students and 12 examinations. The row number and column number are the examination identifier for the examination code. For example, the value 3 in cell C_{74} indicates that 3 students have to sit for both examinations KEL234 and KEH215. It also implies that examination KEL234 and KEH215 has a clash constraint and scheduling these examinations in the same slot violates a hard constraint (clash) in an examination timetabling problem.

The conflict matrix is actually a symmetrical matrix where each conflict is stored twice. The values in diagonal cells are zero because it is not meaningful to store the number of students when i equals j . The zero value cells indicate that there are no students sitting for both examinations in that row and column number. Since there will be many cells with zero values (a sparse matrix), we store the conflict matrix in the form of an array of examination objects. The element in the array is an

examination object and each examination object contains a linked list of examinations that are in conflict.

	i, j	1	2	3	4	5	6	7	8	9	10	11	12
BEL250	1	0	0	0	0	0	0	0	1	0	0	0	1
KEE212	2	0	0	0	3	1	2	2	0	3	4	1	4
KEE222	3	0	0	0	2	0	0	2	0	0	0	0	2
KEH215	4	0	3	2	0	0	0	3	0	0	2	0	4
KEK213	5	0	1	0	0	0	1	0	0	1	1	0	0
KEL114	6	0	2	0	0	1	0	0	0	2	1	1	1
KEL234	7	0	2	2	3	0	0	0	0	1	1	0	3
KEU110	8	1	0	0	0	0	0	0	0	0	0	1	1
KEU210	9	0	3	0	0	1	2	1	0	0	2	1	2
KEU230	10	0	4	0	2	1	1	1	0	2	0	0	3
KJE211	11	0	1	0	0	0	1	0	1	1	0	0	1
KJM203	12	1	4	2	4	0	1	3	1	2	3	1	0

Figure 5.3: Conflict Matrix

The conflict matrix is stored in a file to be read in by a search algorithm.

File Format Specification:

$\langle eid \rangle$ $\langle snu \rangle$ $\langle cnu \rangle$ $\langle cenu \rangle$ $\langle cne[0] \rangle$ $\langle cns[0] \rangle$ $\langle cne[1] \rangle$ $\langle cns[1] \rangle$

where:

$\langle eid \rangle$ is examination id

$\langle snu \rangle$ is total student enrolment for examination $\langle eid \rangle$

$\langle cnu \rangle$ is total number of students in conflict with examination $\langle eid \rangle$

$\langle cenu \rangle$ is number of examinations in conflict with $\langle eid \rangle$

$\langle cne[i] \rangle$ is examination id of the i^{th} examination in conflict with examination $\langle eid \rangle$

$\langle cns[i] \rangle$ is number of students for examination $cne[i]$ in conflict with examination $\langle eid \rangle$

12	7	22	10	1	1	2	4	3	2	4	4	6	1	7	3	8	1	9	2	10	3	11	1
2	6	20	8	4	3	5	1	6	2	7	2	9	3	10	4	11	1	12	4				
9	3	12	7	2	3	5	1	6	2	7	1	10	2	11	1	12	2						
10	4	14	7	2	4	4	2	5	1	6	1	7	1	9	2	12	3						
6	2	8	6	2	2	5	1	9	2	10	1	11	1	12	1								
7	4	12	6	2	2	3	2	4	3	9	1	10	1	12	3								
4	5	14	5	2	3	3	2	7	3	10	2	12	4										
11	2	5	5	2	1	6	1	8	1	9	1	12	1										
5	1	4	4	2	1	6	1	9	1	10	1												
3	2	6	3	4	2	7	2	12	2														
8	2	3	3	1	1	11	1	12	1														
1	1	2	2	8	1	12	1																

Figure 5.4: Sample of a conflict graph file

The conflict matrix is stored in file $\langle fnm.cns \rangle$ known as the conflict graph file (see Figure 5.4). The examinations are sorted by the number of examinations in conflict (in bold). Thus, any examination timetabling solver that uses the same file format specification for its conflict graph can use this static file as an input to the solver. Figure 5.5 shows the algorithm to create the conflict graph.

```

while ( more students )
    read StudentID
    CountExam = 0
    while ( more examination)
        read ExamCode
        ExamID = GetExamID (ExamCode)
        Increment CountExam
        ExamList[CountExam] = ExamID;
    endwhile
    for (i=1 to CountExam-1) do
        for (j=i to CountExam) do
            if (i != j)
                Increment conflict value of row ExamList[i] and column ExamList[j]
    Endwhile

```

Figure 5.5: Algorithm to create a conflict graph

Greedy Constructive Heuristic

A greedy constructive heuristic generates an initial solution by sequentially selecting an examination using a certain criteria and assigning the examination to a feasible slot (without violating any hard constraints). During the assignment of an examination to a slot, a greedy heuristic normally does not waste time trying to find the best slot but instead saves time by using the first available slot.

In this thesis, the greedy constructive heuristic used in finding an initial solution is the largest degree graph colouring heuristic (see Figure 5.6). This algorithm begins with an examination with the largest degree (i.e. with the largest number of

conflicts) and assigns it to the first available slot. If a slot is not available, the examination is put into an unscheduled list and the slot assigned is zero. The *CanSchedule* (Figure 5.6) function is a simple function that determines whether the *slotNumber* is a feasible slot for examination e_i . A slot is feasible if no other examinations, in conflict with the chosen examination is already assigned to that slot.

```

while (more examinations)
    select next examination  $e_i$  to schedule
    slotNumber = 0
    repeat
        Increment slotNumber
    until (CanSchedule( $e_i$ , slotNumber) or no more slots)
    if CanSchedule( $e_i$ , slotNumber)
        assign  $e_i$  to slotNumber
    else
        add  $e_i$  to unscheduled list
    endwhile

```

Figure 5.6: Largest Degree Constructive Algorithm

The same constructive algorithm can be used with the UiTM dataset but the initial solution is very poor because the first slot is a weekend slot and the examination with the largest number of conflicts is assigned to the first weekend slot. It is very difficult to move a very large examination from a weekend slot and therefore, a search algorithm may not find a better slot for it. The constructive algorithm needs to be modified so as to take account of a weekend slot. The new constructive algorithm first tries to find a feasible non-weekend slot. Once, all non-weekend slots are exhausted, it tries to find a weekend slot either of type *A* or type *B* (see section 5.4.1). Figure 5.7 shows the modified constructive algorithm used to construct an initial solution for the UiTM dataset. The initial solution produced using this algorithm is better when compared to the initial solution produced using the algorithm from Figure 5.6. This has an effect on the final solutions produced by

a search algorithm (Chapter 8).

```
while (more examinations)
    select next examination  $e_i$  to schedule
    slotIndex = 0
    repeat (more nonWeekendSlots)
        Increment slotIndex
        slotNumber = nonWeekendSlots[slotIndex]
    until (CanSchedule( $e_i$ , slotNumber) or no more non-weekend slots)
    if CannotSchedule( $e_i$ , slotNumber)
        if enrolmentWithWeekendTypeA < enrolmentWithWeekendTypeB
            {
                slotIndex = 0
                while (more weekendSlotsTypeA)
                    Increment slotIndex
                    slotNumber = weekendSlotsTypeA[slotIndex]
                until (CanSchedule( $e_i$ , slotNumber) or no more slots) }
        else
            {
                slotIndex = 0
                while (more weekendSlotsTypeB)
                    Increment slotIndex
                    slotNumber = weekendSlotsTypeB[slotIndex]
                until (CanSchedule( $e_i$ , slotNumber) or no more slots) }
    if CanSchedule( $e_i$ , slotNumber)
        assign  $e_i$  to slotNumber
    else
        add  $e_i$  to unscheduled list

endwhile
```

Figure 5.7: Modified Largest Degree Constructive Algorithm for UiTM dataset

5.7 Conclusions

This chapter has presented the examination timetabling problem using graph and mathematical models. The UiTM problem formulation is modified from an existing model used to solve benchmark datasets due to the additional constraints added to the UiTM problem.

A description of the conflict matrix graph was presented. Frequently, raw examination timetabling data comes in different formats and pre-processing is required so as to convert it into a general format as input to a solver. By storing the number of students involved in a conflict in a static conflict graph file, we do not need to keep retrieving and counting the number of conflicts from the student file.

The largest degree graph colouring constructive heuristic is used to generate initial solutions for a neighbourhood search algorithm. It is not our aim to provide a good initial solution to the hyper-heuristic framework described in the next chapters. The algorithm will place examinations that cannot be scheduled into an unscheduled list. No attempt was made to un-schedule certain examinations and reschedule the unscheduled examinations because we want the hyper-heuristic to work both with complete and partial solutions. The UiTM problem has an additional soft constraint i.e. to avoid weekend slots. The weekend slots are necessary to get a feasible schedule where all examinations are scheduled. However, it is desirable to avoid weekend slots (see questionnaire results in Chapter 4). Two constructive algorithms were presented, one that does not consider weekend slots and one that tries to avoid or reduce weekend slots.

The work presented in this chapter provides the necessary background and foundation for the hyper-heuristic framework or other meta-heuristic approaches to solve examination timetabling problems.

Chapter 6

Initial Study on a Tabu Search Based Hyper-heuristic for the Examination Timetabling Problem

This chapter reports on research into the design of a new hyper-heuristic framework using a tabu list and adaptive memory with the intention of monitoring and learning the behaviour and performance of low-level heuristics so as to help in making a well-informed decision on applying the best heuristic at each decision point.

Section 6.1 begins with an introduction to the rationale of using a hyper-heuristic in solving combinatorial optimisation problems. A detailed description of the hyper-heuristic framework and strategies are given in Section 6.2 and experimental results and analysis is presented in Section 6.3. The chapter concludes with a summary and future research directions for this thesis.

6.1 Introduction

In many real world optimisation problems, it is very difficult (if not impossible) to find optimal solutions by enumerating all possible solutions and selecting the optimum solution. Firstly, it may be difficult to find methods to enumerate the solutions and secondly, an optimum solution(s) is very problem specific and may be difficult to identify. The solution search space for an instance of a problem is often too large and feasible solutions of the search space may be disconnected and as such, it will be very difficult to design an exact heuristic that can thoroughly explore the

search space finding an optimal solution. Actually, the first obstacle is not the solution methodology, but a clear and precise specification of the important parameters and constraints to consider in constructing the solution method. For example, different problem instances of examination timetabling problems may have different sets of constraints and objectives in producing an acceptable and feasible timetable.

Chapter 2 has already highlighted the various methods and strategies applied by researchers to solve timetabling problems. Many of these methods have successfully solved given problems and some algorithms/heuristics were reported to work well with particular datasets whilst others performed better when presented with different datasets. This indicates that one of the potential research issues in timetabling is to design a high-level algorithm that automatically, and intelligently, chooses a method suitable for a given problem instance (Burke and Petrovic, 2002). The objective of this chapter is to investigate the design of a new hyper-heuristic system that is able to select the most appropriate algorithm for the current instance of a given timetabling problem and able to produce acceptable solutions that are comparable with other solution methodologies.

6.2 Hyper-heuristic Framework and Strategy

The hyper-heuristic framework (see Figure 6.1) can be divided into two modules: a general solver (hyper-heuristic) and the problem specific module (low-level heuristics and evaluation function). Initially, the hyper-heuristic needs to know the number of n heuristics provided by the low-level heuristic module. It will guide the search for good quality solutions by setting up its own strategy of calling and evaluating the performance of each heuristic known by their generic name $H_1, H_2, \dots, H_n$. The hyper-heuristic does not need to know the name, purpose or implementation detail of each low-level heuristic. It just needs to call a specific heuristic, H_i , and the heuristic may modify the solution state (s) and return a new solution (s_i') and a new value of an evaluation function ($f(s_i')$). The low-level

heuristics also return some attributes that assist the hyper-heuristic in making informed and intelligent decisions. The low-level heuristics module can be viewed as a ‘black box’ that hides the implementation detail and only returns a value. The low-level heuristics are the heuristics that deal directly with the problem at hand and require domain knowledge of the problem it is solving

Once a hyper-heuristic and a set of low-level heuristics have been shown to be able to solve and produce competitive solutions for an optimisation problem, we should be able to replace the low-level heuristics for another problem easily without reconstructing the search element (i.e. the hyper-heuristic) of the algorithm.

6.2.1 Tabu Search Based Hyper-heuristic Module

The hyper-heuristic module (see Figure 6.1) is the focus of this research where we need to design and test strategies that can intelligently select an appropriate heuristic from a set of given heuristics. The appropriate selection of heuristics should guide a search algorithm to either intensify or diversify the exploration of the search region.

The tabu search meta-heuristic has been explored in detail and applied to examination timetabling problems by several researchers (see section 2.3). The basic concept of this search is to overcome the problem of becoming stuck in local optimality by a strategy of forbidding (or penalising) certain moves (i.e. making it taboo or ‘tabu’). A move is made tabu mainly to prevent ‘cycling’ (i.e. visiting previous points in the search space or undoing a move that has been made before). The prohibited move is later made available after a certain number of subsequent moves have been made.

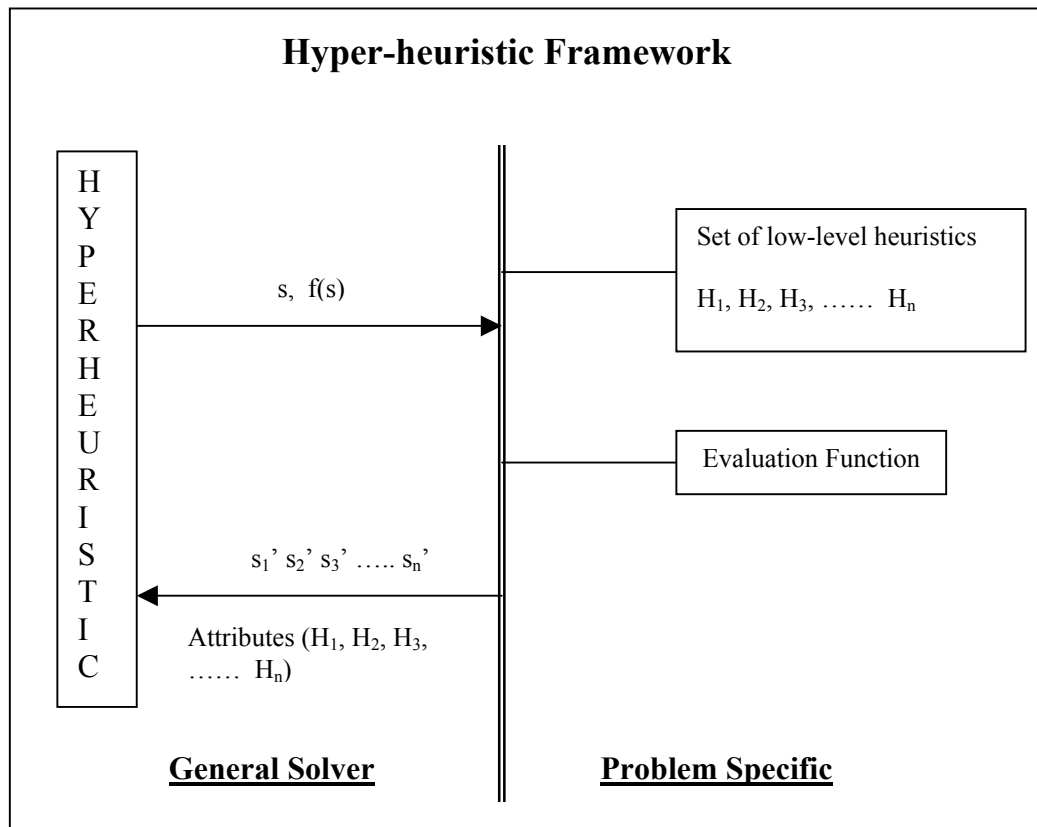


Figure 6.1: Hyper-heuristic Framework

The main issues that were addressed by tabu search researchers and can be explored further in the proposed tabu search based hyper-heuristic are as follows:

- How can we use memory to help store a history of previous moves (adaptive, short-term, long-term etc.)?
- What items should be stored in the tabu list?
- What is the best neighbourhood size?
- What operators should be used to move to neighbouring states of the current solution?
- How do we provide a balance between intensification and diversification?
- What are the conditions for tabu restriction?

- What are the factors that affect tabu tenure?
- What aspiration criteria can be used to avoid missing a potentially good solution?

Some of the above issues are incorporated in tabu search based hyper-heuristic and applied to examination timetabling problem.

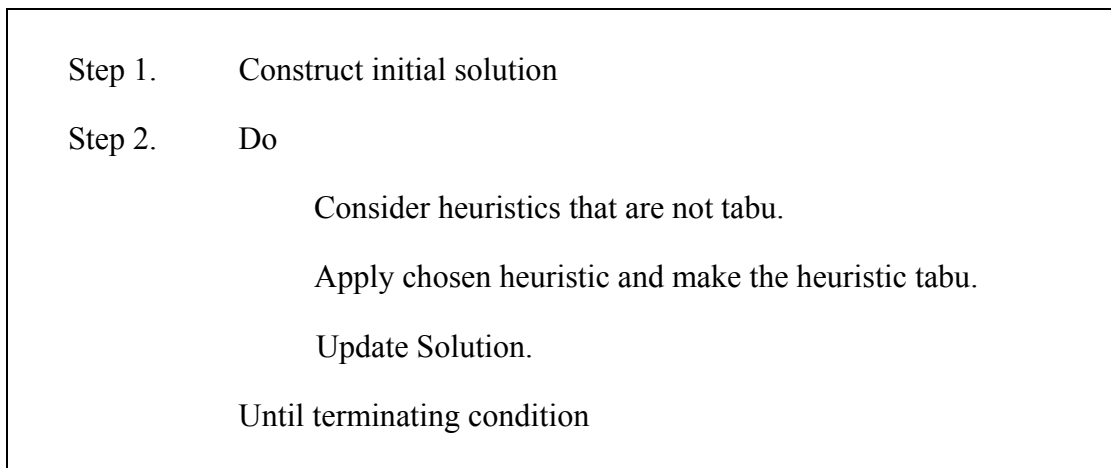


Figure 6.2: A general tabu search based hyper-heuristic procedure.

A general algorithm for the tabu search based hyper-heuristic is as shown in figure 6.2. The search begins (as with other local searches) with an initial solution to the problem. The initial solution is produced using a constructive heuristic (e.g. largest degree or saturation degree (Carter and Laporte (1996))). The initial solution need not be a good solution and it may not be feasible (e.g. some examinations are unscheduled). The algorithm works with infeasible solutions since some of the low-level heuristics will schedule unscheduled examinations. Next, a randomisation (randomly move examinations to other valid slots) is carried out to start different runs with different initial solutions. In Step 2 we explore the neighbourhood to search for a better solution or local optima (and possibly global optima). The framework is similar to a local search except that in Step 2, we explore the neighbourhood by selecting which heuristic to use to update the current solution.

Our tabu search based hyper-heuristic differs from other neighbourhood search

algorithms or meta-heuristics (such as Tabu Search and Simulated Annealing) with respect to the management of several heuristics. The hyper-heuristic manages the heuristics by selecting, which heuristic(s) should be considered and which heuristic(s) should be applied. In fact, the heuristics being considered can be a local search algorithm, a meta-heuristic or just a simple move operator.

The hyper-heuristic is like a manager who employs a team of *heuristic workers*. A good manager does not need to know how the workers do their job but it must be intelligent in recognising when a job is well done. The workers may be good or poor and sometimes a good combination of team workers will produce good solution. Normally when we have a team of workers, rather than asking them to work in sequence, we can ask them to perform their specific task simultaneously and whoever produces the best work will be accepted. Their progress will be monitored so that the manager can learn and recognise each workers' specialisation and will be able to decide and select the next team of workers.

Therefore, we can view the hyper-heuristic as a manager and the collection of heuristics as a team of workers who are given an area in the solution space (heuristic search space as opposed to the search space) and their task is to find a good solution and return it. The hyper-heuristic needs to manage the heuristics by choosing the appropriate heuristic for the current task. The heuristic may be doing a complex task by intensively exploring a large neighbourhood search space or it may just do a simple task of exploring a very small neighbourhood.

In the search for good quality solutions, the hyper-heuristic exhibits a kind of reinforcement learning (Nareyek, 2003), which will assist in making an intelligent action at each decision point. The hyper-heuristic monitors the behaviour of each low-level heuristic by storing information about their performance using an adaptive memory (See figure 6.1). The hyper-heuristic uses a tabu list that is of a fixed length n , where n is the number of low-level heuristics. Instead of storing moves, each tabu entry stores (non-domain) information about each heuristic i.e. the heuristic number,

the recent change in evaluation function and the tabu status (or tabu duration, which is the term we prefer to use). The tabu duration indicates how long a heuristic should remain tabu and, will therefore, not be applied in the current iteration. If the tabu duration is zero, the heuristic is said to be tabu inactive and can be applied to update the solution. If the tabu duration is non-zero, the heuristic is said to be tabu active and may not be used to update the solution. A heuristic is made tabu when it satisfies a specific tabu restriction condition. After each iteration, the tabu duration of each low-level heuristic is decremented until it reaches zero and the heuristic is now tabu inactive. Tabu search uses an aspiration criteria to remove (un-tabu) a tabu move that improves the solution. However, in tabu search based hyper-heuristic, it is not necessary to use an aspiration criteria since heuristics in the tabu list are never considered at all. Furthermore, the heuristic will eventually be removed from the tabu list once its tabu duration reaches zero. For each dataset a deterministic tabu duration between zero and four is used, so as to ascertain which tabu duration produces better quality solutions. We also use a random dynamic tabu duration where the tabu duration value is randomly selected from within a range.

The strategies that can be used when considering which heuristics to evaluate are: consider all heuristics (i.e. no tabu duration), consider heuristics that are not tabu, or consider heuristics that lead to improvement only. Each heuristic differs in how it decides to move, thus creating its own search space region (heuristic search space). At each choice point, the hyper-heuristic needs to decide whether it wants to intensify the search in a particular region by applying the same heuristic or to diversify the search into another region by applying a different heuristic. At this point, the hyper-heuristic can actually choose intelligently when to intensify or diversify because we believe that (by allowing the low-level heuristics to compete at each iteration and selecting the heuristic with the best performance) will help to balance the diversification and intensification of the solution search space. A heuristic that has been applied is made tabu so that in the next iteration we can look at the possibility of other low-level heuristics that may perform well but, perhaps, not as well as the previous heuristics that are now tabu active.

```

Construct initial solution  $s$ 
bestSolution =  $s$ 
initialCost =  $f(s)$ , bestCost =  $f(s)$ 
Set  $TD = t$  (fixed  $td$ ) or Set  $t_{min} = 0$  and  $t_{max} = 4$  (random dynamic  $td$ )
Set tabuStatus = 0 for all heuristics  $H_1, H_2, H_3, \dots H_i \dots H_n$ 
Do
    Call all heuristics that are not tabu (with tabuStatus=0)
        {input:  $s$  and  $f(s)$ , output:  $s_i'$ ,  $f(s_i')$  and heuristic attributes}
    Evaluate  $f(s_1'), f(s_2'), \dots f(s_n')$  and select  $H_c$  with the best feasible  $f(s')$ 
    Decrement tabuStatus of all heuristics with tabuStatus > 0
    Apply  $H_c, s = s_c', f(s) = f(s_c')$ 
    Set  $H_c$  tabuStatus =  $t$  or select at random between  $t_{min}$  and  $t_{max}$ .
    If  $f(s_c') < bestCost$  then bestCost =  $f(s_c')$  and bestSolution =  $s_c'$ 
Until terminating condition

```

Figure 6.3: Tabu search based hyper-heuristic algorithm

A hyper-heuristic with the simplest strategy is implemented (see the algorithm in figure 6.3), i.e., tabu search based hyper-heuristic (TSHH), where all non-tabu heuristics are called and the current solution is updated with the best solution produced by the low-level heuristics in the current iteration. The current best solution will also be updated if the new solution is better than the current best solution. Once a solution from a heuristic is accepted, the heuristic is made tabu (tabu status equals tabu duration) and the heuristic is considered again when its tabu status is zero. Each run of the TSHH will use either a deterministic tabu duration (0,1,2,3 or 4) or a random dynamic tabu duration. Random dynamic tabu duration uses a tabu duration range (t_{min} and t_{max}), and whenever we want to make a heuristic tabu, a tabu duration value is selected at random from the given range. The algorithm iterates for a fixed time or until there is no further improvement for a given number of heuristic calls. The results and analysis of experiments are given in section 6.3.

Burke et al. (2003) also apply a tabu search hyper-heuristic to course timetabling and nurse rostering. See section 3.2 for a description of their approach.

6.2.2 Low-level Heuristics Module

Low-level heuristics used here in the context of hyper-heuristics, are heuristics that provide solutions by directing movement through a solution space that require domain knowledge and are problem dependent. The low-level heuristics (or simply heuristics) transforms the current problem state to a new problem state. Each heuristic creates its own heuristic search space (Storer et al., 1995). The idea is to build a collection of possible simple moves or choices since we would like to provide a library of heuristics that can be selected intelligently by a hyper-heuristic tool. This library, at the moment, will only include simple low-level heuristics and future work will include the possibility of adding other meta-heuristics such as simulated annealing or tabu search.

Each low-level heuristic can be considered as improvement heuristics that returns a move and a change in the penalty function. The best performing heuristic should cause a maximum decrease in penalty (the lowest value). Each move from an individual heuristic may cause the search to probe into the current neighbourhood or to explore into a different neighbourhood. A change in the penalty value means changing the penalty value for each of the soft constraints that were violated (each proximity cost conflict) or moving an examination into an unscheduled list (examination becomes unscheduled and violates hard constraints).

The following low-level heuristics are implemented and classified under four categories:

1. Select and schedule examination – Selecting the next examination to schedule is dependent upon which factor is considered to be important in determining the difficulty of scheduling an examination. Five strategies that had been used in the

literature (see section 2.3.2), and are adapted from the graph colouring heuristics (largest degree, largest weighted degree, largest colour degree, least saturation degree and largest enrolment), are used here. The heuristics select examination from the unscheduled list and schedule it into the best available slot that will maximise the reduction in penalty.

