TWO PHASE HEURISTIC ALGORITHM FOR THE UNIVERSITY COURSE TIMETABLING PROBLEM: THE CASE OF UNIVERSITY OF DAR ES SALAAM

AR Mushi
Department of Mathematics, Box 35062, University of Dar es salaam, Tanzania
allen.mushi@gmail.com

ABSTRACT
University course timetabling is the problem of scheduling resources such as lecturers, courses, and rooms to a number of timeslots over a planning horizon, normally a week, while satisfying a number of problem-specific constraints. Since timetabling problems differ from one institution to another, this paper investigated the case of the University of Dar Es salaam, based on the combination of Simulated Annealing (SA), and steepest descent in a two-phase approach. Solutions have been generated which greatly outperform the manually generated ones. Furthermore, the method compares well with previous work on Tabu Search but with faster execution time and higher quality on rooms allocation. It is concluded that the approach gives good results given a careful selection of parameters.

Keywords: Timetabling Problem, Simulated Annealing, Combinatorial Optimization, Steepest Descent

INTRODUCTION

Timetables play very important roles especially in educational institutions. Essentially, timetabling involves the allocation of resources such as courses, lecturers and rooms to a fixed time period while satisfying a given number of constraints. These constraints can be divided into hard and soft. Hard constraints must be satisfied, while it is desirable, though not essential, to satisfy soft constraints. Many varieties of timetables exist, including high school, University Course, and Examination timetables, where each group differs significantly from another. Timetabling problems are of much interest because of their real life applications. However, they are known to be NP-Hard (Cooper and Kingston, 1996), and vary from one institution to another. This paper addresses the course timetabling problem at the University of Dar es Salaam (UDSM) in Tanzania, East Africa.

Many approaches have been suggested in tackling this problem, including Mathematical Programming (de Werra 1985, Daskalaki *et al.* 2004), Constraint logic programming (Abdennadhar and Marte, 1998; Panagiotis 1998); Graph Coloring (Miner 1995) as optimal solution strategies. However, since it is NP-Hard, no optimal algorithm is known which can give a solution within reasonable time. Many authors have attempted to solve variations of the problem by applying heuristic techniques which do not guarantee an optimal solution but produce a better solution which is tolerable in many timetabling applications. White and Xie (2001) and White et al (2004), successfully applied Tabu Search techniques with longer-term memory to examination timetables. Genetic algorithms have been widely attempted for specific Universities including Corne *et al.* (1993), Hiroaki *et al*. (2004), Hitoshi *et al.* (2002) and Kragelund (1997).

The most recent heuristics include Harmony Search, Al-Betar *et al.* (2008), and Particle collision, Abuhamdah & Ayob (2009). Simulated Annealing has also been used with some success in a number of problems. A comparison of annealing techniques is given by ElMohamed and Fox (1998). Nanthini and Kanmani (2009) provide a survey of Simulated Annealing methodologies for University course timetabling problem for the past ten years. They conclude that despite the success of Simulated Annealing, the real world application depends on the institution where it is applied; therefore finding a standard framework is difficult. In general, articles describing solution procedures can easily be found but few discuss the actual implementation of a practical problem. Some of these practical implementations include Stallaert (1997), Phala (1988) and Daskalaki *et al*. (2004). Mushi (2006) described UDSM course timetabling problem using Tabu Search heuristic and present promising results.

This paper presents another heuristic approach to UDSM course timetabling and compares the results with the previous work on Tabu Search. The rest of the paper is organized as follows; description of the course timetabling at UDSM, description of the two-phase algorithm, and implementation details are discussed before presenting summary of results and conclusion.

