A new approach for university timetabling problems

Article *in* International Journal of Mathematics in Operational Research · January 2013

DOI: 10.1504/IJMOR.2014.059537

3 authors:

Rakesh Prasad Badoni
Indian Institute of Technology Kharagpur
9 PUBLICATIONS 35 CITATIONS
SEE PROFILE

D. K. Gupta
Indian Institute of Technology Kharagpur
107 PUBLICATIONS 528 CITATIONS
SEE PROFILE

Anil Lenka
IBM India Pvt. Ltd, Bangalore
5 PUBLICATIONS 1 CITATION
SEE PROFILE

A new approach for university timetabling problems

Rakesh P. Badoni and D.K. Gupta*

Department of Mathematics,
Indian Institute of Technology,
Kharagpur 721 302, West Bengal, India
E-mail: rakeshbadoni@gmail.com
E-mail: dkg@maths.iitkgp.ernet.in
*Corresponding author

Anil K. Lenka

Wipro Technologies,
Bangalore 560 068, Karnataka, India
E-mail: aklenka@gmail.com

Abstract: A general university timetabling problem formulated as a constraint satisfaction problem is solved by a new backtrack-free algorithm. This algorithm incrementally constructs a solution as is done by the usual backtracking algorithm. Whenever a dead-end is encountered, rather than backtracking to the previously assigned variable, it resolves its cause by using a local search min-conflicts-hill-climbing algorithm and then continues further progressively. The proposed algorithm is tested on a small dataset and its representative timetable is presented. It is also tested on a number of randomly generated university timetabling problems. The performance of the algorithm measured in terms of the number of iterations and the amount of CPU time in seconds required is compared with the backtracking algorithm and the min-conflicts-hill-climbing algorithm. It is observed that our algorithm takes less CPU time and less number of iterations.

Keywords: constraint satisfaction problems; university timetabling problems; backtracking algorithm; min-conflicts-hill-climbing algorithm; backtrack-free algorithm.

Reference to this paper should be made as follows: Badoni, R.P., Gupta, D.K. and Lenka, A.K. (2014) ‘A new approach for university timetabling problems’, *Int. J. Mathematics in Operational Research*, Vol. 6, No. 2, pp.236–257.

Biographical notes: Rakesh P. Badoni received his MSc from Hemwati Nandan Bahuguna Garhwal University in 2005 and MTech from IIT Kharagpur, India in 2011. Currently, he is working as a Research Scholar in the Department of Mathematics, IIT Kharagpur. His area of research is on theoretical computer science.

D.K. Gupta received his PhD in Science from IIT Kharagpur, India in 1985. Later, he joined there as a faculty member in the Department of Mathematics.

Currently, he is working as a Senior Professor. His research interests focus on theoretical computer science, distributed databases, constraint satisfaction problems and numerical analysis.

Anil K. Lenka received his MCA from Berhampur University in 2002 and MS in Computer Science from IIT Kharagpur, India in 2007. Currently he is working as a Senior Consultant in Wipro Technologies, Bangalore, India.

1 Introduction

The university timetabling problems are one of the most important and challenging problems encountered in computer science (CS), operation research (OR) and artificial intelligence (AI). These are high dimensional multi-objectives combinatorial optimisation problems belonging to a class of NP-complete problems. The construction of their solution is extremely difficult and grows exponentially with size. A manual solution typically requires much effort and involves numerous reformulations of the problem's settings in order to find a compromise solution. A general university timetabling problem consists of weekly assignment of lectures for a number of subjects of a number of courses among a number of teachers with preferences in pre-specified time slots per day in a limited number of rooms of specified sizes. A solution to a university timetabling problem is to obtain a weekly schedule of meetings acceptable to all people involved satisfying hard as well as soft constraints as efficiently as possible. Hard constraints are those which can not be violated whereas soft constraints can be relaxed. For example, a hard constraint implies that neither a student nor a teacher can be physically in two different places at the same time. An example of a soft constraint can be to avoid time gaps between lectures on the same day.

In this paper, motivated from the work of Zhang and Zhang (1996) and Yoshikawa et al. (1995), we have proposed a new backtrack-free algorithm for solving a general university timetabling problem. This is done by converting its mathematical formulation as a constraint satisfaction problem (CSP) and proposing the backtrack-free algorithm combining the usual backtracking (BT) algorithm and a local search min-conflicts-hill-climbing (MCHC) algorithm for its solution. This algorithm incrementally constructs a solution as is done by the usual BT algorithm. Whenever a dead-end is encountered, rather than BT to the previously assigned variable, it resolves the cause of dead-end by using the MCHC algorithm and then continues further progressively. The proposed algorithm is tested on a small dataset and its representative timetable is presented. It is also tested on a number of randomly generated university timetabling problems. The performance of the algorithm is measured in terms of the number of iterations and the amount of CPU time in seconds required by the algorithm. On comparison with the BT and the MCHC algorithms, it is observed that our algorithm takes less CPU time and less number of iterations for the university timetabling problems considered.

This paper is organised as follows. Section 1 is the introduction. A brief literature review is given in Section 2. In Section 3, a general university timetabling problem, its mathematical formulation and some methods for its solution is described. In Section 4, preliminaries of CSP and the CSP formulation of general university timetabling problem described in Section 3 is discussed. In Section 5, the BT, MCHC and the new backtrack-free algorithms for solving university timetabling problems are described. In Section 6, these algorithms are tested on a simple dataset and its representative timetable is presented. They are also tested on a number of randomly generated university timetabling problems. The performance of the algorithm measured in terms of the number of iterations and the amount of CPU time in seconds required in comparison with the BT and the MCHC algorithms is summarised in a table. Finally, conclusions are included in Section 7.

2 Literature review

The university timetabling problems are extensively studied by a number of researchers (Cambazard et al., 2012; Abdullah and Turabieh, 2008; Nandhini and Kanmani, 2011; Lewis, 2008) and a number of approaches are proposed for their solutions. They have systematically categorised these problems, presented their mathematical formulations and described both exact and heuristic algorithms for their solutions. One such approach (Razak et al., 2010; Abdullah and Turabieh, 2008; Tiwari and Sharma, 2011; Wijaya and Manurung, 2009) converted it to a graph, in which the nodes correspond to lectures and the edges between the nodes correspond to the constraints and used graph colouring algorithms. The graph colouring algorithm assigns a limited number of colours to the nodes of the graph in such a way that no two nodes connected by an edge have the same colour. The number of colours correspond to the number of available time slots. Lewis (2008) discussed several meta-heuristic algorithms for solving them. Genetic algorithms (GAs) (Abdullah and Turabieh, 2008; Nandhini and Kanmani, 2011; Wong and See, 2009), tabu search (TS) (Wilke and Ostler, 2008) and simulated annealing (SA) (Abdullah et al., 2010; De Causmaecker et al., 2009) are also successfully applied to solve these problems. The timetable of the University of Dar Es salaam is given based on the combination of SA and steepest descent in a two-phase approach by Mushi (2012). The solution to them by a dual-sequence simulated annealing (DSA) algorithm which is employed as an improvement algorithm and then using the round robin (RR) algorithm to control the selection of neighbourhood structures within DSA is presented by Abdullah et al. (2010). De Causmaecker et al. (2009) presented a decomposed heuristic for solving a large, complex real-world university timetabling problem by grouping similar lectures so that the timetabling problems become less complex.

