AN INVESTIGATION OF A TABU-SEARCH-BASED HYPER-HEURISTIC FOR EXAMINATION TIMETABLING

Graham Kendall and Naimah Mohd Hussin
Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science and Information Technology,
University of Nottingham, Nottingham NG8 1BB, UK

Abstract This paper investigates a tabu-search-based hyper-heuristic for solving examination timetabling problems. The hyper-heuristic framework 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. Experiments carried out on examination timetabling datasets from the literature show that this approach is able to produce good quality solutions.

Keywords: hyper-heuristic, examination timetabling, heuristics, tabu search.

1. INTRODUCTION

This paper investigates a hyper-heuristic, based on tabu search, and its application to examination scheduling. The objective is to design a generic system that is able to select the most appropriate algorithm for the current instance of a given timetabling problem. Carter (1986), Carter and Laporte (1996) and Schaerf (1999) have conducted comprehensive surveys on 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 data sets whilst others performed better when presented with different data sets. 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).

This paper will report on our 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 of applying the best heuristics at each decision point. We test our approach on examination timetabling problem using examination timetabling problem dataset publicly available at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/.

The next section reviews the use of hyper-heuristic methodologies and tabu search related to timetabling problems. Section 2 gives a description of the examination timetabling problem. We describe our hyper-heuristic framework and strategy in Section 3 and Section 4 gives our experimental results and analysis. Section 5 concludes with a summary and presents future research directions.

1.1 Hyper-heuristics

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

One example of solving a large-scale university examination timetable problem using a hyper-heuristic approach can be seen in Terashima-Marin *et al.* (1999). Their approach had two phases in the construction of a timetable. Each phase used a different set of heuristics and a switch condition determined when to move from one phase to the other. A genetic algorithm, using a non-direct chromosome representation, was used to evolve the choice of heuristics, switch condition and strategies.

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 exams 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).

Cowling *et al.* (2001) use a choice function in their hyper-heuristic to determine which low level heuristic will be called next. The choice function adaptively ranks the low-level heuristics by considering 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 (Cowling *et al.*, 2002c) and project presentation scheduling (Cowling *et al.*, 2002b; Kendall *et al.*, 2002).

Cowling *et al.* (2002a) use a genetic algorithm based hyper-heuristic (Hyper-GA) to construct a sequence of heuristics that are applied to a trainer scheduling problem.

Nareyek (2001) proposed a learning procedure in a search process that learns to select promising heuristics based on weight adaptation. Their empirical study was carried out on two problems: Orc Quest and Logistics Domain.

Burke *et al.* (2003a) have used a tabu search hyper-heuristic (although different to the one proposed in this paper) and have successfully applied it to course timetabling and rostering problems. They used a ranking mechanism to dynamically rank each low-level heuristics. 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. Our hyper-heuristic differs with respect to how we use the tabu list and how we choose and apply heuristic. Further details are given in Section 3.

There are other papers published on methods that are similar to the concept used in hyper-heuristics. It is not our intention to mention all of them, but nevertheless, it would be interesting to carry out a complete survey and categorise all papers that exhibit hyper-heuristic behaviour. From what we have seen 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.

1.2 Tabu Search (Timetabling)

In this section, we will discuss briefly how other researchers apply tabu search approaches in solving timetabling problem. The basic form of tabu search (TS) is an idea proposed by Glover (1986) to solve combinatorial optimisation problems. The following is a definition 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.

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.

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_{mxp} 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 improvements 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 criterion* 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 exams are to be moved, they maintain two violation lists: list of exams that violate either hard or soft constraints and list of exams 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 the examination timetabling problem, 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 there was 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 single neighbourhood.

Thomson and Dowsland (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. They apply varying *tabu restriction* on different moves and use a frequency-based *diversification* mechanism and penalised attributes that occur very frequently. Some of the modifications included can improve 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 the 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 *et al.* (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 focus 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 significantly improved results.

The tabu search meta-heuristic has been explored in detail and applied to the examination timetabling problem by the above researchers. The main issues that were addressed and can be explored further are as follows:

- How can we use memory to help in storing history of previous moves (adaptive, short-term, long-term etc.)?
- What items should be stored in the tabu list?
- Neighbourhood size.
- Type of moves that dictate the next neighbour of a solution state.
- How to balance and decide when to intensify and diversify the search?
- Conditions for tabu restriction.
- Factors that affect tabu tenure?
- What aspiration criteria can be used to avoid missing a potentially good solution?