2. Move examination i from slot x to slot y .

- Select an examination at random and move to another random slot.
- Move examination i with maximum penalty from randomly selected examinations.
- Move examination i with highest third order conflict from slot x to a new randomly selected slot y .
- Move examination i with highest third order conflict from slot x to a new slot y which maximises the reduction in third order conflict.
- Move examination i with highest second order conflict from slot x to a new randomly selected slot y .

3. Swap.

- Random - Select an examination at random and find another examination at random which can swap slots.
- Min-Max Swap – Swap the slots for examination with minimum penalty and examination with maximum penalty.

4. Remove – Remove a randomly selected examination from examinations already scheduled. This is the only heuristic which will move the search into an infeasible region because any examination may be unscheduled. We make sure that the search can move back into its feasible region by un-scheduling examinations that have other valid slots it can move into in the next iteration.

All of the above low-level heuristics are either 1-opt (one move) or 2-opt (two moves) and there is also a mixture of some randomness and deterministic selection of examinations and slots. The low-level heuristics are purposely designed with simple moves rather than intelligent and complex moves because we want to make sure that the hyper-heuristic can recognise good moves and make an intelligent decision based on these simple moves. Furthermore, the underlying principle of a hyper-heuristic is to make the problem-domain knowledge heuristics easy to implement and the hyper-heuristic more generalised.

6.3 Experimental Results

The tabu search based hyper-heuristic (TSHH) framework was implemented and tested on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The program was coded in C++ using an object-oriented approach. The hyper-heuristic and heuristics were defined and implemented as objects that have a common interface and can interact with each other. Once the hyper-heuristic object is fully defined, implemented and tested with a set of heuristic objects for one application, we can easily reuse the hyper-heuristic object with another set of heuristic objects for a different application. This approach should be cost effective because it can reduce the resources required to build another system. Thus, we can easily produce solutions for users who require “*good enough - soon enough - cheap enough*” (Burke et. al. 2003a) solutions to their problems by implementing several domain specific low-level heuristics with simple moves.

Therefore, the objectives of these experiments are:

- To establish a well defined relationship between a hyper-heuristic module and its domain specific low-level heuristics module
- To compare the quality of the results produced by this hyper-heuristic with other known methods published using similar quality measures.

- To demonstrate that the hyper-heuristic module does not need to rely upon domain knowledge to make its decisions.
- To demonstrate that the hyper-heuristic can manage and choose the low-level heuristics at each decision point in a search.
- To evaluate the performance of the low-level heuristics.
- To determine further improvement in this hyper-heuristic module, which we will investigate in later chapters.

6.3.1 Datasets

The hyper-heuristic system was tested with datasets taken from established datasets made public and used by a number of other examination timetabling researchers. The datasets were provided by Michael Carter and can be downloaded from <ftp://ftp.mie.utoronto.ca/pub/carter/testprob/>. Details on these datasets are described in chapter 5 (table 5.1 shows the characteristics of each dataset). Initial investigation on this method is first carried out using eight benchmark datasets and later extended to twelve benchmark datasets.

6.3.2 Experimental Results and Analysis

The new hyper-heuristic framework, with fixed tabu duration (TSHH), was first tested with eight benchmark datasets. Initially only eight datasets are used to study the effect of tabu duration value and run time. Each dataset is experimented with tabu durations varying from 0 to 4 and with two different terminating conditions (no further improvement for the last 10,000 iterations or running time of 10 minutes). In Table 6.1, the first column shows the file name of each dataset and the next five columns show the tabu search based hyper-heuristic results (best penalty value per student) with a tabu duration varying from 0 to 4. The actual time that it takes to find the best solution is not shown here but the best results are normally found towards the end of the search. After further analysis on the performance graph it was

found that improvements are still being made toward the end of the run time. Therefore, the algorithm was prolonged for four hours with a tabu duration of 2 (many of the datasets work best with tabu duration of 2), to see whether much better solutions can be found if the algorithm runs longer than the 10 minutes that was used previously. The last column in Table 6.1 shows that prolonging the algorithm does improve the solution further and it demonstrates that given more time it is able to avoid being trapped in local optima and can find better solution.

Table 6.1: Comparing results of TSHH with 10 minutes run time (with deterministic tabu duration value) and 4 hours run time

File	Tabu Search based Hyper-Heuristic with fixed TD (best penalty value per student from 8 runs)					TSHH (4 hr. run)	% improve with long run
	TD=0	TD=1	TD=2	TD=3	TD=4	TD=2	
Car-f92	5.94	5.52	5.46	5.63	6.02	4.67	14.47%
Car-s91	6.91	6.47	6.32	6.98	7.10	5.37	15.03%
Ear-f83	47.01	44.58	43.58	43.54	45.16	40.18	7.80%
Hec-s92	13.84	14.09	12.79	12.24	12.86	11.86	7.27%
Kfu-s93	20.53	18.32	18.08	19.22	19.86	15.84	12.39%
Sta-f83	168.4	165.8	165.6	166.3	167.1	157.38	4.96%
Tre-s92	11.01	10.99	9.79	10.99	10.59	8.39	14.30%
Ute-s92	29.34	28.18	27.97	29.20	31.05	27.60	1.32%

When the tabu duration is 0, the hyper-heuristic does not make any heuristics tabu and, since none of the results is the best among the datasets, it can be concluded that a tabu list does contribute towards guiding the hyper-heuristic in its heuristic selection. The tabu based hyper-heuristic strategy is to utilise all the heuristics if applicable and rather than making a non-performing heuristic tabu, the hyper-heuristic make the performing heuristic tabu so that other heuristics can be applied as well. When tabu duration is greater than zero, tabu restriction applies where a heuristic will be tabu active if its solution value has been accepted to update the current solution. The heuristic will remain tabu active for a number of steps equal to tabu duration. A heuristic that has been applied is made tabu so that we can direct the search to other possible heuristics. Eventually the search may go back to a

heuristic once it is no longer tabu active and can give the best solution amongst all tabu inactive heuristics. The best result for six of the datasets is when $TD = 2$ and for two of the datasets, the best result is when $TD = 3$. It is interesting to find that two datasets (ear-f83 and hec-s92) obtain the best result when TD is higher and has a higher conflict matrix density (refer to Table 5.1) i.e. 26.7% and 42.0%. An examination timetabling dataset with a higher conflict matrix density is normally more difficult, and therefore, it may require a higher tabu duration to diversify its exploration of the solution search space by allowing it to move from one heuristic to another. For each of the datasets, except one (hec-s92), the average penalty started to decrease as the tabu duration increased and began to increase again once it reach its minimum average penalty. This shows that the hyper-heuristic does need to choose which tabu duration is best for each dataset because a tabu duration, which is too high or too low, will produce worse solutions.

From table 6.1, we can see that prolonging the run time improves solutions further. Therefore, we continue running the experiments for four hours. The next table (table 6.2) shows the results for twelve benchmark datasets with fixed deterministic tabu duration (0, 1, 2, 3 or 4) and a random dynamic tabu duration. Random dynamic tabu duration uses a tabu duration range (0 - 4), and whenever we want to make a heuristic tabu, a tabu duration value is selected at random from the given range (refer to the algorithm illustrated in figure 6.3). Table 6.2 shows that with a longer run (4 hours), overall performance has improved. Four datasets perform better with random dynamic tabu duration while seven datasets perform better with $TD = 2$ and only one dataset perform well with $TD = 3$. With a random dynamic tabu duration, we can conclude that some datasets work best with a mixed and a dynamic change in tabu duration value during the search. If we observe and compare results from fixed deterministic tabu duration only, the best tabu duration across all datasets (except ute-s92) is 2. The results from the best tabu duration value are also as competitive as the results from the random dynamic tabu duration. For deterministic tabu duration, it is necessary to tune the tabu duration parameter to find the most suitable tabu duration value for a dataset. The maximum tabu duration

value should not be too large because we will only have a small number of heuristics to be tabu. In our experiments, the maximum tabu duration value for this problem is about 30% of the total number of low-level heuristics.

Table 6.2: Results of TSHH for 12 benchmark datasets with fixed deterministic tabu duration and random dynamic tabu duration (4 hours run)

File	Tabu Search Based Hyper-heuristic					
	TD=0	TD=1	TD=2	TD=3	TD=4	RTD
car-f92	5.83	6.05	4.67	5.35	6.05	4.61
car-s91	6.88	6.83	5.37	6.31	6.91	5.43
ear-f83	47.61	44.07	40.18	41.29	44.43	41.56
hec-s92	13.94	13.07	11.86	12.31	13.64	11.97
kfu-s93	19.27	17.69	15.84	16.13	17.45	15.56
Lse-f91	15.04	14.89	13.67	14.53	15.87	13.29
pur-s93	6.13	6.07	6.06	7.31	8.50	5.67
sta-f83	157.64	164.62	157.38	166.32	164.62	157.42
Tre-s92	10.48	10.99	8.39	9.46	10.16	9.21
uta-s92	4.93	4.50	3.92	4.60	5.16	4.23
ute-s92	31.36	27.59	27.60	27.28	29.83	27.34
yor-f83	45.98	43.16	39.42	40.77	42.39	40.04

Figure 6.4 shows the hyper-heuristic performance with different tabu duration on the car-f92 dataset. This dataset is one of the largest with 543 examinations to schedule in 32 slots and with a total number of students of 18,419. This graph demonstrates how the hyper-heuristic explores the search space. The x-axis represents the iteration steps up to 250,000 moves while the y-axis represent overall penalty cost. Note that the timetable quality is measured by taking the average penalty per student. The curve shows that the algorithm begins with an initial solution and rapidly improves the result in less than 10,000 moves. The graph shows fluctuations because at every move a solution from the best performance heuristic is accepted even though it does not improve the solution. The higher tabu duration means that the heuristics will remain tabu longer, thus allowing other heuristics to be applied. By increasing the tabu duration, the next solution accepted may make the

solution much worse but it can still improve the solution in the next move. So, a tabu duration value does help to improve solution quality, and too high a value may make the solution much worse, making it difficult to improve it again. The simplest form of this hyper-heuristic does not limit the range of how much a worse solution may be accepted but further investigation (see section 7.2) on this hyper-heuristic will limit the acceptance of a worse solution.

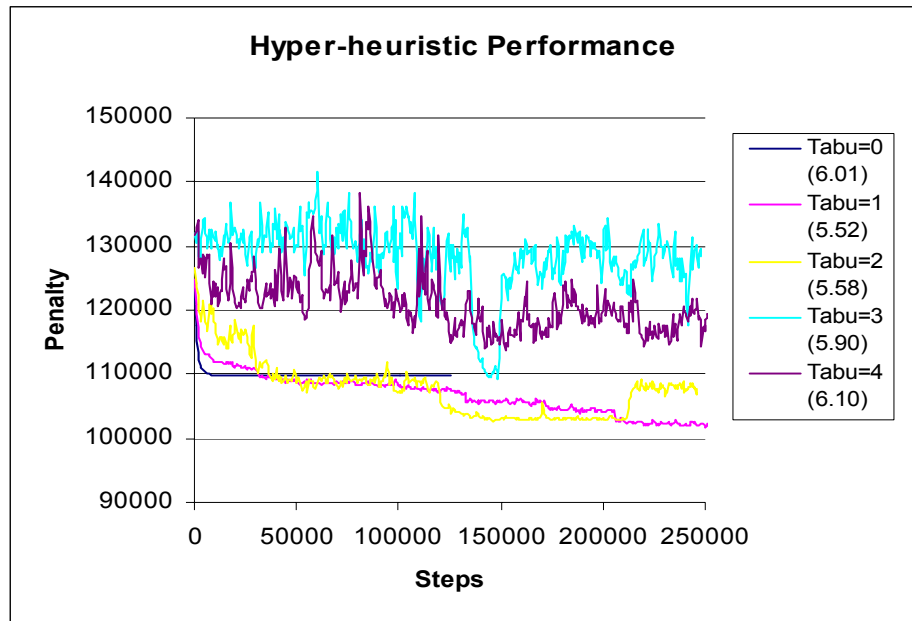


Figure 6.4: Tabu search base hyper-heuristic performance with different TD.

The graphs in figure 6.5 and figure 6.6 show when two of the heuristics were applied and how the two heuristics change the solution state. The larger number of plots in figure 6.5 compared to figure 6.6 indicates that heuristic 6 has been applied more than heuristic 7. Graphs for other heuristics are not shown here because their shape is almost the same except that some are applied more than others. For different datasets certain heuristics are applied more than the others, therefore it is justifiable to use several low-level heuristics that can compete with each other and a hyper-heuristic can then select the best low-level heuristic to be applied, given not only the point in the search space but also a specific problem instance.

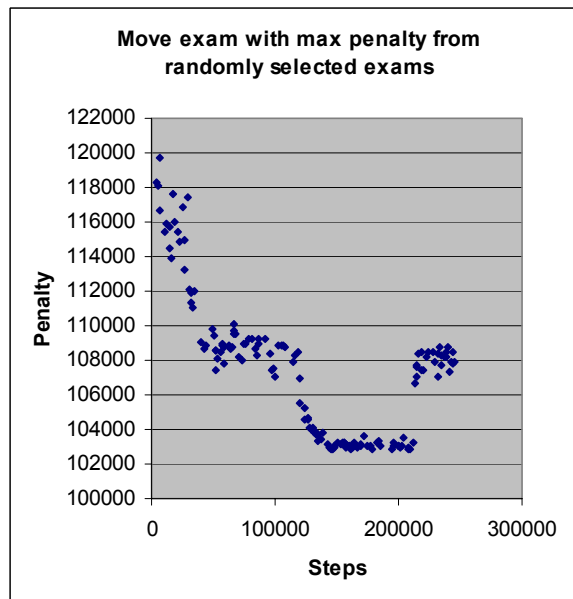


Figure 6.5: Heuristic 6 – Select examination at random and move to random slot

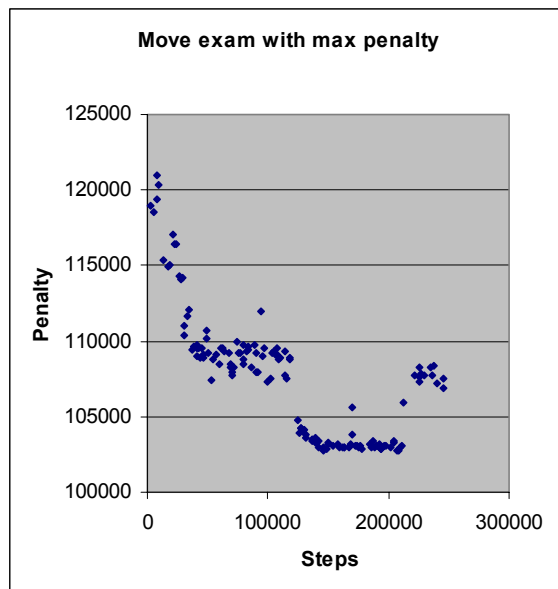


Figure 6.6: Heuristic 7 – Move examination with maximum penalty from randomly selected examinations

Table 6.3 shows the results from when the tabu search based hyper-heuristic is run for four hours compared with the other published results for benchmark datasets. The objective here is to show that the tabu search based hyper-heuristic is able to produce good quality and feasible solutions for examination timetabling problems even though they may not produce the best results.

Table 6.3: Comparing TSHH-4 hour run and published results

File	TSHH (TD=2)	Di Gaspero & Schaerf (2001)	Di Gaspero (2002)	Carter et al. (1996)	Caramia et al. (2001)	Merlot et al. (2003)	Burke & Newall (2004)
Car-f92	4.67	5.2	-	6.2 - 7.6	6.0	4.3	4.32
Car-s91	5.37	6.2	5.68	7.1 - 7.9	6.6	5.1	4.97
Ear-f83	40.18	45.7	39.36	36.4 - 46.5	29.3	35.1	36.16
Hec-s92	11.86	12.4	10.91	10.8 - 15.9	9.2	10.6	11.61
Kfu-s93	15.84	18.0	-	14.0 - 20.8	13.8	13.5	15.02
Lse-f91	13.67	15.5	12.6	10.5 - 13.1	9.6	10.5	10.96
Pur-s93	6.06	-	-	3.9 - 5.0	3.7	-	-
Sta-f83	157.38	160.8	157.43	161.5-165.7	158.2	157.3	161.91
Tre-s92	8.39	10.0	-	9.6 - 11.0	9.4	8.4	8.38
Uta-s92	3.92	4.2	4.1	3.5 - 4.5	3.5	3.5	3.36
Ute-s92	27.60	29.0	-	25.8 - 38.3	24.4	25.1	27.41
Yor-f83	39.42	41.0	39.7	41.7 - 49.9	36.2	37.4	40.77

The first two comparisons are results against Di Gaspero and Schaerf (2001) who use tabu search and Di Gaspero (2002) who uses tabu search with multi-neighbourhoods. The tabu search based hyper-heuristic results are better than the tabu search method in all cases and almost as good as the tabu search with multi-neighbourhoods. Comparisons are also made with results from other methods such as: constructive heuristics with backtracking by Carter et al. (1996); adaption of heuristics orderings by Burke and Newall (2004); greedy constructive heuristic with an optimiser by Caramia et al. (2001); and hybrid of constraint programming, simulated annealing and hill climbing with Kempe chain neighbourhood by Merlot et

al. (2003). Tabu search based hyper-heuristic results are better than Carter et al. (1996) in five cases, better than Caramia et al. (2001) in four cases, better than Burke and Newall (2004) in two cases and better than Merlot et al. (2003) in one case. Two of our results (sta-f83 and tre-s92) are almost the same as the best results (difference of 0.01 and 0.08). We did not perform the worst in any of the datasets except for pur-s93. In the above comparison only two other methods attempted the pur-s93 dataset. This is the largest dataset and not many approaches are tested with this dataset. See table 2.1 for a detailed comparison with other methods.

The results show that the tabu search based hyper-heuristic is able to produce good quality solutions compared to some of the others. In general, the tabu search based hyper-heuristic works reasonably well across all problem instances. Further enhancements in the tabu search based hyper-heuristic heuristic selection method, acceptance criteria of new solution and some adaptive tabu duration should improve the solution quality even further (see next chapter).

6.4 Conclusions

The simplest form of the hyper-heuristic module (tabu search based hyper-heuristic with fixed deterministic tabu duration and random dynamic tabu duration) has been implemented and tested on examination timetabling benchmark datasets. Preliminary results showed that it is not able to beat the best results in the literature but it is able to produce good quality solutions. The main objective of this algorithm is not to beat the best solution but to show that the hyper-heuristic module does produce good solutions that are feasible and will work across all problem instances and other real-world problems. This generic solution methodology can easily be applied to other problems by just changing the low-level heuristics and the evaluation function while the search method remains the same.

The work presented in this chapter was published (Kendall and Mohd Hussin 2005a) and presented at the 1st Multidisciplinary International Conference on

Scheduling: Theory and Applications (MISTA 2003).

The next chapter explores the possibility of including more advanced control techniques and adaptive tabu strategies, and applies the same hyper-heuristic on a larger real world timetabling instance. Two further modifications of the tabu search based hyper-heuristic will also be introduced in the next chapter. Firstly, the selected heuristic will be reapplied and made tabu only if it cannot improve the solutions further (i.e. a hill climbing or steepest descent). Secondly, the acceptance criterion is modified so that worst solution is only accepted to modify the current solution if it is below a threshold value. We will also look at how the unscheduled examination weight can direct the search from feasible to infeasible regions, and vice versa.

Chapter 7

Control Techniques in the Tabu Search Based Hyper-heuristic

Initial results with a simple tabu search based hyper-heuristic described in Chapter 6 and in (Kendall and Mohd Hussin 2005a) are competitive when compared to other results published on the same datasets and using the same objective function. Across all the benchmark datasets tested, the simplest form of a tabu search based hyper-heuristic performs better when compared to a tabu search meta-heuristic. In this chapter, we further explore more advanced control techniques by incorporating intelligent and dynamic decision tasks within the hyper-heuristic.

Section 7.1 presents the motivation for the design of new hybridised control techniques of classical search with a tabu search based hyper-heuristic. A detailed description of the control flow in a hyper-heuristic framework and control strategies are given in Section 7.2. Experimental results and analysis on benchmark datasets and a new dataset is presented in Section 7.3.

7.1 Introduction

It has been shown in the previous chapter that the simple tabu search based hyper-heuristic is an improvement over the classical tabu search methods. However, its performance can still be further improved by considering the strength of other search methods and incorporating it into the tabu search based hyper-heuristic so as to improve its search space exploration.

It is noted in chapter 2 that many solution methodologies incorporate or adopt new or classical strategies from other methods to enhance their performance. Some of these strategies are: using multiple solutions (population) based methods as opposed to single solution based method (GRASP, memetic algorithm, genetic algorithm); deciding to accept worse solutions to replace current solution with a calculated probability (simulated annealing, monte carlo method) or based on a roulette wheel (genetic algorithm) or below an upper bound (threshold acceptance method, great deluge algorithm); fuzzy decision criteria (ordering heuristics by Asmuni et al., 2004); multi-neighbourhood structure (Di Gaspero 2002); adaptive heuristic ordering (Burke and Newall, 2004), etc.

In this chapter, we will make an improvement over the basic tabu search based hyper-heuristic in managing the currently performing heuristic, controlling and directing the search trajectory along promising regions, and prohibiting certain heuristics for a fixed deterministic period of time or a random dynamic period of time. The main issue in hyper-heuristics is, of course, in the management of several low-level heuristics and learning how the low-level heuristics can cooperate with each other in the search for optimality.

7.2 The Tabu Search Based Hyper-heuristic

The general idea of a hyper-heuristic is to operate at a higher level of abstraction than a meta-heuristic by interacting with its domain dependent low-level heuristics via a common interface. Figure 7.1 shows a general structure of a tabu search based hyper-heuristic. The information maintained in this structure is all non-domain knowledge information such as the change in the evaluation function and various attributes of each low-level heuristic (heuristic number, frequency of improving and non-improving moves and tabu status). It is important that we only utilise non-domain information in our hyper-heuristic as this means we can simply replace the low-level heuristics and the evaluation function and apply the hyper-heuristic to a totally different problem.

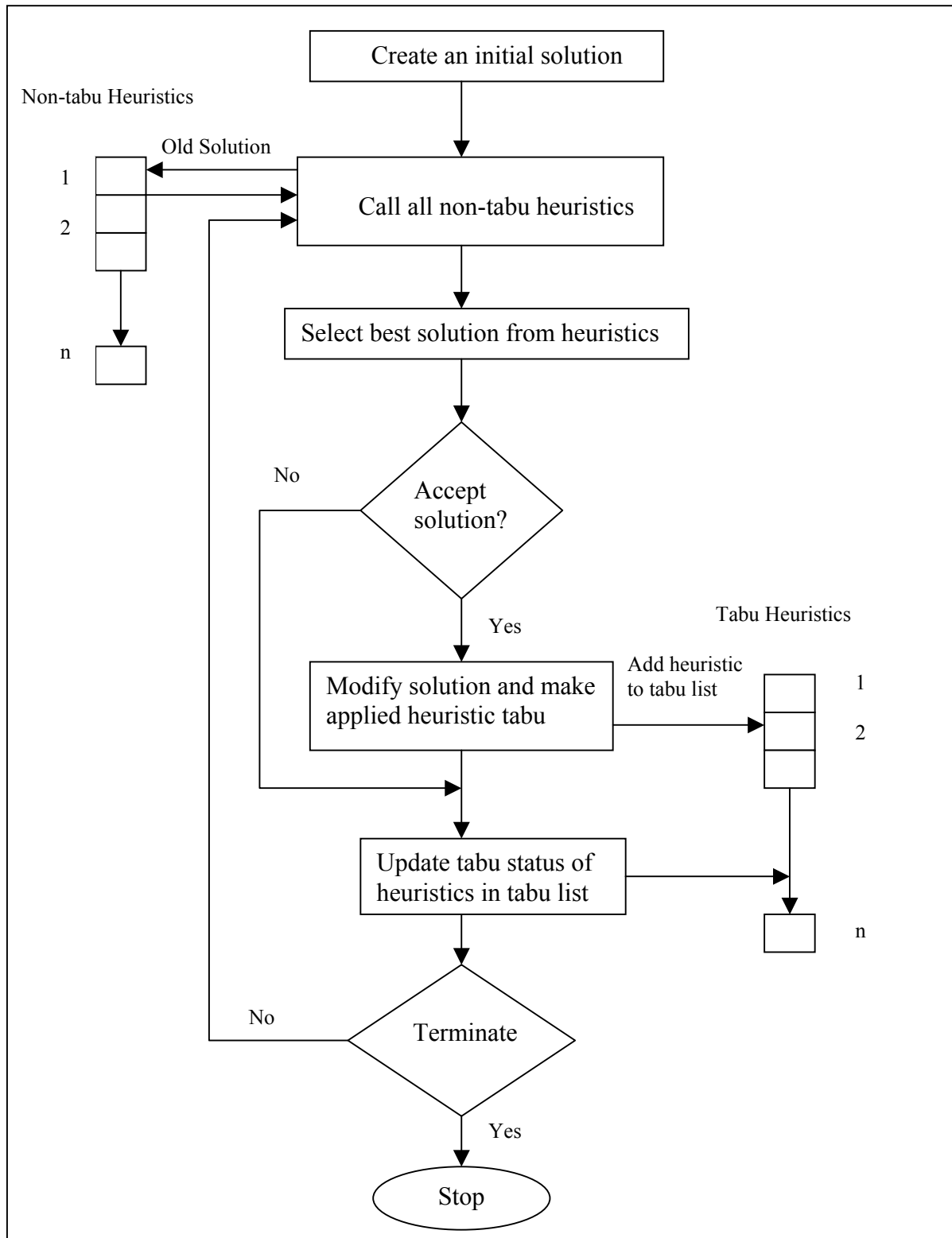


Figure 7.1: Tabu Search based Hyper-heuristic Framework

The main purpose of a tabu list is to keep some heuristics tabu (not evaluated on the current state of the problem). In this case, a heuristic that has performed well in the previous iteration is made tabu to allow other heuristics to be called and chosen next.