Recently, constraint-based reasoning (Cambazard et al., 2012; Alidaee et al., 2009) has gained importance for solving university timetabling problems. Wijaya and Manurung (2009) represented a university timetabling problem as a CSP and solved it by using GA. Banks et al. (1998) have employed an approach in which constraints were iteratively added to the CSP representation of university timetabling problems and then employed BT algorithm to solve them. Zhang and Zhang (1996) had combined a BT with a local search algorithm to find solutions to the university timetabling problems. Yoshikawa et al. (1995) solved a high school timetabling problem by combining a BT algorithm with a MCHC algorithm.

3 A general university timetabling problem

In this section, a detailed description of the general university timetabling problem and its mathematical formulation is described. The development of automated university timetables for academic institutions leads to several benefits such as reduction of time for its creation with minimum errors and satisfaction to all as all the hard and soft constraints are taken into account. A general university timetabling problem consists of weekly assignment of lectures for a number of subjects of a number of courses among a number of teachers with preferences in pre-specified time slots per day in a limited number of rooms of specified sizes. The problem involves courses, subjects, periods, rooms and teachers. Let there are a number of courses each having a specific number of subjects taken by a number of teachers having preferences in a given number of time periods and rooms of specified capacities. A lecture is represented by a quadruple (Course, Subject, Teacher, Room) and implies that a subject of a course taken by a teacher in a room. It is assumed that a subject requires a number of lectures for to be scheduled in distinct time periods. Also, the lectures of the subjects of a course can not be scheduled at the same time. However, lectures for subjects of different courses can be scheduled at the same time. Each lecture is to be taken by a teacher who can teach a number of subjects belonging to different courses. The capacity of each room can not exceed the lecture's strength. Let the time slots per day of the five days week is fixed. The constraints are described as follows.

- Hard constraints (HC)

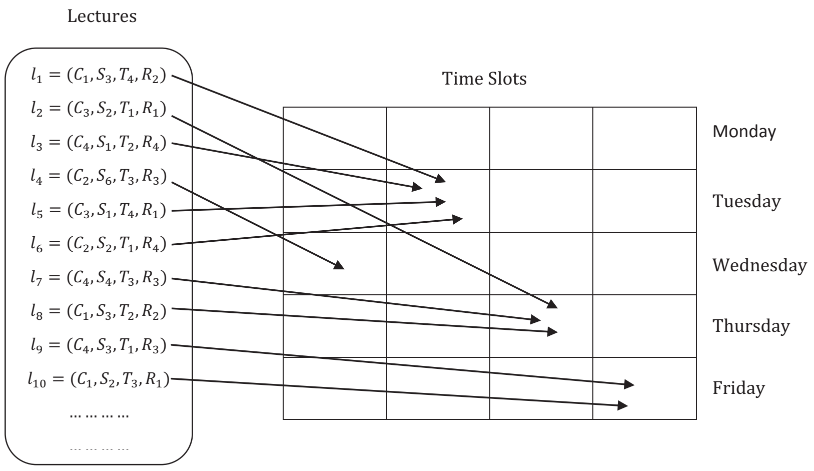
  1. The number of lectures of each subject is fixed.
  2. A teacher can teach only one lecture at a time.
  3. Only one lecture of a course can be held at a time.
  4. A room can host at most one lecture at a time.
  5. Rooms capacities and availabilities must be satisfied.
  6. Each subject of a course may have at most one lecture per day.
  7. The total number of lectures scheduled in a period must be less than or equal to the maximum number of rooms available.
  8. Each course is made up of a fixed number of subjects. A given set of periods must be involved in each subject.
  9. All lectures for a subject of a course must be scheduled at different times.

- Soft constraints (SC)

  1. Few lectures are to be scheduled in fixed periods.
  2. Each teacher may have a minimum and a maximum number of lectures per week.
  3. Teachers have preferences for specific time slots.

The aim is to find the solution as a weekly schedule of meetings acceptable to all people involved satisfying hard as well as soft constraints as efficiently as possible. This leads to a matrix with multi-valued entries, where each entry representing the lectures in a time slot of a day involving subjects of courses taken by teachers in rooms satisfying all the constraints. Figure 1 gives some representative entries to illustrate the solution of our university timetabling problem.

Figure 1 Solution entries for university timetabling problem


Lectures
$l_1 = (C_1, S_3, T_4, R_2)$
$l_2 = (C_3, S_2, T_1, R_1)$
$l_3 = (C_4, S_1, T_2, R_4)$
$l_4 = (C_2, S_6, T_3, R_3)$
$l_5 = (C_3, S_1, T_4, R_1)$
$l_6 = (C_2, S_2, T_1, R_4)$
$l_7 = (C_4, S_4, T_3, R_3)$
$l_8 = (C_1, S_3, T_2, R_2)$
$l_9 = (C_4, S_3, T_1, R_3)$
$l_{10} = (C_1, S_2, T_3, R_1)$
Time Slots
Monday
Tuesday
Wednesday
Thursday
Friday

Many approaches are used for their solutions. The sequential methods (Burke and Petrovic, 2002) order lectures using domain heuristics and then assign them sequentially into valid time periods so that they are not in conflict with each other. Graphs are used here for representing lectures as vertices and conflicts between them by edges. The construction of a conflict-free timetable can therefore be modelled as a graph colouring problem by taking each time period as a different colour. Search methods (Pothitos et al., 2012) explore exhaustively the whole search space consisting of all the possible solutions to timetabling problems. Generally, tree search methods are used. They are classified into two groups, namely BT algorithms and iterative improvement algorithms. In BT solutions are incrementally constructed whereas in iterative improvement, all variables have tentative initial values and some appropriate heuristics are used to minimise the constraint violations till either the optimal or near optimal solutions are obtained. In cluster methods (Shatnawi et al., 2010), the set of lectures is split into groups which satisfy hard constraints and then the groups are assigned to time periods to satisfy the soft constraints. Different optimisation techniques have been employed to solve the problem. The main drawback of these methods is that the clusters of lectures are formed and fixed at the beginning of the method leading to poor quality timetables. A variety of meta-heuristic approaches (Lewis, 2008), mostly inspired from nature and apply nature-like process, such as SA, TS, GAs and hybrid approaches have been investigated for timetabling. Meta-heuristic methods begin with one or more initial solutions and employ search strategies that try to avoid local optima. All of these search algorithms can produce high quality solutions but often have a considerable computational cost. In constraint-based methods (Rossi et al., 2006), a timetabling problem is modelled as a set of variables to which values have to be assigned in order to satisfy a number of hard and soft constraints. Usually, search methods with a large number of heuristics are used to improve the generation of university timetables.

4 Constraint satisfaction design of timetabling

In this section, the basic concepts of CSPs, its formulation of general university timetabling problem is described.