We incorporate some of the above issues into our hyper-heuristic framework and apply it to the examination timetabling problem.

2. PROBLEM DESCRIPTION

Timetabling is a special case of a scheduling problem (Wren, 1996). The layman's term for a timetable is normally used in an academic environment, which refers to a class timetable or examination timetable. 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 some cases conflict cannot be avoided and the objective is to minimise the number of student conflicts.

We can represent the examination timetabling problem using a mathematical model. From the problem definition we know that it is an assignment type problem because we need to assign examinations to slots while minimising an objective function and satisfying a set of constraints. Thus we can formulate the problem as follows:

- E: a set of m examinations $E_1, E_2, \ldots, E_m$;
- S: a set of n slots $S_1, S_2, \ldots, S_n$;

- A final exam timetable T_{mn} such that $T_{ik} = 1$ if exam i is scheduled in slot k, 0 otherwise;
- A conflict matrix C_{mm} such that C_{ij} = total number of students sitting for both exams i and j;
- P_{ik} is a penalty given if exam i is scheduled in slot k.

The examination timetabling problem is to assign examinations to slots subject to some hard constraints that must be satisfied, and minimise soft constraint violation.

Hard constraints that must be satisfied are:

1 *Feasible.* The timetable must be feasible such that all exams must be scheduled and each exam $(E_1, E_2, \ldots, E_m)$ must be scheduled only once:

$$\sum_{k=1}^{n} T_{ik} = 1 \qquad i = 1, \ldots, m$$

2 *Student conflict.* No student should sit for more than one exam in the same slot. If exam i and exam j are scheduled in slot k, the number of students sitting for both exam i and $j(C_{ij})$ must be equal to zero, and this should be true for all exams 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 exam i is scheduled in slot k, is as follows:

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

where j (an exam in conflict with exam i) is scheduled in slot l.

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

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

Other additional soft constraints that are specific to university requirements can be added to this problem. But in this paper, we apply the same method of evaluating solution quality so that we can compare our results with other results published in the literature.

3. HYPER-HEURISTIC FRAMEWORK AND STRATEGY

A hyper-heuristic framework (Burke *et al.* 2003a,b) works at a higher level of abstraction than current (meta-)heuristic approaches and does not require domain knowledge. It only has access to non-domain-specific information that it receives from the heuristics that it operates upon. The 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. 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 $H_1, H_2, \ldots, H_n$. 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 and return the result via an evaluation function. The low-level-heuristic module can be viewed as a "black box" that hides the implementation detail and only returns a value.

3.1 Hyper-heuristic Module

The hyper-heuristic module is the main part of the research area where we need to design and test strategies that can intelligently select the best heuristic that will help guide the search to either intensify or diversify the exploration of the search region.

The general framework for our hyper-heuristic algorithm is as follows:

```
Step 1. Construct initial solution

Step 2. Do

            Consider heuristics that are not tabu

            Apply chosen heuristic and make the heuristic tabu

            Update solution

          Until terminating condition
```

The initial solution is produced using a constructive heuristic (largest degree or saturation degree (Carter and Laporte, 1996). The initial solution need not be a good solution and it may not be feasible (i.e. some exams are unscheduled). The algorithm works with infeasible solutions since some of the low-level heuristics specialised in scheduling unscheduled exams. Next, a randomisation (randomly move exams to other valid slots) is carried out to start different runs with different 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 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 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 good job is 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 produce 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 (the heuristic search space, which is part of the solution search space) and their task is to find a good solution and return it. 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 exploiting a very small neighbourhood of solutions. In the search for good quality solutions, the hyper-heuristic exhibits a kind of reinforcement learning, which will assist in an intelligent action at each decision point. It monitors the behaviour of each low-level heuristic by storing information about the performance using adaptive memory. Our 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., heuristic number, recent change in evaluation function, CPU time taken to run the heuristic, and tabu status (or tabu duration, as the term we prefer to use). Tabu duration indicates how long a heuristic should remain tabu (0–4) and 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 our tabu restriction conditions. We do not use any aspiration criteria (changing a tabu active status to tabu inactive because the heuristic improves the solution) at this point because we wanted to compare which tabu duration produces the best quality solution. Therefore, the only time a heuristic changes its status from tabu active to tabu inactive is when the tabu duration is zero. The tabu duration is set for a heuristic whenever a tabu restriction is satisfied. After each iteration, the tabu duration is decremented until it reaches zero and the heuristic is now tabu inactive. For each test on the dataset we fixed the tabu duration at between zero and four and, in this paper, we compare to find which tabu duration produces better quality solutions.