The first implementation (discussed in Chapter 6 and reported in Kendall and Mohd Hussin, 2005a), of tabu search based hyper-heuristic (TSHH-B) used a deterministic tabu duration where, in each run, a fixed tabu duration is applied. We also test the algorithm with a random dynamic tabu duration. Random dynamic tabu duration uses a tabu duration range (t_{min} and t_{max}), and at each decision point of making a heuristic tabu, a tabu duration value is selected at random from the given range. In general, having low tabu duration allows the exploration to move within the same heuristic search space and a high tabu duration allows exploration into other possible regions. If a tabu duration is too high, the quality of the solution deteriorates since good heuristics are made tabu for too long. If a tabu duration is too low (or equal to zero), we have limited ourselves to just search within a small region in the solution space.

The next issue to address is the mechanism for updating a solution. In each iteration, each heuristic is given a solution state and it needs to return a new solution state, and the best solution is selected by the hyper-heuristic. Three different strategies are used to accept the best solution as a new solution at each decision point. First, is to accept the solution from the best performing heuristic (we call this TSHH-B). This best solution may not improve the current or previous best solution. Second, is to accept the best solution and if this solution improves the previous solution, the same heuristic will be applied until it cannot improve the solution anymore (steepest ascent hill climbing; TSHH-HC). Third, is to accept the best solution only if the solution is less than a boundary penalty (TSHH-GD). The boundary penalty was first introduced by Dueck (1993) in his great deluge algorithm, and Burke et al. (2004a) also develop it for the examination timetabling problem.

In this chapter, both deterministic tabu duration and random dynamic tabu duration are considered. The advanced hyper-heuristic incorporates other decision strategies in modifying the solution state. In the simplest form of the hyper-heuristic it was found that, prolonging the run time can improve results further subject to the correct tabu duration being used. Therefore, in the following experiments a run time of four hours seems reasonable to evaluate a more advanced hyper-heuristic with twelve benchmark datasets, five different fixed deterministic tabu durations and random dynamic tabu duration.

7.2.1 A Tabu Search based Hyper-heuristic with Hill Climbing (TSHH-HC)

In the management scenario (a manager and a group of workers) described in chapter 6, a manager may decide that a good worker can be re-utilised until no further improvement is obtained or it can be asked to take a break and allow other workers to demonstrate their (possible) potential. This management method is conceptualised in the next hyper-heuristic. The second hyper-heuristic adds to the basic tabu search hyper-heuristic (TSHH-B, figure 6.3) a successive call to the best performing heuristic until no further improvement is made. This allows the algorithm to re-use a heuristic until it cannot improve the current solution. Once this happens, it is made tabu. This is motivated by the steepest descent or hill climbing algorithm. We are exploiting the same heuristic search space in order to get a better solution. Implicitly we have also included in our hyper-heuristic the ability to escape from local optima by using one of the other heuristics. Figure 7.2 shows the algorithm for TSHH-HC with modified lines from TSHH-B (see figure 6.3) printed in bold.

```

Construct initial solution  $s$ 
 $bestSolution = s$ 
 $initialCost = f(s)$ ,  $bestCost = f(s)$ 
Set  $TD = t$  (fixed  $td$ ) or Set  $t_{min} = 0$  and  $t_{max} = 4$  (random dynamic  $td$ )
Set  $tabuStatus = 0$  for all heuristics  $H_1, H_2, H_3, \dots, H_n$ 
Do
    Call all heuristics that are not tabu (with  $tabuStatus=0$ )
    {input:  $s$  and  $f(s)$ , output:  $s_i'$ ,  $f(s_i')$  and heuristic attributes}
    Evaluate  $f(s_1')$ ,  $f(s_2')$ , ...  $f(s_n')$  and select  $H_c$  with the best feasible  $f(s')$ 
    Decrement  $tabuStatus$  of all heuristics with  $tabuStatus > 0$ 
    Repeat a call to  $H_c$ , until  $f(s_c') > f(s_c)$ 
    Apply  $H_c$ ,  $s = s_c$ ,  $f(s) = f(s_c)$ 
    Set  $H_c$   $tabuStatus = t$  or select at random between  $t_{min}$  and  $t_{max}$ .
    If  $f(s_c) < bestCost$  then  $bestCost = f(s_c)$  and  $bestSolution = s_c$ 
Until terminating condition

```

Figure 7.2: Tabu search based hyper-heuristic with hill climbing algorithm

7.2.2 A Tabu Search based Hyper-heuristic with Great Deluge (TSHH-GD)

In TSHH-B and TSHH-HC, there is a possibility that all heuristics produce very bad solutions and hence the best solution is still a poor solution. Therefore, we need to control the algorithm to prevent it from accepting very poor solutions. Some strategies used in controlling which solutions to accept can be seen in other meta-heuristics such as simulated annealing, threshold acceptance, Monte Carlo methods and the great deluge algorithm. The third hyper-heuristic is motivated by the great deluge algorithm (GDA) introduced by Dueck (1993). The algorithm deals with feasible solutions and accept worse solutions than the current one if the cost evaluation is less than or equal to a boundary called *level*. The *level* is reduced in each iteration using a *decay rate* that is dependent upon the initial cost evaluation,

desired cost evaluation and the time for the algorithm to achieve this desired cost. A more detailed description of the great deluge algorithm and how this approach has been adapted by Burke and Newall (2003) and Burke et al. (2004a) to solve examination timetabling problems was presented in chapter 2. An advantage of using the great deluge algorithm is the freedom from tuning the algorithm with different parameters. The desired cost evaluation may be the lower bound (if known), best result already published or determined by the user. The time taken can be a fixed run time or a maximum number of iterations.

Figure 7.3 shows the algorithm for TSHH-GD with modified and additional lines from TSHH-B (see figure 6.3) printed in bold. In TSHH-GD, the initial solution may not be a feasible solution and the desired cost is taken from the best result for each benchmark dataset already published. The execution time is limited by a number of steps (i.e. 10,000,000 iterations) and the algorithm will terminate if there is no improvement in the last 10,000 iterations. The time taken by a TSHH-GD iteration is longer compared to GDA because GDA generates only one new solution by doing a random move while TSHH-GD generates several solutions as it calls every heuristic which is not tabu. Each low-level heuristic may also take a longer time than just a random move. The decay rate is calculated as the difference between the desired cost and the initial cost divided by the number of iterations. In each iteration, the boundary is lowered by the decay rate. Burke et al. (2004a) modify the acceptance criteria of the great deluge algorithm by accepting improving moves (as in Hill Climbing) even if it is higher than the boundary. A similar acceptance criterion to Burke et al. (2004) is used in TSHH-GD. A solution from the best performing low-level heuristic is accepted to modify the current solution if it is better than the previous solution or it is lower than the boundary. The time taken in one iteration varies for each dataset because the time is dependent upon the number of examinations, the number of slots and the number of examinations in conflict. Therefore, the TSHH-GD with a fixed number of steps and fixed number of no improvement moves will run for different amounts of time for each dataset.

```

Construct initial solution  $s$ 
bestSolution =  $s$ 
initialCost =  $f(s)$ , bestCost =  $f(s)$ 
nSteps = maximum number of iterations
desiredCost = best cost as published in literature  $\times$  number of students
boundary =  $f(s)$ 
decayRate = (boundary – desiredCost) / nSteps;
Set TD =  $t$  or  $t_{min} = 0$  and  $t_{max} = 4$ 
Set tabuStatus = 0 for all heuristics  $H_1, H_2, H_3, \dots H_n$ 
Do
    Call all heuristics that are not tabu (with tabuStatus=0)
    {input:  $s$  and  $f(s)$ , output:  $s_i'$ ,  $f(s_i')$  and heuristic attributes}
    Evaluate  $f(s_1')$ ,  $f(s_2')$ , ...  $f(s_n')$  and select  $H_c$  with the best feasible  $f(s')$ 
    Decrement tabuStatus of all heuristics with tabuStatus > 0
    If (( $f(s_c') - f(s) < 0$ ) or ( $f(s_c') < \text{boundary}$ ))
        Apply  $H_c$ ,  $s = s_c'$ ,  $f(s) = f(s_c')$ 
        Set  $H_c$  tabuStatus =  $t$  or select at random between  $t_{min}$  and  $t_{max}$ .
        If  $f(s_c') < \text{bestCost}$  then bestCost =  $f(s_c')$  and bestSolution =  $s_c'$ 
    boundary = boundary - decayRate
Until terminating condition

```

Figure 7.3: Tabu search based hyper-heuristic with great deluge algorithm

7.2.3 Low-level Heuristics

The same low-level heuristics (as described in section 6.2.2 and in Kendall and Mohd Hussin, 2005a) are used with the new hyper-heuristics. The low-level heuristics are summarised as follows:

- § Five graph colouring heuristics that schedule examinations from the unscheduled list.
- § Five move heuristics that select an examination, either at random, with maximum penalty, with highest second order conflict or highest third order conflict. This examination is re-scheduled into a new random slot or a new chosen slot that maximises the reduction in either the total penalty, total second order conflict or third order conflict.
- § Two swap heuristics that select an examination, either at random, with maximum penalty or with minimum penalty. The two selected examinations will swap slots subject to no hard constraint violations.
- § A heuristic that removes a randomly selected examination from examinations already scheduled. This is the only heuristic which will move the search into an infeasible region because any examination may be unscheduled. We make sure that the search can move back into its feasible region by un-scheduling examinations that have other valid slots it can move into in the next iteration.

7.3 Experimental Results and Analysis

7.3.1 Datasets

The two tabu search based hyper-heuristic framework were implemented and tested on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The system is tested with datasets used by a number of other examination timetabling researchers, as well as a new dataset we are introducing into the literature. The new dataset is from the MARA University of Technology (uitm-03) and was introduced in chapter 4 and chapter 5 (Kendall and Mohd Hussin, 2005b). Even though the actual problem is different from the benchmark datasets, we have formulated it in such a way that we can solve it in a similar way to the benchmark datasets. In the next chapter we will tackle the problem that is actually faced by the

university.

The benchmark datasets are as in table 5.1 and section 5.5 described some important characteristics of these datasets. For each dataset, (as described in section 5.6) we construct a conflict graph and create an initial solution using largest degree algorithm (Carter et al., 1996). There is no ‘backtracking’ used to schedule the unscheduled examinations. Therefore, 6 of the datasets (ear-f83, hec-s92, ute-s92, lse-f91, uta-s92 and yor-f83) have infeasible solutions where some examinations are not scheduled.

7.3.2 Tabu Search Based Hyper-heuristic Results Analysis

A comparison of results from three variations of the tabu search based hyper-heuristic with deterministic tabu duration (TD=0, etc) and random dynamic tabu duration (RTD) is shown in Table 7.1. A weighted proximity cost x_s is given whenever a student has to sit for two examinations scheduled s periods apart: these weights are $x_1 = 16$, $x_2 = 8$, $x_3 = 4$, $x_4 = 2$ and $x_5 = 1$. An examination i scheduled in slot k will strike a proximity cost for all students sitting examination i . The summation of proximity cost for all examinations divided by the total number of students gives the proximity cost (penalty cost) per student. A lower penalty cost implies that fewer students need to sit for examinations that are close to each other. This method of evaluation is used by many examination timetabling researchers making it possible to compare different solution methodologies applied to the examination timetabling datasets (see table 7.3 for a comparison between tabu search hyper-heuristic results and other solution methodologies).

The columns in table 7.1 shows penalty cost per student for the different fixed tabu duration (TD ranging from 0 to 4) and dynamic tabu duration (RTD) for the three hyper-heuristics: TSHH-B, TSHH-HC and TSHH-GD. The results from chapter 6 (TSHH-B) are shown here again so as to make a comprehensive

comparison between all the methods tested. The best result for each method is in bold while the best result from all three methods is printed in blue.

Both TSHH-B and TSHH-HC run for 4 hours each while TSHH-GD may run for more than 4 hours or less than 1 hour depending upon whether more improvement can be made and the size of dataset. For example, a large dataset such as pur-s93 (2,419 examinations) takes 7 hours 20 minutes while ute-s92 (184 examinations) takes 15 minutes to reach the best result.

For all methods, the results improve as we increase the TD, and after it reaches its best TD, the results deteriorate as the TD is increased further. This implies that the hyper-heuristics are able to manage the low-level heuristics such that results can be improved with the help of a tabu list.

Table 7.1: Comparing results between the three methods: TSHH-B, TSHH-HC and TSHH-GD

File	Basic Tabu Search based Hyper-heuristic (TSHH-B)						TS based Hyper-heuristic with Hill Climbing (TSHH_HC)						TS based hyper-heuristic with Great Deluge (TSHH-GD)					
	TD=0	TD=1	TD=2	TD=3	TD=4	RTD	TD=0	TD=1	TD=2	TD=3	TD=4	RTD	TD=0	TD=1	TD=2	TD=3	TD=4	RTD
Car-f92	5.83	6.05	4.67	5.35	6.05	4.61	5.94	5.72	4.56	5.07	5.92	4.70	5.87	5.88	4.45	4.48	4.40	4.97
Car-s91	6.88	6.83	5.37	6.31	6.91	5.43	6.78	6.88	5.14	6.04	7.10	5.31	6.85	7.01	5.15	4.93	5.01	5.60
Ear-f83	47.61	44.07	40.18	41.29	44.43	41.56	46.58	43.23	39.84	41.51	45.37	41.76	45.55	44.16	39.86	38.14	39.96	42.53
Hec-s92	13.94	13.07	11.86	12.31	13.64	11.97	13.61	13.07	11.63	12.31	11.77	10.74	14.32	12.10	12.19	11.86	11.38	12.01
Kfu-s93	19.27	17.69	15.84	16.13	17.45	15.56	18.47	16.35	15.60	16.42	17.84	15.55	19.18	16.94	15.98	17.11	16.94	16.65
Lse-f91	15.04	14.89	13.67	14.53	15.87	13.29	14.62	15.30	13.36	14.23	15.75	14.33	15.12	15.63	13.18	12.35	12.91	13.55
Pur-s93	6.13	6.07	6.06	7.31	8.50	5.67	6.19	6.09	5.96	7.59	8.48	5.72	6.26	6.17	5.41	5.49	5.53	5.58
Sta-f83	157.64	164.62	157.38	166.32	164.62	157.42	158.55	161.65	158.68	166.67	164.29	158.26	164.31	164.89	160.21	168.30	168.25	160.33
Tre-s92	10.48	10.99	8.39	9.46	10.16	9.21	10.89	10.50	8.72	9.67	9.99	8.66	11.21	10.57	9.03	8.35	8.50	9.34
Uta-s92	4.93	4.50	3.92	4.60	5.16	4.23	4.96	5.00	4.04	4.67	4.87	4.28	4.93	4.61	4.17	3.77	3.91	4.27
Ute-s92	31.36	27.59	27.60	27.28	29.83	27.34	32.01	27.58	27.55	27.12	29.68	27.66	30.61	27.94	28.00	29.31	28.13	28.28
Yor-f83	45.98	43.16	39.42	40.77	42.39	40.04	46.04	43.43	39.94	40.41	42.77	40.24	47.64	44.26	39.01	38.48	37.99	43.01
Uitm-03	2.16	1.55	1.93	3.95	6.37	1.65	2.08	1.55	1.94	3.84	6.33	1.63	2.14	1.44	1.37	1.36	1.44	1.40
Average	6.27	5.85	5.48	6.88	8.46	5.34	6.23	5.82	5.45	6.84	8.39	5.34	6.34	5.81	5.14	5.12	5.17	5.35

The last row in table 7.1 is an averaged penalty per student for all datasets. The averages for all fixed deterministic tabu duration and random dynamic tabu duration of TSHH-B and TSHH-HC are similar. Thus TSHH-B and TSHH-HC are as good as each other. TSHH-GD with TD of 2, 3 and 4 perform better over TSHH-B and TSHH-HC. When TD equals 0 and 1, all methods perform in a similar way. On average, random dynamic tabu duration for all methods also gives the same result. From this simple observation, it is interesting to note that these hyper-heuristics have almost the same averages across different deterministic tabu duration and random dynamic tabu duration. With the same parameters (dataset characteristics and tabu duration), TSHH-B and TSHH-HC algorithms seem to support the No Free Lunch Theorem (NFL) of Wolpert and Macready (Wolpert and Macready, 1995; Wolpert and Macready, 1997; see section 1.1). Even though the problem is of the same class, we can see a different behaviour by the different algorithms on different datasets. There is no algorithm that gets the best solution for all datasets. Nine of the datasets perform better with TSHH-GD, three datasets achieve good results from TSHH-HC while one dataset performs better with TSHH-B. Thus, we can also conclude that TSHH-GD seems superior over TSHH-B and TSHH-HC because most of the best results are by TSHH-GD. The time taken by TSHH-GD is also less than 4 hours with smaller datasets (except car-f92, pur-s93 and uitm-03)

7.3.3 UiTM-03 Performance Analysis

The MARA University of Technology real examination timetabling problem is different from the examination timetabling problem of the benchmark datasets. However, in this chapter, the MARA University of Technology (uitm-03 dataset) problem is solved using a proximity cost evaluation method, similar to the way we evaluate the benchmark datasets. Therefore, this dataset can be added to the benchmark datasets, and further research on this problem can include the uitm-03 dataset. The file format specification for uitm-03 dataset is similar to the benchmark dataset file format specification (refer Appendix B).

Table 7.2: Comparison of result for UiTM dataset with other solution methods

	TSHH-B	TSHH-HC	TSHH-GD	SA	GD	Manual
uitm-03	1.55	1.55	1.36	1.68	1.4	12.83

Table 7.2 compares the hyper-heuristic best result on the uitm-03 dataset with the great deluge algorithm, simulated annealing and the existing manual solution. The great deluge algorithm was run for a maximum of 20,000,000 iterations and the target penalty cost is 1.2. The simulated annealing algorithm uses a standard neighbourhood (Thompson and Dowsland, 1998). The performance across all methods is almost the same and much better than the manual solution. The manual solution is the timetable produced using manual methods at UiTM. Recall that a weight of 16 is given if a student has to sit for two examinations in sequence. Since the penalty cost per student in the manual solution is 12.83, we can generalise that on average each student will have at least two period gap between two examinations.

Here, we will discuss how the algorithm performs when applied to the uitm-03 dataset and compare its behaviour with one other dataset i.e. car-s91 (one of the benchmark dataset). Car-s91 (Carleton University, Ottawa) is one of the largest datasets with 682 examinations to be scheduled in 35 slots, 16,925 students and 56,877 student examinations. Referring to table 7.3, the best result for car-s91 is 4.50 published by Yong and Petrovic (2005). They used a combination of case base reasoning using a fuzzy similarity measure to choose sequential heuristic during the construction phase and the great deluge algorithm. The next best result of 4.65, published by Burke and Newall (2003) use a combination of adaptive initialisation strategies and the great deluge algorithm. Both methods are based upon the idea that good initial solutions fed to great deluge algorithm will produce good solutions. Table 7.1 illustrates that all tabu search based hyper-heuristic methods show similar behaviour across both datasets, where, the best result is obtained using a tabu search hyper-heuristic with great deluge and a tabu duration of 3. The best result for car-s91 and uitm-03 is 4.93 and 1.36 respectively. Thus, the hyper-heuristic is able to produce a good quality solution for the car-s91 dataset, even though the initial

solution is not as good as the initial solutions in Yong and Petrovic (2005) and Burke and Newall (2003).

Figure 7.4 and figure 7.5 show the best proximity cost, with various tabu durations obtained for uitm-03 and car-s91 datasets. In our tabu based hyper-heuristic strategy, we apply the concept of heuristics cooperating with each other rather than penalising a non-performing heuristic. When the tabu duration is greater than zero, we apply a tabu restriction where a heuristic will be tabu active if its solution value has been accepted to update the current solution. The heuristic will remain tabu active for a number of steps equal to tabu duration. We make a heuristic tabu because we want to direct the search to other possible heuristic search spaces. Eventually we may go back to a heuristic search space once it is no longer tabu active and can give the best solution amongst all tabu inactive heuristics. Similar to the tabu search meta-heuristic, we need to decide which tabu duration (or list size in tabu search) works best for a problem instance.

For the uitm-03 dataset (figure 7.4), as the tabu duration increases, solution quality improves, and once it reaches its best tabu duration, the solution quality begins to deteriorate as we increase the tabu duration further. It seems that the search by TSHH-B and TSHH-HC are more sensitive to the tabu duration value compared to the search by TSHH-GD.

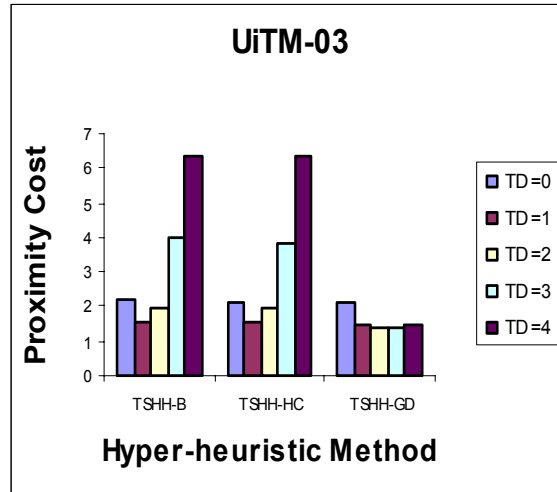


Figure 7.4: Hyper-heuristic methods on uitm-03 dataset

Car-s91 (Figure 7.5) shows only a slight difference where solution quality is better when tabu duration is zero compared to when tabu duration is one and improves again after that for both hyper-heuristics that either incorporates hill climbing or great deluge.

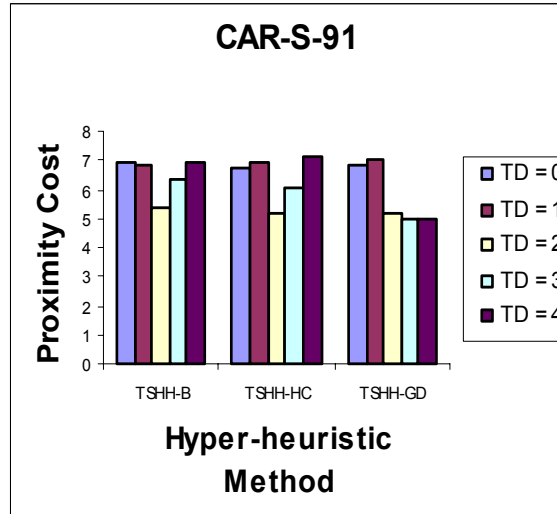


Figure 7.5: Hyper-heuristic methods on Car-f91 dataset

Figures 7.6 – 7.9 show the hyper-heuristic performance with different tabu

durations on the uitm-03 dataset. The graphs show how the hyper-heuristic explores the search space. Both TSHH-B and TSHH-HC show more dispersed points in the graph compared to TSHH-GD because the boundary penalty used in great deluge directs the search on a trajectory. The lower tabu duration does not show much movement in the search space and it might be trapped in local optima, and as we increase the tabu duration, it shows more movement and exploration of the search space and therefore is able to find a better solution. As the tabu duration increases it becomes more difficult to get better solutions since many heuristics stay tabu too long and the hyper-heuristic has no option but to take the best solution from the worst heuristics. TSHH-GD is better at not moving to a worse solution since it is also directed by the boundary penalty and thus will not make bad moves in such a way that it cannot get back to a good solution space.

Figure 7.9 shows a comparison of the TSHH-GD with the great deluge algorithm. We ran both algorithms for 4 hours and traced the penalty evaluation at every 5,000 iterations (steps). The TSHH-GD converges faster in terms of steps but it actually takes longer because each iteration explores various heuristic solution spaces and may take 10 times longer compared to the great deluge algorithm. The great deluge algorithm runs with a higher number of iterations because each iteration does only one random swap, while TSHH-GD needs to evaluate the performance from several low-level heuristics.

