A Hybrid Simulated Annealing with Kempe Chain Neighborhood for the University Timetabling Problem

Mauritsius Tuga, Regina Berretta and Alexandre Mendes
School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, 2308, NSW, Australia
Moris@cs.newcastle.edu.au, {Regina.Berretta,Alexandre.Mendes}@newcastle.edu.au

Abstract

This paper addresses the problem of finding a feasible solution for the University Course Timetabling Problem (UCTP), i.e. a solution that satisfies all the so-called hard constraints. The problem is reformulated through relaxing one of its hard constraints and then creating a soft constraint to address the relaxed constraint. The relaxed problem is solved in two steps. First, a graph-based heuristic is used to construct a feasible solution of the relaxed problem, and then, a Simulated Annealing (SA)-based approach is utilized to minimize the violation of the soft constraint. In order to strengthen the diversification ability of the method in the SA phase, a heuristic based on Kempe Chain neighborhood is embedded into the standard approach. This strategy is tested on a well-known data set, and the results are very competitive compared to the current state of the art of the UCTP.

1. Introduction

The University Course Timetabling Problem (UCTP) is a well-known NP-Hard combinatorial problem. It is defined as the assignment of a set of main academic events related to a course, such as lectures, tutorials or lab sessions, to resources (timeslots and rooms) subject to a set of constraints. In general the set of constraints can be categorized as *hard* and *soft*. Hard constraints are those that are compulsory to be fulfilled. A timetable will not be acceptable if any hard constraint is violated. Soft constraints include some non-compulsory requirements. Even though they could be violated there is a strong demand to minimize such violations. A timetable without any hard constraints violations will be referred to as a *feasible* timetable. The quality of a feasible timetable will be measured by the extension of its soft constraints violations.

Requirements for a feasible timetable, as well as its soft constraints, differ from one university to another, as each university has different rules. However, there might be some common requirements for a timetable to be considered feasible. They are for instance, no students, as well as lecturers, are expected to attend two different events at one particular time; a room should not be double-booked. A quite comprehensive description of various types of those requirements can be found in [3, 9].

Throughout this paper we will consider the hard constraints as described in the International Timetabling competition [16]. They are:

1. H1: No student clash, i.e. no student is expected to attend two different events at the same time slot;
2. H2: All events should be assigned to a room within the given time slots, and the chosen room for an event should meet the specifications required for that event;
3. H3: A room should not be double-booked.

The natural way and perhaps the simplest way of constructing a feasible solution is to assign each event one by one to its suitable time slot and room until a complete timetable is attained. There are three main concerns in the timetabling construction process.

1. Choosing the order of the events to be assigned;
2. Choosing a time slot for the chosen event;
3. Choosing a suitable room for the chosen event.

Many articles relate the feasibility problem to the graph coloring problem [5, 9]. In its basic formulation, a timetable problem can be represented as a graph, where nodes represent events, and there is an edge between two nodes if and only if those two events cannot be assigned to the same time slot due to some constraint. This could happen, for example, if there is at least one student enrolled in both events or both events are given by the same lecturer. In addition, there could also be an edge between two events, if there is just one suitable room available for them. Thus, two events

6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)
0-7695-2841-4/07 $25.00 © 2007 IEEE

are said to be in conflict if there is an edge between them. Consequently, it is not surprising that constructive (or sequential) heuristics for graph coloring are found to be the most common way to generate initial feasible solutions.

The main idea highlighted in the graph coloring heuristics is that nodes or events are chosen sequentially based on some criteria which reflect how difficult it will be to assign an event. As pointed out by Burke [5], the measures commonly used in this approach are *Largest Degree first* (LD), *Largest Weighted Degree first* (LWD), *Least Saturation Degree first* (LSD), *Largest Color Degree first* (LCD) and *Largest Enrollment first* (LE).

In many cases, sequential methods are proven to be very promising for generating the initial solutions [4, 5, 12]. However, using one or a combination of those sequential heuristics still do not guarantee that a feasible solution will be found [1, 2]. Thus, some further processing is required to pursue them. Some authors incorporated some sequential heuristics into the Constraint Programming (CP) approach for the timetabling problem [8, 15, 20, 23]. Metaheuristics are also reportedly used to reach feasibility, such as Ant Colony [19], Tabu Search [7] and Genetic Algorithms [14].