We use several strategies when considering the heuristics: consider all heuristics (i.e. no tabu criteria), 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) in the solution search space. At each choice point, we need to decide whether we want 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 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. Heuristics that have been applied become 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. We have implemented the simplest strategy, i.e. hyper-heuristic with fixed tabu duration (HH-FTD), 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.

3.2 Low-Level Heuristics Module

Low-level heuristics are heuristics that allow movement through a solution space that require domain knowledge and are problem dependent. Each heuristic creates its own heuristic search space that is part of the solution search space. 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, Tabu Search or a Memetic Algorithm.

The heuristics change the current state of a problem into a new state by accepting a current solution and returning a new solution. Each low-level heuristic can be considered as an improvement heuristic that returns a move, a change in the penalty function and the amount of time taken to execute the heuristic. 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 (first-order conflict, second-order conflict, etc) or moving an exam into an unscheduled list (exam becomes unscheduled and violates hard constraints).

We have implemented the following low-level heuristics, grouped into four categories:

1 Select and schedule exam. Selecting the next exam to schedule is dependent upon which factor is considered to be important in determining the difficulty of scheduling an exam. The strategies that have been used in the literature, and that are adapted from the graph colouring heuristics, are used here. Once an exam is selected, it will be scheduled into the best available slot that will maximise the reduction in penalty.

    - Largest enrolment: exam with largest enrolment should be selected since it might be difficult to schedule at a later time.
    - Largest exam conflict: exam that is in conflict with the largest number of exams is normally considered to be more difficult to schedule.
    - Largest total student conflict: exam that has maximum total number of students in conflict.
    - Largest exam conflict already scheduled: exam that has the greatest number of exams in conflict already scheduled would be difficult to schedule since it would have less choice of valid slots.
    - Exam with least valid slots: exam that has the least valid slots should be scheduled now since it may not have any slots available at a later stage.

2 Move exam i from location x to y.

    - Select an exam at random and move to another random slot.
    - Move exam i with maximum penalty from randomly selected exams.

- Move exam i with highest second-order conflict from location x to a new location y.
- Move exam i with highest second order conflict from location x to a new location y which maximises the reduction in second order conflict.
- Move exam i with first order conflict from location x to a new location y which does not result in first-order conflict.

3 Swap.

- Random: select an exam at random and find another exam at random which can swap slots.
- Min–max swap: swap the slots for exam with minimum penalty and exam with maximum penalty.

4 Remove. Remove a randomly selected exam from the examinations already scheduled.

All of the above low-level heuristics are either 1-opt or 2-opt and there is also a mixture of some randomness and deterministic selection of exams and slots. We purposely test low-level heuristics with simple moves rather than low-level heuristic with intelligence 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, we want to make the problem-domain knowledge heuristics easy to implement and the hyper-heuristic more generalised.

4. EXPERIMENTAL RESULTS

We have implemented and tested our tabu-search-based hyper-heuristic (TSHH) framework 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. We defined and implemented the hyper-heuristic and heuristics 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 heuristics object 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 complexity of building another system. Thus, we can easily produce solutions to users who require "good enough–soon enough–cheap enough" (Burke *et al.* 2003a, 2003b) solutions to their problems by implementing several domain specific low-level heuristics with simple moves.

Therefore, the objectives of our experiments are:

Table 1. Characteristics of real problems.

Code	Institution	No. of slots	No. of exams	No. of students exams	No. of student density	Conflict matrix
Car-f92	Carleton University, Ottawa	32	543	18,419	55,522	13.8%
Car-s91	Carleton University, Ottawa	35	682	16,925	56,877	12.8%
Ear-f83	Earl Haig Collegiate Institute, Toronto	24	189	1,125	8,109	26.7%
Hec-s92	Ecoles des Hautes Etudes Commercials, Montreal	18	81	2,823	10,632	42.0%
Kfu-s93	King Fahd University Of Petroleum and Minerals, Dharan	20	461	5,349	25,113	5.6%
Sta-f83	St Andrew's Junior High School	13	139	611	5,751	14.4%
Tre-s92	Trent University, Peterborough, Toronto	23	261	4,360	14,901	5.8%
Ute-s92	Faculty of Engineering, University of Toronto	10	184	2,750	11,793	8.5%

- To establish a well defined interface between our hyper-heuristic module and our low-level heuristics module;
- To compare the quality of results produced by our 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 low-level heuristic;
- To determine further improvement in our hyper-heuristic module.