Course timetabling at UDSM

UDSM has been undergoing Institutional Transformation which involved among other issues, the expansion of student enrolment. Currently, there are approximately 15,000 students in the main campus. The exercise necessitated a central timetable for all programs so as to optimize the use of the available resources. This paper focuses on course timetabling at the main campus which involves approximately 15,000 students. There are two semesters per each academic year with approximately 750 courses per semester. These are also increasing annually due to curriculum reviews compelled by the challenge to meet the rapidly changing needs of the job market. There are currently 106 rooms which include classrooms and laboratories, about 1000 lecturers, and 15,000 students to be scheduled on five days week. Each day is made up of 13 one-hour timeslots starting from 7.00 a.m. to 8 p.m. Thus, there are 65 time slots for the whole timetable period. Lunch breaks are not considered although it is desired to minimize the use of these hours especially on Fridays where Muslims have to go for prayers. The necessary data includes the student-course registrations, lecturer-course assignments, course requirements, room sizes and room types.

The following definitions are applied, as described in the previous work by Mushi (2006);

Course – A unit of subject content to be taught to a particular group of students.

Event – An assignment of lecturer, room and a course to a one hour time interval. A course can have several events according to the number of hours set in the curriculum.

Lecture – A set of events of the same course, which are required to be scheduled together in the same room and the same time. A lecture can have one or more events.

Block – A lecture with more than one consecutive events.

The hard constraints are as follows;

1. No student can attend more than one lecture at the same time

1. No lecturer can teach more than one lecture at the same time

1. No room can occupy more than one lecture at the same time

1. No room can be assigned a lecture with more students than its capacity

1. Some courses are scheduled in blocks of more than one hour, these restrictions must be respected.

The soft constraints include;

1. As much as possible, minimizing the use of early morning (7.00 a.m.), lunch hours (13-14) and evening hours (18-20).

1. Minimizing the room space wastage by assigning lectures to rooms which have a capacity as close to the student numbers as possible. This also increases the chance of having a room-feasible timetable.

1 . Specifically minimizing the use of Friday 13-14 hour and 18-20 hour slots to allow for Muslim and Seventh Day Adventists (SDA) prayers respectively.

1. Minimizing continuous lecturers/blocks of the same course in a day. It is preferred to spread them over the week as much as possible.
1. As much as possible, evening lectures starting from 18 to 20 hours should be assigned to rooms with standby generators so as to minimize loss of lecture hours due to frequent power cuts.
1 . Special preferences by lecturers, students and University administration.

The Two- Phase Algorithm

The algorithm is implemented in two-phases where phase I deals with timeslot movements and phase II deals with room movements. The algorithm is a combination of simulated annealing and steepest descent.

Phase I uses Simulated Annealing; this is a global heuristic technique which tries to avoid falling into local optima by imitating the physical cooling process. Bad solutions are accepted by following an exponential function which depends on temperature value and worsening of the solution (Ingber 1993, Reeves 1993). Simulated Annealing is chosen because of its vast successful applications to combinatorial optimization problems. Timeslot selection subject to curriculum restrictions is the most combinatorial challenging component of the problem. Simulated Annealing is therefore used to tackle this part of the problem, while ignoring room lower bound restrictions for ease of computations.

Phase II uses steepest descent because at this point the timetable is feasible and timeslot optimized. Room moves are essential because they can help to reduce room under-utilization by some events. Shaking up of rooms while fixing timeslots works well with greedy algorithms such as steepest descent. This algorithm accepts only improved solutions and therefore best suited to this phase. The algorithm is as shown in Figure 1 below. It should be noted that, Simulated Annealing and Steepest descent are general algorithms, and the challenging part is design of use of variables; including the structure of the neighbourhood and type of moves suitable for the UDSM timetabling problem and fine-tuning of parameters as addressed in this paper.

Implementation of the two-phase algorithm

It is necessary to discuss all the design features of this algorithmic framework as applied in the implementation. The use of this framework requires careful choice of parameters and input data structures. These include choice of moves, cooling schedules, objective function structure, representation of input data structures including solution, courses, rooms, and modelling of constraints. This section discusses each of these implementation matters;

A move determines the next trial solution in the solution space. Two types of moves have been used in this implementation depending on the phase of the algorithm. The move in phase I is as follows;

1. Select a random event *e* in the set of all possible events
2. Select randomly a new timeslot *t* in the set of all possible timeslots.
3. Assign the new timeslot *t* to the event at position *e*. If *e* is a member of a block of events, assign sequential timeslots including *t* to all members of the block.