There are authors who did not start their search through a sequential construction process. Instead, the search process begins with an infeasible timetable, where events are assigned randomly. To reach feasibility, an appropriate cost where the hard constraints violations are penalized, and some metaheuristic techniques are utilized [18, 22].

To the best of our knowledge, there are only a few papers focusing on solely finding feasible solutions for UCTP. All authors we mention above worked both on finding feasible solutions and minimising the soft constraints violations. Kostuch [12] attempted to find feasible course timetables from the International Timetable Competition instances using less than the number of timeslots provided. In some instances he was successful, but failed in some others. Recently, Lewis *et al.* [6, 14] posed a challenging university timetable data set for which the goal is just to find feasible solutions. Lewis pointed out that using graph-based sequential heuristics such as in [7] and [12] may fail to find any feasible solution for many instances of this data set. Subsequently they developed three methods mainly based on grouping genetic algorithm, but were only partially successful [6, 14].

In this paper we present an approach where the feasibility problem is reformulated by adding a soft constraint to deal with the hard constraint. We utilize graph-based sequential heuristics to generate the initial solution without hard constraint violation and a Simulated Annealing metaheuristic is used to reduce the number of soft constraints violations. We tested this approach in the Lewis' data set, and the results will be compared to those reported in [6].

The paper is organized as follows: In Section II we present the problem reformulation and some implementation issues. We describe our approach in tackling the problem in Section III followed by presenting our results in Section IV. Finally, the conclusions are presented in Section V.

2. Problem Formulation

In this section, we introduce an approach in dealing with the problem formulation, which forms the basis for our method. We start by reconsidering the problem from the graph theory point of view [9]. For each instance I, let $V = \{v_1, v_2, \cdots, v_n\}$ be the set of events that have to be scheduled on a weekly basis. Let $G = (V, E)$ be a graph whose vertices are the set of V, and $\{v_i, v_j\} \in E(G)$ if and only if events v_i and v_j are in conflict. The graph G is called the *conflict* graph of I. A neighbourhood of an event v denoted by $N(v)$ is a set containing all events which are in conflict with v. The degree of v is defined as $|N(v)|$.

Let also $z(v)$ denote the number of students enrolled in event v. If $z(v) = 0$ then the event v will be called a *zero* event. The set of rooms is denoted by R, the set of timeslots is denoted by W and let $B = R \times W$ be a cartesian product over the set of rooms and the set of time slots in a week. The set B will be referred to as *resource* set. In our implementation, each member of B will be represented by a unique integer, in such a way that it is easy to recover which room and time slot a resource belongs to.

Every room has a number of features such as room capacity and the availability of teaching aids among others. On the other hand, every event can only be assigned to a room and timeslot which satisfy all features required by the event. Therefore for each event v_i, there is a specific set $D_i \subseteq B$, which contains all candidate resources that could be used by event v_i and in general, $D_i \neq D_j, \forall i \neq j$. The *domain* matrix D contains $|V|$ rows and each row D_i contains the candidate resources for event v_i.

A solution or a timetable is represented by a one-dimensional array S where $S(i) = b$ means event i is assigned to resource b. Let P_1 be the problem of finding feasible solutions of an instance I with regard to hard constraints H1, H2, and H3 described in Section I. Let P_2 be the problem derived from P_1 where all the constraints are still the same, except H2, which is partially relaxed, that is, the events can be assigned to some extra timeslots. If we let m be the given timeslots in P_1 and m' be the given timeslots in P_2, then we have $m' \geq m$. The number m' will depend on I. Any timeslot later than m will be referred to as *artificial* time slot. To accommodate the original requirement of using just m timeslots, we introduce a soft constraint in P_2, which states that no event is to be assigned to an artificial timeslot. To discourage the violation of this constraint the objective function is utilized.

0-7695-2841-4/07 $25.00

$$f(S) = \sum_{v \in V_1} z(v) \quad (1)$$

where V_1 denotes the set of events currently assigned to the artificial timeslots.

A solution for P_2 with no hard and soft constraints violations is equivalent to a feasible solution of P_1. Therefore, from now on unless otherwise stated, we will work on the problem P_2 and a *feasible* solution will only mean a solution with no hard constraint violation according to the problem P_2.