4.1 Datasets

We tested our implementation 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

Table 2. Results with 10 minute and 4 hour run.

	Hyper-heuristic with fixed TD (best penalty value per student from 8 runs)					HH-FTD (TD = 2, 4 h run)	% improve with long run
File	TD = 0	TD = 1	TD = 2	TD = 3	TD = 4		
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%

ftp://ftp.mie.utoronto.ca/pub/carter/testprob/. Table 1 shows the characteristics of each dataset. Each datasets 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. We test our method on eight of the datasets. We use both data files to construct a conflict graph and the largest degree algorithm (Carter *et al.* 1996) to construct the initial solution. The density of the conflict matrix in Table 1 is calculated as the average number of other exams that each exam conflicts with, divided by the total number of exams. For example, a conflict matrix density of 0.5 or 50% indicates that each exam conflicts with half of the other exams on average.

The numbers 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.

4.2 Experimental Results and Analysis

Our hyper-heuristic, with fixed tabu duration (HH-FTD), was tested with eight benchmark datasets. For each dataset, we 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 2, the first column shows the file name of each dataset and the next five columns show our results (best penalty value per student) with a tabu duration varying from 0 to 4. We do not show here the actual time that it finds the best solution but the best results are normally found towards the end of the search. After further analysis on the performance graph we found that improvements are still being made toward the end of run time. Therefore, we run the algorithm again 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 we run the algorithm longer than the 10 minutes that we used previously. The last column in Table 2 shows that prolonging the algorithm does improve the solution further and it demonstrates that it is robust and able to avoid being trapped in local optima.

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, we can conclude that we do need to use a tabu list to guide the hyper-heuristic in its heuristic selection. 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 TD 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 TD. We made a heuristic that has been applied 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. The best result for six of the datasets is when TD is two 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 best result when TD is higher and has a higher conflict matrix density (see Table 1), i.e. 26.7% and 42.0%. An examination timetabling dataset with higher conflict matrix density would imply that we might have less and sparsely distributed solution points (feasible solution) in our solution space since too many exams are conflicting with each other. Thus, a higher TD may force it to diversify its exploration of the solution search space by allowing it to move from one heuristic search space to another. For each of the datasets, except one dataset (Hec-s92), the average penalty started to decrease as we increased the TD and began to increase again once it reached its minimum average penalty. This shows that the hyper-heuristic does need to decide which TD is best for each dataset because a tabu duration which is too high or too low will produce worse solutions.

Figure 1 shows the hyper-heuristic performance with different TD on car-f92 dataset. This dataset is one of the largest dataset with 543 exams 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 represents 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 we accept a solution from the best performance heuristic even though it does not improve the solution. The higher TD means that the heuristics will remain tabu longer,

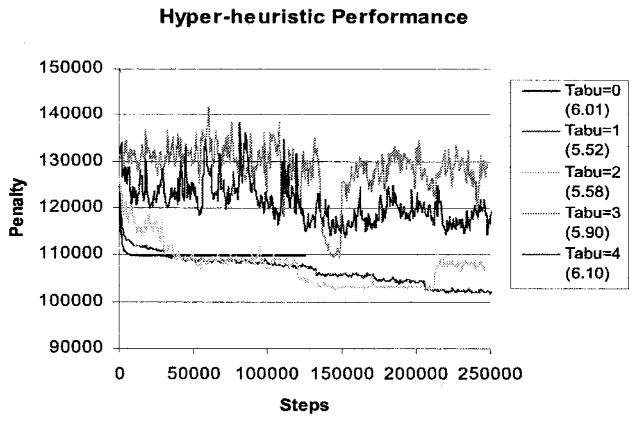

Figure 1. Hyper-heuristic performance with different TD.

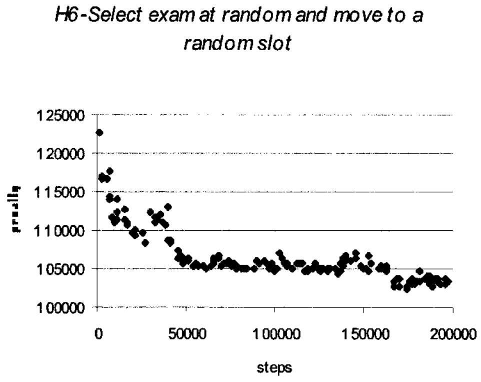

Figure 2. Heuristic 6.

thus allowing other heuristics to be applied next. By increasing the TD, we notice that 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 on this hyper-heuristic will limit the acceptance of worst solution.