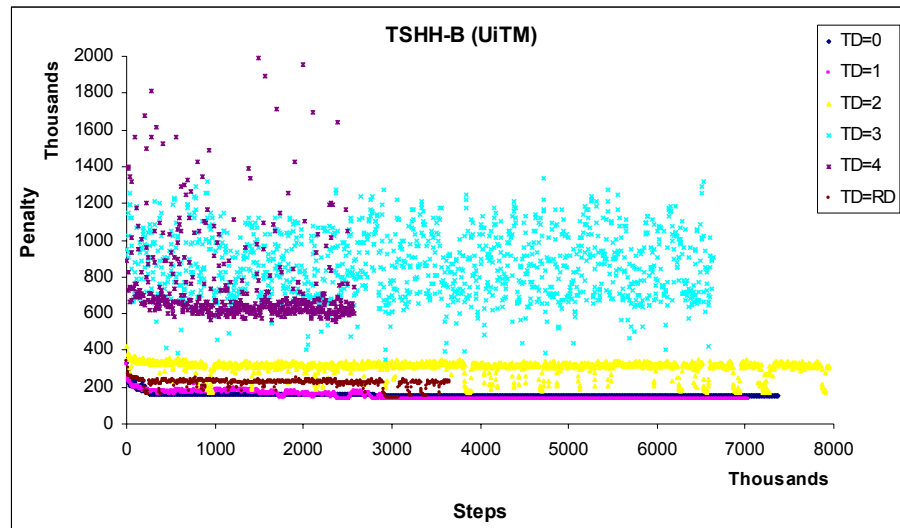


Figure 7.6: Simple Tabu Search Based Hyper-heuristic for UiTM dataset

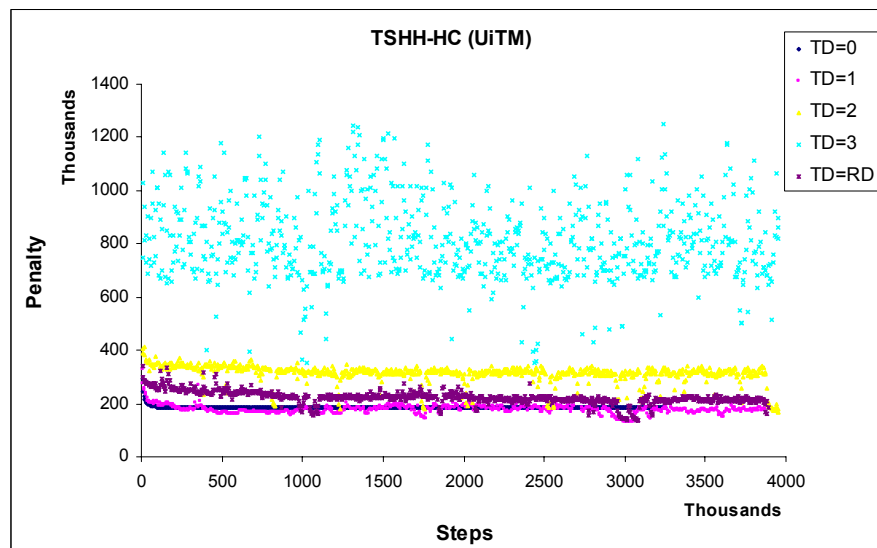


Figure 7.7: Tabu Search Based Hyper-heuristic with Hill Climbing for UiTM dataset

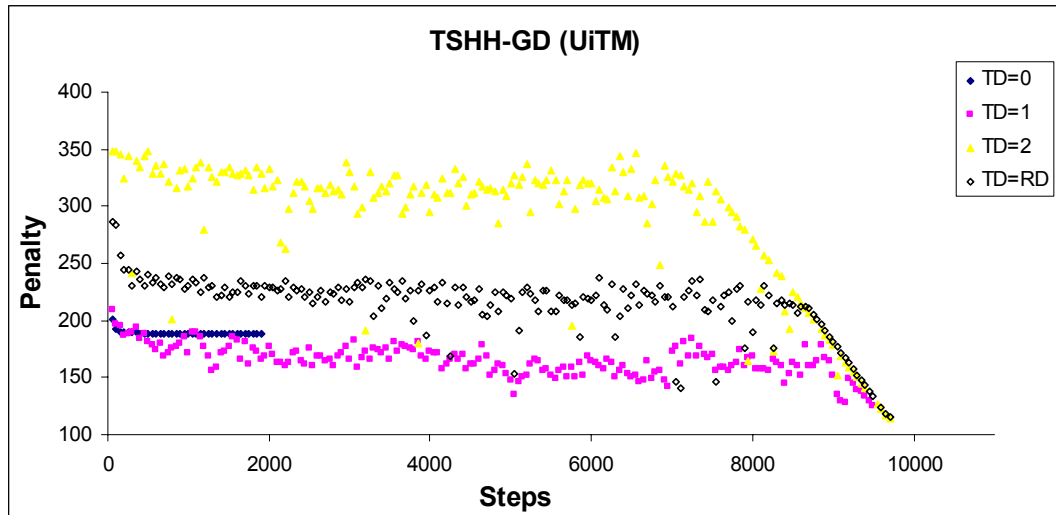


Figure 7.8: Tabu Search Based Hyper-heuristic with Great Deluge for UiTM dataset

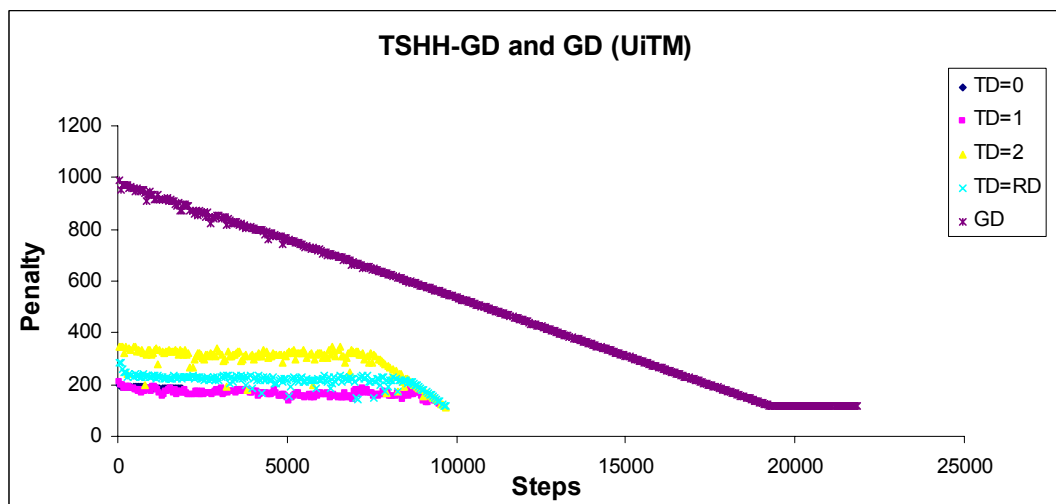


Figure 7.9: Comparing TSHH-GD and Great Deluge Algorithm performance

7.3.4 Tabu Search Based Hyper-heuristic Results Compared to Published Methods

Table 7.3 shows our best results from three hyper-heuristics and our best hyper-

heuristic with fixed TD compared to other published results for benchmark datasets. Our objective here is to show that a tabu search based hyper-heuristic is able to produce good quality and feasible solutions for examination timetabling problems even though they may not produce the best results. The results show that our generic method is able to produce good quality solutions over all datasets, whereas other approaches tend to perform well on some instances but not others. That is, we are presenting a robust algorithm which can be utilised to give good quality solutions across a range of datasets. It also has the potential of being applied to other problems as well.

The first column of results in table 7.3 are results from TSHH-GD with $TD = 3$. This is the method that has the best averaged penalty per student across all dataset and produces the best result for 5 benchmark datasets. The second column of results shows our best results from across the three hyper-heuristics. The next set of results are the results first published on the dataset by Carter et al. (1996). Our best results are better than Carter et al. (1996) in six cases. They used a set of graph colouring heuristics with backtracking (rescheduling examinations until all examinations are scheduled). The next three sets of results are based on tabu search. Di Gaspero and Schaerf (2001) applied tabu search with shifting penalty mechanism, varying tabu list size and two constraint violation lists. Di Gaspero (2002) applied a combination of tabu search with different neighbourhoods (union and composition). He categorised these combinations into local search that specialised in optimising objective function (*recolour*), perturbing current solution (*shake*) or obtaining more improvement (*kick*). The *recolour* and *shake* algorithms were applied in sequence until no further improvement and the algorithm ended with the *kick*. White et al. (2004) applied tabu search with short-term and long term-memory and a controlled tabu relaxation technique (emptying entries in tabu list) to diversify the search. Our method has added a significant improvement over tabu search approaches since our method shows the best results for all datasets. We also compare our results with results from other methods such as: a good initial solution with hill climbing, simulated annealing and great deluge by Burke and Newall (2003); great deluge

algorithm with modified acceptance criteria by Burke et al. (2004a); combination of case base reasoning using a fuzzy similarity measure to choose sequential heuristic during the construction phase and the great deluge algorithm by Yong and Petrovic (2005) and hybrid of constraint programming, simulated annealing and hill climbing with Kempe chain neighbourhood by Merlot et al. (2003). One of our results; hec-s92 is better than the other algorithms based on great deluge. Compared to Merlot et al. (2003), we manage to get two better results for car-s91 and tre-s92.

Table 7.3: Comparing our best results and published results

File	TSHH-GD TD=3	TSHH	Carter et al. (1996)	Di Gaspero, Schaerf (2001)	Di Gaspero (2002)	White et al. (2004)	Burke & Newall (2003)	Burke et al. (2004a)	Yong & Petrovic (2005)	Merlot et al. (2003)
Car-f92	4.48	4.40	6.2 – 7.6	5.2	-	4.63	4.1	4.2	3.93	4.3
Car-s91	4.93	4.93	7.1 – 7.9	6.2	5.7	5.73	4.65	4.8	4.50	5.1
Ear-f83	38.14	38.14	36.4 - 46.5	45.7	39.4	45.8	37.05	35.4	33.71	35.1
Hec-s92	11.86	10.74	10.8 - 15.9	12.4	10.9	12.9	11.54	10.8	10.83	10.6
Kfu-s93	17.11	15.55	14.0 - 20.8	18.0	-	17.1	13.9	13.7	13.82	13.5
lLe-f91	12.35	12.35	10.5 - 13.1	15.5	12.6	14.7	10.82	10.4	10.35	10.5
Pur-s93	5.49	5.41	3.9 – 5.0	-	-	-	-	4.8	-	-
Sta-f83	168.3	157.38	161.5 - 165.7	160.8	157.4	158	168.73	159.1	151.52	157.3
Tre-s92	8.35	8.35	9.6 – 11.0	10.0	-	8.94	8.35	8.3	7.92	8.4
Uta-s92	3.77	3.77	3.5 - 4.5	4.2	4.1	4.44	3.2	3.4	3.14	3.5
Ute-s92	29.31	27.12	25.8 - 38.3	29.0	-	29.0	25.83	25.7	25.39	25.1
Yor-f83	38.48	37.99	41.7 - 49.9	41.0	39.7	42.3	37.28	36.7	36.53	37.4
Average	8.09	7.83	7.93	10.03	9.19	9.72	8.36	7.32	7.99	8.37

Not many approaches are tested with the largest dataset, pur-s93. Even though we could not beat the only other three approaches that are tested with the dataset (see Table 2.1 for a more comprehensive results), our method is already proven to work

across all problem instances. The second last column of Table 2.1 is an average for each method across all instances tested. In general, we have actually produced good solutions over all problem instances since our best results average is only third out of 17 solution approaches (shown in Table 2.1). The best algorithm is by Burke et al. (2004a); great deluge algorithm with good initial solutions and the next best algorithm is by Caramia et al. (2001); greedy constructive heuristic with an optimiser.

7.4 Conclusions

This chapter has presented a study on how a tabu list can help a hyper-heuristic decide which low-level heuristic to call next. The selected heuristics compete to produce a good solution for the current state of the problem. The hyper-heuristic is able to manage and control its low-level heuristics without having domain knowledge of the problem it is solving. The tabu list does not contain operators or moves that are problem specific. It only needs to store important attributes about a heuristic that is required by the hyper-heuristic. The tabu search based hyper-heuristic uses various heuristics thus allowing the search to diversify and explore more of the heuristic search space. It also implicitly explores through infeasible regions (by un-scheduling and rescheduling examinations) of the search space by accepting an infeasible solution without updating the current best solution. In examination timetabling problems, some examinations may be unmovable from their current slot unless other conflicting examinations are rescheduled.

The solution quality from different fixed deterministic tabu duration is sensitive to the tabu duration value indicating that using a correct TD will enhance solution quality. A dynamic mechanism of determining the tabu duration also shows promising results. Currently, the tabu duration is randomly determined and future research will include some intelligence in determining a good combination and sequence of tabu duration value to be used at each decision point.

The tabu search based hyper-heuristic with great deluge performs better compared to the basic tabu search based hyper-heuristic and tabu search based hyper-heuristic with hill climbing. The great deluge acceptance criterion has more control over how much a worsening solution is accepted. Its boundary level directs the search through a trajectory that is closer to the good quality solutions.

The results from experiments, on benchmark datasets, show that a tabu search based hyper-heuristic method is superior over other methods based on the classical tabu search. On average it produces very good results compared to other methods discussed in this chapter. Table 2.1 shows comparison with other methodologies not described in this chapter and results from tabu search hyper-heuristic is as competitive as other methods.

Tabu search based hyper-heuristic generic solution methodology can easily be applied to other problems by just changing the low-level heuristics and the evaluation function while the search method remains the same. In the next chapter, the same hyper-heuristic is applied on a new and large examination timetabling problem with a different objective function and a modified set of low-level heuristics.

Chapter 8

A Tabu Search Based Hyper-heuristic Approach to Examination Timetabling Problem at the MARARA University of Technology

This chapter focuses on the examination timetabling problem faced by the MARARA University of Technology (UiTM), Malaysia. This problem was first introduced in chapter 4 and its mathematical model was presented in chapter 5. The existing tabu search based hyper heuristic presented in chapter 7 was applied on the UiTM real world data but only considered proximity costs. In this chapter, we now introduce a new hard constraint and an additional soft constraint (weekend costs which is very important for this institution). Since this is a new dataset and no solutions have been published in literature, we can only compare the results with an existing manual solution. Section 8.1 gives an overview of UiTM problem characteristics and its complexity. Section 8.2 provides a general description of tabu search based hyper-heuristic and its enhancement to suit UiTM problem specification. Section 8.3 shows how the algorithm works with this new large dataset and how it compares with two initial solutions from two different constructive heuristics. It also compares the effect of dynamic unscheduled examination weights and fixed unscheduled examination weights on the tabu duration value for the benchmark dataset and UiTM dataset. Section 8.4 concludes with a summary and future research directions.

8.1 UiTM Examination Timetable Problem Description

In this thesis, a new timetabling dataset from the MARA University of Technology (UiTM) is introduced and its basic examination timetabling problem (assign examinations to slots) is solved using a hyper-heuristic framework that was shown to produce comparable results for well known benchmark datasets (see Chapter 6 and Chapter 7). A detailed description of the problem is outlined in Section 5.2 and in (Cowling et al., 2002; Kendall and Mohd Hussin, 2005b).

Apart from the constraints that are common for examination timetabling (Carter 1986; Carter and Laporte 1996; Schaerf 1999), UiTM has to consider the following additional hard constraint: If an examination falls on a state public holiday and there are students from that state sitting for the examination then the examination must be moved to another slot. Malaysia has a number of public holidays that are not shared between states. Another hard constraint that need to be included is that some courses need to be held concurrently. An additional soft constraint (weekend constraint) is also included in the problem. UiTM has to schedule examinations on a weekend because different states have different weekend slots. It is quite impossible to remove all the weekend slots because it will be difficult to schedule all examinations into the remaining available slots.

Data on available space for examinations is not considered when moving examinations into slots and therefore every faculty, centre and branch campus needs to give prompt feedback on space availability for examinations already scheduled. Once the examination timetable is in its final draft, it is sent to all faculties and branch campuses for the assignment of rooms and invigilators.

Table 8.1 shows the detailed data characteristics on the number of students, number of student examinations and average examination per student for each of the branch campuses. The last row shows the total for all campuses. The examination data was obtained from the Centre for Integrated Information System (CIIS), MARA

University of Technology. Each faculty and campus update student information through a wide area campus network. The total number of students enrolled in Semester 2, 2002/2003 was 84,675, course enrolment was 472,201 and total number of courses to be scheduled was 2,650. The data supplied by CIIS was in the form of, a list of examinations that must be scheduled, a list of examinations that must be scheduled at the same time (concurrent), a list of students and their course selection (categorised by campuses) and an examination timetable that was used in the May 2003 semester. There is no information regarding capacity or other hard constraints that were imposed via feedback by faculties or campuses.

Table 8.1: Characteristics of UiTM dataset categorised by campuses.

No.	Branch Campuses	No. of Students	No. of Student Exams	Avg Exam Per Student
1.	Shah Alam	40,275	222,559	5.52
2.	Melaka	4,447	25,334	5.70
3.	Negeri Sembilan	315	1,913	6.07
4.	Johor	4,057	22,109	5.45
5.	Perak	5,366	31,518	5.87
6.	Perlis	5,824	32,807	5.63
7.	Kelantan	3,515	19,627	5.58
8.	Terengganu	4,842	27,444	5.67
9.	Pahang	4,607	27,221	5.91
10.	Sarawak	4,800	25,618	5.34
11.	Sabah	2,423	11,470	4.73
12.	Penang	1,453	9,296	6.40
13.	Kedah	2,751	15,285	5.56
	uitm-03 (after pre-processing)	84,675	357,761	4.23

The original list of examinations has 2,650 examinations to be scheduled, but after computing the enrolment for each examination using the student file, examinations with zero enrolments are removed even though these examinations are

present in the examination timetable. Examinations with zero enrolments will have no effect on the way we compute solution quality and therefore can be removed from the timetable. There are 491 examinations with zero enrolment and we can only schedule the remaining 2,159 examinations. Data accuracy is one of the difficulties in processing data from a large university that involves many faculties and campuses. We believe that these examinations have zero enrolment because officers in faculties and campuses may not have the updated information of exact course enrolment and student add or drop courses at the last moment. This should be resolved once the integrated student information system is completed (see section 4.4).

The basic examination timetabling problem that we want to solve is to assign examinations to n number of slots subject to various constraints, so as to minimise various costs. The total number of slots is already fixed and the examination scheduler must schedule examinations into at most n slots. There are 2,159 examinations with some examinations being held concurrently. Further pre-processing was done for all examinations that need to be scheduled at the same time (concurrently). These were grouped together and replaced by one course code. This reduces the number of examinations that need to be scheduled to 2,063. The number of course enrolments is also reduced to 357,761 after removing enrolments in courses that are not in the list of examinations to be scheduled. Details of files needed to process UiTM dataset are listed in Appendix B. Table 5.1 shows various features of the uitm-03 dataset in comparison to the other benchmark datasets.

The examination dates for uitm-03 dataset were from 20th April 2003 to 10th May 2003 with 1st May as a public holiday across all campuses, so no examinations can be scheduled on this day. For this dataset, we are not concerned with other public holidays since none occur on different days for different states. On other occasions there might be a situation where a campus has a public holiday and others do not. Therefore, excluding the 1st May public holiday, the total number of examination days is 20 and each day will have 2 slots (morning and afternoon), i.e. there are 40 slots in which to assign examinations. Even though each examination

may have a different duration (120, 150 or 180 minutes), we will treat each examination as occupying a complete slot of 180 minutes. Each campus is categorised as of type A or type B to indicate which days or slots must be avoided (or minimised) in assigning examinations taken by students in that particular campus. Campuses in category A has weekend: half-day on Saturday and full-day on Sunday while campuses in category B has weekend: half-day on Thursday and full-day on Friday.

8.1.1 Constraints

The following hard constraints are enforced in the UiTM dataset;

- a. **Student conflict:** No student should sit for more than one examination in the same slot.
- b. **Examination conflict:** All examinations must be scheduled in a valid time slot.
- c. **Co-schedule examination conflict:** Examinations that must be scheduled together must be assigned in the same time slot.
- d. All examinations must be scheduled.

Currently, the examination timetable produced by the UiTM examination unit, violates one of the hard constraint i.e. student conflict. Rather than moving the examinations, the students involved are quarantined and are required to sit for the second examination in the next time slot. This increases pressure on the students and also involves other resources such as invigilators and rooms.

8.1.2 Costs

The quality or cost of a timetable is measured by how many soft constraints are not satisfied. It is very difficult to include all the side constraints in evaluating a timetable because data collected might not be enough to enforce a particular constraint or it is difficult to measure the importance of enforcing a particular constraint. For example, scheduling examinations in a specific slot requested by faculty may contradict request(s) from another faculty.

Therefore, additional soft constraints, specific to university requirements, can be added to this problem if we have enough data. Unfortunately adding the constraints will result in a more rugged search space and will make it more difficult to find an optimal, good quality or even a feasible solution. To simplify the problem and make it comparable with other instances of the timetabling problem, proximity constraints are included. In addition, a weekend constraint is added since it is a very important cost that has to be minimised for this particular dataset.

We determine the cost of an examination timetable solution based on the penalty given if certain soft constraints are violated. The soft constraints that we consider are as follows:

- a. **Proximity cost:** A proximity cost x_s is given whenever a student has to sit for two examinations scheduled s periods apart: these weights are $x_1 = 16$, $x_2 = 8$, $x_3 = 4$, $x_4 = 2$ and $x_5 = 1$. This is exactly the same cost we minimised in chapter 6 for all the other benchmarks datasets and for the UiTM dataset.
- b. **Weekend cost:** The current examination timetable scheduler schedules examinations during the weekend. We can try to produce a better quality timetable by penalising examinations that are scheduled during the weekend. A penalty cost of 16 is given whenever a student has to sit for a

weekend examination. This is the extra term that we need to minimise for the UiTM dataset which is not applicable to the benchmark datasets in previous chapters.

Finally, our objective is to optimise the following: i.e. minimise sum of the total proximity cost and the weekend cost.

8.2 The Tabu Search Based Hyper-heuristic

Three tabu search based hyper-heuristic methods have already been implemented and tested on the benchmark datasets (see chapter 6 and chapter 7). The results were found to be competitive with other published results. The methods were also tested on the uitm-03 dataset. In chapter 7, the UiTM problem was solved in a similar way to the benchmark datasets so that other methods published that have been applied on the benchmark datasets can also be applied on the uitm-03 dataset.

In this chapter, we will investigate the same approaches as in chapter 7 but we are now solving the real problem that includes weekend constraints.

8.2.1 Constructive Heuristic

Section 5.2 described the constructive heuristics used to generate an initial solution that is to be used in the hyper-heuristic framework. The constructive heuristic creates an examination timetable by sequentially selecting an examination from a list of exams to be scheduled and assigns the examination to an available slot without violating a hard constraint. Two different constructive heuristics are used to generate initial solutions for the uitm-03 dataset. First, the same constructive heuristic used to generate an initial solution for the benchmark datasets, i.e., select an examination e_i using the largest degree criteria (largest number of examinations in conflict) and assign it to the first feasible slot. The slot is considered to be feasible if no other examinations in conflict with e_i are scheduled in the same slot. The second

constructive heuristic differs in how it selects the first feasible slot. It selects a feasible slot in the following order: a non-weekend slot, a weekend slot with lesser number of students, and the remaining slots. For both constructive heuristics, if no feasible slot is available for an examination, it will be added to a list of examinations unscheduled and an appropriate penalty cost is assigned.

8.2.2 Hyper-heuristic Module

The following hyper-heuristics were implemented (in chapter 6 and chapter 7) and applied to the UiTM dataset.

- § The simplest form, i.e., Basic Tabu Search Based Hyper-heuristic (TSHH-B), where we consider all tabu inactive heuristics and apply the heuristic that has the best improvement only. The algorithm iterates for a fixed time or until there is no further improvement for a given number of heuristic calls (see TSHH-B algorithm in figure 6.3).
- § The second hyper-heuristic is Tabu Search Based Hyper-heuristic with Hill Climbing (TSHH-HC), which adds to TSHH-B a successive call to the best performing heuristic until no further improvement is made (see TSHH-HC algorithm in figure 7.3).
- § The third hyper-heuristic is Tabu Search Based Hyper-heuristic with Great Deluge (TSHH-GD), which updates a solution within a certain boundary only (see TSHH-GD algorithm in figure 7.4).

For each of these hyper-heuristics, we apply both deterministic tabu duration and random dynamic tabu duration.

8.2.3 Low-level Heuristics Module

The low-level heuristics module contains a set of low-level heuristics (that

converts a solution from an initial state to a new state) and an evaluation function (that evaluates the current solution state) for the current problem being solved. For this problem (MARA University of Technology examination timetabling problem), the problem characteristics and the number of constraints to be minimised differ from the benchmark datasets.

The same low-level heuristics as described in chapter 6, chapter 7 and in Kendall and Mohd Hussin (2005a; 2005c) can be applied to the current problem. In addition, another low-level heuristic is added, i.e., move examination with highest weekend conflict. This method allows the hyper-heuristic to select a heuristic that improves the weekend conflict when it arises without being aware that it is doing so. In general, we can easily add the necessary low-level heuristics specific to a particular instance of a problem with additional soft constraints that need to be minimised. In the real problem, we also include both proximity cost and weekend cost in the evaluation function. Therefore, we modify the objective function such that it will also compute the weekend cost (see objective function in section 8.3.1).

8.3 Experimental Results

The tabu search based hyper-heuristic framework was implemented on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The program was coded in C++ using an object-oriented approach. In Kendall and Mohd Hussin, (2005a) and chapter 5, the performance of the simplest form of tabu search based hyper-heuristic module TSHH-B with deterministic tabu duration and random dynamic tabu duration was investigated by testing it on Carter's examination timetabling benchmark data. Chapter 6 extended the work with an implementation of two additional variations of the hyper-heuristic. The three methods were also tested on the uitm-03 dataset that is formulated in a similar way to the benchmark datasets. In all the methods, we are able to find good solutions for all datasets. The tabu search based hyper-heuristic method has also added a significant improvement to tabu search because the results are better in all cases compared to the tabu search

approach by Di Gaspero and Schaerf (2001). The same hyper-heuristic framework with slight modifications to the low-level heuristics, evaluation function and weighted cost, is tested with uitm-03 dataset, and experimental results (with respect to minimising both proximity and weekend cost) are presented in this section.

8.3.1 Proximity and Weekend Cost Evaluation Method

The UiTM dataset has an additional constraint or criterion, i.e., a weekend cost that needs to be minimised. The proximity cost and weekend cost are combined in one objective function with the same weights for proximity cost and a weight of 16 for every student who has to sit for a weekend examination.

The high level part of the hyper-heuristic does not need to be changed. The low-level heuristics that require more knowledge about the problem, need to be adjusted. The cost evaluation function is also modified to reflect a change in the objective function. The unscheduled examination weight needs adjustment (new weight of 10,000) and a new weekend weight added. In the previous implementation, two heuristics minimise the second and third order conflict (a conflict by a student having to sit for examinations, 1 or 2 slots apart). In the current implementation, a new heuristic, reduces a conflict by a student who has to sit for a weekend examination.

In general, we can further improve our low-level heuristics by generalising the number of conflicts and the type of conflicts we want to minimise. A separate low-level heuristic can be used to minimise a specific criteria and the hyper-heuristic can choose which criteria to be optimised in each decision step. The hyper-heuristic approach does this implicitly because it has the intelligence to choose between the low-level heuristic that minimises the first or the second criteria. The only parameter that we need to adjust would be the most suitable weights associated with the conflicts or criteria.

In the UiTM problem, it is important to associate appropriate weights to the

weekend conflict violation and the unscheduled examination. Currently the low-level heuristics are able to optimise the second and third order conflict, and weekend conflict. It can easily be extended to optimise a number of criteria (e.g. minimise usage of smaller rooms). In deciding what is an appropriate weight, we need to consider the importance of a criterion and how it will influence the search direction. A high weekend weight will push the search into a higher proximity cost since the number of examination slots will be reduced. Experiments were conducted with weights of 32 and 16 associated with a weekend conflict. Finally, a weight of 16 is chosen because a weight of 32 was too high and caused the search to become trapped and not able to improve the solution. The weekend weight is also the same (and as important) as the weight for students who have examinations in adjacent period.

The unscheduled examination weight of 5,000 that is being used with benchmark datasets is also not suitable with the UiTM dataset. Normally the unscheduled weight determines whether an examination is better being unscheduled, or scheduled and causing high proximity cost. A 5,000 unscheduled examination weight is not suitable because the total number of students is too large and this will direct the search into an infeasible region since the *remove examination* heuristic will outperform other heuristics by un-scheduling an examination. In the UiTM problem, the unscheduled examination weight is fixed at 10,000.