4.1 Preliminaries

The CSPs are emerging technologies for solving combinatorial optimisation problems and find applications in many areas like planning, scheduling, timetabling, routing, frequency assignment, design problems and so on. A CSP (Rossi et al., 2006) consists of three components which are variables, values and constraints. The problem is to find assignment of values to each variables such that all the constraints are satisfied. A CSP may be defined as a triple $P = (X, D, C)$, where

$X = \{X_1, X_2, \ldots, X_n\}$	is the set of variables.
$D = \{D_1, D_2, \ldots, D_n\}$	is the set of domains. Each domain D_i is a finite set of possible values which can be assigned to the variable X_i.
$C = \{C_1, C_2, \ldots, C_m\}$	is the set of m constraints. A constraint C_i is defined by a relation $C_i \subseteq \{D_{i1} \times D_{i2} \times \ldots \times D_{ik}\}$ defined on a subset $\{X_{i1}, \ldots, X_{ik}\}$ of X.

The solution to a CSP is an assignment of values to all the variable of X from D in such a way that all the constraint of C are simultaneously satisfied. A binary CSP is one in which each of the constraint involves at the most two variables. A graph (X, C) associated with a binary CSP is called a constraint graph. Binary CSPs are generally studied as any general CSP can easily be transformed into an equivalent binary CSP by using constraint binarisation (Bacchus and Van Beek, 1998). Given a CSP, the problem of finding all solutions in a CSP may be called constraint satisfaction generation problem (CSGP) to distinguish it from the enumeration problem (CSEP), search problem (CSSP) and decision problem (CSDP) variants which seek respectively, the total number of solutions, any one solution and answer to the question "is there a solution?". The CSP is in general an NP-Complete Problem. Two well known examples which illustrate a CSP are the N-Queen's problem and map-colouring problem (Dechter, 1990). In general, four approaches are used to solve CSPs. Retrospective algorithms (Dechter, 1990) guide its backtrack strategy when they can not proceed any further by deciding how and which variable to backtrack. One of the standard and simple retrospective algorithm is the BT algorithm. It attempts to assign values to a set of variables in some pre-determined order such that all the constraints among the variables are satisfied. Starting with the first variable it assigns each successive variables, from their respective set of domains, which is consistent with all the previous assignments. The process is continued until either a solution is found or until it may be concluded that no solution exists. If a variable is encountered which does not have a value in its domain which is consistent with the previous assignment, i.e., a dead-end is reached, the algorithm backtracks to the immediately preceding variable. It assigns the next available value to the variable and continues from there. Some of the other standard algorithms in this class are back-jumping, back-marking, deep and shallow learning. These algorithms differ in their backtrack strategies in a dead-end solution. The back-jumping algorithm retreats several level in the search tree in an attempt to locate the source of failure. The back-marking and the learning algorithms employ the concept of constraint recording to record the failures. Prospective algorithms (Nadel, 1989) attempt to achieve some form of local consistency before continuing the search for a solution. Some of the standard algorithms in this class are forward checking, partial look-ahead and full look-ahead. They remove from the future domains the values which are inconsistent with the current variable assignments. Theses algorithms differ in the degree of consistency they attempt to achieve after each assignment to the variables. In the forward checking algorithm the variables are assigned in some pre-determined order. Each time a variable is assigned a value, all the values which are inconsistent with respect to the current variable's assignment are removed from all the future variables domain. In the partial look-ahead algorithm the process of forward checking is followed by the removal of those values from the domain of each of the future variables which does not have at least one consistent value in the domain of all the future variables. Consistency algorithms are also used to prune values from consideration which do not meet local consistency criteria. These algorithms are generally used prior to the application of the algorithms for the search for a solution. These are also called simplification algorithms which reduces the constraint network to an equivalent but a more explicit network. Structure-based algorithms (Dechter, 1990) exploit the topological structure of the constraint graph. Some of the algorithms in this class are known as the cycle cutset, tree-clustering and cyclic clustering methods. These algorithms use the special structure of the constraint graph which admits simple solution to solve the original problem. One such structure of the constraint graph is the tree. A tree structured CSP can be solved in linear time. The cycle cutset method instantiates the variables in some predetermined order using the BT algorithm till the constraint graph reduces to a tree. At this point an algorithm to solve a tree structured CSP is invoked and the remaining problem is solved in linear time. The easily solvable property of a tree structured CSP is also used in the tree-clustering and the cyclic clustering algorithms. Repair-based algorithms (Minton et al., 1992) start with an inconsistent initial solution and then attempt to improve the solution using local repair heuristics. These algorithms generally outperform the constructive method because a complete assignment is more informative in guiding search than a partial assignment.

4.2 CSP formulation of the problem

In this section, the CSP formulation of the general university timetabling problem is described. Let there are g courses, $C = \{C_1, C_2, \ldots, C_g\}$, q subjects, $S = \{S_1, S_2, \ldots, S_q\}$, p periods, $P = \{P_1, P_2, \ldots, P_p\}$, m rooms, $R = \{R_1, R_2, \ldots, R_m\}$, and n teachers, $T = \{T_1, T_2, \ldots, T_n\}$. Let each course contains a number of subjects. A lecture is represented by a quadruple (Course, Subject, Teacher, Room) and implies that a subject of a course taken by a teacher in a room. It is assumed that a subject S_i requires l_i number of lectures for $i = 1, 2, \ldots, q$ to be scheduled in distinct time periods and the subjects are assigned to the teachers as per their specialiation or teaching interest. Also, the lectures of the subjects of a course can not be scheduled at the same time. However, lectures for subjects of different courses can be scheduled at the same time. Each lecture is to be taken by a teacher T_e, who can teach a number of subjects belonging to different courses. Each room R_f has a capacity which can not be exceeded by a lecture strength. Let d be the number of working days and s be the number of time-slots per day. The aim is to find the solution as a matrix $Y = (y_{ik})$, where the multi-valued y_{ik} represents the lectures in the time slot P_k of D_i^{th} day involving subject S_j of course C_l taken by teacher T_e in the room R_f satisfying a number of hard and soft constraints.

4.2.1 CSP modelling

The CSP modelling takes the set of lectures as variables, their domains and a set of constraints which are to be simultaneously satisfied as far as possible.

4.2.1.1 CSP variables

The set of variables are lectures. A lecture is represented by a quadruple (Course, Subject, Teacher, Room) and implies that a subject of a course taken by a teacher in a room. There are number of attributes for each component of a variable given as follows.

1 The attributes of each course are
(Course ID, Course Name, Number of Subjects, Subject Name).

2 The attributes of each teacher are
(Teacher Code, Teacher Name, Minimum Number of Lectures Allotted (MinL), Maximum Number of Lectures Allotted (MaxL), Number of Subjects, Allotted Subjects Names).

3 The attributes of each subject are
(Subject ID, Subject Name, Number of Lectures, Number of Students Enrolled).

4 The attributes of each room are
(Room Name, Room Capacity).

Assigning a value to a variable implies a time period to each of the lectures with different values of course, subject, teacher and room.