The graphs in Figures 2 and 3 show when two of the heuristics were applied and how the two heuristics change the solution state.

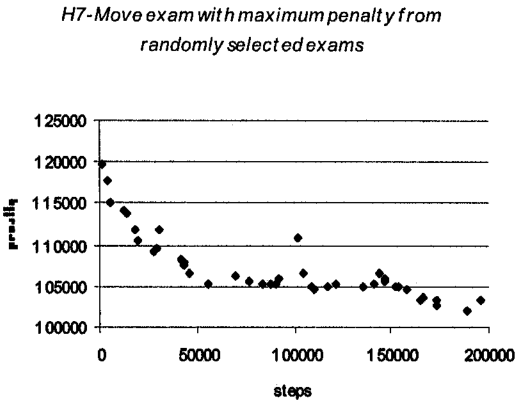

Figure 3. Heuristic 7.

The larger number of plots in Figure 2 compared to Figure 3 indicates that heuristic 6 has been applied more than heuristic 7. We do not show graphs for other heuristics because the shape is almost the same except that some are applied more than others. We also do not show how each heuristic performs in all iterations because we only keep track of its performance when it is applied. We can also see that for different datasets certain heuristics will be applied more than others, therefore it is justifiable to use several low-level heuristics that can compete with each other and a hyper-heuristic can than select the best low-level heuristic to be applied, given not only the point in the search space but also a specific problem instance.

Table 3 shows our four hour run results compared to other published results for benchmark datasets. Our objective here is to show that the HH-FTD is able to produce good quality and feasible solutions for examination timetabling problems even though it may not produce the best results. The results show that our generic method is able to produce good quality solutions compared to the others. The first two that we compare results against are of Di Gaspero and Schaerf (2001) who use tabu search and Di Gaspero (2002) who use tabu search with multi-neighbourhood. Our results are better than the tabu search method in all cases and almost as good as the tabu search with multi-neighbourhood. We also compare our results with results from other methods: constructive heuristics with backtracking of Carter *et al.* (1996); the memetic algorithm of Burke and Newall (1999); the greedy constructive heuristic with an optimiser by Caramia *et al.* (2001); and a hybrid of constraint programming, simulated annealing and hill climbing with Kempe chain neighbourhood by Merlot *et al.* (2003). Our results are better than Carter *et al.* (1996) and

Table 3. Comparing our best results and published results.

File	HH-FTD (TD = 2, 4 h run)	Di Gaspero and Schaerf	Di Gaspero	Carter *et al.*	Caramia *et al.*	Merlot *et al.*	Burke and Newall
Car-f92	4.67	5.2	-	6.2–7.6	6.0	4.3	**4.2**
Car-s91	5.37	6.2	5.68	7.1–7.9	6.6	5.1	**4.8**
Ear-f83	40.18	45.7	39.36	36.4-46.5	**29.3**	35.1	35.4
Hec-s92	11.86	12.4	10.91	10.8–15.9	**9.2**	10.6	10.8
Kfu-s93	15.84	18.0	-	14.0–20.8	13.8	**13.5**	13.7
Sta-f83	157.38	160.8	157.43	161.5–165.7	158.2	**157.3**	159.1
Tre-s92	8.39	10.0	-	9.6–11.0	9.4	8.4	**8.3**
Ute-s92	27.60	29.0	-	25.8–38.3	**24.4**	25.1	25.7

Caramia *et al.* (2001) in four cases. In all cases we could not produce better results than Burke and Newall (1999) and Merlot *et al.* (2003).

As a whole, we can see that our method does not perform the worst or best in any cases, but works reasonably well across all problem instances. We believe that with further enhancements in our hyper-heuristic selection method and some adaptive tabu duration, we can improve our results.

5. CONCLUSIONS AND FUTURE WORK

The simplest form of the hyper-heuristic module HH-FTD has been implemented and tested on exam timetabling benchmark data. 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. Our objective is not to beat the best solution but to show that the hyper-heuristic module produces good solutions that are feasible and will work across all problem instances and other real-world problems. Our 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.