This is identical to several moves described by Thomson *et al*. (1996), ElMohamed (1998) and Reeves (1999). It is a simple move, yet it allows a well balanced mix of events among all timeslots.

```
Timetable_Algorithm
{ Initialize parameters T, Freeze;
Get Initial Solution (So);
Phase I: Time moves
while(T > Freeze){
Get a timeslot-move solution S in neighborhood of So(S∈N(So))
        Given an objective function f, find $\sigma = f(S) - f(so)$;
        if($\sigma < 0$)
                Accept solution (So = S);
        else{ Generate a random number $x$ between (0,1);
                if( $x < e^{-\sigma/T}$ )
                        Accept solution So = S;
                else    Reject solution;
                }
        T = $\alpha$(T); where $\alpha$ is a cooling function.
        }

Phase II: Room moves – Steepest descent
Initialize T;
while(T > Freeze){
Get a room-move solution S in neighborhood of So(S∈N(So))
        Given an objective function $f$, find $\sigma = f(S) - f(so)$;
        if($\sigma < 0$)
                Accept solution (So = S);
        Else    Reject solution;
        T = $\alpha$(T);
        }
```

Figure 1: The general timetable algorithm

The second type of move is implemented in the second phase where the timeslots are fixed and changes are made to the assigned rooms as follows;

1. Randomly select a room r in the set of all possible rooms
2. Find an event e associated with a room with the highest number of empty seats. This is basically a room which is most under-utilised.
3. Assign the new room r to event e. If e is a member of a block of events, assign r to all events of the block.

This increases the chance of finding a better room assignment since we only concentrate on the worst cases.

The size of this neighbourhood is $|N(s)| = |r|$, where $|r|$ = the total number of rooms.

Note that it is not necessary to check for feasibility when making a move, it is penalised highly in the objective function.

Cooling schedules

The rate at which temperature is reduced is vital to the success of the algorithm as it controls the rate of cooling. One of the most widely used schedules involves geometric reduction function;

$$\alpha(t) = at, \quad where \quad a < 1$$

Another cooling schedule as suggested by Lundy and Mees (1986) which involves a much slower cooling rate is also tested;

$$\alpha(t) = \frac{t}{(1 + \beta t)} \quad \text{where } \beta < 1.$$

Creating an initial timetable
The initial solution must be feasible by satisfying all hard constraints but can violate soft constraints. The method used must be quick and easy to generate an initial solution. This is done by assigning events to the earliest available timeslot and available feasible room. To simplify searching, both rooms and events are sorted in increasing order of their sizes.

If no suitable timeslots or rooms have been found for the block of events of a given course, the block is kept aside and reported as infeasible and can be dealt with manually.

Representation of the Problem
A solution is represented as a list s of events, where each event stores a course, timeslot assigned and allocated room as shown in Table 1.

Event		0	1	2	…u-1
Course ID	0	c_o	c_1	c_2	…
Timeslot	1	t_o	t_1	t_2	…
Room ID	2	r_o	r_1	r_2	…

Table 1: Solution data structure

The solution is therefore a matrix s_{ux3}, such that s_{ej} = solution value associated with event e for resource j, and resource j =
$$\begin{cases} 0 & course \\ 1 & timeslot \\ 2 & room \end{cases}$$

Courses are represented by a list of course sizes, units and maximum number of consecutive hours per each lecture (block size). Each unit is equivalent to one lecture hour per week. Thus, a three unit course with a maximum block of 2 hours per lecture will require scheduling of one 2-hour block of lecture session and an additional 1-hour lecture. This is stored as a 3-rows table, where the 1st row stores size, 2nd row stores units and 3rd row stores the maximum block size, as shown in Table 2.

		0	1	2	…n-1
Course size	0	d_o	d_1	d_2	…
Units	1	u_o	u_1	u_2	…
Max block size	2	b_o	b_1	b_2	…

Table 2: Course data structure

Rooms are represented by an array whose index represents room numbers and the content stores the room sizes.
A conflict matrix is used to detect course conflicts. Given a set of n courses, the conflict matrix is an $n_x n$ matrix M such that
$$M_{ij} = \begin{cases} 1 & \text{if course i clashes with course j} \\ 0 & \text{Otherwise} \end{cases}$$

Courses i and j clash if they have at least one student or lecturer in common.
This is clearly a triangular matrix, since course i colliding with course j is the same as course j colliding with course i.