4.2.1.2 Domains of CSP variables

Since all variables are lectures, so their domains are same representing the set of all time periods given by $P = \{P_1, P_2, \ldots, P_p\}$, where p is the total number of periods in a week. The courses, subjects, rooms and teachers domains are $C = \{C_1, C_2, \ldots, C_g\}$, $S = \{S_1, S_2, \ldots, S_q\}$, $R = \{R_1, R_2, \ldots, R_m\}$ and $T = \{T_1, T_2, \ldots, T_n\}$ respectively.

4.2.1.3 CSP constraints

- Hard constraints (HC)

1 The number of lectures of each subject is fixed.

$$\sum\{y_{ik}|k = 1, 2, \dots, p\} = l_i, \text{ for } i = 1, 2, \dots, q;$$

2 A teacher can teach only one lecture at a time. This means that a teacher can not be assigned more than one subject in the same time.

$$\sum\{y_{ik}|i \in T_l\} \leq 1, \text{ for } k = 1, 2, \dots, p;\ l = 1, 2, \dots, n;$$

Such constraints are denoted by a Teacher-Conflict matrix and is given as,

$$\textit{Teachers-Conflict} = (tc_{ij}) \text{ for } i = 1, 2, \dots, n;\ j = 1, 2, \dots, p; \text{ where}$$

$$tc_{ij} = \begin{cases} 1 & \text{if teacher } T_i \text{ is assigned more than one subjects at period } P_j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

3 Only one lecture of a course can be held at a time.

$$\sum\{y_{ik}|i \in C_l\} \leq 1, \text{ for } k = 1, 2, \dots, p;\ l = 1, 2, \dots, g;$$

4 A room can host at most one lecture at a time.

$$\sum\{y_{ik}|i \in R_l\} \leq 1, \text{ for } k = 1, 2, \dots, p;\ l = 1, 2, \dots, m;$$

5 Rooms capacities and availabilities must be satisfied.

R_i*.capacity* $\geq S_j$*.Number of Students Enrolled*,
for $i = 1, 2, \dots, m;\ j = 1, 2, \dots, q;$

where $S_j \in R_i$, i.e., the lecture of subject S_j held in room R_i.

R_i*.availability* $= 1$, if the room R_i $(i = 1, 2, \dots, m)$ is assigned for a lecture.

6 Each subject of a course may have at most one lecture per day.

$$\sum\{y_{ik}|i \in S_j\} \leq 1, \text{ for } j = 1, 2, \dots, q;\ k = 1, 2, \dots, d;$$

7 The total number of lectures scheduled in a period must be less than or equal to the maximum number of rooms available.

$$\sum\{y_{ik}|i = 1, 2, \dots, q\} \leq m, \text{ for } k = 1, 2, \dots, p;$$

8 Each course is made up of a fixed number of subjects. A given set of periods must be involved in each subject, i.e.,

$$\sum\{y_{ik}|i \in S_j\} \geq h_{jk}, \text{ for } j = 1, 2, \dots, q;\ k = 1, 2, \dots, p; \text{ where}$$

$$S_j \in C_l, \text{ for } l = 1, 2, \dots, g;$$

Also, H is a matrix denoted as $H = (h_{jk})$ of order $q \times p$ and is defined as

$$h_{jk} = \begin{cases} 1 & \text{if subject } S_j \text{ is taught at period } P_k, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

9 All lectures for a subject of a course must be scheduled at different times. Let a symmetric square matrix Subject-Conflict of order q represents conflicts between subjects, i.e.,

$$\textit{Subject-Conflict} = (sc_{ij}) \text{ for } i = j = 1, 2, \dots, q; \text{ where}$$

$$sc_{ij} = \begin{cases} 1 & \text{subjects } S_i \text{ and } S_j \text{ belongs to the same course and are} \\ & \text{scheduled at the same time, and} \\ 0 & \text{otherwise.} \end{cases}$$

- Soft constraints (SC)

1 Few lectures are to be scheduled in fixed periods. Let a_{ik} represents the scheduling of a lecture to a subject S_i in the period P_k, then

$$a_{ik} = \begin{cases} 1 & \text{when lecture scheduled at period } P_k, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Also, if b_{ik} represents their reassignment, then

$$b_{ik} = \begin{cases} 1 & \text{if there is a reassignment, and} \\ 0 & \text{otherwise.} \end{cases}$$

where,

$$b_{ik} \leq y_{ik} \leq a_{ik} \text{ for } i = 1, 2, \dots, q; \; k = 1, 2, \dots, p;$$

2 Each teacher may have a minimum and a maximum number of lectures per week.

$$T_l.\mathit{MinL} \leq \sum \{y_{ik} | i \in T_l\} \leq T_l.\mathit{MaxL}, \text{ for } l = 1, 2, \dots, n; \; k = 1, 2, \dots, p;$$

3 Teachers have preferences for specific time slots. This constraints of teachers availability to different periods is denoted by a Teacher-Period matrix and is given as,

$$\textit{Teachers-Period} = (tp_{ij}) \text{ for } i = 1, 2, \dots, n; \; j = 1, 2, \dots, p; \text{ where}$$

$$tp_{ij} = \begin{cases} 1 & \text{if teacher } T_i \text{ is available for period } P_j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Remark 4.1: The maximum number of lectures scheduled in a period are given by $\min(g, m, n)$.

5 Algorithms for university timetabling problems

In this section, in order to make our paper self sufficient, we shall first give the BT and the MCHC algorithms for solving CSP formulation of the general university timetabling problems and then develop a backtrack-free algorithm based on the combination of these two algorithms. Let TN is the total number of lectures to be scheduled over p periods.

5.1 BT algorithm

The BT algorithm incrementally attempts to extend a partial solution towards a complete solution by repeatedly choosing values for the unassigned variables. If a situation occurs when there are no values for an unassigned variables, that is a dead-end occurs then the most recent variable assignment is undone and another assignment for it is attempted. the process continues until a solution is obtained. The BT algorithm to find one solution can be given in Algorithm 1.

Algorithm 1 BT algorithm to find a solution

```
Input: A CSP formulation of University Timetabling Problem
 1:      $X$: the set of $TN$ variables;
 2:      $D$: the set of domains;
 3:      $C$: the set of constraints;
 4:      $S_0$: the solution set initially taken as empty;
Output: Either a solution or no solution.
 5: begin
 6: $i \leftarrow 1$;
 7: $D_i' \leftarrow D_i$;              // Copy $D_i$ in $D_i'$ for recovery in case of dead-end
 8: while $(i \leq TN)$ do
 9:     $X_i = Selectvalue(i)$;    // $Selectvalue(i)$ is a procedure to select a value
                                 for $X_i$ from $D_i$
10:     if $X_i$ is null then       // No value of $X_i$ found consistent with $S_{i-1}$
11:         $i \leftarrow i - 1$;         // Backtrack to the previously added variable
12:         $D_i' \leftarrow D_i$;
13:     else
14:         $S_i = S_{i-1} \bigcup \{a\}$;
15:         $i \leftarrow i + 1$;
16:         $D_i' \leftarrow D_i$;
17:     end if
18: end while
19: if $i = 0$ then
20:     return no solution;
21: else
22:     return $S_i$;                  // $S_i$ is the complete solution of CSP
23: end if
24: end;
```

BT algorithm calls the procedure $Selectvalue(i)$ to find a value for the current variable X_i, that is consistent with the current partial solution S_{i-1}. It returns a value for X_i if it is consistent with the partial solution S_{i-1} else returns null. This implies that a dead-end has occurred. In this case, BT search algorithm restores the domain D_i for X_i and looks for a new value for the previous variable X_{i-1}. The algorithm terminates when the values are assigned to all the variables and hence a solution is obtained. Otherwise it fails to find a solution. In terms of university timetabling problems, the assignment of a lecture X_i is said to be consistent in period a with the current partial solution S_{i-1}, if the course, teacher and room involves in lecture X_i are not engaged in period a before. The time complexity of this algorithm is of $O(p^{(TN)})$, where p is the domain size for each variable.

