Construction of Course Timetables Based on Great Deluge and Tabu Search

Salwani Abdullah[1], Khalid Shaker[1], Barry McCollum[2], Paul McMullan[2]

[1]Center for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
{salwani, khalid}@ftsm.ukm.my
[2]Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom,
{b.mccollum,p.p.mcmullan}@qub.ac.uk

Abstract. The course timetabling problem deals with the assignment of a set of courses to specific timeslots and rooms within a working week subject to a variety of hard and soft constraints. Solutions are called feasible if all the hard constraints are satisfied. The goal is to satisfy as many of the soft constraints as possible whilst constructing a feasible schedule. In this paper, we present a combination of two metaheuristics i.e. great deluge and tabu search approaches. The algorithm is tested over eleven benchmark datasets (representing one large, five medium and five small problems). The results demonstrate that our approach is able to produce solutions that have lower penalty on all the small and medium problems when compared against other techniques from the literature.

Keywords: Great deluge, tabu search, course timetabling.

1 Introduction

In the timetabling literature, significant attention has been paid to the problem of constructing university course timetables. Various techniques have been applied to this complex and difficult problem. Exact solutions, where the optimum solution is guaranteed, can be only possible for problems of a limited size [38]. On the other hand, algorithms employing heuristic based techniques have shown to be highly effective. Examples include simulated annealing (e.g. [1, 2]), tabu search (e.g. [3,4]) and genetic algorithms (e.g. [5,6]). In this paper, a combination of two metaheuristic based techniques i.e. great deluge and tabu search are applied to the university course timetabling problem. The approach is tested over eleven benchmark datasets that were introduced by Socha *et al*. [7]. The results demonstrate that our approach is capable of producing high quality solutions when compared to other techniques in the literature. The paper is organised as follows: The next section describes the course timetabling problem and provides a brief overview of the relevant timetabling literature. Section 3 describes our algorithm and its application to the course timetabling problem. Experimental results are presented in Section 4. A comparison between state-of-the-

art techniques from the literature and some brief concluding comments are presented in Section 5.

2 The University Course Timetabling Problem

In university course timetabling, a set of courses are scheduled into a given number of rooms and timeslots across a period of time. This usually takes place within a week and the resultant timetable replicated for as many weeks as the courses run. Also, students and teachers are assigned to courses so that the teaching delivery activities can take place. The course timetabling problem is subject to a variety of hard and soft constraints. Hard constraints need to be satisfied in order to produce a *feasible* solution. In this paper, we test our approach on the problem instances introduced by Socha *et al*. [7] who present the following hard constraints:

- *No student can be assigned to more than one course at the same time.*
- *The room should satisfy the features required by the course.*
- *The number of students attending the course should be less than or equal to the capacity of the room.*
- *No more than one course is allowed to be assigned to a timeslot in each room.*

Socha *et al*. [7] also present the following soft constraints that are equally penalised:

- *A student has a course scheduled in the last timeslot of the day.*
- *A student has more than 2 consecutive courses.*
- *A student has a single course on a day.*

The problem has

- A set of N courses, $e = \{e_1,\ldots,e_N\}$.
- 45 timeslots.
- A set of R rooms.
- A set of F room features.
- A set of M students.

The objective of this problem is to satisfy the hard constraints and to minimise the violation of the soft constraints.

Several university course timetabling papers have appeared in the literature in the last few years which tackle various benchmark course timetabling problems which have been introduced [39]. The following provides an overview of techniques which have been used to find solutions to various formulations of the course timetabling problem in the past. Socha *et al*. [7] employed a local search and ant based algorithms and tested on eleven problems that were produced by Paechter's[1] course timetabling test instance generator (note that these instances are used to evaluate the method described in this paper). In 2003, Burke *et al.* [9] introduced a tabu-search hyperheuristic which was also tested on a nurse rostering problem. The Great deluge algorithm was employed by Burke *et al.* [10]. Di Gaspero and Schaerf [11] applied a multi neighbourhood search approach and tested on the same instances. In 2004, Lewis and