Objective function is the weighted sum of objectives f_i with weights λ_i representing the importance of each constraint. Thus the objective is to minimize a function of the form $f(s) = \sum_{i=1}^{k} \lambda_i f_i(s)$, where s is a solution for a set of k constraints. The objective function caters for both hard and soft constraints. A higher penalty is assigned to hard constraints so as to discourage their selection. Since there are many soft constraints, it is impossible to satisfy all of them, so that all hard constraints are included but only a selected number of soft constraints in order of their importance to the timetabling requirements. Other constraints, such as special preferences, can be dealt with manually. The hard constraints included in the objective function are student clashes, lecturer clashes, room clashes, and room size violations. Soft constraints implemented are the need to minimize distances between events of the same course, minimizing the use of special

times, evening standby generators, and maximizing room utilization.
The implementation of each type of constraints as part of the objective function is hereby discussed; the same constraints as modelled in Mushi (2006) are used and summarized as follows;

$$\text{Minimize } f(s) = \sum_{i=1}^{7} \lambda_i f_i(s)$$

$$= \text{Minimize } \lambda_1 \sum_{t} \sum_{\substack{(i,j)\in E_t \\ i<j}} M_{ij}$$

$$+ \lambda_2 \sum_{t} \sum_{(i,j)\in E_t} a_{ij}$$

$$+ \lambda_3 \sum_{e:i=s_{e0},r=s_{e2}} b_{ir} + \lambda_4 \sum_{i} \sum_{\substack{(p,q)\in C_i \\ p<q}} \frac{1}{(t_p - t_q)^2}$$

$$+ \lambda_5 \sum_{e\in H} s_{e1} +$$

$$\lambda_6 \max_{e} \{Cap(s_{e2}) - Cap(s_{eo})\}$$

$$+ \lambda_7 \sum_{e} \delta_e$$

Where f_1(s) represents student lecturer collisions, f_2(s) is for room clashes, f_3(s) is for room size violations, f_4(s) is distance between events of the same course, f_5(5) is for use of special times, f_6(s) is for room utilization and f_7(s) takes care of the standby generator constraints.

SUMMARY OF RESULTS

The algorithm was tested on a course timetabling problem previously solved by manual methods for semester 1 of the 2003/2004 academic year. A program is written in C++ and tests run on a 2.4GHz, Pentium 4 processor. Table 3 shows data for the specific problem.

Data	Value
Students	8161
Lecturers	607
Rooms	106
Courses	729
Total events	1570
Total timeslots	65

Table 3: Data for the tested problem at UDSM

Since the number of first years is normally difficult to determine during timetabling, it has been good practice to include a single dummy student to represent first years in each programme, since they have similar core courses per programme. It is difficult to determine the first year numbers because timetables are supposed to be released before first years have arrived and registered to the University. Table 4 shows the weights used in the cost function for each type of constraint.

Weight	Value	Description
λ_1 - λ_3	100	Hard constraints
λ_4	10	Distance between events
λ_5	4	Friday Muslim prayers
	4	Seventh day Adventists
	2	Lunch times
	1	Morning and evening times
λ_6	1	Room utilization
λ_7	3	Standby generators

Table 4: Weights used in the objective function

These weights have been assigned according to the importance of each constraint. Hard constraints have a large value of 100 which was found to be sufficient to discourage infeasible solutions from selection. Weights on the soft constraints were set according to the experience on user needs. For example, Friday prayers are slightly more important than general lunch time violations. These values were picked after thorough testing of different parameters and reflect their results to the actual output requirements.