Algorithm 2 Procedure $Selectvalue(i)$

```
Input: None.
Output: Returns a consistent value from $D_i{}'$ with partial solution $S_{i-1}$.
1: begin
2: while $D_i'$ is not empty do
3:     select an element $a \in D_i'$, and remove $a$ from $D_i'$;
4:     if Consistent$(S_{i-1}, a)$ then     // Check the consistency of $S_{i-1}$ with $X_i = a$.
5:         return $a$;
6:     end if
7: end while
8: return null;
9: end;
```

5.2 MCHC algorithm

In this section, the MCHC algorithm is described. It is a combination of Hill-Climbing (HC) algorithm with some suitable heuristics. First of all, we shall review HC algorithm in brief. It is a local search algorithm. This algorithm starts from an initial solution S_0 obtained either by some method or generated randomly. It then enters in a loop that navigates the search space stepping from one solution S_i to its neighbour solution S_{i+1} until a best solution is obtained. The structure of moves depends on the specific problem, for example, it can be the change of the value of one single variable or the swap of the values of two different variables. The search is driven by a cost function f that estimates the quality of each solution. In our problem, the cost function f represents the total number of constraint violations corresponding to a variable assignment. Although the selection of the moves is based on the cost function, the precise way in which a selection takes place depends on the specific local search technique. The whole procedure stops either when a solution is reached, i.e., the cost function f is equal to zero, or when a predetermined number of iterations have being accomplished. In HC algorithm, a move is only accepted if it produces a solution at least as good as the current solution. The HC algorithm always selects the best assignment out of all the neighbours. In case of optimisation problems, it picks up the neighbour assignment which minimises the cost function among the assignments with the minimal number of violated constraints. When there is no better assignment than the current one, the search is stuck in a local optimum. The HC algorithm usually restarts the search from another randomly selected assignment. This algorithm repeats till the solution is found. The HC algorithm has to explore all the neighbours of the current solution before choosing a move but this takes a lot of time. To avoid exploring all neighbours of the current solution, a number of heuristics were proposed to find the next move. The most popular min-conflict heuristic (Minton et al., 1992) randomly chooses any variable which violates some constraints and then assigns that value to it from its domain which minimises the total number of violated constraints. More precisely, If a variable X_i is in conflict then the min-conflict heuristic finds a value v of X_i from $D(X_i)$ with the minimum number of conflicts. In case of tie, the random selection is made. The HC algorithm combined with min-conflicts heuristic is termed as MCHC algorithm. We take a label $\langle X, v\rangle$, a variable-value pair that represents the assignment of the value v to the variable X. A compound label $(\langle X_1, v_1\rangle, \langle X_2, v_2\rangle, \dots, \langle X_n, v_n\rangle)$ is used to denote the assignment of $(v_1, v_2, \dots, v_n)$ to $(X_1, X_2, \dots, X_n)$ respectively. A cost function is used to measure the total number of constraint violations by the current solution. The MCHC algorithm can be given in Algorithm 3.

Algorithm 3 MCHC algorithm

```
Input: A CSP formulation of University Timetabling Problem
 1:     $X$: the set of $TN$ variables;
 2:     $D$: the set of domains;
 3:     $C$: the set of constraints;
 4:     $f$: the cost function;
Output: A solution $S$ for the university timetabling problem.
 5: begin
 6: $i \leftarrow 0$;
 7: $S_i \leftarrow$ generate an initial solution;
 8: repeat
 9:     $Mincon(X, D, C, f, S_i, S_{i+1})$;
          // Procedure to get a minimum conflicting solution
10:     $i \leftarrow i + 1$;
11: until (No constraints violation or Termination criteria)
12: $S \leftarrow S_i$;
13: end;
```

Algorithm 4 **Procedure** $Mincon(X, D, C, f, S_i, S_{i+1})$

```
Input: None.
Output: Best Solution.
 1: begin
 2: $S \leftarrow S_i$;
 3: Select at random some label $\langle X_i, v_i \rangle$ for $X_i$ violating some constraints of C;
        // Randomly choosing a constraint violating variable
 4: for $v \in D(X_i)$ do   // Check the constraint violation
                             corresponding to each value of variable $X_i$
 5:     $f \leftarrow$ repair-improvement on $(S_i - \langle X_i, v_i \rangle) + \langle X_i, v \rangle$;
        // Calculate the cost function $f$ corresponding to the value $v$ of $X_i$
 6:     $BestSolutionSet \leftarrow$ Set of variable-value pairs with minimum $f$;
 7:     $v_{i+1} \leftarrow$ value of $X_i$ in $BestSolutionSet$;
 8:     $S \leftarrow (S - \langle X_i, v_i \rangle) + \langle X_i, v_{i+1} \rangle$;
 9: end for
10: $S_{i+1} \leftarrow S$;            // $S_{i+1}$ is the new improved solution
11: return $S_{i+1}$;
12: end;
```

We start with a randomly generated solution S_0. We select a conflicting variable X_i with value v_i from the partial solution S_k and calculate its cost function f. Now we calculate the cost function f of X_i corresponding to all the values of domain $D(X_i)$. We assign the value v to X_i if f is minimum corresponding to $X_i = v$ and replace $X_i = v_i$ by $X_i = v$ in the new partial solution S_{k+1}. If multiple selected values tie for the least f, one of them is chosen randomly. We choose another variable X_j for assignment if still $f > 0$ and total number of iterations are less than the maximum number of iteration (defined as a termination criterion). This process will continue until either we get a complete solution, i.e., $f = 0$, or the maximum number of iterations are complete. If at any instance we are not able to improve the solution quality then instead of BT we will start again by choosing a random solution other than the previous one and repeat the process. This instance is termed as local minima. If the probability that solution is trapped into the local minima without restarts is $\frac{1}{w}$, then an average of w restarts is needed. Hence the complexity of the algorithm is $O(w \times (TN)^2 \times p)$, where TN and p are the size of the variable set and domain set respectively.