[1] http://www.dcs.napier.ac.uk/~benp/

Paechter [12] designed several crossover operators and tested on twenty instances generated by Paechter's generator which was used in the competition in 2002 (see http://www.idsia.ch/Files/ttcomp2002). Kostuch and Socha [13] investigated a statistical model in predicting the difficulty of timetabling problems particularly on the competition datasets. In 2005, Kostuch [14] presented a three phase approach which employs simulated annealing and had 13 best results of the 20 competition instances. A variable neighbourhood search with a fixed tabu list has been employed by Abdullah *et al*. [15]. Asmuni *et al*. [16] applied a fuzzy multiple heuristic ordering on the eleven standard benchmark datasets. In 2007, Abdullah *et al*. [17] developed an iterative improvement algorithm with composite neighbourhood structures and later combining this algorithm with a mutation operator (see Abdullah *et al*. [18]). McMullan [19] applied a two phased approach utilizing an adaptive construction heuristic and an extended version of the Great Deluge Algorithm. In 2008, Abdullah and Turabieh [20] employed a genetic and local search approach on 11 benchmark course. Landa-Silva and Obit employed a non linear great and deluge on the same instances [21]. Müller [22] applied a constraint-based solver applied to the curriculum-based course timetabling problems in the 2nd International Timetabling Competition (Track 1 and Track 3) as introduced by Di Gaspero *et al*. [23] and achieved the first place in this competition. Lu and Hao [24] applied a hybrid heuristic algorithm called adaptive tabu search to the same instances. Other papers that tackle curriculum-based course timetabling problems can be found in Clark *et al*. [25], De Cesco *et al*. [26], Geiger [27] and Lach and Lubbecke [28]. Interested readers are referred to Lewis [29] for a comprehensive survey of the university timetabling approaches in recent years, a gap between theory and practice in university timetabling area by McCollum [30] and other related papers on course timetabling in [31,32,33,34,35].

3 The Algorithm

The algorithm presented here is divided into two parts i.e. construction and improvement algorithms. Within the latter stage, four neighbourhood structures have been employed.

3.1 Neighbourhood Structure

The different neighbourhood structures and their explanation are outlined as follows:

N_1: Choose a single course at random and move to a feasible timeslot that can generate the lowest penalty cost.
N_2: Select two courses at random from the same room (the room is randomly selected) and swap timeslots.
N_3: Move the highest penalty course from a random 10% selection of the courses to a new feasible timeslot which can generate the lowest penalty cost.
N_4: Move the highest penalty course to a random feasible timeslot (both courses are in the same room).

3.2 Constructive Heuristic

A least saturation degree is used to generate initial solutions which start with an empty timetable [19]. The events with less rooms available and more likely to be difficult to schedule will be attempted first without taking into consideration the violation of any soft constraints. This process is carried out in the first phase. If a feasible solution is found, the algorithm terminates. Otherwise, phase 2 is executed. In the second phase, neighbourhood moves (N1 and/or N2) are applied with the goal of achieving feasibility. N1 is applied for a certain number of iterations. If a feasible solution is met, then the algorithm stops. Otherwise the algorithm continues by applying a N2 neighbourhood structure for a certain number of iterations. Across all instances tested, solutions were made feasible before the improvement algorithm was applied.

3.3 Improvement Algorithm

During the improvement stage a set of the neighbourhood structures outlined in subsection 3.1 are applied. The hard constraints are never violated during the timetabling process. The pseudo code for the algorithm implemented in this paper is given in Fig. 1.

```
Set the initial solution Sol by employing a constructive heuristic;
Calculate initial cost function f(Sol);
Set best solution Sol_best ← Sol;
do while (not termination criteria)
     Step 1: Great Deluge
     Step 2: Tabu Search
     Step 3: Accepting Solution
          Choose the best between SolbestGD* and SolbestTS*, called
         Sol*
         if (f(Sol*) < f(Sol_best))
             Sol ← Sol*;
             Sol_best ← Sol*;
         end if
end do
```

Fig 1. The pseudo code for the improvement algorithm

There are 3 steps involved in the improvement algorithm. In Step 1, the great deluge (see [36]) algorithm is employed followed by a tabu search (see [37]) in Step 2. Step 3 involves accepting a solution to be used in the search process in the next iteration where the best solutions from Step 1 and Step 2 are chosen (called *Sol**) and compared with the best solution so far (called Sol_{best}). If the quality of the *Sol** is less then the quality of the Sol_{best}, then the current solution will be updated and Sol* will be assigned as Sol_{best}. The pseudo code for Step 1 and Step 2 are outlined in Figures 2 and 3. Note that, the process is repeated and stops when the termination criterion is

met (in this work the termination criteria is set as the number of evaluations i.e. 100,000 and 200000 evaluations or when the penalty cost is zero).