The algorithm was tested by varying the cooling schedules, α, β and freezing points. In both cases, an initial temperature value of 100 was used; higher temperature values did not show any improvement on the final solution. Table 5 shows a summary of results for the tested strategies. Obviously, the initial solution is the same for both cases, and in this case the value was 3094.72. The resulting cost for each phase and the percentage of improvement from the initial solution is shown. The first group of columns show the performance by geometric function, followed by Lundy & Mees (1986). The two schedules have different freezing points as shown.

Table 5 also shows the quality of each solution in terms of the improvement in achieving constraints. It is worth noting that, the presented results comes from the algorithm whose parameters have been thoroughly tested over a wide range of possible values (fine-tuning) to achieve parameters used in the results.

Results presented in each column were obtained by picking the average solution on the set of runs for different random number seeds. In both cases, initial and final solutions were found to be feasible by satisfying all hard constraints. The cost improvement from initial solution is very high. The best solution was obtained by the use of geometric function after 3,323.84 seconds which is approximately 55 minutes. By the nature of timetabling applications, this time is tolerable.

Initial Cost = 3094.72			Geometric			Lundy&Mees	
Phase I Cost			209.9700	204.9590	203.4910	203.6880	203.8340
Final Cost			207.9700	204.9590	201.4910	203.6880	203.8340
% Cost Improvement			0.9322	0.9331	0.9342	0.9335	0.9335
Time(s)			508.9530	2088.2000	3323.8400	2337.1100	3642.4500
α			0.9950	0.9990	0.9995		
β						0.1000	0.0100
Geometric Freezing point			0.0010	0.0001	0.0001		
Lundy&Mees freezing point						0.0010	0.0100
Performance	Initial	Student/Lecturer Collisions	0.0000				
		Room clashes	0.0000				
		Room size violations	0.0000				
		Events spread	2384.7200				
		Special time violations	510.0000				
		Largest room gap	200.0000				
	Final	Student/Lecturer Collisions	0.0000	0.0000	0.0000	0.0000	0.0000
		Room clashes	0.0000	0.0000	0.0000	0.0000	0.0000
		Room size violations	0.0000	0.0000	0.0000	0.0000	0.0000
		Events spread	8.9734	4.9592	3.4911	2.6878	3.8344
		Special time violations	1.0000	0.0000	0.0000	1.0000	0.0000
		Largest room gap	198.0000	200.0000	198.0000	200.0000	200.0000

Table 5: Performance by parameter choices

Since both solutions were feasible, the introduction of the second phase only yielded a small improvement over the output of the first phase. However, this phase is still important, since it improves on room utilization.

Figure 2 represents the improvement in cost by iterations for the best two performances. Iterations presented here are in multiples of a thousand. There is a sharp drop in objective function value within the first 8000 iterations, followed by a slow convergence.

A closer look at Figure 2 shows that the geometric function converges to lower cost values when in the lower temperatures.

Manually generated solution as created in 2003/2004 academic year was used for comparison with the automatically generated best solution. Table 6 is a summary of the performances in terms of constraint violations and compared by previous work on Tabu Search (Mushi, 2006). Both cases were feasible by satisfying all hard constraints.