Note that a feasible solution with zero cost in P_2, generally implies that no event is placed in the artificial timeslots. There might be cases that some instances contain zero events. However, the zero events can be easily rescheduled to a permitted timeslot if required without breaking any hard constraint.

3. Simulated Annealing-based Heuristic

Simulated Annealing (SA) is a stochastic search method based on the use of a local search. At any step, SA either moves to a better neighbor solution if it finds one, or to a worse solution with certain probability. It includes some important elements concerning the probability of accepting a deteriorating solution which will be considered later. SA was introduced by Kirkpatrick [11] in 1983, inspired by annealing process in physics. Many authors reportedly applied this strategy in solving timetabling problem [7, 12, 15, 18, 21, 22]. In addition, it turns out that the winner of the 2002 International Timetabling Competition utilized a SA-based approach to solve the instances [16].

In Fig. 1, we present a pseudo-code of the heuristic to construct the initial solution (**ISheuristic**). Some sequential approaches were tested for the feasible initial solution generation and the combination of LSD (Least Saturation Degree) and LD (Largest Degree) seems to be the best combination. The feasible solution found by **ISheuristic** will be further processed by the SA-based method to minimise the number of soft constraint violations, if necessary.

In Fig. 2, we show the pseudo-code of the Hybrid Simulated Annealing (HSA) implemented in this work. Next, we will describe the three neighborhood structures used in our HSA algorithm.

1. *Simple neighbourhood*: This neighbourhood contains solutions that can be obtained by simply changing the resource of one event.

2. *Swap neighbourhood*: Under the simple neighbourhood, an event is randomly chosen and a new resource is allocated to it. However, this may involve some bias as there might be events which do not have any valid resources left at one stage of the search. This might create a disconnected search space. In the swap neighbourhood, the resources of two events are exchanged, overcoming the disconnection of the search space that might occur in the simple neighbourhood.

3. *Kempe chain neighbourhood*: A standard Kempe chain neighbourhood operates over two selected timeslots and was used by Burke *et al.* [4], Thompson *et al.* [22] and

```
Input: Set of events V and set of candidate resources D
U ← V
While ((#trials < #trials_max) and (U = V))
    D' ← D
    Sort the candidate resources of each event randomly
    Sort events in U using LSD
        Break the tie using (LD) if necessary
    For i = 1 to |V|
        Choose event v_i
        If (v_i has no resource left)
            U ← V; D' ← D
            #trials ← #trials + 1
        else
            Choose the first resource for v_i
            Update D'; U ← U \ {v_i}
        Endif
    Endfor
EndWhile
```

Figure 1. Pseudo-code of the ISHeuristic used to create the initial solution.

```
Input: Incumbent solution S
For i = 0 to maxItSimAnnealing
    initialize T and a
    For j = 0 to a * |V|
        Choose an event v_k at random
        If (v_k still has permitted resources)
            Choose a candidate resource randomly
            Calculate Δ_Cost
            If ((Δ_Cost ≤ 0) or (e^(Δ_Cost/T) < rand[0,1]))
                Accept the move
                Update matrix D
                Update incumbent solution S
            Endif
        Else Find potential event v_s for swapping
            Calculate Δ_Cost
            If ((Δ_Cost ≤ 0) or (e^(Δ_Cost/T) < rand[0,1]))
                Accept the move
                Update matrix D
                Update incumbent solution S
            Endif
        Endif
    Endfor
    If ((No improvement for iter_max iterations)
      and (time limit not reached))
        KCHeuristic(S)
        Adjust the temperature T
    Endif
Endfor
```

Figure 2. Pseudo-code of the HSA algorithm.

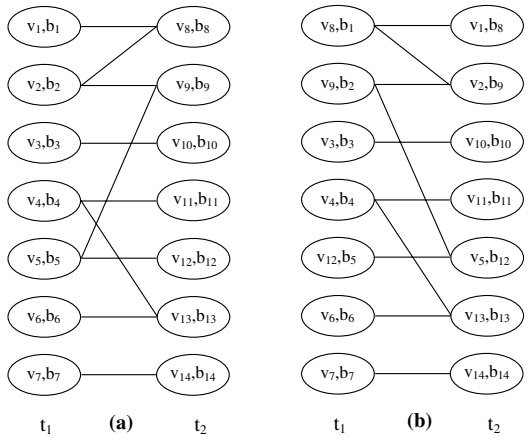

Figure 3. Bipartite graphs induced by the resources and events in timeslots t_1 and t_2 before (a) and after (b) the Kempe Chain move.