Step 1: Great Deluge
Fig. 2 shows the pseudo code for the algorithm in Step 1.

```
SolGD ← Sol;
SolbestGD ← Sol
f(SolGD) ← f(Sol);
f(SolbestGD)← f(Sol)
Set optimal rate of final solution, Optimalrate;
Set number of iterations, NumOfIteGD;
Set initial level: level ← f(SolGD);
Set decreasing rate ∆B = ((f(SolGD)–Optimalrate)/(NumOfIteGD);
Set iteration ← 0;
Set not_improving_counter ← 0, not_improving_ length_GDA;
do while (iteration < NumOfIteGD)
    Apply neighbourhood structure N_i where i ∈ {1,…,K} on
    SolGD,TempSolGD_i;
    Calculate cost function f(TempSolGD_i);
    Find the best solution among TempSolGD_i where i ∈ {1,…,K} call new
    solution SolGD*;
    if (f(SolGD*) < f(SolbestGD))
        SolGD ← SolGD*;
        SolbestGD ← SolGD*;
        not_improving_counter ← 0;
        level = level - ∆B;
    else
        if (f(SolGD*)≤ level)
            SolGD ← SolGD*;
            not_improving_counter ← 0;
        else
           not_improving_counter++;
           if (not_improving_counter == not_improving_length_GDA)
                level= level + random(0,3);
    Increase iteration by 1;
end do;
return SolbestGD;
```

Fig 2. The pseudo code for the great deluge (Step 1 in Fig. 1)

Let K be the total number of neighbourhood structures to be used in the search (K is set to be 4 (applied in the preliminary experiments, see subsection 4.2) and 2 (applied in subsection 4.3)) and $f(SolGD)$ is the quality measure of the solution *Sol*. At the start, the best solution, *SolbestGD* is set to be *Sol*. In a *do-while* loop, a set of neighbourhoods i where $i \in \{1,\ldots,K\}$ is applied to *SolGD* to obtain $TempSolGD_i$. The best solution among $TempSolGD_i$ is identified, called, *SolGD**. The $f(SolGD^*)$ is compared to the $f(SolbestGD)$. If it is better, then the current and best solutions are updated. Otherwise $f(SolGD^*)$ will be compared against the level. If the quality of *SolGD** is less than the level, the current solution, *SolGD* will be updated as *SolGD**. Otherwise, the level will be increased with a certain number (between 1 and 3 in this experiment) in order to allow some flexibility in accepting a worse solution. The process is repeated until the termination criterion is met.

Step 2: Tabu Search
Fig. 3 shows the pseudo code for the algorithm in Step 2.

```
SolTS ← Sol;
SolbestTS ← Sol
f(SolTS) ← f(Sol);
f(SolbestTS)← f(Sol)
Set tabu list = 10, called TL;
Set number of iterations, NumOfIteTS which is equal to the number TL;
Set iteration ← 0;
Set OptimalValue = 0/50/500(for small/medium/large instances);
do while (iteration < NumOfIteTS or Converge == OptimalValue)
    Apply neighbourhood structure N_i where i ∈ {1,2} on SolTS,called
    TempSolTS_i;
    Calculate cost function f(TempSolTS_i);
    Find the best solution among TempSolTS_i where i ∈ {1,2} call new
    solution SolTS*;
    Keep neighbourhood structure that generate SolTS*, called N_TS;
    Moved = False;
    while !Moved
         if (f(SolTS*) < f(SolbestTS))
             SolTS ← SolTS*;
             SolbestTS ← SolTS*
              Converge = f(SolTS);
             Moved = True;
        endif
        if (N_TS is not tabu)
             Push N_TS into tabu list, TL;
    end while
end do;
return SolbestTS;
```

Fig 3. The pseudo code for the tabu search (Step 2 in Fig. 1)

In Step 2, a similar process as in Step 1 is applied where a tabu search approach is employed on a different set of neighbourhood structures. In this experiment two neighbourhood structures are used to obtain $TempSolTS_i$ where $i \in \{1,2\}$. The best solution among $TempSolTS_i$ is identified, called *SolTS**. The *f*(*SolTS**) is compared to the *f*(*SolbestTS*). If it is better, then the current and best solutions are updated. Our tabu search algorithm uses only a short term memory. We add any moves that generate *SolTS** to the tabu list (if currently not in the tabu list) denoted as *TL*. These moves are not allowed to be part of any search process for a certain number of iterations (the tabu tenure). The tabu tenure is decreased after each iteration until it reaches zero. All tabu moves will change to non tabu status when the tabu tenure is zero. In these experiments, we set the tabu tenure to be 10. The determination of these values was based upon a series of experiments. The process is repeated until the termination criterion is met.