Therefore, for the UiTM dataset, the weighted penalty function that we are minimising is:

$$\frac{\sum_{i=1}^m \sum_{k=1}^n \sum_{j=1}^m T_{ik} x_{|k-l|} C_{ij}}{2} + \sum_{i=1}^m \sum_{k=1}^n T_{ik} A_k E_i + \sum_{i=1}^m \sum_{k=1}^n T_{ik} B_k F_i + 10000 \sum_{i=1}^m U_i \quad (8.1)$$

where:

m = number of examinations

n = number of slots

T_{mn} = final examination timetable, if $T_{ij} = 1$, exam i is scheduled in slot j .

C_{mm} = conflict matrix, C_{ij} = number of students sitting both examination i and examination j .

$x_{|k-l|} = 2^{(5-|k-l|)}$ for $0 < |k-l| \leq 5$

U_m = list of unscheduled examinations, $U_i = 1$ if unscheduled, 0 otherwise.

A_n and B_n = list of slots for weekend type A and weekend type B, 1 if it is a weekend and, 0 otherwise.

E_m and F_m = list of enrolments, i.e. E_i = number of students with weekend type A sitting for examination i .

8.3.2 Results Analysis

Table 8.2 shows the same hyper-heuristic being applied but with a different objective function (i.e. (8.1)). The first constructive heuristic described in section 8.2.1 is used to construct the initial solution. The initial solution is poor with respect to weekend conflict since all slots are treated as equal. The unscheduled weight is fixed at 10,000 and a weight of 16 per student who has to sit for a weekend examination. The results should be higher than the results in chapter 7 (see table 7.2) since we are also taking into account the penalty for weekend examinations. On average, we could say that students may not have to sit for weekend examinations or sit for an adjacent examination since the best solution (cost of 7.12 by TSHH-GD

with tabu duration = 3) is less than the weight for a weekend examination or adjacent examinations. However, this is not true since we do need to look at the overall timetable and count the number of students who have to sit for adjacent examinations or weekend examinations.

Table 8.2: Uitm-03 dataset (minimised proximity and weekend cost)

	UiTM-03		
Tabu Duration	TSHH-B	TSHH-HC	TSHH-GD
0	10.10	9.29	12.12
1	9.27	11.20	11.46
2	9.87	9.70	10.04
3	17.36	17.31	7.12
4	16.42	17.16	7.92
Random	9.51	9.65	8.04

Table 8.3 and table 8.4 show the number of students and percentage of students involved in a conflict that occur in two of the solutions produced. Table 8.3 is an analysis of the best result by tabu search based hyper-heuristic with great deluge (TD = 3). Table 8.4 is an analysis of the best result by tabu search based hyper-heuristic with hill climbing (TD = 1). The columns show each of the constraints that we are minimising. The gap number indicates a gap between two examinations that a student is sitting. The results seem consistent with the objective of minimising proximity and weekend cost. The best solution has lower cost in all of the individual constraints that we wish to minimise.

The weekend cost contributes a high percentage to the total cost in both solutions (last row in table 8.3 – 8.4). Therefore, it is beneficial to look at how the initial solution and the unscheduled examination weight affect the hyper-heuristic performance.

Table 8.3: Data characteristic of solution with 7.12 proximity and weekend cost (by TSHH-GD with TD = 3).

	Gap=1	Gap=2	Gap=3	Gap=4	Gap=5	Weekend
Total cost	88,464	86,536	59,440	45,104	29,488	294,208
Average (Cost per student)	1.04	1.02	0.70	0.53	0.35	3.47
Number of Students	5,529	10,817	14,860	22,552	29,488	18,388
% of total students	6.53%	12.77%	17.55%	26.63%	34.82%	21.72%
% of cost	14.66%	14.35%	9.85%	7.48%	4.89%	48.77%

Table 8.4: Data characteristic of solution with 9.27 proximity and weekend cost (by TSHH-B with TD = 1).

	Gap=1	Gap=2	Gap=3	Gap=4	Gap=5	Weekend
Total cost	172,688	115,512	69,344	53,930	30,579	343,296
Average (Cost per student)	2.04	1.36	0.82	0.64	0.36	4.05
Number of Students	10,793	14,439	17,336	26,965	30,579	21,456
% of total students	12.75%	17.05%	20.47%	31.85%	36.11%	25.34%
% of cost	21.99%	14.71%	8.83%	6.87%	3.89%	43.71%

8.3.3 New Initial Solution and Dynamic Unscheduled Examination Weight

Previously, the constructive heuristic used to generate the initial solution for the uitm-03 dataset is the same as the constructive heuristic for the benchmark datasets. With that initial solution, the hyper-heuristic is able to produce a feasible solution that is much better when compared to the existing manual solution. The constructive heuristic for benchmark datasets does not need to differentiate between weekend and non-weekend slots, and therefore does not prioritise which slots to assign an examination to. The results for uitm-03 dataset in table 8.3 and table 8.4 show that the weekend cost contributes almost half to the total cost. Therefore, it is well worth

considering prioritising non-weekend slots in the constructive heuristic, i.e. we need to inject more knowledge about the problem domain so as to produce an initial solution that starts from a reasonable point in the search space that does not violate one of the important constraints that we want to minimise. The only modification in the constructive heuristic is in determining which is the first available slot for an examination. The new constructive algorithm will only consider weekend slots if non-weekend slots are unavailable. During the search, the algorithm will also use the same criteria when moving an examination. This method should direct the search along a trajectory that avoids weekend slots. A description of this new constructive heuristic is described in section 5.6 and illustrated in figure 5.7.

Another issue that needs to be addressed is whether the unscheduled weight of 10,000 is suitable for the uitm-03 dataset. In deciding which low-level heuristic to select, the hyper-heuristic uses two important factors, i.e. cost of the objective function and non-tabu low-level heuristics. If the remove examination heuristic un-schedules an examination with a proximity cost higher than the unscheduled weight, it would seem advantageous to un-schedule the examination. The heuristics that reschedule examinations might find it difficult to improve the objective function since the new proximity cost may be greater than the un-schedule cost. This will eventually cause some examinations not being scheduled at the end of the search. The unscheduled weight does play an important role in deciding whether we need to un-schedule some examinations so that other conflicting examination can be scheduled into the vacated slot. Therefore, instead of using a fixed unscheduled weight, we can use a dynamic un-scheduled weight that is dependent upon the number of students. Thus, in this implementation, the cost of un-scheduling an examination differs for each examination, i.e., the cost of un-scheduling an examination e_i is calculated as, 60 multiplied by the number of students enrolled in examination e_i or 5,000 (value used for benchmark datasets), whichever is the maximum.

Table 8.5: Results with two different initial solutions and two unscheduled weight mode.

	TD	Dynamic Weight		Fixed Weight	
		Old Soln (1 hr run)	New Soln (1 hr run)	Old Soln (4hrs run) – same as table 7.2	New Soln (1 hr run)
Basic tabu search based hyper-heuristic (TSHH-B)	0	6.98	4.40	10.10	18.43
	1	7.67	4.91	9.27	20.60
	2	10.23	6.64	9.88	7.16
	3	16.99	8.73	17.36	8.67
	4	16.75	15.11	16.42	15.43
Tabu search based hyper-heuristic with hill climbing (TSHH-HC)	0	6.77	4.25	9.29	16.64
	1	7.70	6.01	11.20	16.88
	2	10.12	5.93	9.70	6.94
	3	16.91	8.95	17.31	9.94
	4	16.30	14.74	17.16	16.09

Table 8.5 shows a comparison of the results obtained using two different initial solutions and fixed and dynamic unscheduled weights. The old solution indicates an initial solution used in previous experiment described in section 8.3.1 and table 8.2. The new solution is produced using the new constructive heuristic. The last 4 columns in table 8.5 show the results produced by TSHH-B and TSHH-HC with fixed deterministic tabu duration between 0 and 4. Each run takes one hour. Since we already have results from previous experiment (table 8.2) of a 4-hour run with fixed unscheduled weight and old initial solution, we did not run it again for one hour. Moreover, the one-hour run with new parameters has already improved the results compared to the 4-hour run. The TSHH-GD is not tested with this parameter because we want to observe how the dynamic unscheduled weight and new initial solution affect the final results and it should be sufficient to test the first two methods only.

From table 8.5, we can conclude that with the new solution, both methods are able to produce better solutions compared to the best method (TSHH-GD) that was initialised with the old solution (table 8.2). A dynamic unscheduled weight also assists the hyper-heuristic in deciding which low-level heuristic is in favour. Since the best solution is when TD equals zero, this implies that the hyper-heuristic does not even need a tabu list to help manage the low-level heuristics.

Tables 8.6 – 8.8 present a detailed analysis of the best solution with different parameters. With the new initial solution that avoids weekend slots, the hyper-heuristic is able to improve the weekend conflicts, implying that with more knowledge about the problem domain injected into the initial solution construction phase and the low-level heuristics, the hyper-heuristic can perform better. With this good initial solution and fixed unscheduled weight, we can also get a reasonable result compared to solutions with dynamic unscheduled weight. As a general method and without the extra knowledge, the hyper-heuristic can also produce competitive result (compared to a manual solution) for the UiTM dataset specification (as shown in section 7.3.3 and table 7.3).

Table 8.6: Data characteristic of solution with 4.40 proximity and weekend cost. Best solution for TSHH-B with new initial solution and dynamic weight

	Gap=1	Gap=2	Gap=3	Gap=4	Gap=5	Weekend
Total cost	80,672	73,984	58,820	50,476	30,881	78,064
Average (Cost per student)	0.95	0.87	0.69	0.60	0.36	0.92
Number of Students	5,042	9,248	14,705	25,238	30,881	4,879
% of total students	5.95%	10.92%	17.37%	29.81%	36.47%	5.76%
% of cost	21.63%	19.84%	15.77%	13.54%	8.28%	20.93%

Table 8.7: Data characteristic of solution with 6.98 proximity and weekend cost. Best solution for TSHH-B with old initial solution and dynamic weight

	Gap=1	Gap=2	Gap=3	Gap=4	Gap=5	Weekend
Total cost	91,360	89,440	55,808	47,600	29,472	277,440
Average (Cost per student)	1.08	1.06	0.66	0.56	0.35	3.28
Number of Students	5,710	11,180	13,952	23,800	29,472	17,340
% of total students	6.74%	13.20%	16.48%	28.11%	34.81%	20.48%
% of cost	15.46%	15.13%	9.44%	8.05%	4.99%	46.93%

Table 8.8: Data characteristic of solution with 6.94 proximity and weekend cost. Best solution for TSHH-HC with new initial solution and fixed weight

	Gap=1	Gap=2	Gap=3	Gap=4	Gap=5	Weekend
Total cost	135,360	92,664	89,080	58,090	32,723	89,768
Average (Cost per student)	1.60	1.09	1.05	0.69	0.39	1.06
Number of Students	8,460	11,583	22,270	29,045	32,723	5,611
% of total students	9.99%	13.68%	26.30%	34.30%	38.65%	6.63%
% of cost	27.20%	18.62%	17.90%	11.67%	6.58%	18.04%

8.3.4 Benchmark Datasets with Dynamic and Fixed Unscheduled Weights

From section 8.3.3, we observe that a dynamic unscheduled examination weight has an effect on the tabu search based hyper-heuristic methods. Therefore, we extend the experiments to include a test on the benchmark datasets. Table 8.9 shows the results of a one-hour run of basic tabu search based hyper-heuristic (TSHH-B) and tabu search based hyper-heuristic with hill climbing (TSHH-HC) with a dynamic unscheduled examination weights. Table 8.10 shows the results of a one-hour run of basic tabu search based hyper-heuristic (TSHH-B) and tabu search based hyper-

heuristic with hill climbing (TSHH-HC) with fixed unscheduled examination weights. In previous implementations (see chapter 6 and chapter 7), the unscheduled examination weight is fixed at 5,000. If an examination is not scheduled in a slot, a penalty of 5,000 is given. With dynamic unscheduled examination weight, if an examination is unscheduled, a maximum penalty of either 5,000 or 60 times the number of students sitting for the examination is given. In both tables 8.9 and 8.10, the results of the best tabu duration value for each dataset using each method is printed in bold. There is not much difference across all the datasets for both methods with respect to the best tabu duration where best results are obtainable when tabu duration value is 2. The last row in both tables is the average penalty across all datasets with the best average result printed in red. TSHH-B performs a little better with dynamic unscheduled weight while TSHH-HC performs better with fixed unscheduled weight. However, the difference is not significant and therefore, we can summarise that for the benchmark datasets, the unscheduled examination weights does not influence the results produced by the hyper-heuristic.

Table 8.9: Benchmark datasets with dynamic unscheduled weight (1 hour run)

File	TSHH-B					TSHH-HC				
	TD=0	TD=1	TD=2	TD=3	TD=4	TD=0	TD=1	TD=2	TD=3	TD=4
car-f92	5.96	6.13	5.21	5.62	6.28	5.98	6.29	4.90	5.58	6.39
car-s91	7.19	6.99	5.95	6.62	7.66	7.01	7.09	5.94	6.84	7.61
ear-f83	48.98	47.39	43.82	44.51	47.09	48.28	46.32	45.12	45.71	47.45
hec-s92	14.35	12.72	12.05	13.22	15.71	12.93	13.26	11.97	14.36	13.77
kfu-s93	20.09	18.93	16.16	17.10	18.71	19.25	16.81	17.60	16.34	19.99
lse-f91	15.44	15.39	15.03	14.53	17.25	15.35	15.06	14.26	15.32	17.14
pur-s93	6.24	6.20	5.99	6.99	8.07	6.19	6.10	6.24	6.92	8.32
sta-f83	171.20	168.38	158.68	165.83	166.21	165.66	167.42	158.28	169.96	165.79
tre-s92	11.70	11.63	9.34	9.41	10.48	11.99	10.53	9.00	9.51	10.21
uta-s92	4.94	4.99	4.26	4.44	5.33	6.59	6.71	4.33	4.70	5.20
ute-s92	32.38	30.46	27.64	27.22	27.75	34.94	30.43	28.14	28.15	28.64
yor-f83	46.51	41.81	42.50	43.36	45.13	48.53	45.36	40.90	40.50	43.99
Average	9.85	8.93	8.02	8.59	9.67	9.43	9.15	8.10	8.68	9.74

The same conclusion cannot be made with regards to the uitm-03 dataset (see section 8.3.2). The way we calculate the unscheduled weights did not make any significant difference on which tabu duration is suitable for a particular instance of a problem on benchmark datasets. In contrast, the dynamic unscheduled examination weight has a greater influence on how the hyper-heuristic behaves with the uitm-03 dataset. Thus, the uitm-03 has a different characteristic and structure compared to benchmark datasets.

Table 8.10: Benchmark datasets with fixed unscheduled weight (1 hour run)

File	TSHH-B					TSHH-HC				
	TD=0	TD=1	TD=2	TD=3	TD=4	TD=0	TD=1	TD=2	TD=3	TD=4
car-f92	6.02	5.92	5.23	5.64	6.38	5.84	5.71	5.24	5.64	6.33
car-s91	7.01	6.85	5.75	6.43	7.34	6.86	6.79	5.91	6.80	7.55
ear-f83	47.47	44.72	43.12	46.91	46.73	44.33	44.96	42.50	41.88	47.30
Hec-s92	13.50	12.37	12.32	12.65	12.96	16.21	13.03	11.71	13.10	12.72
kfu-s93	20.43	16.62	16.15	16.77	18.27	18.76	18.33	16.22	16.66	17.62
lse-f91	15.31	15.03	14.40	15.47	16.98	15.31	15.39	14.75	16.82	16.74
pur-s93	6.20	6.14	6.19	6.83	8.24	6.13	6.20	6.07	7.01	8.41
sta-f83	167.82	166.06	157.57	165.01	166.19	173.20	159.69	159.85	167.30	165.97
tre-s92	10.64	10.92	9.10	9.75	10.74	10.89	11.66	9.16	9.65	10.30
uta-s92	6.25	4.77	4.39	4.51	5.21	6.38	6.37	4.31	4.54	5.04
ute-s92	32.58	28.03	28.48	28.64	28.31	32.83	29.44	28.57	28.50	29.11
yor-f83	46.32	44.22	40.34	41.76	44.67	44.76	43.84	39.85	40.56	44.94
Average	10.00	8.57	8.05	8.58	9.57	9.22	9.04	8.01	8.67	9.58

8.4 Conclusions

Collecting data from a large university is a difficult task, but with the help of a distributed network and a frequent update of student information, we can facilitate the automated process of producing examination schedules that are feasible. The

uitm-03 dataset is a large real world examination timetabling problem. Its data may not exactly reflect all the required information such as validated student data and implicit constraints, however, other crucial data such as the actual timetable used and the list of examinations to be scheduled are consistent with the real problem. All data inconsistencies are removed prior to processing so that the dataset presented is as close as possible to the actual problem. The important constraints are considered in the search engine with a flexibility of adding additional constraints if necessary.

Tabu search based hyper-heuristics have been shown to produce feasible and good quality solutions on other benchmark datasets (Kendall and Mohd Hussin 2004a and chapter 7). The same hyper-heuristic framework was tested with uitm-03 dataset. An additional criterion of minimising the weekend cost is added to the objective function and with no modification to the hyper-heuristic framework it is able to produce good solutions. It is obvious that the nature of the uitm-03 dataset is different from benchmark datasets because of an additional objective criteria, a larger number of students and the maximum number of examinations in conflict is high compared to the average number of examinations in conflict (see table 5.1). Thus by adding such knowledge in the form of a new constructive heuristic in the initialisation phase, prioritising non-weekend slots and dynamic unscheduled examination weights, the hyper-heuristic framework is able to produce a much better solution. It is such knowledge that differentiates problem instances and as such the hyper-heuristic is a general method that was proven on benchmark datasets and also on UiTM problem.

Currently, the tabu search hyper-heuristic method is dependent upon the tabu duration value. In the next chapter we investigate an adaptive tabu strategy of selecting the best tabu duration at each decision point. This method should free the hyper-heuristic from the problem of parameters tuning. Execution time can be pre-determined by the user and it is reasonable to let the system run over the weekend with no difficulty.

Chapter 9

Reactive Hyper-Heuristic with Tabu Heuristic List

This chapter reports on our work to investigate if we can add more intelligence to the hyper-heuristic by adapting its parameters (tabu duration, termination criteria and tabu restrictions) during the search for good quality solutions. Section 9.1 provides a background on adaptive strategies. Section 9.2 provides a detailed description of the techniques and feedback mechanism to automate the tabu duration parameter in a hyper-heuristic framework. Results from a set of experiments on benchmark datasets and UiTM dataset (introduced in this thesis) are given in section 9.3. Section 9.4 concludes with a summary.

9.1 Introduction

The basic principle underlying a local search algorithm is to move from one solution state to another new, and improved, solution state by applying a simple move. The move that produces the best configuration for the new solution is selected from a set of sampled moves (neighbourhood moves). The size of the neighbourhood moves may be the entire size of the neighbourhood or just a sample of the neighbourhood (in cases where the neighbourhood size is too large). The algorithm stops once it reaches a local optimum. Most meta-heuristics attempt to overcome the problems of being trapped in a basin of local optima. Modern meta-heuristics such as simulated annealing, tabu search, etc., (see chapter 2) are improvements over local search by including strategies to diversify the search into other regions, modifying acceptance criteria rules, learning search characteristics

using adaptive memory, using multi-neighbourhood structure; and others. More details can be found in Osman and Kelly (1996), Voß et al. (1999), Ribeiro and Hansen (2001).

One of the strategies to escape from local optima is to repeat the search by starting from a different point in the search space (see GRASP in Festa and Resende, 2004). The search can restart from different points generated by different or stochastic constructive heuristics, from a previously visited solution or from a completely random solution. In this chapter, we apply a restart strategy to our hyper-heuristic when there is a sign of no improvement in the search. A feedback-based tuning parameter that automates the tabu duration parameter is included in the algorithm. This mechanism is inspired by the reactive tabu search by Battiti and Tecchiolli (1994) and reactive search by Battiti (1996). In the reactive schemes, an appropriate tabu list size is determined automatically (by feedback-based parameter tuning) by reacting to the occurrence of cycles in the solution configuration. The list size is increased if there is a need for diversification. A balance of intensification and diversification is obtainable through a feedback mechanism that monitors the repetition of configurations. If there are repetitions, diversification is needed and if too many configurations occur too many times, a large jump is made by executing a number of random steps that moves the trajectory away from the current region. Another mechanism to automate the tabu list size is by using a functional representation of the tabu list size for a particular application proposed by Salhi (2002). An attribute is penalised according to the change in the cost function it produced. In other words, a removal of an attribute that yielded a larger improvement will have a higher value for the tabu size compared to the removal of an attribute that produced a smaller or negative improvement.

In this chapter, the objectives are:

- To automate an adaptive tabu duration parameter based on a feedback mechanism.

- To include a robust diversification technique by monitoring search performance.
- To impose a mixture of tabu duration value on heuristics based on their performance history.

9.2 Methodology

The design, implementation and results of the basic tabu search based hyper-heuristic (TSHH-B), tabu search based hyper-heuristic with hill climbing (TSHH-HC) and tabu search based hyper-heuristic with great deluge (TSHH-GD) was reported in the previous chapters. TSHH-B and TSHH-HC performance is similar across all benchmark datasets while TSHH-GD is superior by performing better on a greater number of benchmark datasets.

The hyper-heuristic methods were tested with a range of fixed deterministic tabu duration and random dynamic tabu duration. Chapters 6 - 8 reported that the tabu duration value has varying effect on different benchmark datasets, i.e. not all benchmark datasets achieve the best result with the same tabu duration value. Previous experiments show that we need to tune the tabu duration parameter because as we increase the tabu duration values, the results improved and begin to deteriorate again once the tabu duration value is too large. This chapter addresses the issue of incorporating an adaptive parameter tuning within the hyper-heuristic so as to alleviate the necessity to tune the hyper-heuristic for a particular problem instance. Since the range of the tabu duration is small, it will be possible to allow the hyper-heuristic to make a dynamic decision of applying different tabu duration at different stages in the search.

The following strategies are tested:

- § **Adaptive tabu duration value:** The hyper-heuristic ranks the tabu duration value from a set of values $\{0, 1, 2, 3, 4\}$ based on their performance for the first 5 minutes. The hyper-heuristic generates a tabu duration sequence containing a

sequence of tabu duration values to be applied in the improvement phase. The best tabu duration value will be applied more than the worst tabu duration value. Assume that R is a list of tabu durations ranked from the worst to the best and R_i is tabu duration value with rank i . For example if $R = \{0, 4, 1, 3, 2\}$, the worst tabu duration value is 0 and the best tabu duration value is 2. The number of times each tabu duration will be applied is as follows $\{1, 2, 4, 8, 16\}$ i.e. tabu duration value 2 will be applied 16 times and so on. This allows the best tabu duration value to be applied twice as frequently as the next best tabu duration value. The sequence is reshuffled randomly to create a mixture of different tabu duration values to be applied each time while preserving the frequency of each tabu duration value to be applied. The hyper-heuristic selects the next tabu duration value to apply from this sequence. Once the hyper-heuristic exhausts the sequence it restarts from the beginning of the sequence.

§ **Performance based tabu duration value:** The hyper-heuristic monitors the performance of each heuristic for the first 5 minutes by identifying which heuristics direct the search into infeasible regions (heuristics that unscheduled examination), identifying which heuristics direct the search from an infeasible region to a feasible region (heuristics that schedule examination) and keeping account of the number of improvements made by each heuristic. The heuristic(s) that direct the search into infeasible region are assigned the maximum tabu duration value since we do not want to apply this heuristic too frequently. The heuristic(s) that cause a move from an infeasible region to feasible region will not be made tabu at all (tabu duration equals 0). If a heuristic improves solution less than 20% of the time it was called, the tabu duration value is set to 3. In other words, a heuristic that normally does not improve solution will be made tabu longer. Other heuristics that do not fall under any of the previous categories will be assigned the best tabu duration value.

§ **Restart the search with a different tabu duration value:** The search restarts with the next tabu duration value in the tabu duration sequence each time the hyper-heuristic is stuck in a local optima and encounters a no improvement phase for a period of time (10 seconds, 5 minutes or 50,000 iterations).

§ **Restart the search with previous best solution or randomised solution:** The search restarts with the previous best solution stored in memory if there was at least a 0.05% improvement from its last restart. If the improvement is less than 0.05%, it will reinitialise the solution by randomly rescheduling examinations into a different slot.

We can control the time taken by TSHH-B and TSHH-HC since the termination condition is fixed according to the run time. However, TSHH-GD varies in run time depending upon the size of a dataset and can either take longer time or a much shorter time compared to TSHH-B and TSHH-HC. Therefore, in the following experiments we apply either TSHH-B or TSHH-HC or combine both TSHH-B and TSHH-HC. TSHH-GD is tested only once in one of the experiment.

For each of the experiments, the algorithm runs for 5 minutes with the tabu duration varying from 0 to 4 (called the learning stage). In the learning stage, the hyper-heuristic monitors the performance of each low level heuristic with respect to their performance in:

- Improving the solution.
- Moving the search from a feasible to an infeasible region.
- Moving the search from an infeasible region to a feasible region.

After 5 minutes, the hyper-heuristic is able to rank the tabu duration value based on their performance. The tabu duration value that produces the best solution will have the highest rank. The hyper-heuristic then generates a sequence of tabu duration value to be applied in the search. The sequence contains all the tabu duration values {0, 1, 2, 3, 4} described above as adaptive tabu duration value and performance based tabu duration value.