Algorithm 5 Backtrack-free algorithm

```
Input: A CSP formulation of University Timetabling Problem
 1:        $X$: the set of $TN$ variables;
 2:        $D$: the set of domains;
 3:        $C$: the set of constraints;
 4:        $S_0$: initial solution taken as empty;
Output: best solution.
 5: begin
 6: BestSet = $S_0$;
 7: Assigned = $\{\}$;
 8: $i \leftarrow 1$;
 9: $k \leftarrow 0$;
10: while (all variables are not completely assigned in $S_k$) do
11:     Unassigned = $\{X\backslash$ Assigned$\}$;
12:     $a$ = SelectValue($X_i$);     // Procedure SelectValue($X_i$)
                                      chooses value $a$ for variable $X_i$
13:     if ($a$ is not found) then
14:         if (Unassigned = $\{\}$) then     // Last variable checked
15:             return solution;
16:         end if
17:         repeat
18:             Variable = SelectVariable($S_k$, Unassigned);
                // This procedure select a variable from Unassigned
19:             $S_k$ = MakeMove($S_k$, Variable);
                // This Procedure find the solution $S_k$ using MCHC algorithm
20:             if ($S_k$ is better than BestSet) then
21:                 BestSet = $S_k$;
22:             else
23:                 $S_k$ is stuck in local minima.
                    Choose another random solution and repeat the whole process;
24:             end if
25:             $S_{k+1} \leftarrow$ BestSet;
26:             Assigned = Assigned $\bigcup$ {Variable};
27:             $i \leftarrow i + 1$;
28:             $k \leftarrow k + 1$;
29:         until Termination Criteria;
30:     else
31:         $S_{k+1} = S_k \cup \{X_i = a\}$;
32:         Assigned = Assigned $\bigcup$ $\{X_i\}$;
33:         $i \leftarrow i + 1$;
34:         $k \leftarrow k + 1$;
35:     end if
36: end while
37: return $S_k$;
38: end;
```

5.3 Backtrack-free algorithm

In this section, a backtrack-free algorithm is developed by combining the BT algorithm with MCHC algorithm to solve the general university timetabling problem. The BT algorithm incrementally extends a partial solution until a dead-end is reached. At this point, a MCHC algorithm is used to resolve the dead-end by making local changes in the current partial solution. The process is continued until either a complete solution is reached or a predetermined number of iterations are completed. The MCHC algorithm is guided by a cost function representing the number of violated constraints in a partial solution. A number of heuristics can be used for obtaining the solution to general university timetabling problem.

The backtrack-free algorithm assigns the lectures to the time periods one by one such that the constraints are satisfied in order to get a feasible timetable. When a dead-end is encountered, the MCHC algorithm using a cost function resolves the reason of dead-end and progresses iteratively. This means that it looks randomly for a lecture that causes at least one conflict and moves it to the period in which it creates the minimum number of conflicts. This method accepts the selected move only if the cost function is improved or left unchanged. It has the capability of navigating plateaus where they are all trapped by local minima. It is terminated when the best solution obtained can not be further improved for predetermined number of iterations. If the probability that solution is trapped into the local minima without restarts is $\frac{1}{w}$, then an average of w restarts is needed. Hence the complexity of the algorithm is $O(w \times TN \times p)$, where TN and p are the size of the variable set and domain set respectively. Thus, the backtrack-free algorithm will always give a solution if $TN \leq p \times min(g, m, n)$, where g, m, and n are the number of courses, rooms and teachers respectively.

6 Experimental results

In this section, we have implemented the algorithms BT, MCHC and backtrack-free to solve our general university timetabling problem described in previous section using an object-oriented approach C++ running under Red Hat Linux Operating System on Intel Pentium® 4 PC with 512 MB RAM. A number of randomly generated datasets are used for this purpose. The test problems are randomly generated based on four parameters.

1 The number of variables.

2 The number of values in the domain of each variable. Each variable has a domain of the same size.

3 Number of binary constraints. The constraints are chosen at random from a uniform distribution. This number is specified as an integer or as a fraction between 0 and 1. For example, if a problem has 20 variables then the maximum number of constraints is $\frac{20 \times 19}{2}$. A particular problem could be specified with number of constraint equals 95 or 0.5.

4 The tightness of each constraint. All constraints have the same tightness. Tightness refers to the number of value pairs which are disallowed by the constraint. The specific pairs are chosen at random from a uniform distribution. Tightness may be specified either as an integer or as a fraction between 0 and 1. For instance, if a problem has variables with domain size of 5, then the maximum number of value pairs disallowed by a constraint is $5 \times 5 = 25$. A particular problem could be specified with tightness of 5 or 0.2.

Algorithm 6 **Procedure** SelectValue(X_i)

```
Input: None.
Output: returns a value of $X_i$.
1: begin
2: Assigned = {all assigned variable in $S_k$};
3: value = $\{a \in D(X_i) \mid S_k \cup \{X_i = a\}$ is consistent};
4: if (value is found) then
5:     return value;
6: else
7:     return null;
8: end if
9: end
```

Algorithm 7 **Procedure** SelectVariable(S_k, Unassigned)

```
Input: None.
Output: returns a variable from Unassigned.
1: begin
2: Assigned = {all assigned variable in $S_k$};
3: Unassigned = {$X \setminus$ Assigned};
4: $A$ = select a variable from Unassigned;
5: return $A$;
6: end
```

Algorithm 8 **Procedure** MakeMove(S_k, Variable)

```
Input: None.
Output: returns a value of Variable.
1: begin
2: for $v \in D$(Variable) do        // check the confliction corresponding
                                     to each value of Variable
3:     $f$ ← number of conflicts for Variable value $v$ with $S_k$;
       // Calculate the cost function $f$ corresponding to the value $v$ of Variable with $S_k$
4:     BestValue ← value $v$ of Variable with minimum $f$;
5: end for
6: return ($S_k \bigcup$ {Variable = BestValue});
7: end
```

For the sake of simplicity, we have also considered a simple dataset given below as well to illustrate the representative time tables.

1 the data used for the construction of university timetable is taken as

> (Number of Courses, Number of Subjects,
> Number of Teachers, Number of Rooms,
> Periods Per Day, Periods per Week) = (4, 15, 6, 4, 4, 20)

2 the data for Subjects is given as shown in Table 1

3 the data for Courses is given in Table 2

4 the data for Teachers is given in Table 3

5 the data for Periods per day is from

> (9–11), (11–13), (13–15) and (15–17)

6 the notation for the days of the week are in Table 4

7 the data for Periods per week is given in Table 5

8 the data for Rooms is shown in Table 6

9 the data giving information about Subjects given in specific Periods are listed in Table 7

10 the Teacher's preferences for specific periods are listed in Table 8.