4 Experimental Results

4.1 Problem instances and experimental protocol

The algorithm is coded using Matlab under Windows XP. We ran the experiments over the weekend for medium and large datasets. For the small datasets, the algorithm is able to produce an optimal solution in less than a minute. We note that course timetabling is a problem that is usually tackled several months before the schedule is required. This is a scheduling problem where the time taken to solve the problem is often not critical. We evaluate our results on the instances taken from Socha *et al*. (2002) and which are available at http://iridia.ulb.ac.be/~msampels/tt.data/. The experiments for the course timetabling problem discussed in this paper were tested on the benchmark course timetabling problems proposed by the Metaheuristics Network that need to schedule 100-400 courses into a timetable with 45 timeslots corresponding to 5 days of 9 hours each, whilst satisfying room features and room capacity constraints. They are divided into three categories: *small*, *medium* and *large*. We deal with 11 instances: 5 *small*, 5 *medium* and 1 *large*. The parameter values defining the categories are given in Table 1.

Table 1. The parameter values for the course timetabling problem categories

Category	*small*	*medium*	*large*
Number of courses	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Number of students	80	200	400

4.2 Results under preliminary experiments

The aim of this preliminary experiment is to measure the effectiveness of the neighbourhood structures used. We tested four neighbourhood structures as discussed in subsection 3.1 on all instances using the great deluge algorithm (as described in Fig. 2).

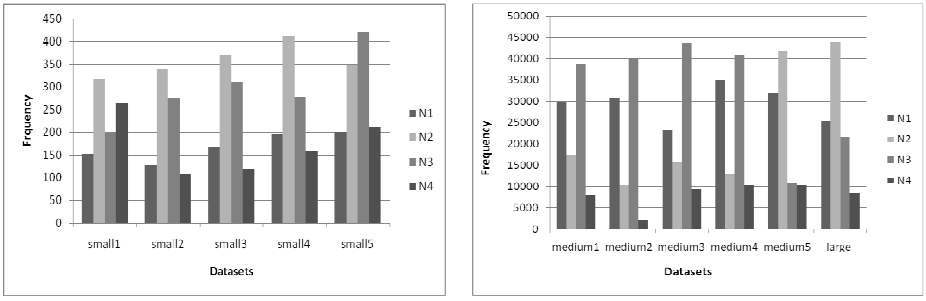

Fig 4. Frequency of the neighbourhood structures used for the *small*, *medium* and *large* datasets

Fig. 4 shows the frequency charts of the neighbourhood structures that have been used by the great deluge algorithm for all datasets. The x-axis represents the datasets while the y-axis represents the frequency of the neighbourhood structures being employed throughout the search. It can be seen, from Figure 4, in most of the *small* and *medium* instances, neighbourhood structures *move2* (N2) and *move3* (N3) are the most popular structures used. The popular structures for the *large* dataset are *move1* (N1) and *move2* (N2). This shows that different size or complexity of the problem might need different neighbourhood structures in order to help the search algorithm to explore the search space. It is also evident that some neighbourhood structures do not contribute to a good quality solution. Further preliminary experiment is carried out to test the effect of using a set of different neighbourhood structures on different instances; for the *small* and *medium* instances, N2 and N3 are used while the *large* instance uses N1 and N3 neighbourhood structures. Table 2 shows the comparison results on different neighbourhood structures when applied to great deluge algorithm for 100000 iterations.

Table 2. Comparison on different moves

Data set	Initial solution	Great Deluge with 4 moves	Great Deluge with a set of 2 moves
Small1	266	0	0
Small2	225	0	0
Small3	289	0	0
Small4	209	0	0
Small5	382	0	0
Medium1	957	186	182
Medium2	896	225	202
Medium3	957	223	226
Medium4	835	170	178
Medium5	711	167	135
Large	1613	980	845

The preliminary results demonstrate that good moves (regardless of the number of moves) will help the algorithm to find better solutions. This shows the advantage of combining several good neighbourhood structures against the type of structure alone in order to help compensate against the ineffectiveness of the other neighbourhood structures.