The following is a description of the nine experiments we have conducted:

1. ***TSHH-B_TSHH-B_BestFirst***. Run TSHH-B during the learning stage and restart TSHH-B with the previous best solution starting from the best tabu duration value to the worst tabu duration value (5 minutes each).
2. ***TSHH-B_TSHH-GD_BestFirst***. Run TSHH-B during the learning stage and restart TSHH-GD with previous best solution starting from best tabu duration value to the worst tabu duration value (5 minutes each).
3. ***TSHH-B_TSHH-HC-BestOnce***. Run TSHH-B during learning stage and run TSHH-HC once with the best tabu duration value for 4 hours.
4. ***TSHH-B_TSHH-HC_OncePerfTD***. Run TSHH-B during the learning stage and run TSHH-HC once with performance based tabu duration value for 4 hours.
5. ***TSHH-HC_RestartRandomiseSequenceTD***. Run TSHH-HC during the learning stage and restart TSHH-HC with a randomised solution using tabu duration sequence. TSHH-HC restarts after 5 minutes of no improvement (maximum run time is 4 hours).
6. ***TSHH-HC_RestartBestSequenceTD***. Run TSHH-HC during the learning stage and restart TSHH-HC with the previous best solution using tabu duration sequence. TSHH-HC restarts after 5 minutes of no improvement (maximum run time is 4 hours).
7. ***TSHH-HC_RestartBestRandomiseSequenceTD***. Run TSHH-HC during the learning stage and restart TSHH-HC with the previous best solution or randomised solution (if previous solution has less than 0.05% improvement)

using tabu duration sequence. TSHH-HC restarts after 5 minutes of no improvement (maximum run time is 4 hours).

8. ***TSHH-HC__RestartBestRandomiseSequenceTD (12 hrs).*** Run TSHH-HC during the learning stage and restart TSHH-HC with the previous best solution or randomised solution (if previous solution has less than 0.05% improvement) using tabu duration sequence. TSHH-HC restarts after 5 minutes of no improvement (maximum run time is 12 hours).
9. ***TSHH-HC__RestartBestRandomiseMixedTD.*** Run TSHH-HC during the learning stage and restart TSHH-HC with the previous best solution or randomised solution (if previous solution has less than 0.05% improvement) using tabu duration sequence (mixture of adaptive tabu duration values and performance based tabu duration value). TSHH-HC restarts after 10 seconds or 50,000 iterations of no improvement (maximum run time is 4 hours).

Note that experiment number 9 (***TSHH-HC__RestartBestRandomiseMixedTD***) incorporates many of the strategies outlined at the beginning of this section. Figure 9.1 illustrates the algorithm for experiment number 9.

The above nine experiments were carried out on 12 benchmark datasets. The objectives of these experiments are:

- To determine the best strategies combination for the 12 benchmark datasets.
- To compare results from these experiments with the best results from TSHH-B, TSHH-HC and TSHH-GD
- To compare results from the best experiment with the results from random dynamic tabu duration of TSHH-B, TSHH-HC and TSHH-GD.

- To apply the best algorithm on a new dataset (i.e. uitm-03), with fixed unscheduled examination weight and dynamic unscheduled examination weight.

Construct initial solution s

For $td = 0$ to 4

Run TSHH-HC with tabu duration = td and save best solution for each td value

Rank the td values based on their best solution

Assign each heuristic a td value based on their performance

Generate a sequence of td values to be applied (fixed td or performance based td)

Set $TD =$ next td value in the sequence

Do

Call TSHH-HC with tabu duration = TD and terminate if no improvement for 50,000 iterations or for 10 seconds (whichever comes first)

If $\text{improvement}(\text{current solution}, \text{previous solution}) < 0.05\%$

Create new initial solution s

Else $\text{initial solution} = \text{current best solution}$

Set $TD =$ next td value in the sequence

Until *terminating condition*

Figure 9.1: Algorithm for experiment number 9

9.3 Results and Analysis

The tabu search based hyper-heuristic framework was implemented and tested on a PC with an AMD Athlon 1 GHz processor, 128 Mb RAM and Windows 2000. The nine experiments were carried out on the examination timetabling benchmark datasets (refer to table 5.1 for a detailed description of the datasets). The best algorithm from the nine experiments is applied on the UiTM dataset.

9.3.1 Results of Nine Experiments for Benchmark Datasets

The results in table 9.1 are from the nine experiments outlined in section 9.2. The column labelled *Best* is the best results from the 9 experiments for each dataset, while the last column is the best result from chapter 7 (best of TSHH-B, TSHH-HC and TSHH-GD). The results that are better than the results from chapter 7 (sta-f83 and ute-s92) are printed in blue. Note that the best results from chapter 7 are the results from different tabu search hyper-heuristic methods and with different fixed tabu duration. In chapter 7, all values of the tabu duration parameter were tested to give results that we can compare to. Therefore, the hyper-heuristic is still dependent upon the tabu duration parameter. In the current implementation, we can run the algorithm without entering any parameters. The only parameters needed are the dataset and their characteristics. The algorithm can be left running for any period of time.

Table 9.1: Results from 9 experiments

File	Experiment Number									Best	Chap7
	1	2	3	4	5	6	7	8	9		
Car-f92	5.69	5.21	4.41	5.54	4.70	5.38	4.99	4.97	4.61	4.41	4.40
Car-s91	6.45	5.82	5.71	7.05	5.57	5.72	5.76	5.31	5.38	5.31	4.93
Ear-f83	43.09	42.52	42.44	44.60	38.83	39.66	40.11	39.08	38.57	38.83	38.14
Hec-s92	11.28	12.02	10.91	12.35	11.55	11.48	11.64	10.62	10.82	10.62	10.74
Kfu-s93	16.60	16.27	15.76	17.25	16.16	16.71	16.54	16.02	16.22	15.76	15.55
lse-f91	14.95	14.21	14.11	14.67	14.65	13.64	12.53	12.08	13.00	12.08	12.35
Pur-s93	6.20	6.20	6.17	6.10	5.99	6.11	5.87	5.77	5.84	5.77	5.67
sta-f83	158.34	165.48	166.36	158.24	157.84	156.56	156.42	156.44	154.19	154.19	157.38
tre-s92	9.58	8.74	9.30	9.28	9.25	8.91	9.24	8.94	8.98	8.74	8.35
Uta-s92	4.45	4.38	4.23	4.48	4.33	4.46	4.30	4.19	4.04	4.04	3.77
Ute-s92	27.78	29.49	27.26	29.36	28.18	27.07	26.89	27.33	27.28	26.89	27.12
Yor-f83	43.10	43.10	39.38	41.95	41.56	38.98	39.73	39.46	38.58	38.58	37.99
Avg	8.95	8.78	8.47	9.08	8.47	8.61	8.43	8.23	8.18	8.08	7.83

In table 9.1, the best result across all experiments for each dataset is printed in bold. The last row is an average penalty per student across all dataset (best average printed in red). In general (refer to last row), the average penalty across all datasets does not show lots of difference (range between 8.23 to 9.08). Even though all experiments are equally good, experiment number 9 is considered to be superior amongst the others. Experiment number 9 incorporates most of the strategies outline in section 9.2. It uses TSHH-HC to rank the tabu duration values and uses the performance history to determine a performance based tabu duration value for each heuristic. It has more frequent restarts because the no improvement time is 10 seconds compared to 5 minutes of no improvement time for other experiments. We now refer to experiment 9 as reactive tabu search based hyper-heuristic.

9.3.2 Reactive Tabu Search Based Hyper-heuristic (Experiment 9) Analysis

Previously, we have mentioned that experiment 9 (reactive tabu search based hyper-heuristic) incorporates most of our adaptive strategies, and we propose to evaluate this method further. In chapter 8, we have shown that the choice of using either fixed unscheduled examination weight or dynamic unscheduled examination weight does not have as much impact on the benchmark datasets compared to the uitm-03 dataset. However, we will also evaluate the reactive tabu search based hyper-heuristic with the dynamic unscheduled examination weights for the benchmark datasets. We will compare these results with results using fixed unscheduled examinations weight from TSHH-B, TSHH-HC and TSHH-GD with random dynamic tabu duration (from table 7.2) and reactive tabu search based hyper-heuristic. Random dynamic tabu duration does not need to include parameter tuning since the tabu duration value is selected randomly from the tabu duration range (t_{min} - t_{max}). However, since it is random, each tabu duration value has an equal chance of being selected at each decision point. In reactive tabu search based hyper-heuristic, the choice of tabu duration value is more systematic based on a feedback mechanism that uses previous history and performance.

Therefore, in table 9.3 we show that reactive tabu search based hyper-heuristic (RTSHH in the last two columns) is superior compared to the random dynamic tabu duration (RTD) of TSHH-B, TSHH-HC and TSHH-GD methods. Both RTSHH with the fixed unscheduled examination weight and dynamic unscheduled examination weights perform almost similar. This confirms our previous findings that an unscheduled examination weights does not have much effect on the final solution for the benchmark datasets.

Table 9.2: Reactive Hyper-heuristic Methods

File	TSHH-B (RTD)	TSHH-HC (RTD)	TSHH-GD (RTD)	RTSHH (fixed unscheduled examination weight)	RTSHH (dynamic unscheduled examination weight)
car-f92	4.61	4.70	4.97	4.61	4.65
car-s91	5.43	5.31	5.60	5.38	5.38
ear-f83	41.56	41.76	42.53	38.57	39.24
hec-s92	11.97	10.74	12.01	10.82	10.82
kfu-s93	15.56	15.55	16.65	16.22	16.28
lse-f91	13.29	14.33	13.55	13.00	12.45
pur-s93	5.67	5.72	5.58	5.84	5.79
sta-f83	157.42	158.26	160.33	154.19	154.25
tre-s92	9.21	8.66	9.34	8.98	8.85
uta-s92	4.23	4.28	4.27	4.04	4.14
ute-s92	27.34	27.66	28.28	27.28	27.04
yor-f83	40.04	40.24	43.01	38.58	39.72
Avg	8.26	8.27	8.47	8.18	8.19

We now compare the results from the fixed unscheduled examination weight of RTSHH with the TSHH-B, TSHH-HC and TSHH-GD. Six datasets have better results compared to the other three methods and 1 dataset is the same as the best result of the other three methods. The results of the remaining 5 datasets are not better than the other three methods but they are competitive. The average penalty per student across all datasets for RTSHH is also better than the other 3 methods.

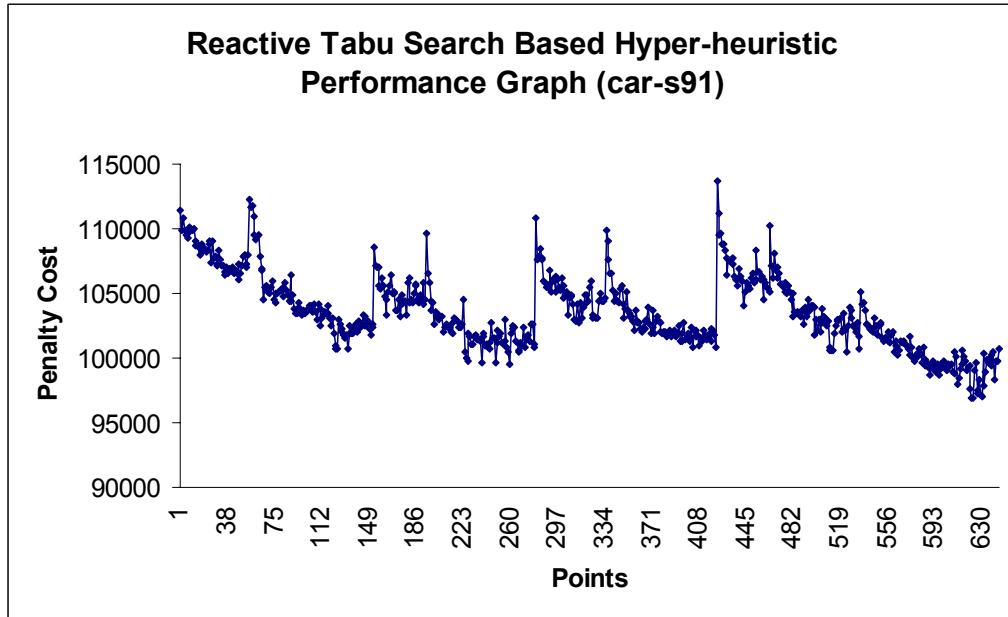


Figure 9.2: A snapshot of Reactive Tabu Search Based Hyper-heuristic performance graph for car-s91 dataset.

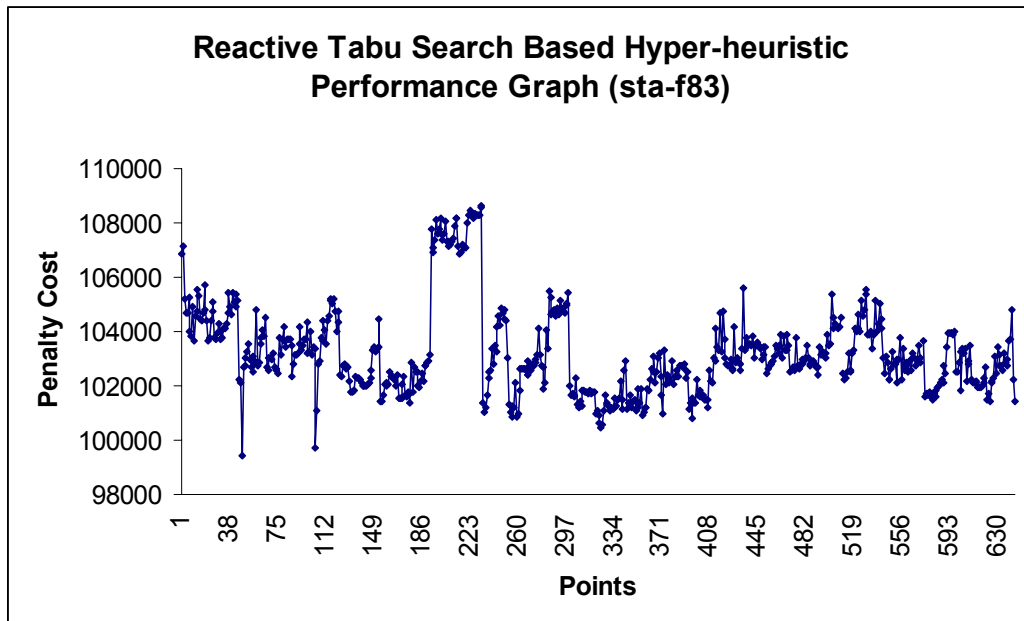


Figure 9.3: A snapshot of Reactive Tabu Search Based Hyper-heuristic performance graph for sta-f83 dataset.

9.3.3 Performance Analysis

Figure 9.1 and figure 9.2 illustrate snapshots of the reactive tabu search based hyper-heuristic performance graph on car-s91 and sta-f83 datasets. Each point in the graph is the current penalty cost monitored after every 5,000 iterations. The high points in the graph indicate a restart in the search. Thus, by starting the search again (after no improvement) from different points, we are diversifying and carrying out a more intensive exploration of the search space. In figure 9.3, at point 223, the search starts at a very good point and within the time taken (10 seconds) or in 50,000 iterations, the search could not further improve the solution. As we change the tabu duration value, the search restarts from the previous best solution stored in memory.

Figure 9.4 shows another snapshot of one cycle in a search for sta-f83 dataset. One cycle starts with a new solution and dynamically change its tabu duration when there is no improvement and ends when the search diversifies with a new solution. The final result for this search is 156.05. The points highlighted by a square indicate that the search change its tabu duration value (selected from the tabu duration sequence) to be applied. The search starts from a new solution with the following sequence of tabu duration {mixed TD, mixed TD, mixed TD, 1, 4} where mixed TD indicates a tabu duration value based on performance (mixed TD values for each heuristics). Once it cannot further improve the solution, the tabu duration value changes (select next TD from the sequence) and begins with the previous best solution from memory. Note that this combination of TD values has produced better results than previous hyper-heuristic methods (chapter 7) with TD values of 1 or 4.

Table 9.3 shows how the hyper-heuristic monitors the performance of each low-level heuristics. Refer to chapter 6 for a detailed description of each low-level heuristic. The column headings of table 9.3 are the attributes of each low-level heuristic that is updated after every iteration. The first 5 heuristics are heuristics that reschedule un-scheduled examinations. The number of examinations to be scheduled

is normally very small and all 5 heuristics normally schedule the same examination resulting in the same change in penalty. Since the hyper-heuristic will always select the first performing heuristic it will always select the first heuristic (see row 2, column 3) to schedule an examination. The most applied heuristic is 6 (randomly select an examination and schedule into a randomly selected slot). The next heuristic that is applied most is heuristic 7, which moves examinations with highest penalty selected from a set of randomly selected examinations. Heuristic 9 (reschedule examination with highest third order conflict) improves the solution more than the other heuristics, followed by heuristic 10 (reschedule examination with highest second order conflict). Heuristic 13 removes examinations already scheduled, thus it always moves the search to an infeasible region.

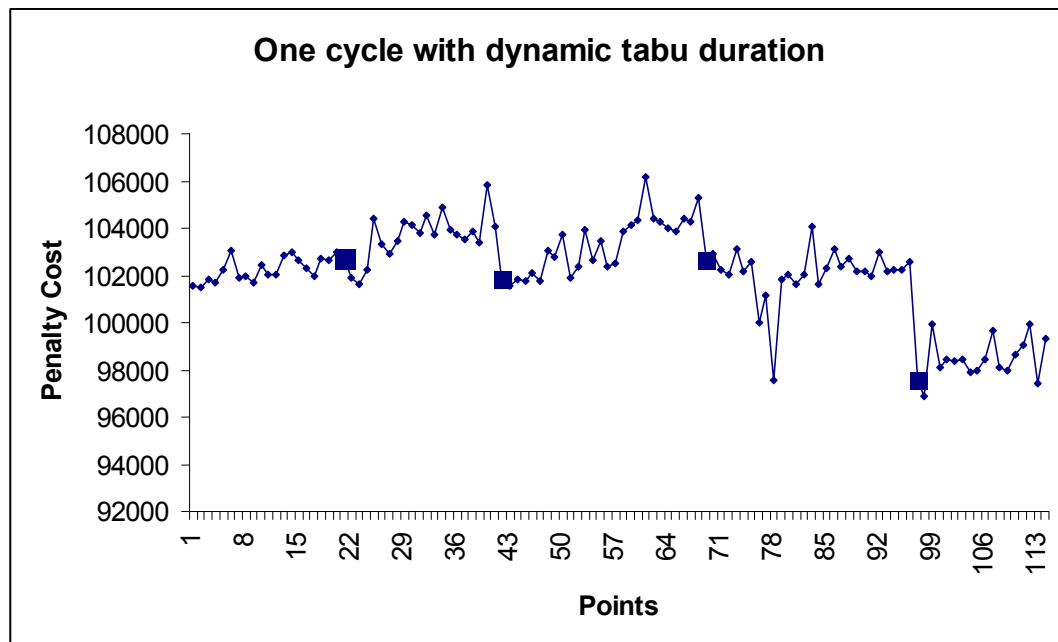


Figure 9.4: One cycle of Reactive Tabu Search Based Hyper-heuristic search graph (sta-f83 dataset).

Table 9.3: A snapshot of low-level heuristic performance

Heuristic ID	Number of times being considered	Number of times applied	No change in solution	Number of iterations made tabu	Number of times improve solution	No Improvement	Move from feasible to infeasible region	Move from infeasible to feasible region
1	77,357,952	270,399	77,077,796	1,079,856	280,156	0	0	280,156
2	78,437,768	1	78,157,611	1	280,157	0	0	280,157
3	78,437,769	0	78,157,612	0	280,157	0	0	280,157
4	78,437,769	0	78,157,612	0	280,157	0	0	280,157
5	78,437,769	0	78,157,612	0	280,157	0	0	280,157
6	42,254,278	34,995,521	227,300	36,184,186	388,981	5,568,191	0	14,145
7	52,945,610	21,787,623	18,544	25,496,740	964,921	15,063,511	0	131
8	60,639,046	14,090,067	14,654	17,798,783	53,727	7,564,772	0	0
9	76,953,147	1,302,619	55,998,230	1,484,946	5,846,832	70,935,665	0	0
10	78,296,785	128,813	61,044,923	141,044	2,198,497	76,032,464	0	0
11	77,959,861	372,687	73,758,616	477,908	622,022	9,043,575	0	0
12	70,147,475	5,210,439	49,175	8,298,757	1,874,113	32,631,015	0	0
13	77,318,971	280,158	0	1,118,798	0	77,318,971	77,318,971	0

9.3.4 Reactive Tabu Search Based Hyper-heuristic with Fixed and Dynamic Unscheduled Examinations Weight (UiTM Dataset)

In section 9.3.1, we presented results from the reactive tabu search based hyper-heuristic tested on the benchmark datasets. The hyper-heuristic went through a modification that involves an adaptive tabu duration parameter (sequence of tabu duration values based on history of previous performance) and a reactive mechanism (when to restart with a previous best solution or a new solution). In this section we apply the hyper-heuristic without any modification on the uitm-03 dataset (see chapter 8). We run two experiments; firstly, using a fixed unscheduled examination weight and secondly, using dynamic unscheduled examination weight for uitm-03 proximity and weekend cost problem. Table 9.4 shows the results from the experiment and a comparison with TSHH-B, TSHH-HC and TSHH-GD with random

dynamic tabu duration (from table 8.2). The reactive tabu search based hyper-heuristic is superior compared to other previous methods.

Table 9.4: Comparison of results for uitm-03 dataset.

	TSHH-B (RTD)	TSHH-HC (RTD)	TSHH-GD (RTD)	RTSHH (fixed unscheduled examination weight)	RTSHH (dynamic unscheduled examination weight)
Uitm-03	9.51	9.65	8.04	6.40	5.33

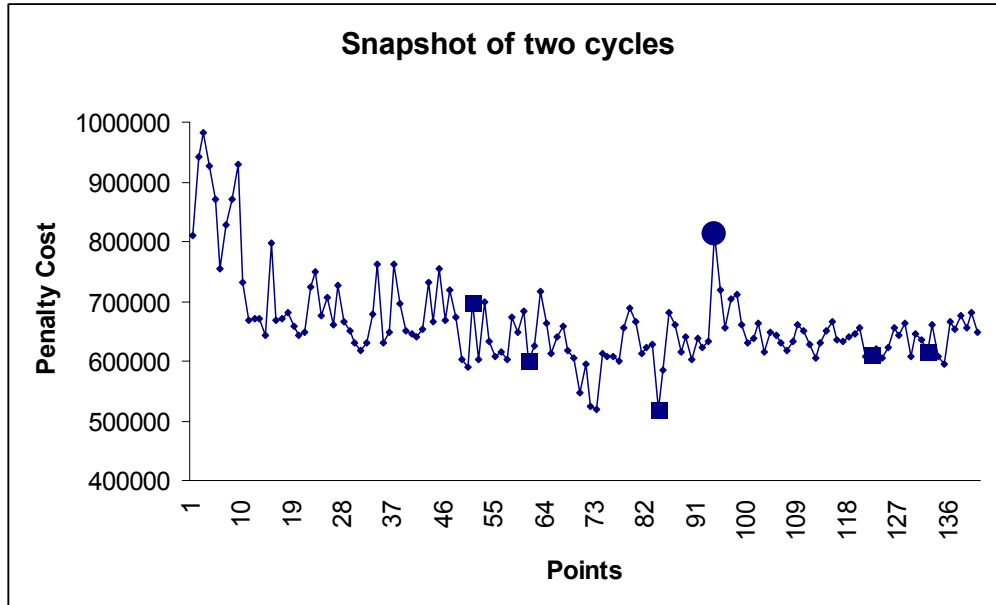


Figure 9.5: A snapshot of two cycles in the search.

Figure 9.5 shows a snapshot of two cycles in the search. The circle indicates when a new cycle starts with a new solution. The square points indicate when the search changes its tabu duration value. Again, we can see the search starting from different points in the search space, continue improving the solution by dynamically changing the tabu duration value and restarting with a new solution once its improvement is below 0.05% (almost zero percent).

9.4 Conclusions

In this chapter, we have included strategies that monitor heuristic performance and provide feedback so as to facilitate the automated determination of the tabu duration parameter. The tabu duration sequence generated includes a combination of tabu duration values that work in favour of the best tabu duration value. The performance based tabu duration value for each heuristic included in the sequence can differentiate the heuristics in favour of heuristics that make more improvement and are capable of directing the search to a feasible and promising region of the search space. The reactive tabu search based hyper-heuristic is able to recognise when diversification is needed (when there is no improvement and the search is not doing anything fruitful). It restarts from a previous best solution or a randomised solution with a different tabu duration value. With a different tabu duration value, the heuristics will stay tabu for a shorter or longer period of time causing a different combination of heuristics being applied.

The best experiment tested on benchmark datasets is proposed as the reactive tabu search based hyper-heuristic because of its superiority and integration of various strategies that makes the hyper-heuristic more dynamic and robust. The reactive tabu search based hyper-heuristic has also produced good quality solution on a large dataset with an additional constraint (weekend constraint) and different evaluation function. The adaptive and reactive mechanism of the hyper-heuristic does not use domain knowledge of problem being solved and as such it should be applicable on a wider range of problems.

Chapter 10

Conclusions

There were four objectives of the research presented in this thesis. Firstly, to develop a hyper-heuristic framework that can solve the basic examination timetabling problem of assigning examinations to slots without violating a clash constraint. Secondly, to demonstrate how a hyper-heuristic can intelligently manage several low-level heuristics that are specific to a particular application. Thirdly, to present a new examination timetabling dataset with a new objective function and new constraints not present in other benchmark datasets. Finally, to demonstrate hyper-heuristic framework flexibility in adapting to new examination timetabling datasets with different problem characteristics, constraints and evaluation function.

The research carried out to achieve the above objectives can be summarised in the following research contributions:

§ **A Tabu search based hyper-heuristic framework:** A new hyper-heuristic incorporating memory (tabu list) to manage a set of low-level heuristics that guides a search towards good quality solution was implemented and tested with benchmark datasets. Three tabu search based hyper-heuristic methods: basic tabu search based hyper-heuristic (TSHH-B), tabu search based hyper-heuristic with hill climbing (TSHH-HC) and tabu search based hyper-heuristic with great deluge (TSHH-GD) were implemented. Each of these methods requires a tabu duration parameter and was tested with fixed deterministic tabu duration (0 – 4) and random dynamic tabu duration. TSHH-B and TSHH-HC produced similar results (best tabu duration value is 2) and by prolonging the run time (4 hours), better results were produced. TSHH-GD produced better results for most

benchmark datasets and the run time varies according to the size of the dataset. Larger size dataset took longer since the algorithm terminates after a fixed number of iterations or after a fixed number of idle iterations. These methods are as competitive as other known methods applied on the same benchmark datasets and using a similar quality measure.