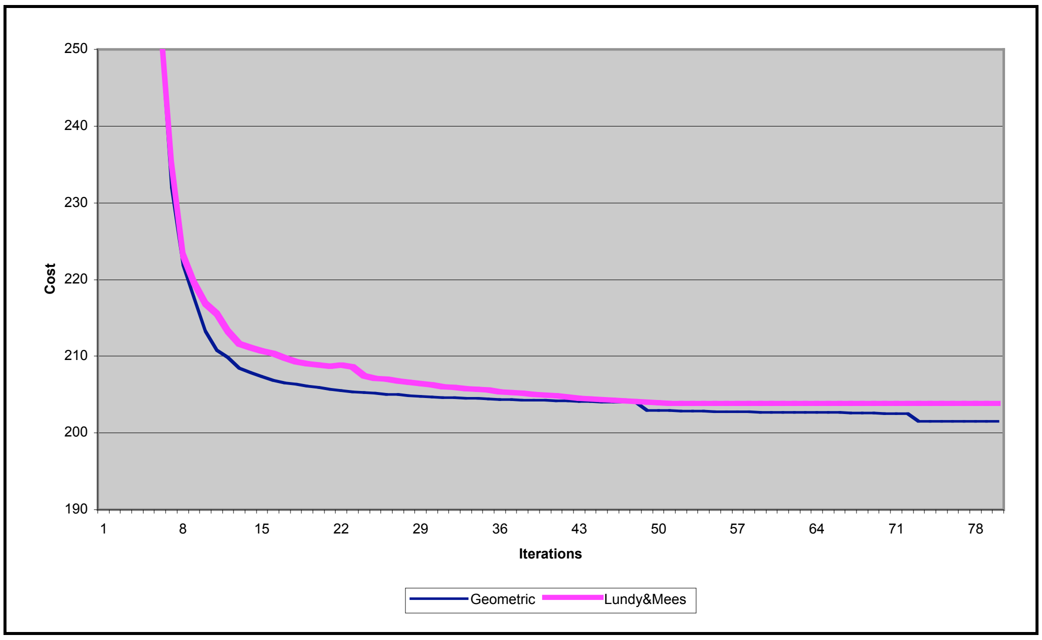

Figure 2: Performance Comparison between Geometric and Lundy & Mees

Constraints	**Violations in Manual**	**Violations in automatic - Two-Phase method**	**Violations in automatic - Tabu Search**
Student/Lecturer Collision	0	0	0
Room clashes	0	0	0
Room size violations	0	0	0
Course event gaps	253.75	3.491	1.74
Special time penalties	659	0	0
Largest room gap	488	198	Not Tested
Standby generators	0	0	0
Total violation cost	1400.75	201.491	1.74

Table 6: Comparison of Performances

There were more violations of the soft constraints in the manual than the automatic one. Clearly, the automatic system performs better than the manual system. Tabu Search seems to have performed better than the Two-Phase approach in terms of course-event gaps. However, Tabu Search method did not consider the process of reducing the largest room gap which has been improved by phase II of the current method. Furthermore, Tabu Search took 1 hour and 18 minutes to compute the best solution, while the Two-Phase method took only 55 minutes to do the same.

SUMMARY AND CONCLUSION

The paper has presented a Two-Phase approach which combines both Simulated Annealing and Steepest Descent. The Steepest descent has been used to improve on the room utilization which is not considered in the Simulated Annealing case. Summary of Results indicates that Simulated Annealing and steepest descent combination performs better, faster and with much less effort compared to manually generated results. The Geometric function performs generally better than Lundy & Mees (1986) although Lundy & Mees (1986) show better performance on higher temperatures. Change in parameter selections significantly changes the performance of the algorithm. Fine tuning of parameters is therefore essential in obtaining a better solution. It has been observed that the Tabu Search performs slightly better in terms of constraint violation detection, but took much longer than the presented algorithm. It has been clearly demonstrated that timetabling at UDSM can be improved by automated procedures.

Further Research

Traditionally, course timetabling at UDSM has been manually done. Since timetabling problem varies between universities, there is a room for further research on UDSM course timetabling. Specifically, only Tabu Search and this two-phase technique have been implemented for UDSM timetabling. Many general algorithmic techniques exists; including Genetic Algorithms (Ashish *et al.* 2010), Particle-Collision (Anmar and Masri 2009), Harmony Search (Mohammed *et al.* 2010) and many others which might bring better results. Furthermore, the current paper did not cover all soft constraints due to their vast numbers and conflicts. It is worth improving the quality of solutions by further addition of soft constraints.

REFERENCES

Abdennadher S Marte M 1998 University Timetabling using Constraint Handling rule, *In Actes des Journ'ees Francophones de Programmation en Logique et Programmation par Contraintes*

Abuhamdah A Ayob A 2009 Experimental Result of Particle Collision Algorithm for Solving Course Timetabling Problems, *Int. J. Comput. Sci. Network Secur.* **9**(9): 134-142

Al-Betar M Khader A Gani T 2008 A harmony search Algorithm for university course timetabling, In 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008) Montreal Canada

Anmar A Masri A 2009: Experimental Results of Particle Collision Algorithm for Solving Course Timetabling Problems, *Int. J. Comput. Sci. Network Secur.* **9(**9): 134-142