4.3 Results under more computational resources

In this experiment, we evaluate the search potential of our algorithm with a relaxed stop condition. For this purpose, we run our algorithm for 200000 iterations (which takes approximately twelve hours) with different set of moves as presented in our preliminary experiment. The best results out of 5 runs obtained are presented. Table 3 shows the comparison of the approach in this paper with other available approaches in

the literature on all instances i.e. genetic algorithm and local search by Abdullah and Turabieh (2008), randomised iterative improvement algorithm by Abdullah *et al*.(2007a), graph hyper heuristic by Burke *et al*. (2007), variable neighbourhood search with tabu by Abdullah *et al*. (2007b), hybrid evolutionary approach by Abdullah *et al*. (2007c), extended great deluge by McMullan (2007), and non linear great deluge by Landa-Silva and Obit (2008). Note that the best results are presented in bold. The best results out of 5 runs obtained are presented. It can be seen our approach has better results on all datasets except *large*.

Table 3. Best results and comparison with other algorithms under relaxed stop condition

Dataset	Our method			M1	M2	M3	M4	M5	M6	M7
	Min	Max	Ave.							
Small1	**0**	2	0.8	**0**	**0**	6	**0**	**0**	**0**	3
Small2	**0**	4	2	**0**	**0**	7	**0**	**0**	**0**	4
Small3	**0**	3	1.4	**0**	**0**	3	**0**	**0**	**0**	6
Small4	**0**	1	1	**0**	**0**	3	**0**	**0**	**0**	6
Small5	**0**	1	0.6	**0**	**0**	4	**0**	**0**	**0**	**0**
Medium1	**78**	143	132.2	175	242	372	317	221	80	140
Medium2	**92**	186	124.6	197	161	419	313	147	105	130
Medium3	**135**	180	162.0	216	265	359	357	246	139	189
Medium4	**75**	160	111.2	149	181	348	247	165	88	112
Medium5	**68**	151	113.1	190	151	171	292	130	88	141
Large	556	835	738.6	912	-	1068	-	**529**	730	876

Note:
M1: Genetic algorithm and local search by Abdullah and Turabieh (2008)
M2: Randomised iterative improvement algorithm by Abdullah *et al*. (2007a)
M3: Graph hyper heuristic by Burke *et al*. (2007).
M4: Variable neighbourhood search with tabu by Abdullah *et al*. (2007b)
M5: Hybrid evolutionary approach by Abdullah *et al*. (2007c)
M6: Extended great deluge by McMullan (2007).
M7: Non linear great deluge by Linda-Silva and Obit (2008)

Fig. 5(a), (b) and (c) show the box plots of the cost when solving *small*, *medium* and *large* instances, respectively. The results for the *large* dataset is less dispersed compared to *medium* and *small* (worse dispersed case in these experiments). We believe that the neighbourhood structures (N1 and N2) applied to the *large* datasets are able to force the search algorithm to diversify its exploration of the solution space by moving from one neighbourhood structure to another even though there may be fewer and more sparsely distributed solution points in the solutions space since too many courses are conflicting with each other. When comparing between *small* and *medium* datasets, Figure 5 (b) shows less dispersion of solution points compared to Figure 5 (a). Again, applying the same neighbourhood structures (N2 and N3) for both instances most likely does not result in similar behaviour of the search algorithm. This can be supported by Figure 5 (a) where the dispersion of solution points for

small datasets is not consistent from one to another. For example *small2* in Figure 5 (a) shows worse dispersion compared to *small4*. From these experiments, we believe that the size of the search space may not be dependent on the problem size due to the fact that the dispersion of solution points are significantly different from one to another, even though the problems are from the same group of datasets with the same parameter values.