Table 1 Data for subjects

Subject ID	*Subject name*	*Number of lectures*	*Number of students enrolled*
M1	MA1	4	28
M2	CS1	5	58
M3	EE1	5	26
M4	ECE1	5	56
M5	BT1	5	29
M6	PHY1	5	28
M7	MA2	5	60
M8	CS2	5	26
M9	EE2	4	24
M10	BT2	5	21
M11	ME1	5	58
M12	ME2	5	27
M13	ECE2	5	65
M14	EE3	5	58
M15	ECE3	5	68

Table 2 Data for courses

Course ID	*Course name*	*Number of subjects*	*Subject ID*
MTech1	Master of Technology 1st semester	4	M1, M2, M3, M4
MTech2	Master of Technology 2nd semester	4	M5, M6, M7, M8
MTech3	Master of Technology 3rd semester	4	M9, M10, M11, M12
MTech4	Master of Technology 4th semester	3	M13, M14, M15

Table 3 Data for teachers

Teacher code	*Teacher name*	*MinL*	*MaxL*	*Number of subjects*	*Subject ID*
DKG	Dinesh Kumar Gupta	4	13	2	M1, M2
SK	Sanjay Kumar	7	12	3	M3, M4, M14
PK	Parimal Kumar	4	10	2	M5, M6
MB	Manohar Biswal	8	16	3	M7, M8, M13
RD	Rupali Datta	6	13	2	M9, M10
UCG	Upendra Chandra Gupta	5	14	3	M11, M12, M15

Table 4 Days of the week

Notation for days	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Day name	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday

Table 5 Data for periods per week

Mon (9–11)	Mon (11–13)	Mon (13–15)	Mon (15–17)
Tue (9–11)	Tue (11–13)	Tue (13–15)	Tue (15–17)
Wed (9–11)	Wed (11–13)	Wed (13–15)	Wed (15–17)
Thu (9–11)	Thu (11–13)	Thu (13–15)	Thu (15–17)
Fri (9–11)	Fri (11–13)	Fri (13–15)	Fri (15–17)

Table 6 Data for rooms

Room name	A	B	C	D
Room capacity	35	60	60	70

Table 7 Information about subject-periods

Subject ID	*Period*	*Subject ID*	*Period*	*Subject ID*	*Period*
M1	Mon (9–11)	M5	Thu (13–15)	M10	Tue (11–13)
M1	Tue (11–13)	M5	Mon (13–15)	M10	Fri(13–15)
M1	Wed (11–13)	M5	Tue (11–13)	M12	Mon (11–13)
M2	Mon (11–13)	M5	Wed (11–13)	M12	Tue (13–15)
M2	Tue (9–11)	M9	Mon (15–17)	M14	Tue (13–15)
M2	Thu (9–11)	M9	Fri (9–11)	M14	Wed (13–15)

Table 8 Data for teachers preferences

Teacher code	DKG	SK	PK	UCG
Preferences	Mon (9–11), Mon (11–13)	Tue (11–13)	Wed (13–15)	Tue (9–11)

Table 9 The general university timetable

	9–11	*11–13*	*13–15*	*15–17*
Monday	(MTech1, M1, DKG, A)	(MTech1, M2, DKG, B)	(MTech2, M5, PK, A)	(MTech3, M9, RD, A)
	(MTech2, M6, PK, B)	(MTech3, M12, UCG, A)	(MTech1, M3, SK, B)	(MTech1, M4, SK, B)
	(MTech3, M11, UCG, C)	(MTech2, M7, MB, C)	(MTech3, M10, RD, C)	(MTech2, M8, MB, C)
	(MTech4, M13, MB, D)	(MTech4, M14, SK, D)	(MTech4, M15, UCG, D)	
Tuesday	(MTech3, M11, UCG, B)	(MTech1, M3, SK, A)	(MTech3, M12, UCG, A)	(MTech1, M4, SK, B)
	(MTech1, M2, DKG, C)	(MTech2, M5, PK, B)	(MTech4, M14, SK, B)	(MTech2, M8, MB, A)
	(MTech2, M6, PK, A)	(MTech3, M10, RD, C)	(MTech1, M1, DKG, C)	(MTech3, M9, RD, C)
	(MTech4, M13, MB, D)	(MTech4, M15, UCG, D)	(MTech2, M7, MB, D)	
Wednesday	(MTech1, M2, DKG, B)	(MTech1, M1, DKG, A)	(MTech2, M5, PK, A)	(MTech1, M4, SK, B)
	(MTech2, M6, PK, A)	(MTech2, M7, MB, B)	(MTech3, M9, RD, B)	(MTech2, M8, MB, A)
	(MTech3, M10, RD, C)	(MTech3, M11, UCG, C)	(MTech1, M3, SK, C)	(MTech3, M12, UCG, C)
	(MTech4, M13, MB, D)	(MTech4, M14, SK, D)	(MTech4, M15, UCG, D)	
Thursday	(MTech1, M2, DKG, B)	(MTech1, M1, DKG, A)	(MTech2, M5, PK, A)	(MTech1, M4, SK, B)
	(MTech2, M6, PK, A)	(MTech2, M7, MB, B)	(MTech1, M3, SK, B)	(MTech2, M8, MB, A)
	(MTech3, M12, UCG, C)	(MTech3, M11, UCG, C)	(MTech3, M10, RD, C)	
	(MTech4, M13, MB, D)	(MTech4, M14, SK, D)	(MTech4, M15, UCG, D)	
Friday	(MTech3, M9, RD, A)	(MTech1, M2, DKG, B)	(MTech3, M10, RD, A)	(MTech2, M8, MB, A)
	(MTech1, M3, SK, B)	(MTech2, M6, PK, A)	(MTech1, M4, SK, B)	(MTech3, M12, UCG, B)
	(MTech2, M5, PK, C)	(MTech3, M11, UCG, C)	(MTech2, M7, MB, C)	
	(MTech4, M13, MB, D)	(MTech4, M14, SK, D)	(MTech4, M15, UCG, D)	

The general university timetable, course wise timetable for one of the course namely MTech2, and the teacher wise timetable for a teacher namely DKG obtained are displayed in Tables 9, 10, and 11 respectively.

Table 10 Timetable for MTech2

	9–11	*11–13*	*13–15*	*15–17*
Monday	(M6, PK, B)	(M7, MB, C)	(M5, PK, A)	(M8, MB, C)
Tuesday	(M6, PK, A)	(M5, PK, B)	(M7, MB, D)	(M8, MB, A)
Wednesday	(M6, PK, A)	(M7, MB, B)	(M5, PK, A)	(M8, MB, A)
Thursday	(M6, PK, A)	(M7, MB, B)	(M5, PK, A)	(M8, MB, A)
Friday	(M5, PK, C)	(M6, PK, A)	(M7, MB, C)	(M8, MB, A)

Table 11 Timetable for DKG

	9–11	*11–13*	*13–15*	*15–17*
Monday	(MTech1, M1, A)	(MTech1, M2, B)		
Tuesday	(MTech1, M2, C)		(MTech1, M1, C)	
Wednesday	(MTech1, M2, B)	(MTech1, M1, A)		
Thursday	(MTech1, M2, B)	(MTech1, M1, A)		
Friday		(MTech1, M2, B)		

Next, the performance of the BT, MCHC and backtrack-free algorithms is measured in terms of CPU time in seconds and the number of iterations required for finding solutions to general university time tabling problems on a number of randomly generated datasets with 73 and 100 variables. The result obtained are summarised in Table 12.