Ashish J Suresh J Chande P K 2010 Formulation of Genetic Algorithm to Generate Good Quality Course Timetable, *Int. J. Innov. Manag. Technol.* **1(**3) 248-251

Cooper Tim B Kingston J H 1996 The Complexity of Timetable Construction Problems, Springer Lecture Notes in Computer Science **1153**: 283-295

Corne D Fang H L Mellis C 1993 Solving the Modular Exam scheduling problem with Genetic Algorithms, Industrial and Engineering Applications of AI and Exp. Sys: Proc. of 6th International Conference, Edinburgh 1992

Daskalaki S Birbas T Housos E 2004 An Integer Programming formulation for a case study in University timetabling, *Eur. J. Operat. Res.* **153**: 117-135

de Werra D 1985 An Introduction to Timetabling, *European Eur. J. Operat. Res.* **19**: 151-162

ElMohamed M Fox G 1998 A Comparison of Annealing Techniques for Academic Course Scheduling, Practice and Theory of Automated Timetabling II, Selected Papers from the 2nd International Conference, PATAT'97, Edmund Burke, Mike Carter (Eds.), Lecture Notes in Computer Science, Springer

Hiroaki U Daisuke O Kenichi T Tetsuhiro M 2004 Comparisons of Genetic Algorithms for Timetabling Problems, *Systems and Computers in Japan* **35(7)**: 691-701

Hitoshi K Kondo M Sugimoto M 2002 Solving Timetabling Problems using Genetic Algorithms based on minimizing conflict heuristics, in Giannakoglou K Tsahalis Periaux Papailiou Fogrty(Eds), Evolutionary methods for design, Optimisation and Control, CIMNE, Barcelona

Ingber A L 1993 Simulated Annealing: Practice versus Theory, *J. Math. Comput. Modelling* **18(**11) 29-57

Kragelund L V 1997 Solving Timetabling Problem using Hybrid Genetic Algorithms, *Software – Practice and Experience*, **27**(10) 1121-1134

Lundy M Mees A 1986 Convergence of an Annealing Algorithm, *Math. Prog.* **34**: 111-124

Miner S ElMohamed Yau H W 1995 Optimizing Timetabling Solutions Using graph Colouring, In Timetabling Techniques, North-East Parallel Architecture Centre (NPAC), NPAC REU Program, Syracuse University, New York

Mohammmed A A Ahamad T K Taufiq A G 2010 A Harmony Search Algorithm for University Course Timetabling, Annals of Operations Research, Springer, DOI. 10.1007/s 10479-010-0769-z

Mushi A R 2006 Tabu Search Heuristic for University Course Timetabling Problem, *Afr. J. Sci. Technol. (AJST),* Science and Engineering Series, **7(1):** 34-40

Nandhini M Kanmani S 2009 A Survey of Simulated Annealing Methodology for University Course Timetabling, *Int. J. Recent Trend. Engin.,* **1**(2): 255-257

Panagiotis S Vigla E Karaboyas F 1998 Nearly Optimum Timetable Construction through CLP and Intelligent search, *Int. J. Artific. Intel.*

Tools **7(**4): 415-442

Phala J M 1988 A University Course Timetabling Problem, ORiON, **4(**2): 92-102

Reeves C R (Ed) 1993 Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford

Reeves C 1999 Landscapes, operators and heuristics search, *Ann. Operat. Res.* **86**: 473-490

Stallaert J 1997 Automated Timetabling Improves Course Scheduling at UCLA, *Interfaces* **27(**4): 67-81

Thompson J Dowsland K 1996 Variants of simulated annealing for the examination timetabling problem, *Ann. Operat. Res.* **63**: 105-128

White G M Xie S B Zonjic S 2004 Using Tabu Search with longer term memory and relaxation to create examination timetables, *Eur. J. Operat. Res.* **153**: 80-91

White G B Xie 2001 Examination Timetabling and Tabu Search with longer term memory. In E Burke and W Erben, (Eds), The Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science **2079**: 85-103, Springer-Verlag