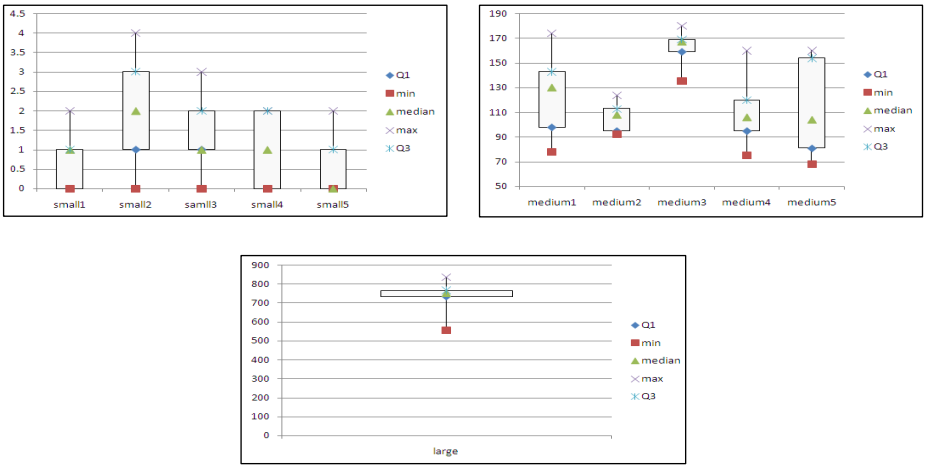

Fig. *5*(a), (b) and (c). Box plots of the penalty costs for *small*, *medium* and *large* datasets, respectively

5 Conclusion and Future Work

This paper has focused on investigating the application of a combination of a great deluge and tabu search algorithms with a set of neighbourhood structures. Preliminary comparisons indicate that this algorithm is competitive with other approaches in the literature. Indeed, it produced seven solutions that were better than or equal to the published penalty values on these eleven instances although it did require significant computational time for the medium/large problems. Future research will be aimed to test this algorithm on the International Timetabling Competition datasets (ITC-2007) introduced by the University of Udine.

References

1. R Bai, EK. Burke, G Kendall and B McCollum. “A Simulated Annealing Hyper-heuristic for University Course Timetabling Problem”. (Abstract) PATAT '06, Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, ISBN 80-210-3726-1, pp345-350 (2006).
2. MAS. Elmohamed, P Coddington, and G Fox. A comparison of annealing techniques for academic course scheduling. The Practice and Theory of Automated Timetabling II:

Selected Papers from 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT II), Toronto, Canada, Lecture Notes in Computer Science 1408, Springer-Verlag. (Editors: E.K. Burke and M. Carter), pp 92-112 (1998).
3. EK Burke, G Kendall and E Soubeiga, A tabu search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9(6), 451-470 (2003).
4. R Alvarez-Valdes, E Crespo and JM Tamarit. Design and implementation of a course scheduling systems using tabu search: Production, manufacturing and logistics. European Journal of Operational Research, 137, pp 512-523 (2002).
5. R Lewis and B Paechter. New crossover operators for timetabling with evolutionary algorithms. *Proceedings of the 5th International Conference on Recent Advances in Soft Computing* (ed. Lotfi), UK, December 16th-18th, pp 189-194 (2004).
6. R Lewis and B Paechter. Application of the groping genetic algorithm to university course timetabling. Evolutionary Computation in Combinatorial Optimisation (eds. Raidl and Gottlieb), Springer Lecture Notes in Computer Science Volume 3448, pp 144-153 (2005).
7. K Socha, J Knowles and M Samples, A max-min ant system for the university course timetabling problem. *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, Springer Lecture Notes in Computer Science Volume 2463, 1-13 (2002).
8. B McCollum, A Schaerf, B Paechter, P McMullan, R Lewis, A Parkes, L Di Gaspero, R Qu, EK Burke. Setting the research agenda in automated timetabling: The second international timetabling competition, Accepted for publication to INFORMS Journal of Computing (to appear 2009).
9. EK Burke, G Kendall and E Soubeiga, A tabu search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9(6), 451-470 (2003).
10. EK Burke, Y Bykov, J Newall and S Petrovic, A Time-Predefined Approach to Course Timetabling, *Yugoslav Journal of Operational Research* (YUJOR), Vol 13, No. 2, 139-151 (2003).
11. L Di Gaspero and A Schaerf. Multi-neighbourhood local search with application to course timetabling. In Emund Burke and Patrick De Causmaecker, editors, Proc. Of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2002), selected papers, volume 2740 of Lecture Notes in Computer Science, Springer, 262–275 (2003).
12. R Lewis and B Paechter, New crossover operators for timetabling with evolutionary algorithms. *Proceedings of the 5th International Conference on Recent Advances in Soft Computing* (ed. Lotfi), 189-194 (2004).
13. P Kostuch and K Socha, Hardness Prediction for the University Course Timetabling Problem, Proceedings of the Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004), Coimbra, Portugal, April 5-7, 2004, Springer Lecture Notes in Computer Science Volume 3004, 135-144 (2004).
14. P Kostuch, The university course timetabling problem with a three-phase approach. Practice and Theory of Automated Timetabling V (eds. Burke and Trick), Springer Lecture Notes in Computer Science Volume 3616, 109-125 (2005).
15. S Abdullah, EK Burke and B McCollum, An investigation of variable neighbourhood search for university course timetabling. *The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications* (*MISTA 2005*), 413-427 (2005).
16. H Asmuni, EK Burke and JM Garibaldi, Fuzzy multiple heuristic ordering for course timetabling. The Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI05), 302-309 (2005).
17. S Abdullah, EK Burke and B McCollum, Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling. In: Metaheuristics Progress in Complex Systems Optimization, Springer, 153-169 (2007).
18. S Abdullah, EK Burke and B McCollum, A Hybrid Evolutionary Approach to the University Course Timetabling Problem. IEEE Congres on Evolutionary Computation, ISBN: 1-4244-1340-0, 1764-1768 (2007).