Currently, we are testing a more advanced hyper-heuristic module that includes more tabu criteria such as tabu criteria based on CPU time, tabu based on change in penalty function and a probabilistic heuristic selection. In the future, we will experiment on adaptive tabu strategies and apply our method on a larger timetabling instance as well as other applications.

References

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

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

Burke, E. K. and Newall, J. P. (2004), Solving examination timetabling problems through adaptation of heuristics orderings. *Annals of Operations Research*, **129**:107–134.

Burke, E. K. and Newall, J. P. (1999) A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, **3**:63–74.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. and Schulenburg, S. (2003b) 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.

Caramia, M., Dell'Olmo, P. and Italiano, G. F. (2001) New algorithms for examination timetabling. In *Proceedings of 4th Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, Vol. 1982, Springer, Berlin, pp. 230–242.

Carter, M. W. and Laporte, G. (1996) Recent developments in practical examination timetabling. *The Practice and Theory of Automated Timetabling I*, Lecture Notes in Computer Science, Vol. 1153, E. K. Burke and P. Ross (Eds.), Springer, Berlin, pp. 3–21.

Carter, M. W. (1986) A Survey of practical applications of examination timetabling algorithms. *Operations Research Society of America*, **34**:2, March–April.

Carter, M. W., Laporte, G. and Lee, S. Y. (1996) Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, **47**:373–383.

Cowling, P., Kendall, G. and Soubeiga, E. (2001) A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, Lecture Notes in Computer Science, Vol. 2079, E. K. Burke and W. Erben, (Eds.), Springer, Berlin, pp. 176–190.

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.

Cowling, P., Kendall, G. and Soubeiga, E. (2002b) Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Applications of Evolutionary Computing: Proceedings of EVO Workshop 2002*, Lecture Notes in Computer Science, Vol. 2279, S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf and R. Günther (Eds.), Springer, Berlin, pp 1–10.

Cowling, P., Kendall, G. and Soubeiga, E. (2002c) Hyperheuristics: A robust optimisation method applied to nurse scheduling. *7th International Conference on Parallel Problem Solving from Nature, PPSN2002*, Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin, pp. 851–860.

Di Gaspero, L. (2002) Recolour, shake and kick: A recipe for the examination timetabling problem. In *Proceedings of the Fourth International Conference on the Practice and Theory of Automated Timetabling*, Gent, Belgium, August 2002, E. Burke and P. De Causmaecker (Eds.), pp. 404–407.

Di Gaspero L. and Schaerf A. (2001), 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, Vol. 2079, Springer, Berlin, pp. 104–117.

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. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 262–275.

Downsland, K. (1998) Off-the-peg or made to measure: timetabling and scheduling with SA and TS. In *Practice and Theory of Automated Timetabling II*, E. Burke and M. Carter (Eds.), Lecture Notes in Computer Science, Vol. 1408, Springer, Berlin, pp. 37–52.

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, **13**:533–549.

Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer, Boston, MA.

Gratch, J. M. and Chien, S. A. (1996) Adaptive problem-solving for large scale scheduling problems: A case study. *Journal of Artificial Intelligence Research*, **4**:365–396.

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, SEAL'02*, pp. 667–671.

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. Burke and P. De Causmaecker (Eds.), Lecture Notes in Computer Science, Vol. 2740, Springer, Berlin, pp. 207–231.

Nareyek, A. (2001) An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. *Proceedings of 4th Metaheuristics International Conference, MIC'2001*, pp. 211–216.

Schaerf, A. and Schaerf, M. (1995) Local search techniques for high school timetabling. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT'95)*, E. K. Burke and P. Ross (Eds.), pp. 313–323.

Schaerf, A. (1999) A survey of automated timetabling. *Artificial Intelligence Review*, **13**:87–127.

Terashima-Marín, H., Ross, P. M. and Valenzuela-Rendón, M. (1999) Clique-based crossover for solving the timetabling problem with gas. *Proceedings of the Congress on Evolutionary Computation*, Washington, DC, July 6–9, pp. 1200–1206.

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, Vol. 2079, Springer, Berlin, pp. 85–103.

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**:80–91.

Wren, A. (1996) Scheduling, Timetabling and Rostering—a special relationship? In *Practice and Theory of Automated Timetabling I*, E. K. Burke and P. Ross (Eds.), Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 46–76.

Wright, M. (2001) Subcost-guided search—experiments with timetabling problems. *Journal of Heuristics*, **7**:251–260.

Sports Scheduling