Merlot *et al.* [15] to tackle examination timetabling problems. It swaps the timeslot of a subset of events in such a way that the feasibility is maintained. As our concern is in the course timetabling problem, we develop a quite different version of Kempe chain than those used in [4, 15, 22]. This version will be referred to as *Pair-wise Kempe Chain* (PKC) which is basically a class of neighbourhood structures containing those what we call *k-pair Kempe Chain* where k is a positive integer, representing the number of timeslot pairs involved. For $k = 1$ for example, a 1-pair Kempe Chain neighborhood can be obtained by initially choosing a pair of time slots randomly. Then, we create a Kempe Chain based on the chosen timeslots, swapping the timeslots of all events in the chain. To illustrate the idea, consider two time slots, say t_1 and t_2 of a timetable S. A bipartite graph can be induced by considering the resources used within those two timeslots as vertices, and an edge between them if and only if the two resources are used by two conflicting events or they share the same room. In the Figure 3(a), we assume events $v_1, v_2, ..., v_7$ are currently assigned to resources $b_1, b_2, ..., b_7$ in timeslot t_1, and events $v_8, v_9, ..., v_{14}$ are currently assigned to resources $b_8, b_9, ..., b_{14}$ in timeslot t_2. The edge between (v_5, b_5) in timeslot t_1 and (v_9, b_9) in timeslot t_2 indicates that the events v_5 and v_9 are in conflict. A Kempe chain can be formed by choosing one resource randomly from timeslot t_1; and the corresponding event will trigger a chain which is basically a connected subgraph. If, for instance, the resource b_2 of timeslot t_1 is chosen then a chain will be created triggered by event v_2, that will contain the events $\{v_1, v_8, v_2, v_9, v_5, v_{12}\}$. A new timetable S' can be obtained by reassigning each event in this chain to their pair's timeslot in the same room (Fig.1(b)). In Fig. 4, we show the pseudo-code for the Kempe Chain based heuristic, which is called from the HSA (see Fig. 2).

The probability used in SA is defined by using a parameter called *temperature*. The higher the temperature, the higher is the probability of accepting a worse solution. As the search progresses, the probability of accepting a worse solution decreases. A *cooling schedule* in SA is a general term used to manage the temperature during the SA process. It includes the choice of the initial temperature, the rate of decrease of the temperature and the number of trials in one temperature level. The whole SA process is proven to be very sensitive to the choice of cooling schedule. Despite many authors have investigated this aspect [10, 21, 22] it turned out that none of their recommendations was suitable for our problem at hand. We then had to carry out some preliminary tests to tune in our cooling schedule:

1. *Initial temperature*: The initial temperature is chosen such that the probability of accepting a worse solution is sufficiently high $(\pm 40\%)$;
2. *Cooling equation*: We tested many cooling equations and found out that the best one is similar to the one used by Kostuch [12], i.e $T = 1/((1/T) + \beta)$ where β is chosen between 0.001 and 0.0005;
3. *Number of trials*: The number of trials in each temperature level is set to $a.|V|$ where a is linearly increased. Initially, a is initially set to 10.

In order to save CPU time, some problem specific knowledge are incorporated into the search process. For instance, the cost calculation is done using delta evaluation (Δ_{Cost}) [17]. Therefore, given a new solution, its cost is not calculated from scratch. Instead, it incorporates some unchanged cost components from its predecessor. In addition, moving an event using *simple* neighborhood from an artificial times-

```
Input: Incumbent Solution S
For i = 0 to maxIterKempe
    Randomly chose k numbers of pairs
    Mark all timeslots as unvisited
    U ← ∅
    For j = 0 to k
        Choose two unvisited timeslots t1 and t2
        Choose a trigger event in t1
        Build a Kempe chain
        Let Uj denote the set of moving events
        U ← U ⋃ Uj
        Mark the two timeslots as visited
    Endfor
    Move each event in U to their pair timeslot
    Calculate Δ_Cost
    If (Δ_Cost ≤ 0)
        Accept the move
        Update incumbent solution S
    Endif
    If (no improvement for iter_max iterations)
        Return incumbent solution S
    Endif
Endfor
```

Figure 4. Pseudo-code of the KCHeuristic.