19. P McMullan. An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487, pp538-545 (2007).
20. S Abdullah and H Turabieh, Generating university course timetable using genetic algorithms and local search. The Third 2008 International Conference on Convergence and Hybrid Information Technology ICCIT, vol. I, 254-260 (2008).
21. D Landa-Silva and JH Obit. Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), IEEE Press, 8.11-8.18 (2008).
22. T Müller, ITC2007: Solver Description, Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, (2008).
23. Z. Lu and J. Hao. Adaptive Tabu Search for Course Timetabling. European Journal of Operational Research (2009), doi:10.1016.j.ejor.2008.12.007.
24. L Di Gaspero, B McCollum and A Schaerf, The Second International Timetabling Competition (ITC2007): Curriculum-based Course Timetabling Track, the 14th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (2007).
25. M Clark, M Henz and B Love. QuikFix: A repair-based timetable solver. In Proceedings of the 7th PATAT Conference (2008).
26. F De Cesco, L Di Gaspero and A Schaerf. Benchmarking curriculum-based course timetabling: Formulations, data format, instances, validation and results. In Proceedings of the 7th PATAT Conference (2008).
27. MJ Geiger. An application of the threshold accepting metaheuristic for curriculum-based course timetabling. In Proceedings of the 7th PATAT Conference (2008).
28. G Lach and ME Lubbecke. Curriclum-based course timetabling: Optimal solutions to the udine benchmark instances. In Proceedings of the 7th PATAT Conference (2008).
29. R Lewis. A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum 30 (1), 167-190 (2008).
30. B McCollum, A perspective on bridging the gap between theory and practice in university timetabling. LNCS 3867, Springer-Verlag, 3-23 (2007).
31. EK 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(1), 177-192 (2007).
32. B McCollum, EK Burke, P McMullan. A review and description of datasets, formulations and solutions to the University Course Timetabling Problem. To be submitted April 2009 to the Journal of Scheduling.
33. M Chiarandini, M Birattari, K Socha and O Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling* 9(5), pp 403-432 (2006).
34. Z. Lu, J. Hao. Solving the Course Timetabling Problem with a Hybrid Heuristic Algorithm. AIMSA 2008, LNAI 5253, pp. 262–273, 2008. Springer-Verlag Berlin Heidelberg 2008.
35. M Dimopoulou and P Miliotis. An automated university course timetabling system developed in a distributed environment: A case study. *European Journal of Operational Research* 153(1), pp 136-147 (2004).
36. G Dueck. New Optimization Heuristics. The great deluge algorithm and the record-to-record travel. Journal of Computational Physics 104, pp 86-92 (1993).
37. F Glover and M Laguna. Tabu Search. Kluwer Academic, Boston (1997).
38. E.K. Burke and G. Kendall. Search methodologies. Introductory tutorials in optimization and decision support techniques. Springer (2005).
39. L. Di Gaspero, B.McCollum and A. Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, Belfast, United Kingdom, August (2007).