§ **Tabu list as a control management tool:** The hyper-heuristic used tabu list as a guide to manage co-operation between low-level heuristics and to choose the low-level heuristics at each decision point in a search. When a low-level heuristic has been applied, it was made tabu and other low-level heuristics not in the tabu list will compete to produce the next best move. Without the tabu list, all heuristics are equally likely to be selected and some heuristics may not be selected at all. From all three methods, the fixed deterministic tabu duration value has a small effect on the results. The tabu duration value was first set to zero and as the tabu duration value increased, the results improved and as soon as it reached its best tabu duration value the results began to deteriorate. The average results for TSHH-B and TSHH-HC using random dynamic tabu duration were better compared to using fixed deterministic tabu duration. From this observation, we can conclude that a combination of different tabu duration value applied improved solution.

§ **A well-defined relationship between the hyper-heuristic module and the low-level heuristics module:** The hyper-heuristic framework utilised only general knowledge (such as change in objective function, a move into infeasible or feasible region, etc.) about any application and does not rely upon domain knowledge to make its decisions. A well-defined interface between hyper-heuristic framework and low-level heuristics was established so that the hyper-heuristic can work with a different set of low-level heuristics for another problem.

§ **A well-defined module that encapsulate a set of low-level heuristics and an evaluation function for an application:** The module contains a set of low-level

heuristics for the examination timetabling problem and an evaluation function. The heuristics gave the flexibility for the search to explore in an infeasible region, direct the search back into feasible region, improve a specific soft constraint and perform one or two moves. The evaluation function for the benchmark datasets was similar to the evaluation function used by other researchers. The same evaluation function was necessary so as to make a comparison between results from different approaches.

- § **A parameter free reactive tabu search based hyper-heuristic:** A reactive tabu search based hyper-heuristic was implemented that used a feedback mechanism to automate an adaptive tabu duration parameter. The hyper-heuristic monitored the performance of each low-level heuristics, ranked them and generated a sequence of tabu duration value to be applied at each point. In addition to the deterministic tabu duration for all low-level heuristics, suitable performance based tabu duration was also assigned to each low-level heuristic. Even though only one best result was obtained on the benchmark datasets compared to the result from the best method and best tabu duration value for each benchmark dataset, other results were as competitive. The main achievement is that this method has no parameters to tune and we can just run it any number of times.
- § **A new final examination dataset and its problem formulation:** An examination timetabling dataset from the MARA University of Technology (UiTM) was presented in a format similar to the benchmark datasets. This dataset can be used by examination timetabling researchers using existing or new approaches that solved examination timetabling benchmark datasets. In addition, the dataset contains information relevant to solve its weekend slot assignment problem. A new problem model to solve the UiTM problem was presented that included an additional constraint of minimising weekend slots. If an examination is scheduled in a weekend slot a penalty is assigned equivalent to weekend weight multiplied by number of students involved in this weekend slot. Not all students have the same weekend slot. A student who has to sit for an examination during a weekend slot is as important as a student sitting for two

examinations in sequence and, hence, both were given the same penalty weight.

- § **Flexibility in the hyper-heuristic framework:** The framework is easily adaptable to other problems with multi-criteria by simply adding a new low-level heuristic that optimise a specific criterion and a new objective function. In the UiTM problem, the hyper-heuristic module was not modified. The objective function that minimises proximity cost was replaced with a new objective function that minimises proximity cost and weekend cost. A new low-level heuristic that optimises the weekend constraint was added to the set of low-level heuristics. Other constraints that need to be minimised can be added in the same manner. A dynamic unscheduled examination weight was used instead of fixed weights because the number of students was too large and un-scheduling an examination may be more profitable compared to scheduling an examination.
- § **Competitive solutions:** The hyper-heuristic framework is shown to produce competitive solutions across all current benchmark datasets. It is also shown to produce good solutions for the new and large dataset introduced in this thesis.

Future research work with the reactive tabu search hyper-heuristic is in the integration of several problem characteristics (that has an effect over the overall performance of the hyper-heuristic) in the low-level heuristics module. Good initial solutions, proper penalty given to an incomplete solution and dynamic weights can further enhance the hyper-heuristic performance. In this research we concentrated more on the dynamics of the hyper-heuristic level rather than the low-level heuristic module. Currently, the low-level heuristics are simple moves that do not actually explore a larger search space. Future work can include a more intelligent low-level heuristics with variable neighbourhood structure and with a history performance based feedback mechanism.

We can see the robustness of the hyper-heuristic in prolonging its search time without being stuck in local optima and in solving another problem with a new

evaluation function and additional criterion (i.e. the UiTM dataset with weekend constraint). It would be interesting to see how it would perform with a totally new set of low-level heuristics for a similar problem domain to examination timetabling.

Course scheduling and examination timetabling are closely related problems which, when integrated together in a fully automated system, will provide a comprehensive solution to an academic institution. A hyper-heuristic approach has the potential to provide the necessary search engine in a multi-layer system that could solve both the course scheduling and examination timetabling problems.

The MARA University of Technology examination timetabling dataset was the actual data based on student registration for courses at the beginning of the semester. Throughout the semester, some updates were not completed and as a result the data is incomplete. We believe that the actual problem is more difficult, and it would be beneficial to collect another set of data from a different semester so as to compare the problem characteristics with the current set of data. Future work will also include adding the current tabu search hyper-heuristic module into the existing examination timetabling module (part of the Academic Affairs System developed at the Centre for Integrated Information Systems, UiTM) such that it can generate a new examination timetable every semester. Currently, all students are required to register for courses using a web-based information system called the Integrated Course Registration and Scheduling System (available at <http://icress.uitm.edu.my>). Thus, with a more accurate and updated student registration, we can produce a good quality timetable as early as possible.

The number of time slots for the MARA University of Technology examination timetabling problem was the actual number of slots practised every semester. Future work will include an optimisation of number of time slots required using current tabu search based hyper-heuristic and other graph colouring heuristics. Certainly, with a reduction in the number of time slots, proximity and weekend cost will increase, and so will the capacity for each time slot. Even with the current situation, UiTM has to

rent halls to accommodate all their students, so it may not be advisable to reduce number of time slots. Therefore, any changes to the number of time slot will require further analysis in terms of cost of halls rental, invigilators pay and difficulty in finding feasible solution.

Appendix A

Student and Invigilator Questionnaire



UNIVERSITI TEKNOLOGI MARA

NAIMAH MOHD HUSSIN

School of Computer Science and IT, University of Nottingham
Jubilee Campus, Nottingham NG8 1BB, United Kingdom

email: nbh@cs.nott.ac.uk

url: <http://www.cs.nott.ac.uk/~nbh>



STUDENT FINAL EXAMINATION SCHEDULE QUESTIONNAIRE

BACKGROUND: My name is Naimah Mohd Hussin. I am a researcher and a member of Automated Scheduling, Optimisation and Planning Group (ASAP) in the Department of Computer Science and Information Technology, University of Nottingham. We are investigating novel methods of handling the various university timetabling problems and comparing the experiences in Malaysian and in the UK. Various projects can be found at our University Timetabling Research page.

This questionnaire will help us to identify what are the possible constraints and requirements that can be included so as to produce a good final examination schedule. **Please answer the following questions if you take examinations at the end of each semester. Your comments will help in the design of a system that will improve our final examination schedule.**

Name: _____ Email: _____

Degree and Area of Specialisation: _____

Faculty: _____ University: _____

PART A

1. How many examination papers do you sit for in a normal semester? _____
2. How long is your examination period? _____ days
3. What should be the minimum day gap between two examination papers?

4. Are you able to suggest revisions to the first draft of the final examination timetable? _____
5. If yes, how? _____
6. How are examination timetables delivered to you?

7. How long before the examination period are timetables delivered to you?

8. Is the method of delivery satisfactory? YES / NO

PART B. Circle your choice

		disagree completely				agree completely
1	I prefer to have my examinations spread evenly over the examination period.	1	2	3	4	5
2	I prefer to sit for the most difficult examinations earlier	1	2	3	4	5
3	I prefer to have morning examinations	1	2	3	4	5
4	I prefer to have afternoon examinations	1	2	3	4	5
5	I prefer to have weekday examinations	1	2	3	4	5
6	I prefer to have weekend examinations	1	2	3	4	5
7	I prefer to have projects due date before final examination period	1	2	3	4	5

PART C.

1. Name some weaknesses of your previous examination timetables.

2. What do you consider as the most important things to take into account when producing examination timetables?

3. What additional requirements would you like to be taken into account by an automated examination scheduler?

4. Additional comments related to examination timetabling

Would you like (sent by email and through web page) further information on the survey results? YES / NO

THANK YOU



UNIVERSITI TEKNOLOGI
MAR A

NAIMAH MOHD HUSSIN

School of Computer Science and IT, University of Nottingham

Jubilee Campus, Nottingham NG8 1BB, United Kingdom

email: nbh@cs.nott.ac.uk

url: <http://www.cs.nott.ac.uk/~nbh>



EXAMINATION INVIGILATION QUESTIONNAIRE

BACKGROUND: My name is Naimah Mohd Hussin. I am a researcher and a member of Automated Scheduling, Optimisation and Planning Group (ASAP) in the Department of Computer Science and Information Technology, University of Nottingham. We are investigating novel methods of handling the various university timetabling problems and comparing the experiences in Malaysia and in the UK. Various projects can be found at our University Timetabling Research page

This questionnaire will help us to identify what are the possible constraints and requirements that can be included so as to produce a good invigilation schedule. I would appreciate it if you could answer the questions below if **you are required to invigilate and your university produces a common invigilation schedule**

Name: _____ Email: _____

Title: _____ University: _____

1. How would you rate your last invigilation schedule?

1) Poor

2) Fair

3) Good

4) Excellent

and why?

2. Is your invigilation schedule prepared at university: _____ or faculty: _____
or department _____ level?

3. Do you ever experience a clash in your invigilation schedule? YES / NO

If yes, what action do you have to take? _____

4. In your experience, have you ever invigilated: on your own? _____
with another invigilator? _____ with two other invigilators? _____
with more than two other invigilators? _____

5. How many days are allocated for your examination timetable? _____ days

6. How many periods are there on weekdays? _____ on Saturday? _____ and on
Sunday? _____

7. Does your university use the policy that:

a. Lecturers should invigilate their own papers/modules YES / NO

b. Lecturers should not invigilate their own papers/modules. YES / NO

c. Lecturers may or may not invigilate their own papers / modules.
YES / NO

8. What is an appropriate minimum number of invigilation duties? _____

9. What is an appropriate maximum number of invigilation duties? _____

10. What should be the minimum period gap between two invigilation duties?
_____ periods

11. If weekend invigilation is considered as double duty I would prefer a
weekend duty. YES / NO

12. All invigilators should have the same amount of invigilation irrespective of
workload. YES / NO

13. How were you informed about your invigilation schedule?

14. How long before the examination period are your invigilation schedule
delivered to you?

15. Is the method of delivery satisfactory? YES / NO

16. Name some weaknesses of your previous invigilation schedule

17. What do you consider as the most important thing to take into account when producing examination invigilation schedule?

18. What additional requirements would you like to be taken into account by an automated examination scheduler?

19. Additional comments related to examination invigilation schedule.

Would you like (sent by email and through web page) further information on the survey results YES / NO

THANK YOU

Appendix B

Data File Format and Specification

Benchmark Datasets

Each set of test data come in two files, one file contains the list of courses and enrolments and the other file contains a list of student course selections. All files can be retrieved from `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/`. The following is a list of data files related to benchmark datasets:

1. **Course files:** (end with the extension `.crs`). The course data file is in the following format:

`<eCode> <enrl>`

where:

`<eCode>` is examination code

`<enrl>` is total enrolment for examination `<eCode>`

e.g.

0001 10

0002 9

.

`<eof>`

The course file is in ascending order, sorted by `<eCode>`.

2. **Student files** (end with extension `.stu`): The student file is in the following format:

`<eCode[1]> <eCode[2]> <eCode[3]> ....<eCode[n]>`

where:

`<eCode[1]>..<eCode[n]>` are list of courses taken by a student.

e.g.

0001 0005 0010 0100

0003 0005 0013 0123

<eof>

The student course selections are sorted in ascending order.

3. **Conflict graph files** (end with extension .cns): The file is produced using Construct Conflict Graph algorithm (Chapter 4). The file is in the following format:

<eid> <snu> <gnu><genu> <cne[0]> <cns[0]> <cne[1]>

where:

<eid> is examination id

<snu> is total student enrolment for examination <eid>

<gnu> is total number of students in conflict with examination <eid>

<genu> is number of examinations in conflict with <eid>

<cne[i]> is examination id of the i^{th} examination in conflict with examination <eid>

<cns[i]> is number of students for examination cne[i] in conflict with examination <eid>

The conflict graph file is sorted by number of examinations in conflict in descending order.

4. **Solution file** (end with extension .s06): The file is produced using Construct Initial Solution Algorithm (Chapter 4). The file is in the following format:

<eCode> <eid> <SlotNumber>

where:

<eCode> is examination code

<eid> id examination identifier

<SlotNumber> is initial slot for examination <eid> (slot zero if <eid> is unassigned)

UiTM Dataset

The MARA University of Technology examination timetabling dataset comes in four files: student file, course file, concurrent file, examination timetable. From these four files and other relevant data, we can generate the following files to be used with an examination timetabling solver.

1. **Student files** (end with extension .stl). There are 13 student files, one file for each campus. Another student file contain a list of all students in all campuses. The file is in the following format:

<StudentID> <CampusCode><course[1]>...<course[n]>

e.g.

97340897	S	ARB102	MGT318	ACC161
97340913	S	MGT318	MGT358	MGT375
97345197	R	MAT153	FIN310	

<eof>

The student data file is sorted in ascending order by *studentID*

2. **Course file**: The original course file does not contain total enrolment for each course. Therefore we generate enrolment using the student files. There are two enrolments where the first enrolment has weekend type A and the second enrolment has weekend type B. The file is in the following format:

<CourseCode><EnrolmentTypeA><EnrolmentTypeB>

ACC027	33	21
ACC030	208	16

<eof>

The course data file is sorted in ascending order by *<CourseCode>*.

3. **Concurrent file** is in the following format:

<Examination1><Examination2>

ACC100	ACC100
ACC100	ACC101
ACC100	ACC105
ACC100	ACC111
ACC100	ACC114
ACC161	ACC161
ACC161	ACC164
ACC211	ACC165
ACC211	ACC211
ACC211	MAC392
.....	

The above shows which examinations must be scheduled in the same time slot. Example: ACC100 must be scheduled at the same time as ACC101, ACC105, ACC111 and ACC114.

4. The **CampusCategory** file is in the following format:

<CampusCode><CampusCategory>

E.g.

A	A
R	A
D	B
T	B

The above shows that campus A and R are of type A indicating weekend type A.

5. **Weekend file:** The file is generated using the start and end date of the examination period. The data is 1 if it is a weekend slot and 0, otherwise. Two files are generated: one for weekend type A and one for weekend type B. Type A weekend begins Saturday afternoon and whole day Sunday while Type B weekend begins Thursday afternoon and whole day Friday.

e.g.

1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 <eof>

The above shows that slot 1 and slot 2 are weekend slots and etc.

6. **Conflict graph file:** This file is similar to the conflict graph file generated for benchmark datasets.
7. **Solution file:** The format for this file is similar to the solution file generated for benchmark datasets. The only difference is a different algorithm is used to generate the initial solution.

Bibliography

1. Aarts, E. and Korst, J. (Eds.) (1998). Simulated Annealing and Boltzman Machines. Wiley.
2. Aarts, E. and Lenstra, J. K. (Eds) (1997) Local search in combinatorial optimisation. Chichester: John Wiley and Sons.
3. Abdullah, S., Ahmadi, S., Burke, E. K. and Dror, M. (2005) Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. In revision for OR Spektrum.
4. Abramson, D. (1991) Constructing school timetables using simulated annealing: sequential and parallel algorithms. Management Science, Volume 37 Issue 1, pp. 98-113.
5. Abramson, D. A., Dang, H., and Krisnamoorthy, M. (1999) Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research, Volume 16, pp. 1-22.
6. Appleby, J. S., Blake, D. V. and Newman, E. A. (1960) Techniques for producing school timetables on a computer and their application to other scheduling problems. The Computer Journal, Volume 3, pp. 237-245.
7. Asmuni, H., Burke, E. K., Garibaldi, J. M. and McCollum, B. (2004) Fuzzy multiple heuristic orderings for examination timetabling. In: Practice and Theory of Automated Timetabling V, E. K. Burke and M. Trick (Eds.), Lecture Notes in Computer Science Vol. 3616, Springer-Verlag, pp. 334 - 353.
8. Awad, R. and Chinneck, J. (1998) Proctor assignment at Carleton University. In: Interfaces Volume 28, No. 2, pp. 58-71.

9. Balakrishnan, N., Lucena, A., and Wong, R. T. (1992) Scheduling examinations to reduce second order conflicts. In: *Computers and Operations Research* 19, pp. 353-361.
10. Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C. and Stewart, W. R. (1995) Designing and reporting on computational experiments with heuristic methods. In: *Journal of Heuristics*, Volume 1, pp. 9-32.
11. Battiti, R. (1996) Reactive search: Toward self-tuning heuristics. In: *Modern Heuristic Search Methods*, V. J. Rayward-Smith (Ed.), John Wiley and Sons Ltd, pp. 61-83.
12. Battiti, R. and Tecchiolli, G. (1994) The reactive tabu search. *Journal on Computing*, Vol. 6, Issue 2, pp. 126-140.
13. Brelaz D. (1979) New methods to colour the vertices of a Graph. *Communications of the A.C.M.* 7, pp. 494-498.
14. Bremermann, H. (1962) Optimisation through evolution and recombination. In: *Self-Organising Systems*, M. Yovits, G. T. Jacobi and G. Goldstine (Eds.), Spartan Books, Washington DC, pp. 93-106.
15. Broder, S. (1964) Final examination scheduling. In: *Communications of the ACM*, 7, pp. 494–498.
16. Broder, S. (1966) On the problem of timetable construction. In: *Communications of the ACM*, 9: pp. 257.
17. Broder, S. (1968) A heuristic algorithm for timetable construction. *Journal of Educational Data Processing*, 5(3): pp.117–123.

18. Burke, E. K. and Carter, M. W. (Eds.) (1998) Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, Volume 1408, Springer-Verlag.
19. Burke, E. K. and De Causmaecker, P. (Eds.) (2003) Practice and Theory of Automated Timetabling IV, Lecture Notes in Computer Science Volume 2740, Springer-Verlag.
20. Burke, E. K. and Erben, W. (Eds.) (2001) Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science, Volume 2079, Springer-Verlag.
21. Burke, E. K. and Ross, P. (Eds.) (1996) Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science, Volume 1153, Springer-Verlag.
22. Burke, E. K. and Trick, M. (Eds.) (2005) Practice and Theory of Automated Timetabling V, Lecture Notes in Computer Science, Volume 3616, Springer-Verlag.
23. Burke, E. K. and Newall, J. P. (1999) A multi-stage evolutionary algorithm for the timetable problem. The IEEE Transactions on Evolutionary Computation, Volume 3.1, pp. 63-74.
24. Burke, E. K. and Newall, J. P. (2003) Enhancing timetable solutions with local search methods. In: Practice and Theory of Automated Timetabling IV, E. K. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science Vol. 2740, Springer-Verlag, pp. 195-206.
25. Burke, E. K. and Newall, J. P. (2004) Solving examination timetabling problems through adaption of heuristic orderings. Annals of Operations Research 129 (1-4): pp. 107-134.

26. Burke, E. K. and Landa Silva, J. D. (2004) The design of memetic algorithms for scheduling and timetabling problems. In: Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing, N. Krasnogor, W. Hart and J. Smith (Eds.), Vol. 166, Springer-Verlag, pp. 289-312.
27. Burke, E. K. and Petrovic, S. (2002) Recent research directions in automated timetabling. European Journal of Operational Research 140: pp. 266–280.
28. Burke, E. K., Elliman, D. G. and Weare, R. F. (1994a) A university timetabling system based on graph colouring and constraint manipulation. Journal of Research on Computing in Education, 27(1), pp.1-18.
29. Burke, E. K., Elliman, D. G. and Weare, R. F. (1994b) A genetic algorithm based university timetabling system. In: 2nd East-West International Conference on Computer Technologies in Education, Volume 1, pp. 35-40.
30. Burke, E. K., Elliman, D. G. and Weare, R. F. (1995) A hybrid genetic algorithm for highly constrained timetabling problems. In: Proceedings of the 6th International Conference on Genetic Algorithms, pp. 605-610.
31. Burke, E. K., Elliman, D. G., Ford, P. H. and Weare, R. F. (1996a) Examination timetabling in British universities - a survey. In: Practice and Theory of Automated Timetabling, E.K. Burke and P. Ross (Eds.), Lecture Notes in Computer Science Volume 1153, Springer-Verlag, pp. 76-92.
32. Burke, E. K., Newall, J. P. and Weare, R. F. (1996b) A memetic algorithm for university exam timetabling. In: Practice and Theory of Automated Timetabling, E. K. Burke and P. Ross (Eds.), Lecture Notes in Computer Science Volume 1153, Springer-Verlag, pp. 241-250.

33. Burke, E. K., Newall, J. P. and Weare, R. F. (1998) Initialisation strategies and diversity in evolutionary timetabling. *IEEE Transactions on Evolutionary Computation*, Volume 6.1, pp.81 - 103.
34. Burke, E. K., MacCarthy, B., Petrovic S. and Qu., R. (2001a) Case-based reasoning in course timetabling: An attribute graph approach. In: *Proceedings of the 4th. International Conference on Case-Based Reasoning, Lecture Notes in Artificial Intelligence*, Volume 2080, Springer-Verlag, pp. 90-104.
35. Burke, E. K., Bykov, Y. and Petrovic, S. (2001b) A Multicriteria approach to examination timetabling. In: *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), *Lecture Notes in Computer Science*, Volume 2079, Springer-Verlag, pp. 118-131.
36. Burke, E. K., Cowling, P. and Landa Silva, J. D. (2001c) Three methods to automate the space allocation process in UK universities. In: *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), *Lecture Notes in Computer Science*, Volume 2079, Springer-Verlag, pp. 254-272.
37. Burke E. K., Petrovic, S. and Qu. R. (2002) Case-based heuristic selection for examination timetabling. In: *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning*, pp. 277-281.
38. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S. (2003a) Hyper-heuristics: An emerging direction in modern search technology. In: *Handbook of Meta-Heuristics*, F. Glover and G. Kochenberger (Eds.), Chapter 16, Kluwer, Dordrecht, pp. 457–474.

39. Burke, E. K., Kendall, G. and Soubeiga, E. (2003b) A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9, pp. 451–470.
40. Burke, E. K., Bykov, Y., Newall, J. and Petrovic, S. (2004a) A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, Volume 36, No. 6, pp. 509-528.
41. Burke, E. K., de Werra, D. and Kingston, J., (2004b), Chap 5.6: Applications to timetabling. In: *Handbook of Graph Theory*, J. Gross and J. Yellen (Eds.), Chapman Hill/CRC Press, pp. 445-474.
42. Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S. and Qu, R. (2004c) Analysing similarity in exam timetabling. In: *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, pp. 89-106.
43. Burke, E. K., De Causmaecker, P., Vanden Berghe, G. and Van Landeghem, H. (2004d) The state of the art of nurse rostering. *Journal of Scheduling*, Vol. 7, No. 6, pp. 441-499.
44. Burke, E. K., Dror, M., Petrovic, S. and Qu, R. (2005a) Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems. In: *Proceedings of the Ninth INFORMS Computing Society Conference*, pp. 79 - 91.
45. Burke E. K., Landa Silva J.D. and Soubeiga, E. (2005b) A multiobjective hyperheuristic approach for space allocation and timetabling. To appear in: Ibaraki T., Nonobe K., Yagiura M. (Eds.), *Meta-heuristics: Progress as Real Problem Solvers*, Springer, 2005.
46. Burke, E. K., Meisels, A., Petrovic, S., Qu, R. and McCollum, B. (2006a) A Graph-Based Hyper Heuristic for Timetabling Problems. Accepted for

publication in the European Journal of Operational Research, to appear 2006.

47. Burke, E. K., Mac Carthy, B., Petrovic, S. and Qu, R. (2006b) Multiple-retrieval case based reasoning for course timetabling problems. Accepted for publication in the Journal of the Operational Research Society, to appear 2006.
48. Burke, E. K., Petrovic, S. and Qu, R. (2006c) Case based heuristic selection for timetabling problems. Accepted for publication in the Journal of Scheduling, volume 9 issue 2, 2006.
49. Burke, E. K., De Causmaecker, P., Petrovic, S. and Vanden Berghe, G. (2006d) Metaheuristics for handling time interval coverage constraints in nurse scheduling. Accepted for publication in Applied Artificial Intelligence, to appear 2006.
50. Cambazard, H., Demazeau, F., Jussien, N. and David P. (2004) Interactively solving school timetabling problems using extensions of constraint programming. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds.), pp. 107-124.
51. Caprara, A., Fischetti, M., Guida, P. L., Monaci, M., Sacco, G. and Toth, P., (2001) Solution of real-world train timetabling problems. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, pp. 1057 -1066.
52. Caramia, M., Dell' Olmo, P. and Italiano, G. F. (2001) New algorithms for examination timetabling. In: Algorithm Engineering 4th International Workshop, S. Näher and D. Wagner (Eds), Lecture Notes in Computer Science, Volume 1982, Springer-Verlag, pp. 230-241.