Table 12 Comparison of performance measures

S. no.	*No. of variables*	*Backtracking*		*Min-conflicts-hill-climbing*		*Backtrack-free*	
		CPU time	*No. of iterations*	*CPU time*	*No. of iterations*	*CPU time*	*No. of iterations*
1	73	10.921	319	7.921	231	5.889	173
2	100	37.989	971	23.918	738	16.572	511

It can easily be seen from the the experimental results that our backtrack-free algorithm takes less number of iterations and CPU time in comparison with BT and MCHC algorithms in both general university timetabling problems.

7 Conclusions

A backtrack-free algorithm combining a BT algorithm and a MCHC algorithm is developed for solving general university timetabling problems. This proposed algorithm incrementally constructs a solution as is done by the usual BT algorithm. Whenever a dead-end is encountered, rather than BT to the previously assigned variable, it resolves the cause of dead-end by using the MCHC algorithm and then continues further progressively. If the total number of lecturers, periods, courses, rooms and teachers are represented by TN, p, g, m and n, respectively, then this algorithm always leads to a feasible solutions if $TN \leq p \times min(g, m, n)$. The proposed algorithm is tested on a small dataset and its representative timetable is presented. It is also tested on a number of randomly generated university timetabling problems. The performance of the algorithm measured in terms of the number of iterations and the amount of CPU time in seconds required is compared with the BT algorithm and the MCHC algorithm. It is observed that our algorithm takes less CPU time and less number of iterations. Further, the computational complexities of all the algorithms are calculated and observed that the proposed algorithm has linear complexity in terms of TN. In future, this approach can be modelled for finding an optimal solution by introducing an objective function with the objective of minimising the violation of soft constraints.

Acknowledgements

The authors thank the referees for their valuable comments which have improved the presentation of the paper.

References

Abdullah, S. and Turabieh, H. (2008) 'Generating university course timetable using genetic algorithms and local search', *The 3rd International Conference on Convergence and Hybrid Information Technology ICCIT*, IEEE, Vol. 1, pp.254–260.

Abdullah, S., Shaker, K., McCollum, B. and McMullan, P. (2010) 'Dual sequence simulated annealing with round-robin approach for university course timetabling', *Evolutionary Computation in Combinatorial Optimization*, LNCS, Springer Berlin, Heidelberg, Vol. 6022, pp.01–10.

Alidaee, B., Kochenberger, G.A., Lewis, K. and Lewis, M. (2009) 'Computationally attractive non-linear models for combinatorial optimisation', *International Journal of Mathematics in Operational Research*, Vol. 1, No. 1, pp.9–19.

Bacchus, F. and Van Beek, P. (1998) 'On the conversion between non-binary and binary constraint satisfaction problems', *Proceedings of the 15th National Conference on Artificial Intelligence*, AAAI Press/MIT Press, pp.311–318.

Banks, D., Van Beek, P. and Meisels, A. (1998) 'A heuristic incremental modeling approach to course timetabling', *Advances in Artificial Intelligence*, LNCS, Vol. 1418, pp.16–29.

Burke, E.K. and Petrovic, S. (2002) 'Recent research directions in automated timetabling', *European Journal of Operational Research*, Vol. 140, No. 2, pp.266–280.

Cambazard, H., Hebrard, E., O'Sullivan, B. and Papadopoulos, A. (2012) 'Local search and constraint programming for the post enrolment-based course timetabling problem', *Annals of Operations Research*, Vol. 194, No. 1, pp.111–135.

De Causmaecker, P., Demeester, P. and Vanden Berghe, G. (2009) 'A decomposed metaheuristic approach for a real-world university timetabling problem', *European Journal of Operational Research*, Vol. 195, No. 1, pp.307–318.

Dechter, R. (1990) 'Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition', *Artificial Intelligence*, Vol. 41, No. 3, pp.273–312.

Lewis, R. (2008) 'survey of metaheuristic-based techniques for university timetabling problems', *OR Spectrum*, Vol. 30, No. 1, pp.167–190.

Minton, S., Johnston, M.D., Philips, A.B. and Laird, P. (1992) 'Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems', *Artificial Intelligence*, Vol. 58, No. 1, pp.161–205.

Mushi, A.R. (2012) 'Two phase heuristic algorithm for the university course timetabling problem: the case of University of Dar Es Salaam', *Tanzania Journal of Science*, Vol. 37, No. 1, pp.73–83.

Nadel, B.A. (1989) 'Constraint satisfaction algorithms', *Computational Intelligence*, Vol. 5, No. 3, pp.188–224.

Nandhini, M. and Kanmani, S. (2011) 'Design of GAmut-Lssahc: a solver for course timetabling problem', *International Journal of Mathematics in Operational Research*, Vol. 3, No. 6, pp.595–618.

Pothitos, N., Stamatopoulos, P. and Zervoudakis, K. (2012) 'Course scheduling in an adjustable constraint propagation schema', *ICTAI 2012: 24th International Conference on Tools with Artificial Intelligence*, IEEE, Athens, pp.335–343.

Razak, H.A., Ibrahim, Z. and Hussin, N.M. (2010) 'Bipartite graph edge coloring approach to course timetabling', *International Conference on Information Retrieval & Knowledge Management, (CAMP '10)*, IEEE, pp.229–234.

Rossi, F., Van Beek, P. and Walsh, T. (2006) 'Handbook of constraint programming', *Foundations of Artificial Intelligence*, Elsevier Science Inc., New York, Vol. 35.

Shatnawi, S., Al-Rababah, K. and Bani-Ismail, B. (2010) 'Applying a novel clustering technique based on FP-tree to university timetabling problem: a case study', *in ICCES 2010: 6th International Conference on Computer Engineering and Systems*, IEEE, pp.314–319.

Tiwari, P. and Sharma, N.K. (2011) 'Enumeration of spanning trees of graph: alternative methods', *International Journal of Mathematics in Operational Research*, Vol. 3, No. 2, pp.170–185.

Wijaya, T. and Manurung, R. (2009) 'Solving university timetabling as a constraint satisfaction problem with genetic algorithm', *Proceedings of the International Conference on Advanced Computer Science and Information Systems ICACSIS '09*, Depok.

Wilke, P. and Ostler, J. (2008) 'Solving the school time tabling problem using tabu search, simulated annealing, genetic and branch & bound algorithms', *7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008*, Vol. 1, pp.01–04.

Wong, K.Y. and See, P.C. (2009) 'A genetic ant colony optimisation system (GenANT) for quadratic assignment problems', *International Journal of Mathematics in Operational Research*, Vol. 1, No. 4, pp.456–475.

Yoshikawa, M., Kaneho, K. and Watanabe, Y. (1995) 'A constraint-based approach to high-school timetabling problems: a case study', *Proceedings of the 12th National Conference on Artificial Intelligence*, AAAI Press/MIT Press, pp.1111–1116.

Zhang, J. and Zhang, H. (1996) 'Combining local search and backtracking techniques for constraint satisfaction', *Proceedings of the 13th National Conference on Artificial Intelligence*, AAAI Press/MIT Press, pp.369–374.