0-7695-2841-4/07 $25.00

lot to another artificial timeslot will not change the cost. This also applies to swapping two events in the same artificial timeslot and to the *Kempe chain* neighbourhood.

4. Computational Results and Analysis

We tested our heuristics on 60 instances posed by Lewis *et al.* in [6]. These instances were created using an instance generator and they are difficult to solve using some sequential heuristics. Even though, there is at least one feasible solution for each of them. The instances are divided into 3 categories: small, medium and large. For the small instances, $200 \leq |V| \leq 225$ and $|R| = 5$ or 6; for the medium instances $390 \leq |V| \leq 425$ and $|R| = 10$ or 11; and for the large instances $1000 \leq |V| \leq 1075$ and $|R| = 25$ to 28. All events of all instances must be assigned within $m = 45$ timeslots. Additional parameters and other details on the instances can be found in [6].

ISHeuristic is used to generate the initial solution for the relaxed problem. The maximum number of trials is set at 500, which takes about 5 seconds of CPU time to be completed. **ISHeuristic** successfully generates initial solutions for 13 instances from Lewis data set in just few seconds using the 45 time slots available. The rest of the instances are more difficult and it was necessary to add artificial timeslots. A preliminary test were carried out for these instances in order to find out their minimum number of artificial timeslots. Having found this number, we performed some experiments in order to minimize the cost of the objective function (1) using the **HSAHeuristic**.

We carried out 20 runs per instance with distinct random seeds on a PC Pentium IV 3.2 GHz running under Linux. Basically we terminated our algorithm if no improvement was found within a fixed number of iterations. In addition, we set time limits of 200, 400 and 1,000 seconds for small, medium and large instances, respectively.

Table 1 presents the best results found by our SA algorithm (HSA) compared to those found by Lewis for the small, medium and large instances. Lewis I are results obtained by the method as described in [14]. Lewis I, Lewis II and Lewis III results can be found in [6]. It is important to emphasize that we still do not have access to the details of Lewis' methods Lewis II and Lewis III, as they have not been published yet [13]. The four methods have comparable performances in the small problems, with a similar number of feasible solutions found. For the medium instances, the HSA obtains superior results, improving those from the three Lewis methods. Finally, in the large instances the HSA algorithm shows a performance improvement in comparison with the others. It found twice as many feasible solutions than the best Lewis method. Also worth mentioning is the number of times that the HSA found feasible solutions, which shows an intrinsic robustness of the method.

Table 1. Comparison of the HSA and the Lewis I, II and III methods for the small, medium and large instances.