53. Carter, M. W. (1986) A survey of practical applications of examination timetabling algorithms. *Operations Research Society of America*, Volume 34 No 2, pp. 193-202.
54. Carter, M. W. (1997) EXAMINE: A general examination timetabling system. *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, pp. 363.
55. Carter M. W. and Johnson, D. G. (2001) Extended clique initialisation in examination timetabling. *Journal of the Operational Research Society* (2001) 52, pp. 538-544.
56. Carter, M. W. and Laporte, G. (1996) Recent developments in practical examination timetabling. In: *The Practice and Theory of Automated Timetabling*, E. K. Burke and P. Ross (Eds), *Lecture Notes in Computer Science* Vol. 1153, Springer-Verlag, pp. 3-21.
57. Carter, M. W., Laporte, G. and Lee, S. Y. (1996) Examination timetabling: algorithmic strategies and applications. *Journal of the Operational Research Society* Volume 47 Issue 3, pp. 373-383.
58. Casey, S. and Thompson, J. (2003) GRASPing the examination scheduling problem. In: *Practice and Theory of Automated Timetabling*, E. K. Burke and P. De Causmaecker (Eds.), *Lecture Notes in Computer Science* Vol. 2740, Springer-Verlag, pp. 232 –244.
59. Cole, A. J. (1964) The preparation of examination timetables using a small-store computer, *Computer Journal* 7, pp. 117-121.
60. Colorni, A., Dorigo, M. and Maniezzo, V. (1998) Metaheuristic for high-school timetabling. *Computational Optimisation and Applications*, 9, Kluwer Acad. Publ., Dodrecht, NL, pp. 275-298.

61. Comm, C.L. and Mathaisel, D.F.X (1988) College course scheduling. A market for computer software support. *Journal of Research of Computing in Education*, Volume 21, pp. 187-195.
62. Cooper, T. B. and Kingston, J. H. (1996) The complexity of timetable construction problems. In: *Practice and Theory of Automated Timetabling*, E. K. Burke and P. Ross (Eds), *Lecture Notes in Computer Science Volume 1153*, Springer-Verlag, pp. 283-295.
63. Corne, D. and Ross, P. (1995) Comparing genetic algorithms, simulated annealing and stochastic hill climbing on timetabling problems. In: *Evolutionary Computing*, T. E. Fogarty (Ed.), *Lecture Notes in Computer Science Volume 993*, Springer-Verlag, pp. 94-102.
64. Corne, D. and Ross, P. (1996) Peckish initialisation strategies for evolutionary timetabling. In: *Practice and Theory of Automated Timetabling*, E. K. Burke and P. Ross (Eds.), *Lecture Notes in Computer Science Volume 1153*, Springer-Verlag, pp. 227–240.
65. Corne, D., Dorigo, M. and Glover, F. (Eds.) (1999) *New Ideas in Optimisation*, McGraw Hill.
66. Côté, P., Wong, T. and Sabourin, R. (2005) A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In: *Practice and Theory of Automated Timetabling V*, E. K. Burke and M. Trick (Eds.), *Lecture Notes in Computer Science Volume 3616*, Springer-Verlag, pp. 294-312.
67. Cowling, P., Kendall, G. and Soubeiga, E. (2001) A hyperheuristic approach to scheduling a sales summit. In: *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), *Lecture Notes in Computer Science Volume 2079*, Springer-Verlag, pp. 176-190.

68. Cowling, P., Kendall, G. and Han, L. (2002a) An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling Problem. In Proceedings of Congress on Evolutionary Computation (CEC2002), pp. 1185-1190.
69. Cowling, P., Kendall, G. and Mohd Hussin, N. (2002b) A survey and case study of practical examination timetabling problems. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Gent, 2002, pp. 258-261.
70. Cowling, P., Kendall, G and Soubeiga, E. (2002c) Hyperheuristics: A robust optimisation method applied to nurse scheduling. Seventh International Conference on Parallel Problem Solving from Nature, PPSN2002, Springer LNCS, pp. 851-860.
71. Cowling, P., Kendall, G. and Soubeiga, E. (2002d) Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In: 2nd European Conference on Evolutionary Computing for Combinatorial Optimisation, S. Cagani, J. Gottlieb, E. Hart, M. Middendorf, R. Günther (Eds.), Lecture Notes in Computer Science Volume 2279, Springer-Verlag, pp. 1–10.
72. Csima, J. and Gotlieb, C. C. (1964) Tests on a computer method for construction school timetables. Communication of the ACM, Volume 7, Number 3, pp. 160-163.
73. Dawkins, R. (1990) The Selfish Gene, Oxford University Press.
74. De Jong, K. (1975) An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. thesis, Department of Computer and Communication Sciences, University of Michigan. Available at http://cs.gmu.edu/~eclab/kdj_thesis.html.

75. Desroches, S., Laporte, G. and Rousseau, J. M. (1978) Horex: A computer program for the construction of examination schedules, *INFOR* 16, pp. 294-298.
76. de Werra, D. (1985) An introduction to timetabling. *European Journal of Operational Research* 19, pp. 151-162.
77. de Werra, D. (1997) The combinatorics of timetabling. *European Journal of Operational Research* 96, pp. 504-513.
78. Di Gaspero, L. (2002) Recolour, shake and kick: A recipe for the examination timetabling problem. In: *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, E.K. Burke and P. De Causmaecker (Eds.), pp. 404-407.
79. Di Gaspero, L. and Schaerf, A. (2001) A tabu search techniques for examination timetabling. In: *Practice and Theory of Automated Timetabling III*, E. K. Burke and W. Erben (Eds.), *Lecture Notes in Computer Science*, Volume 2079, Springer-Verlag, pp 104 -117.
80. Di Gaspero, L. and Schaerf, A. (2003) Multi-neighbourhood local search with application to course timetabling. In: *Practice and Theory of Automated Timetabling IV*, E. K. Burke and P. De Causmaecker (Eds.), *Lecture Notes in Computer Science*, Volume 2740, Springer-Verlag, pp. 262-275.
81. Dorigo M., Maniezzo, V. and Colorni, A. (1996). Ant System: Optimisation by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1), pp. 29-41.
82. Dowsland, K. A. (1995) Chapter 2: Simulate annealing. In: *Modern heuristics techniques for combinatorial problems*, C. Reeves, (Ed.), McGraw Hill, pp. 20-69.

83. Dowsland, K. A. (1998) Off-the-peg or made-to-measure? Timetabling and scheduling with SA and TS. In: Practice and Theory of Automated Timetabling II, E. K. Burke and M. Carter (Eds.) Lecture Notes in Computer Science, Volume 1408, Springer-Verlag, pp. 37-52.
84. Dueck, G. (1993) New optimisation heuristics for the great deluge algorithm and the record-to-record travel. Journal of Computational Physics, Volume 104, pp. 86-92.
85. Easton, K., Nemhauser, G. and Trick, M. (2004) Sports scheduling. In: The Handbook of Scheduling: Algorithms, Models, and Performance Analysis. J. Leung (Ed.).
86. Erben, W. (2001) A grouping genetic algorithm for graph colouring and timetabling, In: Practice and Theory of Automated Timetabling III, E. K. Burke and W. Erben (Eds), Lecture Notes in Computer Science 2079, Springer-Verlag, pp 132-158.
87. Ergul, A. (1996) GA-based examination scheduling experience at Middle East Technical University. In: Practice and Theory of Automated Timetabling, E. K. Burke and P. Ross (Eds), Lecture Notes in Computer Science, Volume 1153, Springer-Verlag, pp. 212-226.
88. Festa, P. and Resende, M. G. C. (2004) An annotated bibliography of GRASP, European Journal of Operational Research, pp. 1 – 50.
89. Fisher, H. and Thompson, G. L. (1963) Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, Industrial Scheduling, pp. 225–251.
90. Fogel, D. B. and Anderson R. W. (2000) Revisiting Bremermann's genetic algorithm. I. simultaneous mutation of all parameters. In: Proc. 2000

Congress on Evolutionary Computation, IEEE, San Diego, CA, pp. 1204-1209.

91. Fogel, D.B. and Fraser, A. S. (2000) Running races with Fraser's recombination. In: Proc. 2000 Congress on Evolutionary Computation, IEEE, San Diego, CA, pp. 1217-1222.
92. Forster, G. (1995) Syllabus Plus: A state-of-the-art planning and scheduling system for universities and colleges. In: Proceedings of the 1st Intl. Conference on the Practice and Theory of Automated Timetabling PATAT 1, pp. 244-252.
93. Foxley, E. and Lockyer, K. (1968) The construction of examination timetables by computer. The Computer Journal, Volume 11, Issue 3, pp. 264-268.
94. Fraser, A. (1957) Simulation of genetic systems by automatic digital computers. Australian Journal of Biological Science, 10, pp. 484-491.
95. Glover, F. (1977) Heuristics for integer programming using surrogate constraints. Decision Sciences 8(1), pp. 156-166.
96. Glover, F. (1986) Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13, pp. 533-549.
97. Glover, F. W. and Kochenberger, G. A. (Eds.) (2003). Handbook of Metaheuristics. Kluwer Academic Publishers.
98. Glover, F. and Laguna, M. (1997) Tabu search, Boston : Kluwer Academic Publishers.
99. Glover, F., Laguna, M. and Martí, R. (2000) Fundamentals of scatter search and path relinking. In: Control and Cybernetics, vol. 39, no. 3 pp. 653-684.

100. Goldberg, D. (1989), Genetic algorithms in search, optimisation, and machine learning, Addison-Wesley, Reading, MA.
101. Goltz, H.-J. and Matzke, D. (2000) ConBaTT - constraint-based timetabling. Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, E. K. Burke and W. Erben (Eds.), pp. 491.
102. Gratch, J. M. and Chien, S. A. (1996) Adaptive problem-solving for large scale scheduling problems: A case study. Journal of Artificial Intelligence Research Vol. 4, pp. 365-396.
103. Han, L. and Kendall, G., (2003a) Guided operators for a hyper-heuristic genetic algorithm. In: Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence, Tamás D Gedeon and Lance Chun Che Fung (Eds.), pp. 807-820.
104. Han, L. and Kendall, G. (2003b) Investigation of a tabu assisted hyper-heuristic genetic algorithm. In: Proceedings of Congress on Evolutionary Computation, pp. 2230-2237.
105. Han, L., Kendall, G. and Cowling, P. (2002) An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02), pp. 267-271.
106. Hansen, M. P. and Vidal, R. V. V. (1995) Planning of high school examinations in Denmark. European Journal of Operational Research, Volume. 87, pp. 519-534.
107. Hart, E., Ross, P. M. and Nelson, J. (1998) Solving a real-world problem using an evolving heuristically driven schedule builder, Evolutionary Computing, Volume 6 No. 1, pp. 61-80, MIT Press.

108. Hertz, A. (1991) Tabu search for large scale timetabling problems. *European Journal of Operational Research*, vol 54, pp. 39-47.
109. Hertz, A, Taillard, É. D. and de Werra, D. (1997) A Tutorial on tabu search. In: *Local search in combinatorial optimisation*, E. Aarts and J. K. Lenstra (Eds.), J. Wiley & Sons Ltd., pp. 121-136.
110. Holland, J. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Harbor.
111. Ho, W. K, Lim, A. and Oon, W. C. (2001) Maximising paper spread in examination timetabling using a vehicle routing method. *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pp. 335–342.
112. Ibaraki, T., Nonobe, K. and Yagiura, M (Eds.) (2005) *Metaheuristics: Progress as Real Problem Solvers*, Kluwer Academic Publishers.
113. Isaai, M. T. and Singh, M. G., (2001) Hybrid applications of constraint satisfaction and meta-heuristics to railway timetabling: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Volume: 31 Issue: 1, pp. 87-95.
114. Johnson, D. (1990) Timetabling university examinations. *Journal of the Operational Research Society* 41, pp. 39-47
115. JTAP (1998) Central timetabling by computer: A review of existing information. Report by JISC Technology Applications Programme, June 1998 url: <http://www.jtap.ac.uk>
116. Karp, R. M. (1972) Reducibility among combinatorial problems. In: *Complexity of Computer Computations.*, Plenum Press, New York, pp. 85-104.

117. Kendall, G. and Mohd Hussin, N. (2005a) An investigation of a tabu search based hyper-heuristic for examination timetabling. In: Multidisciplinary Scheduling: Theory and Applications, G. Kendall, E. K. Burke, S. Petrovic and M. Gendreau (Eds.), Springer (2005), pp. 309-328. An extended abstract of this paper also appeared in the Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA2003) pp. 226-233.
118. Kendall, G. and Mohd Hussin, N. (2005b) A tabu search hyper-heuristic approach to the examination timetabling problem at The MARA University of Technology. In: Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds), Lecture Notes in Computer Science, Volume 3616, Springer-Verlag, pp. 270-293. An earlier version of this paper also appeared in the Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds.), pp. 199-217.
119. Kendall G. and Mohd Hussin N. (2005c), Control techniques in a tabu search based hyper-heuristic, In: Proceedings of the 6th Metaheuristics International Conference (MIC2005).
120. Kendall, G., Soubeiga, E. and Cowling, P. (2002) Choice function and random hyperheuristics. Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, pp. 667-671.
121. Kendall, G., Burke, E. K., Petrovic, S. and Gendreau, M. (Eds.) (2005) Multidisciplinary Scheduling: Theory and Applications. Selected papers from MISTA 2003, Kluwer Publication.
122. Kingston, J. H. (2004) A tiling algorithm for high school timetabling. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, E. K. Burke and M. Trick (Eds.) pp 233-250.

123. Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimisation by simulated annealing, *Science*, 220, pp. 671-380.
124. Kitano, H. (1990) Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4, pp. 461–476.
125. Kolodner J (1993). *Case-Based Reasoning*. Morgan Kaufmann.
126. Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.
127. Kwok, L.-F., Kong, S.-C., Kam, Y.-Y. (1997) Timetabling in Hong Kong secondary schools. *Computers Education*, Volume 28 Number 3, pp. 173-183.
128. Landa Silva J. D., Burke E. K. and Petrovic S. (2004) An introduction to multiobjective metaheuristics for scheduling and timetabling. In: *Metaheuristic for Multiobjective Optimisation*, X. Gandibleux, M. Sevaux, K. Sorensen, V. T'kindt (Eds.), *Lecture Notes in Economics and Mathematical Systems*, Vol. 535, Springer-Verlag, pp. 91-129.
129. Laporte, G. and Desroches, S (1984) Examination timetabling by computer. *Computers and Operations Research*, Volume 11, Issue 4, pp. 351-360.
130. Leong, T. Y. and Yeong, W. Y. (1987) Examination scheduling: A quadratic assignment perspective. In: *Proceedings of the International Conference on Optimisation: Techniques and Applications*, pp. 550–558.
131. Leong, T. Y. and Yeong, W. Y. (1990) A hierarchical decision support system for university examination scheduling. Working paper, National University of Singapore, September 1990.

132. Lotfi, V. and Cervený, R. (1991) A final-exam-scheduling package. *Journal of the Operational Research Society* 42, pp. 205-216.
133. Marti, R., Lourenço, H. and Laguna, M. (2000) Assigning proctors to exams with scatter search. *Computing Tools for Modeling, Optimisation and Simulation*, M. Laguna and J. L. Gonzalez Velarde (Eds), Kluwer Academic publishers, pp. 215-227.
134. McCollum, B. (1997) The Implementation of a central timetabling system in a large civic university. In: *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, E. K. Burke and M. W. Carter (Eds.), pp 205 - 214.
135. McCollum, B. and Newall, J. (2000) Introducing Optime: Examination timetabling software. In: *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, E. K. Burke and W. Erben (Eds.), pp. 485.
136. Merlot, L. T. G., Boland, N., Hughes, B. D. and Stuckey, P. J. (2003) A hybrid algorithm for the examination timetabling problem. In: *Practice and Theory of Automated Timetabling IV*, E. K. Burke and P. De Causmaecker (Eds.), *Lecture Notes in Computer Science Volume 2740*, Springer-Verlag, pp. 207-231.
137. Metropolis, N., Rosenbluth A. W., Rosenbluth, M. N., Teller A. H. and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6), pp.1087-1092.
138. Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd. Ed., Springer-Verlag.
139. Mirrazavi, S. K., Mardle, S. J. and Tamiz, M. (2003) A two-phase multiple objective approach to university timetabling utilising optimisation and

evolutionary solution methodologies. *Journal of the Operational Research Society* 54, pp.1155 - 1166.

140. Mitchell, M. (1996) *An introduction to genetic algorithms*, Massachusetts Institute of Technology, USA.
141. Moscato, P. (1989) *On evolution, search, optimisation, genetic algorithms and martial arts: towards memetic algorithms*, Report 826, Caltech Concurrent Computation Program, California Institute of Technology.
142. Moscato, P. (1999). *Memetic algorithms: A short introduction*. In: *New Ideas in Optimisation*, D. Corne, M. Dorigo and F. Glover (Eds.), McGraw Hill, pp. 219-234.
143. Nareyek, A. (2003) *Choosing search heuristics by non-stationary reinforcement learning*. In: *Metaheuristics: Computer Decision-Making*, M. G. C. Resende and J. P. de Sousa (Eds.), Kluwer Academic Publishers, pp. 523-544.
144. Norman, M. G. and Moscato, P. (1989) *A competitive and cooperative approach to complex combinatorial search*, Report 790, Caltech Concurrent Computation Program, California Institute of Technology. (expanded version published at the *Proceedings of the 20th Informatics and Operations Research Meeting*, Buenos Aires, August 1991).
145. Osman, I. H. and Kelly, J. P. (Eds.) (1996) *Meta-heuristics: Theory and applications*, Kluwer Academic Publishers.
146. Osman, I. H. and Laporte, G. (1996). *Metaheuristics: A bibliography*. *Annals of Operations Research*, 63, pp. 513-623.
147. Paechter B., Fogarty T. C., Cumming A. and Rankin R.C. (2000), *Stone Soup?* In: *Proceedings of the 3rd International Conference on the Practice*

and Theory of Automated Timetabling, E. K. Burke and W. Erben (Eds.), pp 103-106.

148. Paquete, L. and Stützle, T. (2004) Empirical analysis of tabu search for the lexicographic optimisation of the examination timetabling problem. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), E. K. Burke and P. de Causmaecker (Eds.), pp. 413-420.
149. Peck, J. E. L. and Williams, M. R. (1966) Algorithm 286: Examination scheduling. Communications of the ACM, 9, pp. 433–434.
150. Petrovic, S. and Burke, E. K. (2004) University timetabling. In: The Handbook of Scheduling: Algorithms, Models, and Performance Analysis. J. Leung (Ed.)
151. Petrovic, S. and Bykov, Y., (2002) A multiobjective optimisation technique for exam timetabling based on trajectories. In: Practice and Theory of Automated Timetabling (E.K. Burke and P. de Causmaecker (Eds.), Lecture Notes in Computer Science Vol. 2740, Springer-Verlag, pp. 181-194.
152. Qu, R. (2002). Case-based reasoning for course timetabling problem. PhD Thesis, School of Computer Science and Information Technology, The University of Nottingham.
153. Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK.
154. Reeves, C. (Ed.) (1995) Modern heuristics techniques for combinatorial problems, McGraw Hill.

155. Registrar (2001) Pamphlet from Examination Unit, Office of Registrar, University Technology MARA.
156. Resende, M. G. C. and Ribeiro, C. C. (2003) Greedy randomised adaptive search procedures. In: Handbook of Metaheuristics, F. Glover and G. Kochenberger (Eds.), Kluwer Academic Publishers, pp. 219-249.
157. Resende, M. G. C. and Ribeiro, C. C. (2005) GRASP with path-relinking: Recent advances and applications. In: Metaheuristics: Progress as Real Problem Solvers, T. Ibaraki, K. Nonobe and M. Yagiura, (Eds.), Springer, pp. 30-63.
158. Ribeiro, C. C. and Hansen, P. (2001) Essays and Surveys in Metaheuristics, Kluwer Academic Publishers.
159. Ribeiro Filho, G. and Lorena, L. A. N (2001). A constructive evolutionary approach to school timetabling. In: Applications of Evolutionary Computing, E. J .W Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl and H. Tijink (Eds.), Lecture Notes in Computer Science, Volume 2037, Springer-Verlag, pp. 130-139.
160. Rogalla, S. (1997) CELCAT: A practical solution to scheduling problems. In: Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling, pp. 368.
161. Romero, B. P. (1982) Examination scheduling in a large engineering school: A computer-assisted participative procedure. Interfaces 12, pp. 17-23.
162. Ross, P., Corne, D., Fang, H. (1994) Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation, PPSN III, Springer Verlag.

163. Ross, P., Hart, E. and Corne, D. (1998) Some observations about GA-based exam timetabling. In: Practice and Theory of Automated Timetabling II, E. K. Burke and M. Carter (Eds.), Lecture Notes in Computer Science Volume 1408, Springer-Verlag, pp. 115-129.
164. Ross, P., Schulenburg, S., Marín-Blázquez, J. G. and Hart, E. (2002) Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. GECCO 2002, pp. 942-948.
165. Rossi-Doria, O., Samples, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L. and Stützle, T. (2003) A comparison of the performance of different metaheuristics on the timetabling problem. In: Practice and Theory of Automated Timetabling IV, E. K. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science Volume 2740, Springer-Verlag, pp. 329-354.
166. Salhi, S. (2002) Defining tabu list size and aspiration criterion within tabu search methods. Computers and Operations Research 29, pp. 67-86.
167. Salman, F. S., Kalagnaman, J. R., Murthy, S. and Davenport, A. (2002) Cooperative strategies for solving bicriteria sparse multiple knapsack problem. Journal of Heuristics, 8, pp. 215-239.
168. Schaerf, A. (1999) A survey of automated timetabling. Artificial Intelligence Review. n.13, pp. 87-127.
169. Schaerf, A. and Di Gaspero, L. (2001) Local search techniques for educational timetabling problems. In: Proceeding of the 6th International Symposium on Operational Research in Slovenia, pp.13-23.

170. Schaerf, A. and Schaerf, M. (1995) Local search techniques for high school timetabling. In: Proceedings of the 1st Intl. Conference on the Practice and Theory of Automated Timetabling, pp. 313-323.
171. Soubeiga, E. (2003) Development and application of hyperheuristics to personnel scheduling. PhD Thesis, Department of Computer Science, University of Nottingham, UK.
172. Storer, R. H., Wu, D. S. and Vaccari, R. (1995) Problem and heuristic space search strategies for job shop scheduling. In: ORSA Journal on Computing, 7(4), Fall, pp. 453-467.
173. Taillard, É. D., Gambardella, L. M., Gendreau, M. and Potvin, J. Y. (2001) Adaptive memory programming: A unified view of meta-heuristics. European Journal of Operational Research 135 (1), pp. 1-16.
174. Talbi, E. G. (2002) A taxonomy of hybrid metaheuristics. Journal of Heuristics, 8: pp. 541–564.
175. Terashima-Marín, H., Ross, P.M. and Valenzuela-Rendón, M. (1999a) Clique-based crossover for solving the timetabling problem with GAs. In: Proceedings of the Congress on Evolutionary Computation, pp. 1200-1206.
176. Terashima-Marín, H., Ross, P.M. and Valenzuela-Rendón, M. (1999b) Evolution of constraint satisfaction strategies in examination timetabling. In: Proceedings of the Genetic and Evolutionary Conference, pp. 635-642.
177. Thompson, J. M. and Dowsland, K. A. (1996) General cooling schedules for a simulated annealing based timetabling system. In: Practice and Theory of Automated Timetabling, E. K. Burke and P. Ross (Eds.), Lecture Notes in Computer Science, 1153, Springer-Verlag, pp. 345-363.

178. Thompson, J. M. and Dowsland, K. A. (1998) A robust simulated annealing based examination timetabling system. *Computers Operational Research* Vol. 25, No 7/8, pp. 637-648.
179. Trick, M.A. (2001) A schedule-then-break approach to sports timetabling. In: *The Practice and Theory of Automated Timetabling III*, E.K. Burke and W.Erben (Eds.), *Lecture Notes in Computer Science*, 2079, Springer-Verlag, pp. 242-253.
180. Voß, S. (2001) Meta-heuristics: The State of the Art. In: *Local Search for Planning and Scheduling*, pp. 1-23.
181. Voß, S., Martello, R., Osman, I. H. and Roucairol, C. (Eds.) (1999) *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers.
182. Wan Ya and Baharudin, N. (2001), Interview with Manager and System Analyst, Examination Unit, Center for Integrated Information System, MARA University of Technology, August 2001.
183. Weare, R. F. (1995) Automated examination timetabling. PhD thesis, Department of Computer Science, University of Nottingham, U.K.
184. Welsh, D. J. A. and Powell, M. B. (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, Volume 10, Issue 1, pp. 85-86.
185. White, G. M. and Chan, P. W. (1979) Towards the construction of optimal examination timetables. *INFOR* 17, pp. 219-229.
186. White, G. M. and Xie, B. S. (2001) Examination timetables and tabu search with longer term memory. In: *Practice and Theory of Automated*

Timetabling III, E. K. Burke and W. Erben (Eds.), Lecture Notes in Computer Science, 2079, Springer-Verlag, pp. 85-103.

187. White, G.M., Xie, B.S. and Zonjic, S. (2004) Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research* 153, pp. 80–91.
188. Williams, H. P. (1999) *Model building in mathematical programming*, Chichester, Wiley.
189. Wolpert, D. and Macready, W. (1995) No free lunch theorems for search. References. Technical Report SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, USA.
190. Wolpert, D. and Macready, W. (1997) No free lunch theorems for optimisation. *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 67-82.
191. Wong, T., Côté, P., and Gely, P. (2002) Final Exam Timetabling: A Practical Approach. In: *Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 726-731.
192. Wong, T., Côté, P. and Sabourin, R. (2004) Hybrid MOEA for the capacitated exam proximity problem. In: *IEEE Congress On Evolutionary Computation*, pp. 1495-1501.
193. Wood, D.C. (1968) A system for computing university examination timetables. *The Computer Journal*, Volume 11, Issue 1, pp. 41-47.
194. Wren, A. (1996) Scheduling, timetabling and rostering - a special relationship? In: *Practice and Theory of Automated Timetabling*, E. K. Burke and P. Ross (Eds), Lecture Notes in Computer Science Volume 1153, Springer-Verlag, pp. 46-76.

195. Wright, M. (2001) Subcost-guided search - Experiments with timetabling problems. *Journal of Heuristics*, 7(3), Kluwer Academic Publishers, pp. 251-260.
196. Yong Y. and Petrovic, S. (2005), A novel similarity measure for heuristic selection in examination timetabling. In: *Practice and Theory of Automated Timetabling V*, E. K. Burke and M. Trick (Eds), *Lecture Notes in Computer Science Volume 3616*, Springer-Verlag, pp. 247-269.