Small instances						
Instance	HSA			Lewis I	Lewis II	Lewis III
name	min	ave	CPU(s)			
s_1	$0_{(20)}$	0	0	0	0	0
s_2	$0_{(20)}$	0	0	0	0	0
s_3	$0_{(20)}$	0	9	0	0	0
s_4	$0_{(20)}$	0	0	0	0	0
s_5	$0_{(20)}$	0	5	5	0	0
s_6	$0_{(20)}$	0	0	0	0	0
s_7	$0_{(20)}$	0	0	0	0	0
s_8	$0_{(1)}$	1.9	79	12	4	0
s_9	$0_{(2)}$	3.85	84	4	0	0
s_{10}	$0_{(20)}$	0	15	0	0	0
s_{11}	$0_{(20)}$	0	0	0	0	0
s_{12}	$0_{(20)}$	0	0	0	0	0
s_{13}	$0_{(9)}$	1	15	0	0	0
s_{14}	$3_{(1)}$	5.95	136	17	3	0
s_{15}	$0_{(20)}$	0	0	0	0	0
s_{16}	$0_{(20)}$	0	13	0	0	0
s_{17}	$0_{(20)}$	0	13	0	0	0
s_{18}	$0_{(11)}$	0.45	36	3	0	0
s_{19}	$0_{(11)}$	1.2	25	3	0	0
s_{20}	$0_{(20)}$	0	0	0	0	0
Total	19			14	18	20
Medium instances						
Instance	HSA			Lewis I	Lewis II	Lewis III
name	min	ave	CPU(s)			
m_1	$0_{(20)}$	0	0	0	0	0
m_2	$0_{(20)}$	0	0	0	0	0
m_3	$0_{(20)}$	0	8	0	0	0
m_4	$0_{(20)}$	0	3	0	0	0
m_5	$0_{(20)}$	0	85	8	0	0
m_6	$0_{(20)}$	0	20	15	0	0
m_7	$1_{(1)}$	4.15	440	41	34	14
m_8	$0_{(20)}$	0	12	21	9	0
m_9	$0_{(1)}$	4.9	269	30	17	2
m_{10}	$0_{(20)}$	0	0	0	0	0
m_{11}	$0_{(20)}$	0	25	12	0	0
m_{12}	$0_{(20)}$	0	54	0	0	0
m_{13}	$0_{(12)}$	0.5	172	23	3	0
m_{14}	$0_{(20)}$	0	59	0	0	0
m_{15}	$0_{(19)}$	0.05	72	10	0	0
m_{16}	$1_{(2)}$	5.15	733	50	30	1
m_{17}	$0_{(20)}$	0	39	21	0	0
m_{18}	$0_{(2)}$	6.05	429	15	0	0
m_{19}	$0_{(3)}$	5.45	511	51	0	0
m_{20}	$2_{(1)}$	10.6	457	15	0	3
Total	17			7	15	16
Large instances						
Instance	HSA			Lewis I	Lewis II	Lewis III
name	min	ave	CPU(s)			
b_1	$0_{(20)}$	0	0	0	0	0
b_2	$0_{(20)}$	0	283	0	0	0
b_3	$0_{(20)}$	0	447	0	0	0
b_4	$0_{(20)}$	0	406	32	30	8
b_5	$0_{(6)}$	1.1	743	31	24	30
b_6	$5_{(1)}$	8.45	893	90	71	77
b_7	$47_{(1)}$	58.3	966	150	145	150
b_8	$0_{(20)}$	0	210	35	30	5
b_9	$0_{(19)}$	0.05	419	26	18	3
b_{10}	$0_{(6)}$	1.25	660	36	32	24
b_{11}	$0_{(14)}$	0.35	444	43	37	22
b_{12}	$0_{(20)}$	0	240	4	0	0
b_{13}	$0_{(20)}$	0	274	23	10	0
b_{14}	$0_{(20)}$	0	271	8	0	0
b_{15}	$0_{(20)}$	0	255	120	98	0
b_{16}	$0_{(2)}$	2	755	120	100	19
b_{17}	$76_{(1)}$	89.9	998	260	243	163
b_{18}	$53_{(1)}$	62.6	764	199	173	164
b_{19}	$109_{(1)}$	127	998	262	253	232
b_{20}	$40_{(1)}$	46.7	827	186	165	149
Total	14			3	5	7

The results represent the number of events in the artificial timeslots
The number of times (out of 20 trials) that the HSA found the optimal solution is shown in subscript under the *min* column.

Also important to emphasize is that a zero cost found in zero seconds means that our initial solution generator could already find the feasible solution for the instance by using just 45 timeslots.

5. Conclusion

This paper presented a new Hybrid Simulated Annealing (HSA) for the University Course Timetabling Problem (UCTP). The method relaxes the hard constraint associated to the number of timeslots in the original problem and uses a penalty function to minimize the use of slots and reach a feasible solution. The method uses an improved Kempe Chain neighbourhood that allows a better diversification in the search for a feasible solution. Three sets of instances were used, divided according to their sizes. The HSA performance was compared against three different algorithms from Lewis *et al.* [6, 14], which are the current state of the art for the problem. The results indicate a comparable performance for the set of smaller instances; slightly better results for the medium-sized instances; and strongly better for the larger instances. Also worth mentioning is the robustness showed by the HSA, with the method repeatedly finding feasible solutions in the majority of the trials.

References

[1] S. Abdullah, E. Burke, and B. McCollum. An investigation of variable neighbourhood search for university course timetabling. In *Proceedings of MISTA2005: The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, pages 413–427, New York, USA, 2005.

[2] S. Abdullah, E. Burke, and B. McCollum. A randomised iterative improvement algorithm with composite neighbourhood structures for university course timetabling. In *Proceedings of MIC05: The 6th Metaheuristic International Conference*, Vienna, Austria, 2005.

[3] V. Bardadym. Computer-aided school and university timetabling: The new wave. In E. Burke and P. Ross, editors, *Proceedings of PATAT'95*, volume 1153 of *Lecture Notes in Computer Science*, pages 22–45. Springer-Verlag, Berlin, 1995.

[4] E. Burke, A. Eckersley, B. McCollum, S. Petrovic, and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling. Technical Report NOTTCS-TR-2006-2, University of Nottingham, School of CSiT, 2006.

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

[6] Centre of Emergent Computing. http://www.emergentcomputing.org/timetabling/harderinstances.htm, 2007.

[7] M. Chiarandini, K. Socha, M. Birattari, and O. Rossi-Doria. An effective hybrid approach for the university course timetabling problem. Technical Report AIDA-03-05, TU Darmstadt, FG Intellektik, 2003.

[8] S. Deris, S. Omatu, and H. Ohta. Timetabling planning using the constraint-based reasoning. *Computers & Operations Research*, 27:819–840, 2000.

[9] J. K. E.K. Burke and D. de Werra. Applications to timetabling. In J. Gross and J. Yellen, editors, *Handbook of Graph Theory*, pages 445–474. Chapman Hall/CRC Press, 2004.

[10] M. Elmohamed, P. Coddington, and G. Fox. A comparison of annealing techniques for academic course scheduling. In E. Burke and M. Carter, editors, *Proceedings of PATAT'97*, volume 1408 of *Lecture Notes in Computer Science*, pages 92–114. Springer-Verlag, Berlin, 1997.

[11] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 4598:671–680, 1983.

[12] P. Kostuch. The university course timetabling problem with a three-phase approach. In E. Burke and M. Trick, editors, *Proceedings of PATAT 2004*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125. Springer-Verlag, Berlin, 2004.

[13] R. Lewis and B. Paechter. Finding feasible timetables using group based operators. *IEEE Transactions on Evolutionary Computation*, to appear.

[14] R. Lewis and B. Paechter. Application of the grouping genetic algorithm to university course timetabling. In G. Raidl and J. Gottlieb, editors, *Proceedings of EvoCOP 2005*, volume 3448 of *Lecture Notes in Computer Science*, pages 144–153. Springer-Verlag, Berlin, 2005.

[15] L. Merlot, N. Boland, B. Hughes, and P. Stuckey. A hybrid algorithm for the examination timetabling problem. In E. Burke and P. De Causmaecker, editors, *Proceedings of PATAT 2002*, volume 2740 of *Lecture Notes in Computer Science*, pages 207–231. Springer-Verlag, Berlin, 2002.

[16] Metaheuristic Network. http://www.idsia.ch/files/ttcomp2002, 2007.

[17] P. Ross, D. Corne, and H. Fang. Improving evolutionary timetabling with delta evaluation and direct mutation. *Lecture Notes in Computer Science*, 866:556–565, 1994.

[18] O. Rossi-Doria et al. A comparison of the performance of different metaheuristics on the timetabling problem. In E. Burke and P. De Causmaecker, editors, *Proceedings of PATAT 2002*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer-Verlag, Berlin, 2002.

[19] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In M. Dorigo et al., editor, *Proceedings of ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, Berlin, 2002.

[20] V. Tam and D. Ting. Combining the min-conflicts and look-forward heuristics to effectively solve a set of hard university timetabling problems. In *Proceedings of ICTAI'03: The 15th IEEE International Conference on Tools with Artificial Intelligence*, page 492, Sacramento, USA, 2003.

[21] J. Thompson and K. Dowsland. General colling schedules for a simulated annealing based timetabling system. In E. Burke and P. Ross, editors, *Proceedings of PATAT'95*, volume 1153 of *Lecture Notes in Computer Science*, pages 345–363. Springer-Verlag, Berlin, 1995.

[22] J. Thompson and K. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25:637–648, 1998.

[23] G. White and J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. In E. Burke and M. Carter, editors, *Proceedings of PATAT'97*, volume 1408 of *Lecture Notes in Computer Science*, pages 187–200. Springer-Verlag, Berlin, 1997.

0-7695-2841-4/07 $25.00 © 2007 IEEE